ARL-TR-8768 ● JULY 2020

DEVCOM ARMY RESEARCH LABORATORY

Integrated Sensor Architecture (ISA) Database/Media Storage Tool Software Package Documentation Updated

by Jesse Kovach and Laurel Sadler

NOTICES

Disclaimers

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

Citation of manufacturer's or trade names does not constitute an official endorsement or approval of the use thereof.

Destroy this report when it is no longer needed. Do not return it to the originator.

ARL-TR-8768 ● JULY 2020

Integrated Sensor Architecture (ISA) Database/Media Storage Tool Software Package Documentation Updated

by Jesse Kovach and Laurel Sadler
Sensors and Electron Devices Directorate, CCDC Army Research Laboratory

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED (From - To)
July 2020	Technical Report	1 January–31 July 2019

4. TITLE AND SUBTITLE
Integrated Sensor Architecture (ISA) Database/Media Storage Tool Software Package Documentation Updated

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Jesse Kovach and Laurel Sadler

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
CCDC Army Research Laboratory
ATTN: FCDD-RLS-SI
2800 Powder Mill Road
Adelphi, MD 20783-1138

8. PERFORMING ORGANIZATION REPORT NUMBER
ARL-TR-8768

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES
ORCID ID: Laurel C Sadler, 0000-0001-8697-2246

14. ABSTRACT
The Database/Media Storage (DMS) tool is a configurable software package for retrieving and storing media and message data on an Integrated Sensor Architecture (ISA) network. The DMS tool can be configured to store selected message contents to a fully queryable PostgreSQL database and store media data linked from selected ISA messages to disk. This report describes the functions of the DMS tool and provides installation instructions and configuration examples for the tool.

15. SUBJECT TERMS
Integrated Sensor Architecture, sensor networks, unattended ground sensors, data analytics, PostgreSQL

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Jesse Kovach
Unclassified	Unclassified	Unclassified	UU	34	19b. TELEPHONE NUMBER (Include area code) (301) 394-3988

Contents

Summary

The Database/Media Storage (DMS) tool is a configurable software package for retrieving and storing media and message data on an Integrated Sensor Architecture (ISA) network. The DMS tool can be configured to store selected message contents to a fully queryable PostgreSQL database and store media data linked from selected ISA messages to disk. This report describes the functions of the DMS tool and provides installation instructions and configuration examples for the tool.

1. Introduction

1.1 Integrated Sensor Architecture (ISA) Overview

According to the US Army Night Vision and Electronic Sensors Directorate (NVESD), US Army Combat Capabilities Development Command (CCDC) Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance Center (C5ISR),[1]

> [ISA] is a U.S. Army Service-Oriented Architecture (SOA) developed by NVESD. ISA provides capabilities that enable Soldiers to exchange information between their own sensors and those on other platforms in a fully dynamic and shared environment. ISA enables Army sensors and systems to readily integrate into an existing network and dynamically share information and capabilities to improve situational awareness in a battlefield environment.

According to Poltronieri et al.,[2]

> ISA identifies common standards and protocols, which support a net-centric system-of-systems integration. Utilizing a common language, these systems are able to connect, publish their needs and capabilities, and interact dynamically. ISA provides an extensible data model with defined capabilities, and provides a scalable approach across multi-echelon deployments, which when coupled with dynamic discovery capabilities, cybersecurity, and sensor management, provides a system which can adjust and adapt to dynamic environments.

1.2 Database/Media Storage (DMS) Tool Overview

The DMS tool has two primary functions: 1) storing selected ISA message contents in a queryable PostgreSQL[3] database and 2) storing media data linked from selected ISA messages to a file system. When storing media data, the software can optionally serve this stored media via an embedded Web server. It can also optionally generate ISA update messages to inform other applications of the new location/Uniform Resource Locator (URL) for the media and send commands requesting deletion of media stored at the originating sensor. The software can be configured with a custom ISA subscription to only process and store specified messages. Additionally, the DMS tool can be used to store ISA messages to a file in either a human readable format or a binary format.

The DMS tool fills a different role from the Historical Data Service provided by the ISA program office. The Historical Data Service archives binary ISA message data in the same form as originally transmitted and provides an interface for other

ISA components to query and retrieve the archived information. The database storage function of the DMS tool does not store the data in the originally received form, as it is intended to support advanced data-analytics applications and not provide archival storage of message data. The media storage functions of the DMS tool augment the Historical Data Service, as the Historical Data Service does not archive linked media from ISA messages as of this writing. The file-based message-storage functions of the DMS tool are intended to facilitate development and testing of ISA components and do not provide a query/retrieval interface.

1.3 DMS Tool Use Cases

The DMS tool can be used in a number of different applications. Primary uses include the following:

- Storage of media items sent via ISA. As discussed, ISA messages containing media do not actually contain the media data; instead, they contain a "Resource Descriptor" type that includes a URL pointing to data residing on a Web server.[4] Tools that record ISA message data will store these URLs but will not store the media data, as they are not actually included in the message. The DMS tool can be configured to download and save the data pointed to by resource descriptor URLs in any media messages that it receives, making it a useful complement to message-archiving tools in scenarios where media data need to be stored as well.
- "Rehosting" of media data. Since ISA media messages contain URLs, a Web server must be provided to serve the data. Typically, this Web server is hosted on the same system as the ISA component producing the media. In some scenarios, storing imagery on or retrieving imagery from the producing system may not be desirable. For example, the producing system may not be located in a secure area, network security policy may restrict Hypertext Transfer Protocol (HTTP) or Hypertext Transfer Protocol Secure (HTTPS) connections to the producing system, or the communications link to the producing system may be disadvantaged, intermittent, or limited. In these cases the media data must be retrieved from the producing system and rehosted elsewhere. The DMS tool includes an embedded Web server that will serve the images that it has archived and can be configured to send ISA message updates using the standard ISA Event Reference mechanism[5] to inform other applications of the new URL for the archived image. Additionally, it can be configured to send a custom Delete Resource command to the image producer after the image has been successfully

archived. Data producer components can implement this command to delete their local copy of the data after it has been successfully archived.

- Storage of ISA messages in a queryable database. This is useful for sensor system testing, sensor performance analysis, data fusion applications, and many other scenarios where the full power of a Structured Query Language (SQL) database is needed for processing and analysis of sensor data.
- Storage of ISA messages in files on disk. This is useful for sensor system testing and sensor performance analysis or when a database application is unavailable.
- Storage of ISA messages in files on disk in a binary format. This is useful for sensor system testing, sensor performance analysis and evaluation of complex sensor systems using simulated or recorded data when access to actual sensors is inhibited.

1.4 Usage within the US Army Combat Capabilities Development Center (CCDC) Army Research Laboratory (ARL) Networked Sensor Research Test Bed

The DMS tool is used to collect and store test data within the CCDC Army Research Laboratory Networked Sensor Research Test Bed (NSRTB). The NSRTB, an extension of the ARL Campus Sensor Network, is an environment designed for developing and testing new sensing and data/video analytics capabilities for tactical applications. It incorporates commercial off-the-shelf, government off-the-shelf (GOTS), and experimental sensor systems and software. All components of the NSRTB are interconnected using ISA.

Sensors currently employed within the NSRTB include Axis Communications Q1615-E MkII surveillance cameras, EPC (Electronic Product Code) Gen2 RFID (radio frequency identification) readers from multiple vendors, TRSS (Tactical Remote Sensor System) seismic/acoustic sensors, and experimental sensor systems under development by ARL. Data management and monitoring software within the NSRTB includes the RaptorX mapping system, multiple GOTS video management systems, the ISA diagnostics package, and custom visualization applications developed by ARL engineers. The NSRTB is being used to support the development of data and video analytics systems for multiple government sponsors.

2. DMS Tool Installation Instructions

2.1 Assumptions

This report assumes the user has a working knowledge of basic ISA component configuration and ISA certificate use. See the ISA software development kit documentation for more information regarding these topics.

2.2 Prerequisites/System Requirements

The DMS tool has the following system requirements and prerequisites:

- A Java Runtime Environment (JRE) for Java 8 must be installed. The software has been built and tested with OpenJDK 8. It should also function with Oracle JRE 8, but this configuration has not been tested by the authors. At the time of this writing, use of Oracle JRE 8 in a production environment requires a paid license.
- PostgreSQL 11 must be installed if the database functions are used. PostgreSQL is not needed for the media storage function.

2.3 Installation Procedure

1) Obtain an ISA certificate and private key for this application. The certificate needs to be provided as a `.jks` file. Generating/obtaining certificates is outside the scope of this document; contact your local ISA system administrator for assistance.
2) Extract the distribution zip file to a suitable location (e.g., `c:\isa\database-archiver`). The path to the installation folder cannot contain spaces.
3) Copy the certificate file from Step 1 into the folder you just created.
4) Change into the folder created in Step 2. Copy the `connection_sample.properties` file to `connection.properties`.
5) Open `connection.properties` in a text editor. Change the settings in the file as needed for your application (see Section 3). Save the file and exit the editor.

On a Windows platform, the Database Archiver tool can optionally be configured to run as a Windows service. To do so, perform the following steps:

1) Perform all steps of the installation procedure.

2) Open the install_service.bat file in a text editor. Locate the line beginning

```
set LOCAL_JAVA_HOME=
```

3) Edit this line to specify the path to your JRE installation. If you have a full Java Development Kit (JDK) installation, make sure you specify the path to the JRE contained within the JDK and not to the JDK itself. For example, if you have OpenJDK installed in `c:\openjdk\jdk8u172-b11`, the setting should read

   ```
   set LOCAL_JAVA_HOME=C:\openjdk\jdk8u172-b11\jre\
   ```

   Alternatively, if the `JAVA_HOME` environment variable is properly set on your system, you can skip this step and the script will use the value of `JAVA_HOME`.

4) Ensure that the `NT AUTHORITY\NETWORK SERVICE` account has "Full Control" permissions to the installation folder. The simplest way to do this is to open an administrator command prompt and run the following command:

   ```
   icacls "<path-to-installation-folder>" /grant "NT AUTHORITY\NETWORK SERVICE":(OI)(CI)F
   ```

5) Install the service by opening an administrator command prompt and running `install_service.bat.` The script will attempt to remove any previously installed versions of the service prior to installing. If the service was not previously installed, you will see errors related to this. These errors can be safely ignored.

6) Verify that the service is running by opening the Windows service control panel and verifying that the service is shown as "Running". The name of the service will be the name of the installation folder.

If the service does not start after installation, check the logs in the "ServiceLogs" folder for error messages.

The service may be uninstalled by opening an administrator command prompt and running `install_service.bat /u`.

2.4 Starting the Software

On all platforms, the DMS tool can be started using the following command:

```
java -jar mil.arl.apps.isaToSensorDB.jar
```

Alternatively, on Windows the software can be started using the included `run.bat` script.

3. DMS Tool Configuration Instructions

This section discusses each of the configuration settings in the `config.properties` file, including how they are set and what they mean. This will allow the user to configure the software to accomplish exactly what the user requires and no more.

`isaControllerHost`: IP address or hostname of the ISA controller.

`isaControllerPort`: Port of the ISA controller.

`isaControllerUCI`: Universal Component Identifier (UCI) of the ISA controller. If this does not match the UCI configured on the controller, the software will not connect.

`myUCI`: UCI to use for this instance of the DMS tool. It must match the name on the certificate being used.

`isaKeystorePath`: Path/file name of the keystore (`.jks`) file containing the ISA certificates to use. The names embedded in the certificates contained within the file must match the UCI being used. Generally, the name of this file will match the UCI.

`isaKeystorePassword`: Password for the keystore file.

`isaKeyPassword`: Password for the private keys in the keystore file. This is usually the same as the keystore password.

`isaEncoding`: ISA protocol encoding to use. This should always be set to `ipl_3v7` unless special circumstances apply.

`isaEnableTLS`: If `true`, the software will use transport layer security (TLS) authentication and encryption for the ISA connection. This should always be `true` unless special circumstances apply.

`isaSubscription`: Subscription that selects the type of ISA messages that will be processed by the application and stored in the database (if enabled). This is in standard ISA Query Language (IQL) format. Currently, the same subscription is used for the media storage function, the database function, and the delete-from-source function. Future versions will allow a different subscription for each.

Some example subscriptions follow:

- To receive all admin, event, status, and config messages:
  - `isaSubscription=select admin, event, status, config`
- To receive config and event messages only (no admin or status):
  - `isaSubscription=select config, event`
- To receive all config messages, and events containing detections:
  - `isaSubscription=select config, event matching EventType{\"DETECTION\"} == type()`
- To receive only events containing detections:
  - `isaSubscription=select event matching EventType{\"DETECTION\"} == type()`

For more information on IQL syntax, see the IQL documentation.[6]

`mediaLocation`: Location of the folder in which to save archived media files. The tool will create additional folders under this folder to store media from different sources.

`enableDatabase`: If `true`, the database function will be enabled and received messages will be stored to a PostgreSQL database in a queryable manner. If `false`, the database function will not be used.

`enableStoreMedia`: If `true`, the media retrieval and storage function will be enabled and media items referenced from received messages will be stored in the folder specified in `mediaLocation`.

`enableWriteMessageToFile`: If this parameter is true, the human readable text format of the ISA messages received by the component will be written to a file as defined in `txtFileLocation`. Other capabilities of the component may also be performed if they have been enabled.

`enableWriteBinaryMessageToFile`: If this parameter is true, the binary format of the ISA messages received by the component will be written to a file as defined in `txtFileLocation`.

`enableWriteMessageToFileOnly`: If this parameter is true, the component will only write the ISA messages received by the component to a file as defined in `txtFileLocation`. No other capabilities of the component will be performed.

`txtFileLocation`: This parameter defines the location and name of the output text and binary files when writing messages to a file is enabled.

`enableJettyServer`: If `true`, start an embedded Jetty Web server to serve the archived media files.

The following options only apply if `useJettyServer` is `true`:

> `httphostOverride`: Listen IP address for the embedded Jetty Web server. Set this to an externally accessible IP of the machine. This IP address will also be used as the host subcomponent of any media URLs generated by this system. (The correct IP to use cannot be reliably autodetected, so it must be set by hand.)
>
> `httpPort`: Port used for the Jetty Web server. If this parameter is set to –1, HTTP will be disabled.
>
> `httpsPort`: Port used for the Jetty Web server secure. If this parameter is set to –1, HTTPS will be disabled.
>
> `useHttpsAsPrimary`: This parameter controls whether the URLs sent by this component over ISA will be HTTP URLs or HTTPS URLs. If this parameter is set to false, the HTTP URL will be sent; otherwise, if it is set to true, the HTTPS URL will be sent.
>
> `httpsKeystorePath`: Sets the keystore path containing the HTTPS certificates.
>
> `httpsKeyPassword`: Sets the key password for the HTTPS certificates. If this is left blank, the ISA certificate for Key will be used.
>
> `httpsKeyStorePassword`: Sets the keystore password for the HTTPS certificates. If this is left blank, the ISA certificate for Key Store will be used.
>
> `disableHttpsCertificateValidation`: Setting this parameter to true disables the HTTPS certificate validation when downloading media. This is necessary if the media producers do not have certificates that were issued by a JRE-trusted certificate authority.
>
> `sendnewURL`: If `true`, send an updated ISA message of type "OTHER" containing the URLs to the new locations of the media files.
>
> `sendDeleteMediaAtSiteCommand`: If `true`, send a custom ISA "Delete Resource" command back to the device from which the media was received to delete the media at the originating site. This is for specific applications. Most components do not implement this command.

The following options only apply if `enableDatabase` is `true`:

`databaseServer`: Name of the PostgreSQL server to use (usually `localhost`).

`databaseName`: Name of the PostgreSQL database to use. This database will be created if it does not already exist.

`databaseUser`: PostgreSQL database username. This should usually be "`postgres`" (the default database-administrative user). Using a nonadministrative user is possible but beyond the scope of this report.

`databasePassword`: Password for the PostgreSQL user. Set to match your PostgreSQL configuration.

`databaseLocale`: Locale setting to use when creating new database. The allowed values for this setting will vary with your platform and PostgreSQL configuration, and incorrect values will cause database creation to fail. For Windows installations of PostgreSQL with default settings, use

```
English_United States.1252
```

For Ubuntu packaged versions of PostgreSQL with default settings, use

```
en_US.UTF-8
```

4. Example Usage Scenarios

This section provides example configurations for different DMS tool use scenarios.

4.1 Store Media Only

This configuration does not connect to a database. It subscribes to the ISA controller for event messages of type "Detection" with observables containing media. The media will be stored on the local file system.

This is accomplished by setting

```
isaSubscription=select event matching exists(
  value: /get_event()/observables[/name=="Media"] )
```

and

```
enableStoreMedia=true
```

and

```
enableDatabase=false
```

and

```
sendnewURL=false
```

and

```
enableJettyServer=false
```

and

```
sendDeleteMediaAtSiteCommand=false
```

4.2 Store Media and Send Update Message with New URL

This configuration does not connect to a database. It subscribes to the ISA controller for config messages and event messages of type "Detection" with observables containing media. The media will be stored on the local file system and made available through the embedded Web server. The tool will send ISA update messages with the new URLs for the media.

This is accomplished by setting

```
isaSubscription=select event matching exists(
  value: /get_event()/observables[/name=="Media"])
```

and

```
enableStoreMedia=true
```

and

```
enableDatabase=false
```

and

```
sendnewURL=true
```

and

```
enableJettyServer=true
```

and

```
sendDeleteMediaAtSiteCommand=false
```

4.3 Store Media, Send Update Message with New URL, and Delete Media from Source

This configuration does not connect to a database. It subscribes to the ISA controller for config messages and event messages of type "Detection" with observables containing media. The media will be stored on the local file system and made available through the embedded Web server. The tool will send ISA update

messages with the new URLs for the media and will send an ISA command to the originating component requesting that the media be deleted from the originator site.

The ISA config messages are needed to send the delete command to the originating component, so the subscription used must include config messages in addition to Event messages containing media of interest.

This is accomplished by setting

```
isaSubscription=select config, event matching
  !exists(value: /get_event()) || exists( value:
  /get_event()/observables[/name=="Media"])
```

and

```
enableStoreMedia=true
```

and

```
enableDatabase=false
```

and

```
sendnewURL=true
```

and

```
enableJettyServer=true
```

and

```
sendDeleteMediaAtSiteCommand=true
```

4.4 Store Media and Add Messages Received to the Database

In this mode the software attempts to connect to the database specified by `databaseServer` and `databaseName` in the configuration file. If the database connection fails, the software assumes the database does not exist and attempts to create a new database with the provided name.

This configuration subscribes to the ISA controller for the following message types: admin, event, status, and config. All received messages are submitted to the database, and all media is retrieved and stored on the local file system.

This is accomplished by setting

```
isaSubscription=select admin, event, status, config
```

and

```
enableStoreMedia=true
```

and

`enableDatabase=true`

and

`sendnewURL=false`

and

`enableJettyServer=false`

and

`sendDeleteMediaAtSiteCommand=false`

4.5 Store Media, Add Message to Database, and Send Update Message with New URL

In this mode the software attempts to connect to the database specified by `databaseServer` and `databaseName` in the configuration file. If the database connection fails, the software assumes the database does not exist and attempts to create a new database with the provided name.

This configuration subscribes to the ISA controller for the following message types: admin, event, status, and config. All received messages are submitted to the database. All media will be stored on the local file system and made available through the embedded Web server. The tool will send ISA update messages with the new URLs for the media.

This is accomplished by setting

`isaSubscription=select admin, event, status, config`

and

`enableStoreMedia=true`

and

`enableDatabase=true`

and

`sendnewURL=true`

and

`enableJettyServer=true`

and

`sendDeleteMediaAtSiteCommand=false`

4.6 Store Media, Add Message to Database, Send Update Message with New URL, and Delete Media from Source

In this mode the software attempts to connect to the database specified by `databaseServer` and `databaseName` in the configuration file. If the database connection fails, the software assumes the database does not exist and attempts to create a new database with the provided name.

This configuration subscribes to the ISA controller for the following message types: admin, event, status, and config. All received messages are submitted to the database. All media will be stored on the local file system and made available through the embedded Web server. The tool will send ISA update messages with the new URLs for the media, and will send an ISA command to the originating component requesting that the media be deleted from the originator site.

This is accomplished by setting

`isaSubscription=select admin, event, status, config`

and

`enableStoreMedia=true`

and

`enableDatabase=true`

and

`sendnewURL=true`

and

`enableJettyServer=true`

and

`sendDeleteMediaAtSiteCommand=true`

4.7 Write ISA Messages to a File

The DMS tool can be used solely for the purpose of writing the ISA messages to a file by setting the `enableWriteMessageToFileOnly` to true in the configuration file. However, this capability may also be used in combination with any or all of the previously described capabilities (Sections 4.1–4.6) by setting `enableWriteMessageToFileOnly` to false.

In this mode the software attempts to write the ISA messages to a file location specified by `txtFileLocation` in the configuration file. If the designated file path does not exist, the software attempts to create the user-defined path.

When `enableWriteMessageToFile` is true, the human readable text format of the ISA messages received by the component will be written to a file as defined in `txtFileLocation`. The messages are further sorted into folders based upon their message type. For example, a message of type Event would be stored in a file located in the `txtFileLocation`/Event and a message of type Status would be stored a file located in `txtFileLocation`/Status.

When `enableWriteBinaryMessageToFile` is true, the binary format of the ISA messages received by the component will be written to a file as defined in `txtFileLocation` with the subdirectory BinaryXXXXX, where XXXXX is an integer starting with 00000, that is incremented for each additional Binary folder that is created. A new folder is created after every 10,000 messages. All messages types are stored together in the same folder. The first 10,000 messages will be written to the folder `txtFileLocation`/Binary00000.

The following configuration only writes ISA messages to a file. It subscribes to the ISA controller for the following message types: admin, event, status, and config. All received messages are written to both human readable files and binary files.

This is accomplished by setting

```
isaSubscription=select admin, event, status, config
```

and

```
enableStoreMedia=false
```

and

```
enableDatabase=false
```

and

```
sendnewURL=false
```

and

```
enableJettyServer=false
```

and

```
sendDeleteMediaAtSiteCommand=false
```

and

```
enableWriteMessageToFileOnly=true
```

and

`enableWriteMessagesToFile=true`

and

`enabledWriteBinaryMessagesToFile=true`

and

`txtFileLocation=C:/ISAMessages`

5. Message Database Structure

The message database stores ISA messages and therefore directly follows the structure of the ISA messages with additional columns for easy/rapid queries.

The database is divided into tables based on the individual ISA message types. The database contains five tables: `administration`, `config`, `configconverged`, `event`, and `status`. Note there is a `config` and a `configconverged` table although a message of type `configconverged` does not exist. As per the philosophy of ISA, when a configuration message is updated, only a partial configuration message containing the changes is sent. This partial configuration message is stored in the `config` table exactly as received. This config message is then merged with the previous config message using `withCcd` from the standard ISA libraries, and the resulting full configuration is stored in the `configconverged` table.

The ISA tables are then broken into columns based on the ISA message composite data type. Generally, these composite data types are converted from their native ISA NameValuePair representation and stored in the database in JavaScript Object Notation (JSON) format. However, there are additional columns in each table for database record keeping/maintenance (`tableid`, `databaseguid`, and `receivertime`). These allow for easy database queries on items within composite data types that are often queried (such as location and timestamp values) as well as human readable values for ease of the user. The user can use SQL to query the database and can also query into the JSON-formatted composite-data types using the PostgreSQL JSON query functions.

The ISA message tables and columns are described in the following sections. Columns marked with an asterisk (*) contain data directly from the ISA message. The definitions for these columns are drawn from the ISA Data Model Specification.[3] While summary definitions are included here for informative purposes, the definitions in the data model specification should be regarded as

authoritative. Columns without an asterisk (*) are locally generated by the database tool and are defined as specified in this document.

Certain fields such as `priority` and `source` appear in multiple message types with identical names and definitions. The ISA Data Model Specification does not contain common definitions for these fields; rather, the fields are defined separately in each message where they appear. This structure is paralleled within the database design and this report.

Unless otherwise noted, numeric date/time values are stored in the database as the number of milliseconds elapsed since midnight 1 January 1970 Coordinated Universal Time (UTC) (i.e., Java time).

5.1 Administrative Table

This table contains the administrative messages, which share messages about the ISA network as a whole. It is the mechanism for announcing when a device's connection state changes or when anything noteworthy occurs on the network.

1) *`priority` – `bigint`: Optional value in the ISA message. Priority of the given message. Zero is the highest. If not provided, the default value is 80.
2) *`stale` – `bigint`: Optional value in the ISA message. UTC time at which data contained in the message should be regarded as invalid.
3) *`source` – `character varying`: Required value in the ISA message containing the UCI of the message source.
4) *`identifier` – `bigint`: Required value in the ISA message describing a conversation-unique identifier for the message.
5) *`code` – `character varying`: Required value in the ISA message that defines what the message is about.
6) *`about` – `jsonb`: One or more UCIs describing who the message is about.
7) *`extras` – `jsonb`: Zero or more Name Value Pair. This is a container to be used for passing arguments or defining properties. Named value representing the new value for the pair.
8) *`messagetime` – `bigint`: Message date/time from the ISA message.
9) `receivetime` – `bigint`: Date/time at which the message was received by the database tool.

10) `tableid` – `bigint`: Auto-incrementing integer identifying the row of data in the table.

11) `databaseguid` – `character varying`: Unique identifier used for data message identification to avoid multiple copies of data being added to the database.

12) `stalestring` – `character varying`: Human readable staletime value.

13) `receivetimestring` – `character varying`: Human readable receivetime value.

14) `messagetimestring` – `character varying`: Human readable messagetime value.

5.2 Config Table

This table contains messages that are used by a component to describe itself to the other components in a deployment. The database breaks the ISA Component Capability Declaration (CCD) into four separate columns (`properties`, `commands`, `observables`, and `customtypes`) for easy queries.

1) *`priority` – `bigint`: Optional value in the ISA message. Priority of the given message. Zero is the highest. If not provided, the default value is 80.

2) *`source` – `character varying`: Required value in the ISA message containing the UCI of the message source.

3) *`identifier` – `bigint`: Required value in the ISA message describing a conversation-unique identifier for the message.

4) *`properties` – `jsonb`: Zero or more Property Declarations. Describes the components properties. Part of the CCD.

5) *`commands` – `jsonb`: Zero or more Command Declarations. Describes the commands that the component handles. Part of the CCD.

6) *`observables` – `jsonb`: Zero or more Observables Declarations. Describes the observables that the component publishes. Part of the CCD.

7) *`propertystates` – `jsonb`: Zero or more Property States. The current usable state of the properties.

8) *`observablestates` – `jsonb`: Zero or more Observable States. The current usable state of the observables.

9) *commandstates - jsonb: Zero or more Command States. The current usable state of the commands.

10) *extras – jsonb: Zero or more Name Value Pair. Additional data that can be optionally attached to a config message.

11) *customtypes – jsonb: Zero or more custom types. Describes types unique to the component. Part of the CCD.

12) *messagetime – bigint: Message date/time from the ISA message.

13) receivetime – bigint: Date/time at which the message was received by the database tool.

14) tableid – bigint: Auto-incrementing integer identifying the row of data in the table.

15) databaseguid – character varying: Unique identifier used for data message identification to avoid multiple copies of data being added to the database.

16) receivetimestring – character varying: Human readable receivetime value.

17) messagetimestring – character varying: Human readable messagetime value.

18) staletimestring – character varying: Human readable staletime value.

5.3 Configconverged Table

This table contains the entire configuration file for each sensor after it has been merged with the changes from each config message. The column definitions for this table are identical to the Config Table described in Section 5.2.

5.4 Event Table

This table contains component or controller-published declared observables.

1) *priority – bigint: Optional value in the ISA message. Priority of the given message. Zero is the highest. If not provided, the default value is 80.

2) *source – character varying: Required value in the ISA message containing the UCI of the message source.

3) *identifier – bigint: Required value in the ISA message describing a conversation-unique identifier for the message.

4) *stale – bigint: Optional value in the ISA message. UTC time at which data contained in the message should be regarded as invalid.

5) *messagetime – bigint: Message date/time from the ISA message.

6) *eventsymbol – character varying: MIL-STD-2525[7] symbol code for the entity described in the message. From the identity observable in the ISA message.

7) *eventid – bigint: Required value containing an integer key, unique per component, that allows that event to be updated and referred to at a later point.

8) *type – character varying: Required value describing the reason the event was created. For example: Detection, Alert, Alarm, and Other. Default is Other.

9) *observables – jsonb: Zero or more ISA name value pairs; set of fields published within any event message.

10) *detectorproperties – jsonb: Zero or more ISA name value pairs. Properties of the detector when the event occurred if those properties are different from the last status.

11) *media – jsonb: Media items linked from the ISA message. Removed from observables and placed in its on column for easy queries.

12) *eventrefid – bigint: Optional field containing part of a unique ID that can be used to update previous event information. It is the Event ID of the original event.

13) *eventrefcreator – character varying: Required if eventrefid is present. UCI of the creator of the original event.

14) *eventreftime – bigint: Required if eventrefid is present. The time of the original event.

15) *eventlat – double precision: Latitude (World Geodetic System [WGS84] decimal degrees[*]) of the entity at the point at which it was observed. The observation could be a detection or a measurement. The

[*] The ISA data model specification is ambiguous regarding the coordinate system used to represent geographic positions. In practice, WGS84 with the EGM96 gravitational model is used.

entity could be visible (e.g., tank, mortar) or invisible (e.g., signal, atmosphere). From the Position *observable* in the ISA message.

16) `*eventlon` – `double precision`: Longitude (WGS84 decimal degrees) of the entity at the point at which it was observed. From the Position *observable* in the ISA message.

17) `*eventalt` – `double precision`: Altitude (meters above Earth Gravitational Model 96 [EGM96] mean sea level) of the entity at the point at which it was observed. From the Position *observable* in the ISA message.

18) `*detectorlat` – `double precision`: Altitude of the ISA component at the time the observation was produced. From the Position *detector property* in the ISA message.

19) `*detectorlon` – `double precision`: Longitude of the ISA component at the time the observation was produced. From the Position *detector property* in the ISA message.

20) `*detectoralt` – `double precision`: Altitude of the ISA component at the time the observation was produced. From the Position *detector property* in the ISA message.

21) `*detectorsymbol` – `character varying`: MIL-STD-2525[7] symbol code for the detector. From the `identity` *detector property* in the ISA message.

22) `*bso` – `character varying`: Unique identifier assigned by the component to a confirmed or suspected entity on the battlefield.

23) `receivetime` – `bigint`: Date/time at which the message was received by the database tool.

24) `tableid` – `bigint`: Auto-incrementing integer identifying the row of data in the table.

25) `Databaseguid` – `character varying`: Unique identifier used for data message identification to avoid multiple copies of data being added to the database.

26) `receivetimestring` – `character varying`: Human readable receivetime value.

27) `messagetimestring` – `character varying`: Human readable messagetime value.

28) `staletimestring` – `character varying`: Human readable staletime value above.

29) `imagefilename` – `text []`: Contains a list of file paths identifying the location (on the local file system) of the stored media acquired from the media URLs provided in the ISA message.

30) `distance` – `double precision`: The distance from the observing component to the observed entity.

31) `bearingyaw` – `double precision`: Derived from the line of bearing representing the rotation from the current orientation of the observing component to the observed entity. Yaw represents the angle about the z-axis.

32) `bearingpitch` – `double precision`: Derived from the line of bearing representing the rotation from the current orientation of the observing component to the observed entity. Pitch represents the angle about the y-axis.

5.5 Status Table

This list contains messages used for sending periodic updates on component status. The frequency of these messages is based on the thresholds designated by property values as well as the status interval property.

1) *`priority` – `bigint`: Optional value in the ISA message. Priority of the given message. Zero is the highest. If not provided, the default value is 80.

2) *`source` – `character varying`: Required value in the ISA message containing the UCI of the message source.

3) *`identifier` – `bigint`: Required value in the ISA message describing a conversation-unique identifier for the message.

4) *`stale` – `bigint`: Optional value in the ISA message. UTC time at which data contained in the message should be regarded as invalid.

5) *`messagetime` – `bigint`: Message date/time from the ISA message.

6) *`properties` – `jsonb`: Zero or more property states. The current state of the properties.

7) `latitude` – `double precision`: Optional field containing the latitude (WGS84 decimal degrees) of the component from the `position` property in the ISA message.

8) `longitude – double precision`: Optional field containing the longitude (WGS84 decimal degrees) of the component from the `position` property in the ISA message.

9) `altitude – double precision`: Optional field containing the altitude (meters above EGM96 mean sea level) of the component from the `position` property in the ISA message.

10) `sensorsymbol – character varying`: MIL-STD-2525 symbol code for the component. From the `identity` property in the ISA message.

11) `receivetime – bigint`: Date/time at which the message was received by the database tool.

12) `tableid – bigint`: Auto-incrementing integer identifying the row of data in the table.

13) `databaseguid – character varying`: Unique identifier used for data message identification to avoid multiple copies of data being added to the database.

14) `receivetimestring – character varying`: Human readable receivetime value.

15) `messagetimestring – character varying`: Human readable messagetime value.

16) `staletimestring – character varying`: Human readable staletime value.

5.6 Sample Queries

The PostgreSQL query language can be used to match individual elements within a JSON field. The following syntax can be used for JSON matching:

```
select * from table where column @> '[desired match inside the json]'
```

For example, when looking for the JSON object

```
"[{"url":"http://192.168.1.1:8001/obs/afc3141e-7f06-4a15-bada-a11327eec4c4/media","mimetype":"jpeg","mediaType":"image"}]"
```

the following queries will all match:

```
SELECT * from event where media @> '[{"url":"http://
192.168.1.1:8001/obs/afc3141e-7f06-4a15-bada-
a11327eec4c4/media","mimetype":"jpeg",
"mediaType":"image"}]'

SELECT * from event where media @> '[{"url":"http://
192.168.1.1:8001/obs/afc3141e-7f06-4a15-bada-
a11327eec4c4/media"}]'

SELECT databaseguid, imagefilename from event where
media @> '[{"url":"http://
192.168.1.1:8001/obs/afc3141e-7f06-4a15-bada-
a11327eec4c4/media"}]'
```

Many other JSON queries are possible. For more information, see the PostgreSQL documentation.[8] Note that unlike many other database systems, PostgreSQL query syntax is case-sensitive.

6. References

1. Integrated Systems Architecture (ISA). Fort Belvoir (VA): Army Night Vision and Electronic Sensors Directorate, US Army Combat Capabilities Development Command (CCDC) Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center (US); n.d. [accessed 2019 July 18]. https://confluence.di2e.net/display/ISA/.

2. Poltronieri F, Sadler L, Benincasa G, Gregory T, Harrell JM, Metu S, Moulton C. Enabling efficient and interoperable control of IoBT devices in a multi-force environment. Proceedings of the 2018 IEEE Military Communications Conference (MILCOM); 2018 Oct 29–31; IEEE, c2019. p. 757–762.

3. The PostgreSQL Global Development Group. PostgreSQL. Ver. 11; 2018 [accessed 2019 June 25]. https://www.postgresql.org/.

4. ISA data model specification, release 6.0, document revision 7. Fort Belvoir (VA): US Army Night Vision and Electronic Sensors Directorate, Army Combat Capabilities Development Command (CCDC) Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center (US); 2019 Jan 16.

5. OTTO – BSOs and event references. Fort Belvoir (VA): Army Night Vision and Electronic Sensors Directorate, US Army Combat Capabilities Development Command (CCDC) Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center (US); 2016 Dec 20.

6. ISA data query language user's guide, release 6.0, document revision 3. Fort Belvoir (VA): US Army Night Vision and Electronic Sensors Directorate, Army Combat Capabilities Development Command (CCDC) Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance (C5ISR) Center (US); 2016 Sep 6.

7. MIL-STD-2525C. Common warfighting symbology. Washington (DC): Department of Defense (US); 2008 Nov 17.

8. The PostgreSQL Global Development Group. JSON functions and operators. [accessed 2019 July 25]. https://www.postgresql.org/docs/11/functions-json.html.

List of Symbols, Abbreviations, and Acronyms

ARL	Army Research Laboratory
C5ISR	Command, Control, Computers, Communications, Cyber, Intelligence, Surveillance and Reconnaissance
CCD	Component Capability Declaration
CCDC	US Army Combat Capabilities Development Command
DMS	Database/Media Storage
EGM	Earth Gravitational Model
EPC	Electronic Product Code
GOTS	government off-the-shelf
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IP	Internet Protocol
IQL	ISA Query Language
ISA	Integrated Sensor Architecture
JDK	Java Development Kit
JRE	Java Runtime Environment
JSON	JavaScript Object Notation
NSRTB	Networked Sensor Research Test Bed
NVESD	Night Vision and Electronic Sensors Directorate
SOA	Service-Oriented Architecture
SQL	Structured Query Language
TLS	transport layer security
TRSS	Tactical Remote Sensor System
UCI	Universal Component Identifier
URL	Uniform Resource Locator
UTC	Coordinated Universal Time

WGS	World Geodetic System

1 (PDF)	DEFENSE TECHNICAL INFORMATION CTR DTIC OCA
1 (PDF)	CCDC ARL FCDD RLD CL TECH LIB
2 (PDF)	CCDC ARL FCDD RLS SI J KOVACH L SADLER
2 (PDF)	CCDC C5ISR NVESD C MOULTON M HARRELL