Intelligent Ship Artificial Intelligence Network (ISAIN)

Progeny Task 25: WP 2.1

Developer Guide

Issue: 4.0

Document Reference: SSL/11145/DOC/0004

Conditions Of Supply - Full Rights

The document is supplied to MOD as a FULL RIGHTS VERSION under the terms of DEFCON 703 (Edn 11/02), with ownership of the outputs herein vested in the Authority.

Authorisation

Role	Name(s)
Author	Carl Froom, Jacob Pennels, Ross Walker
Reviewed	Mark Burnett, Emlyn Purvis, James Bloor, Adam Mo, Matt Fisher
Authorised for Release	Mark Burnett

Distribution

Copy Number(s)	Recipient
1	Dstl
2	CGI

Revision History

Version	Date	Author	Description
0.1	27/02/2020	Carl Froom	Initial draft
0.2	25/03/2020	Carl Froom, Jacob Pennels	Updated after internal review
0.3	04/05/2020	Carl Froom	Updated after internal review; Completed section 10
0.4	05/05/2020	Carl Froom	Updated after internal review; Added section 5.4
1.0	06/05/2020	Carl Froom	Added section 10.1.1; Initial release
1.1	29/05/2020	Carl Froom	Added section 5
2.0	05/05/2020	Carl Froom	Added section 11.1.2; Updated after internal review
3.0	13/07/2020	Carl Froom	Added references to the OACS and OARIS data formats as part of ISAIN data format in section 2 and 5; Updated the architecture overview diagram in section 2; Added appendices C and D
4.0	08/02/2022	Ross Walker	Updated to reflect status at delivery of Phase 2

Table of contents

1 Introduction

1.1 Background

The Defence Science and Technology Laboratory (Dstl) has embarked on an Intelligent Ship programme, which will revolutionise ship design by harnessing automation and Artificial Intelligence (AI) to transform naval doctrine.

The Intelligent Ship Artificial Intelligence Network (ISAIN) will provide Dstl with a framework to support a programme of experimentation with AI collaboration and human-machine teaming. This will act as a ‘playground’ for AIs: a ‘sandpit’ to support inter-relationships between applications and human users, with the focus on demonstrating innovative, challenging and revolutionary concepts and opportunities

1.2 Purpose

This document is aimed at software developers intending to integrate a new or existing ship system or AI, (collectively referred to as applications throughout this document) into ISAIN. It describes the development, build and deployment steps required to add a new input processor, output processor or data transformer. It assumes the reader is familiar with the software stack described in section 3 and refers to each stack component’s own documentation for further detailed information where appropriate.

1.3 References

ID	Reference	Title	Version	Date
1	OACS/TD/35	OACS Data Fusion IFS (Dstl GFI)	1	19/01/2016
2	formal/2016-03-02	Open Architecture Radar Interface Standard (OARIS) (https://www.omg.org/spec/OARIS/1.0)	1.0	March 2016

Table 1 - References

2 Overview

ISAIN provides a means to interconnect between disparate ship systems, humans and AIs, building upon the Apache NiFi open source data flow engine, MongoDB and Docker.

Integration of a new application is achieved through configuration of a NiFi flow to include input and output processors to connect to the new application, and the addition of appropriate data converters to/from ISAIN's internal data model. The internal data model currently in use is the Open Architecture Combat System (OACS) data format (OACS Datafusion IFS [1]) and the Open Architecture Radar Interface Standard (OARIS) data format [2], however this may be subject to change in later phases.

In addition to the out-of-the-box NiFi input and output processors, Data Distribution Service (DDS) publish and subscribe processors built using OpenSplice Community Edition DDS and a processor to receive input from Simulation Gateway and VBS3 have been added.

Adding a new data converter to/from ISAIN's data model involves adding a Java implementation of the application's data type and implementing a single Java interface to perform the conversion. Where the application's data type is specified as a DDS Interface Description Language (IDL) file, the Java class(es) should be generated using the OpenSplice tools. The compiled classes are then added as a NiFi extension.

It is anticipated that future releases will expose the ISAIN data model for inclusion in applications, thereby removing the need for the data converters.

Figure 1 shows the individual components that together form ISAIN. Ship systems and AIs are connected together via data flows within Apache NiFi, with a record of AI output being written to a MongoDB collection. Where an AI output has been configured to be Human In The Loop (HITL)[1] or Human on The Loop (HOTL)[2], the AI output is approved/rejected/overridden with the ISAIN Console, whereas Human Out Of The Loop (HOOTL)[3] AI output can only be viewed with the ISAIN Console. ISAIN Console authentication is provided by the LDAP Server. The component log files are routed to Elastic Stack to provide consolidated viewing and searching of the logs.

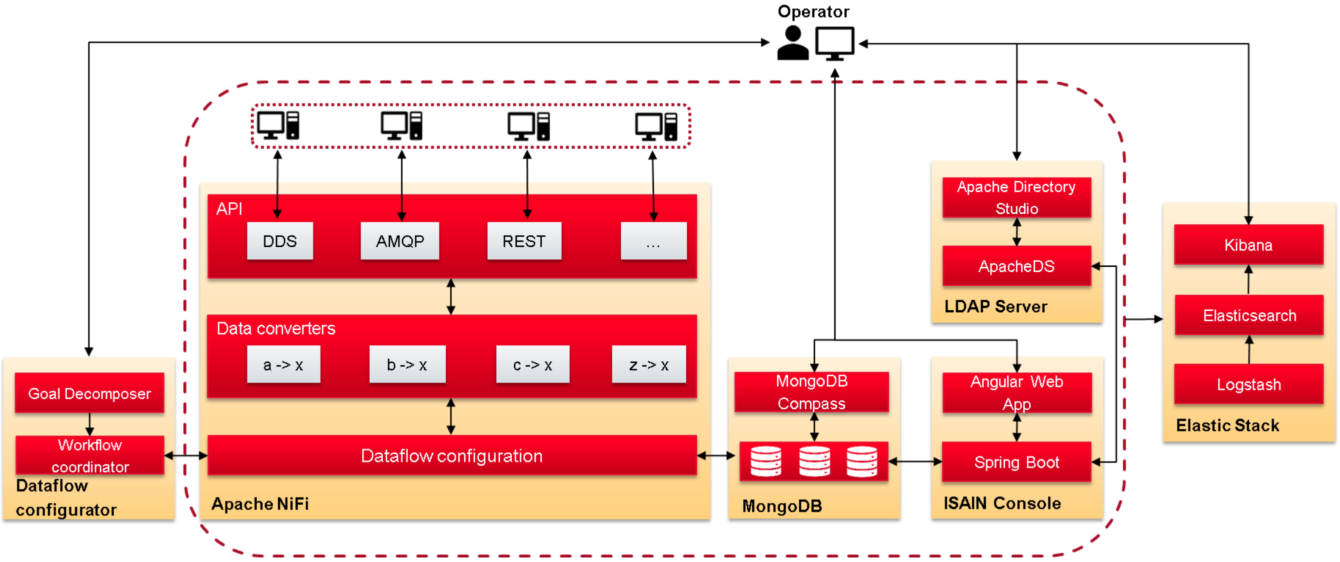


Figure 1 - Architecture overview

[1] *HITL – an AI system makes a recommendation, but the human involved has to make a decision and the AI system is not allowed to proceed otherwise*

[2] *HOTL – an AI system can make decisions by itself, but those decisions can be vetoed by a human*

[3] *HOOTL – an AI system where a human is not required at all*

3 Software Stack

ISAIN is built with, and upon, the software stack listed in Table 2.

Name	Version	URLs	Development License Required	Runtime License Required	Purpose
OpenJDK	1.8.0	https://openjdk.java.net	No	No	Runtime environment for NiFi and the ISAIN Console
Git	2.17.1	https://git-scm.com	No	No	Source control
Jenkins	LTS	https://jenkins.io	No	No	Continuous Integration
Apache Maven	3.6.3	https://maven.apache.org	No	No	Dependency management and build tool
NPM	6.13.4	https://www.npmjs.com	No	No	Dependency management and build tool
Apache NiFi	1.11.2	https://nifi.apache.org https://javadoc.io/doc/org.apache.nifi	No	No	Facilitates the connection of ship systems and AIs
Apache Zookeeper	3.5.6	https://zookeeper.apache.org	No	No	Orchestration manager required by NiFi
ApacheDS	2.0.0.AM25	https://directory.apache.org	No	No	LDAP server for authentication and authorisation
Apache Directory Studio	2.0.0.v20180908-M14	https://directory.apache.org/studio	No	No	Admin UI for ApacheDS
MongoDB	4.2.3	https://www.mongodb.com	No	No	Database to store AI output
MongoDB Compass Community	1.20.5	https://www.mongodb.com/products/compass			Admin UI for MongoDB
Docker	18.09.7	https://www.docker.com	No	No	Facilitates distributing each ISAIN component along with all of its software dependencies
Elastic Stack	7.6.0	https://www.elastic.co	No	No	Application log consolidation tool
Ubuntu	18.04 LTS	https://ubuntu.com	No	No	Base docker image upon which the ApacheDS and ISAIN Console docker images are built
Spring Boot	2.2.4	https://spring.io/projects/spring-boot	No	No	The framework upon which the ISAIN Console server component is built
Angular	8.2.14	https://angular.io	No	No	The framework upon which the ISAIN Console client component is built
ADLINK OpenSplice Community Edition	6.9.190925 OSS	https://github.com/ADLINK-IST/opensplice/releases	No	No	Implementation of the DDS standard within ISAIN
Fast Downward	20.06	https://www.fast-downward.org	No	No	Planning system used within the IDDC component
NGINX	TBC	https://www.nginx.com	No	No	Web server to provide links to the different Web UIs

Table 2 - Software stack

4 Source Code Repositories

The source code is maintained within separate Git repositories to facilitate easier development, with each repository relating to a specific piece of functionality. Table 3 lists each repository at the time of writing.

Name	Description
parent_pom	Parent Maven pom file from which all other Maven pom files inherit
apache_ds	Configuration for the ApacheDS LDAP server
console	The ISAIN Console
dds_datatypes	IDL files for the DDS datatypes to be supported via the DDS processors
deployment	Configuration files for a deployment
mock_startle	A mock STARTLE implementation
nifi_isain_processors	All OFFICIAL NiFi processors
nifi_isain_processors_os	All OFFICIAL SENSITIVE NiFi processors
nifi_isain_common	Common supporting classes for NiFi
nifi	The consolidated ISAIN NiFi component
sdk/isain_nifi_test	SDK REST API tests
sdk/sdk_data_publisher	Publishes sample system track, own ship position and own ship course and speed data
sdk/sdk_deployment	Creates an SDK deployment release
sdk/sdk_geoserver	GeoServer instance to provide map layers
sdk/sdk_mongodb	Prepopulated database
sdk/sdk_nginx	Web proxy for the different SDK components
sdk/sdk_postgis	PostGIS database for GeoServer
sdk/sdk_rabbitmq	RabbitMQ instance with predefined queues

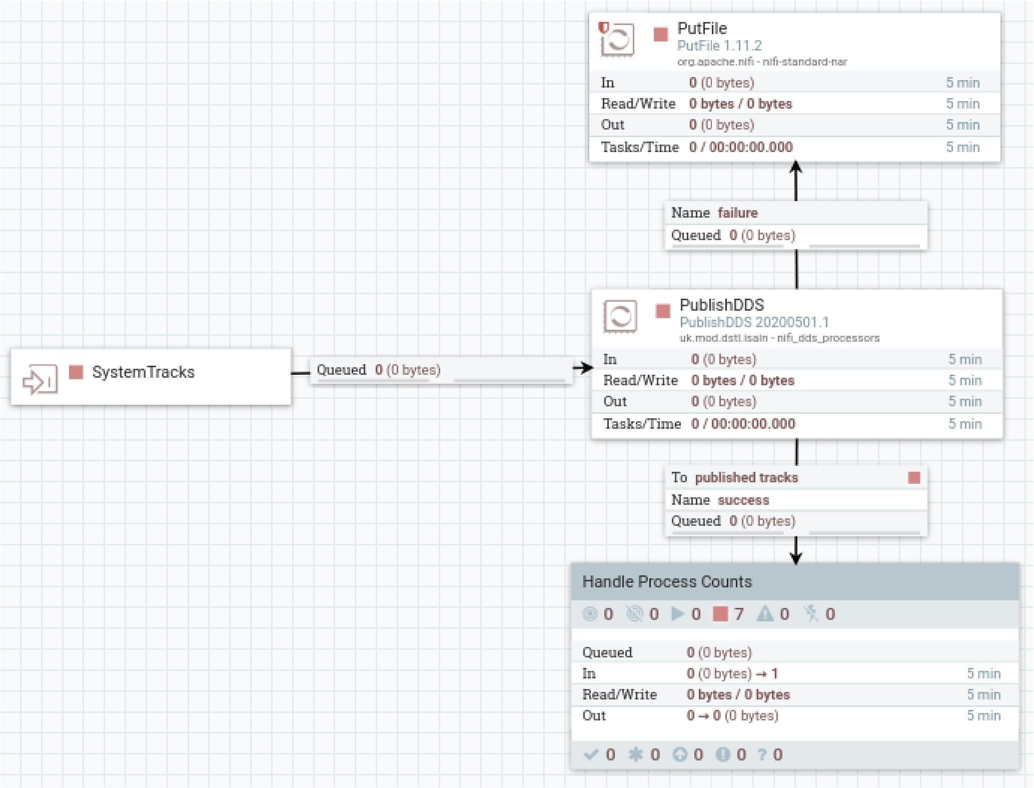
Table 3 - Source Code Repositories

5 AI/Ship System Integration Overview

ISAIN uses Apache NiFi as its core component and therefore familiarity with NiFi is key to integrating new AIs and ship systems. “Getting started” information is available at https://nifi.apache.org/docs/nifi-docs/html/getting-started.html and numerous tutorials are available on YouTube. In summary, NiFi is a data flow engine that enables users to build up complex data flows to manage the flow of information between different systems using an extensive library of built-in processors, as well as providing a Java API allowing users to add their own. In the case of ISAIN, these different systems are AIs, ship systems and ISAIN’s other constituent parts such as the database.

New AIs/ship systems are integrated by adding new input and output processors to the data flow and connecting them to the existing processors in the flow, in order for them to receive the relevant data. If a new AI/ship system uses a data type not yet supported by ISAIN, then conversion processors are required to convert the received/sent data between ISAIN’s data format and the AI/ship system’s data format. Currently ISAIN’s data format consists of the Open Architecture Combat System (OACS) data format (OACS Datafusion IFS [1]) and the Open Architecture Radar Interface Standard (OARIS) data format [2], with the addition of a hostility assessment data type. Refer to Appendix C – OACS IDL Files, Appendix D – OARIS IDL Files and Appendix E – ISAINService IDL for further details.

Figure 2 shows a simplified example invoking an AI via the DDS protocol and sending it systems tracks that have already been received in the OACS format elsewhere in the dataflow. Figure 3 shows the DDS configuration for the PublishDDS processor where properties such as the topic name are specified.



PutFile
PutFile 1.11.2
org.apache.nifi - nifi-standard-nar

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

Name failure
Queued 0 (0 bytes)

SystemTracks

Queued 0 (0 bytes)

PublishDDS
PublishDDS 20200501.1
uk.mod.dstl.isain - nifi_dds_processors

In	0 (0 bytes)	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 (0 bytes)	5 min
Tasks/Time	0 / 00:00:00.000	5 min

To published tracks
Name success
Queued 0 (0 bytes)

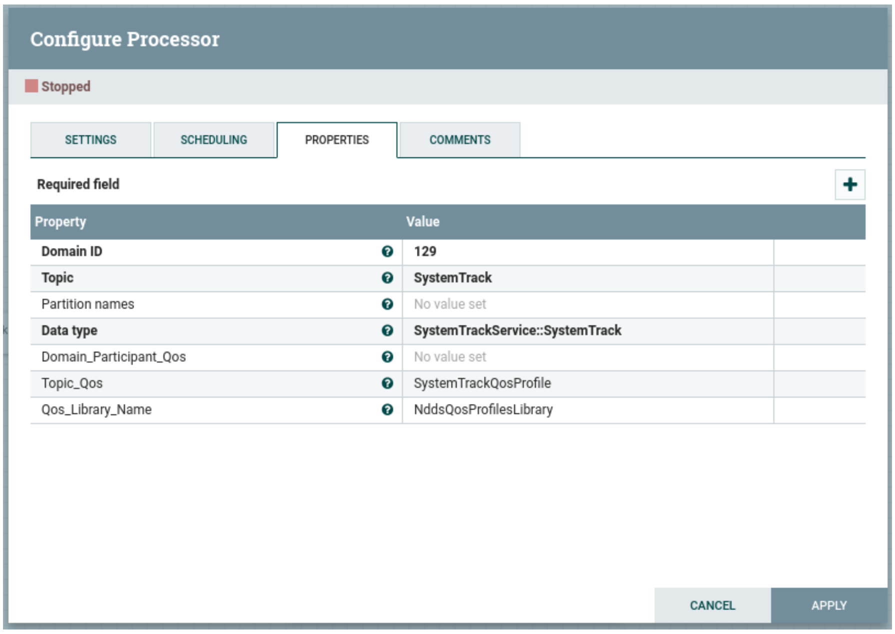
Handle Process Counts

◎ 0 ⊘ 0 ▶ 0 ■ 7 ⚠ 0 ✗ 0

Queued	0 (0 bytes)	
In	0 (0 bytes) → 1	5 min
Read/Write	0 bytes / 0 bytes	5 min
Out	0 → 0 (0 bytes)	5 min

✓ 0 ✱ 0 ⊙ 0 ❗ 0 ? 0

Figure 2 – Example AI input flow



Configure Processor

Stopped

Required field

Property	Value
Domain ID	129
Topic	SystemTrack
Partition names	No value set
Data type	SystemTrackService::SystemTrack
Domain_Participant_Qos	No value set
Topic_Qos	SystemTrackQosProfile
Qos_Library_Name	NddsQosProfilesLibrary

Figure 3 – PublishDDS processor configuration

When integrating a new AI, in addition to modifying the NiFi data flow, the ISAIN Console configuration will need to be updated if the AI belongs to a new domain, e.g. hostility, damage control, etc. Where the console configuration already contains the domain, no configuration changes are required. An example of the console configuration for a new domain is shown in Figure 4, this configuration is held within ISAIN's MongoDB database.

```
{
    "id": "DAMAGE_CONTROL",
    "title": "Damage Control",
    "domain": "damageControl",
    "exclusive": true,
    "headers": [{
        "displayName": "Condition",
        "key": "condition",
        "selectable": true,
        "aiOptions": {
            "IBIS": [
                "RUNNING",
                "FAULTY",
                "NO POWER"
            ]
        }
    }, {
        "displayName": "Action",
        "key": "action",
        "selectable": true,
        "aiOptions": {
            "IBIS": [
                "MONITOR",
                "INVESTIGATE",
                "RESET FUSE",
                "EXTINGUISH"
            ]
        }
    }],
    "selectorKey": "location",
    "authorities: []
}
```

Figure 4 - Example Tab configuration

6 NiFi

6.1 Custom Processors

In addition to the processors included out of the box with NiFi, eg: ConsumeJMS, ConsumeKafka, ConsumeAMQP, ListenHTTP, etc, Table 4 lists the additional custom processors that have been added.

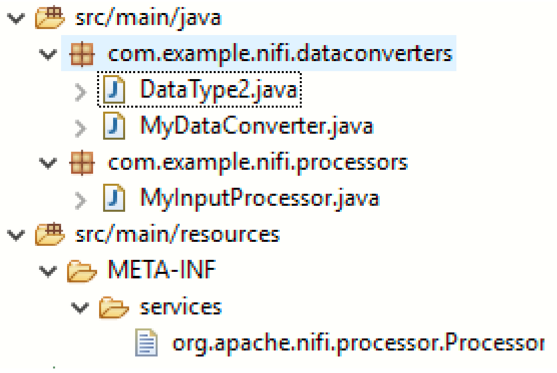
Name	Description	Class Name
SadmNifiBridge	Connects to SADM and reads incoming messages	uk.mod.dstl.isain.sadm.SadmNifiBridge
ConsumeDDS	Consumes DDS messages	uk.mod.dstl.isain.dds.ConsumeDDS
PublishDDS	Publishes DDS messages	uk.mod.dstl.isain.dds.PublishDDS
DataConversionProcessor	Converts data from one data format to another	uk.mod.dstl.isain.data.conversion.DataConversionProcessor

Table 4 - Custom Processors

6.2 Application Interfaces

Where a new application is to be integrated and input and/or output processors are not available for the application’s Application Programming Interface (API), a new input and/or output processor will need to be added. (ConsumeDDS and PublishDDS are examples of input and output processors.) Adding a new processor involves extending NiFi’s `org.apache.nifi.processor.AbstractProcessor` class, adding the processor’s class name to a `META-INF/services/org.apache.nifi.processor.Processor` file, packaging into a nar file and copying the nar file into NiFi’s `lib` directory.

An example of this for a new input processor is shown in Figure 5, Figure 6, Figure 7 and Figure 8.



```
src/main/java
  com.example.nifi.dataconverters
    DataType2.java
    MyDataConverter.java
  com.example.nifi.processors
    MyInputProcessor.java
src/main/resources
  META-INF
    services
      org.apache.nifi.processor.Processor
```

Figure 5 - Example custom processor Maven project layout

```java
package com.example.nifi.processors;

import java.util.Arrays;
import java.util.Collections;
import java.util.List;
import java.util.Set;

import org.apache.nifi.annotation.behavior.InputRequirement;
import org.apache.nifi.annotation.behavior.InputRequirement.Requirement;
import org.apache.nifi.annotation.documentation.CapabilityDescription;
import org.apache.nifi.annotation.documentation.Tags;
import org.apache.nifi.annotation.lifecycle.OnStopped;
import org.apache.nifi.components.PropertyDescriptor;
import org.apache.nifi.expression.ExpressionLanguageScope;
import org.apache.nifi.processor.AbstractProcessor;
import org.apache.nifi.processor.ProcessContext;
import org.apache.nifi.processor.ProcessSession;
import org.apache.nifi.processor.Relationship;
import org.apache.nifi.processor.exception.ProcessException;
import org.apache.nifi.processor.util.StandardValidators;

@InputRequirement(Requirement.INPUT_FORBIDDEN)
@Tags({ "ingest", "my_apps_protocol", "listen" })
@CapabilityDescription("Receives incoming messages through my_apps_protocol and transforms them into
FlowFiles.")
public class MyInputProcessor extends AbstractProcessor {

    public final static Relationship REL_SUCCESS = new Relationship.Builder().name("success")
        .description("Relationship for successfully received FlowFiles").build();

    public static final PropertyDescriptor PORT = new PropertyDescriptor.Builder().name("Listening Port")
        .description("The Port to listen on for incoming connections").required(true)
        .expressionLanguageSupported(ExpressionLanguageScope.VARIABLE_REGISTRY)
        .addValidator(StandardValidators.POSITIVE_INTEGER_VALIDATOR).build();

    private static final Set<Relationship> RELATIONSHIPS;
    private static final List<PropertyDescriptor> PROPERTIES;
    static {
        RELATIONSHIPS = Collections.singleton(REL_SUCCESS);
        PROPERTIES = Arrays.asList(PORT);
    }

    @Override
    public Set<Relationship> getRelationships() {
        return RELATIONSHIPS;
    }

    @Override
    public List<PropertyDescriptor> getSupportedPropertyDescriptors() {
        return PROPERTIES;
    }

    @Override
    public void onTrigger(ProcessContext context, ProcessSession session) throws ProcessException {
        // TODO receive message and convert to FlowFile
    }

    @OnStopped
    public void cleanup() {
        // TODO close connection
    }
}
```

Figure 6 - Example AbstractProcessor implementation

```
com.example.nifi.processors.MyInputProcessor
```

Figure 7 - Example org.apache.nifi.processor.Processor file

Nar files can be created using the `nifi-nar-maven-plugin` Maven plugin and setting the POM's packaging attribute to `nar`.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">


    <modelVersion>4.0.0</modelVersion>
    <groupId>sandpit</groupId>
    <artifactId>sandpit</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>nar</packaging>

    <properties>
        <maven.compiler.source>1.8</maven.compiler.source>
        <maven.compiler.target>1.8</maven.compiler.target>
        <nifi.version>1.11.2</nifi.version>
        <nifi.nar.plugin.version>1.3.1</nifi.nar.plugin.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-api</artifactId>
            <version>${nifi.version}</version>
        </dependency>
        <dependency>
            <groupId>org.apache.nifi</groupId>
            <artifactId>nifi-processor-utils</artifactId>
            <version>${nifi.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.nifi</groupId>
                <artifactId>nifi-nar-maven-plugin </artifactId>
                <version>${nifi.nar.plugin.version}</version>
                <extensions>true</extensions>
            </plugin>
        </plugins>
    </build>
</project>
```

Figure 8 – Example Nar file Maven pom.xml

Refer to the NiFi documentation referenced in Table 2 for further information on writing custom processors.

6.3 Data Type Conversion

Conversion between different data types is performed by the generic `uk.mod.dstl.isain.data.conversion.DataConversionProcessor` processor, which can be configured to invoke a specified converter class which implements the `uk.mod.dstl.isain.data.conversion.DataConverter` interface. Facilitating conversion to/from a new data type and ISAIN's internal data model consists of adding the necessary Plain Old Java Objects (POJOs) for the new data type, adding an implementation of `DataConverter` to perform the conversions, building the classes into a jar file and copying the jar into NiFi's `lib` directory. Note that converting from DataType1 to DataType2 and vice versa requires two separate `DataConverter` implementations. Figure 9 shows an example `DataConverter` implementation.

```
package com.example.nifi.dataconverters;

import java.util.Map;

import uk.mod.dstl.isain.data.conversion.DataConverter;
import uk.mod.dstl.isain.example.DataType1;

public class MyDataConverter implements DataConverter<DataType1, DataType2> {

    @Override
    public DataType2 convert(DataType1 source) {
        DataType2 converted = new DataType2();
        // perform conversion of DataType1 (source) -> DataType2 (converted)
        return converted;
    }

    @Override
    public String getSource() {
        return DataType1.class.getName();
    }

    @Override
    public String getDestination() {
        return DataType2.class.getName();
    }

    @Override
    public void setAttributes(Map<String, String> attributes) {
        // set external attributes required for the conversion not contained within DataType1
    }
}
```

Figure 9 - Example DataConverter implementation

For new DDS data types, the POJOs should be generated using the OpenSplice Community Edition tooling. The dds_datatypes repository listed in Table 3 contains an existing set of IDL files and the Maven POM has been configured to invoke the OpenSplice Community Edition tooling in order to regenerate, compile and package the POJOs during each build.

6.4 Java Object Serialisation

Serialisation and deserialisation of POJOs within the custom processors listed in Table 4 is done through a combination of XStream (https://x-stream.github.io) and Underscore-java (https://github.com/javadev/underscore-java). XStream is used to serialise the POJO into XML, the XML is passed to Underscore-java to convert into JavaScript Object Notation (JSON), and the JSON string is converted to a UTF-8 encoded byte array. The byte array is then written to the NiFi flow file. Deserialisation is the reverse process.

```java
import org.apache.nifi.processor.AbstractProcessor;
import com.github.underscore.lodash.U;
import com.thoughtworks.xstream.XStream;

public class MyProcessor extends AbstractProcessor {

    private XStream xStream;

    public MyProcessor() {
        this.xStream = new XStream();
        this.xStream.setMode(XStream.NO_REFERENCES);
    }

    public byte[] serialiseObject(Object object) {
        final String xml = this.xStream.toXML(object);
        final String json = U.xmlToJson(xml);
        return json.getBytes(StandardCharsets.UTF_8);
    }

    public Object deserialiseObject(byte[] bytes) {
        final String json = new String(bytes, StandardCharsets.UTF_8);
        final String xml = U.JsonToXml(json);
        return this.xStream.fromXML(xml);
    }
}
```

Figure 10 - Example POJO serialisation and deserialisation

```xml
<dependencies>
    <dependency>
        <groupId>com.thoughtworks.xstream</groupId>
        <artifactId>xstream</artifactId>
        <version>1.4.7</version>
    </dependency>
    <dependency>
        <groupId>com.github.javadev</groupId>
        <artifactId>underscore</artifactId>
        <version>1.53</version>
    </dependency>
</dependencies>
```

Figure 11 – Example POJO serialisation Maven pom.xml dependencies

```java
public class Person {
    private String name;
    private int age;
    private String address;
    public Person(String name, int age, String address) {
        this.name = name;
        this.age = age;
        this.address = address;
    }
    public String getName() {
        return name;
    }
    public int getAge() {
        return age;
    }
    public String getAddress() {
        return address;
    }
}
```

Figure 12 – Example POJO classs

```json
{
  "uk.mod.dstl.isain.example.Person": {
    "name": "John Smith",
    "age": "30",
    "address": "1 High Street, London"
  },
  "#omit-xml-declaration": "yes"
}
```

Figure 13 – Example serialised POJO

This serialisation process was chosen as it provides a reliable mechanism for serialising and deserialising POJOs into JSON without requiring the POJO’s to be modified with any annotations. This is particularly important for generated Java classes such as DDS classes defined in IDL files as these annotations would need to be reinstated whenever the Java classes are regenerated.

7 Dataflow Configuration

Configuration of the dataflow controlling the receiving, manipulation and routing of data is performed through the NiFi User Interface (UI), an example which is shown in Figure 14. The default Uniform Resource Locator (URL) for the UI is http://hostname:8080/nifi.

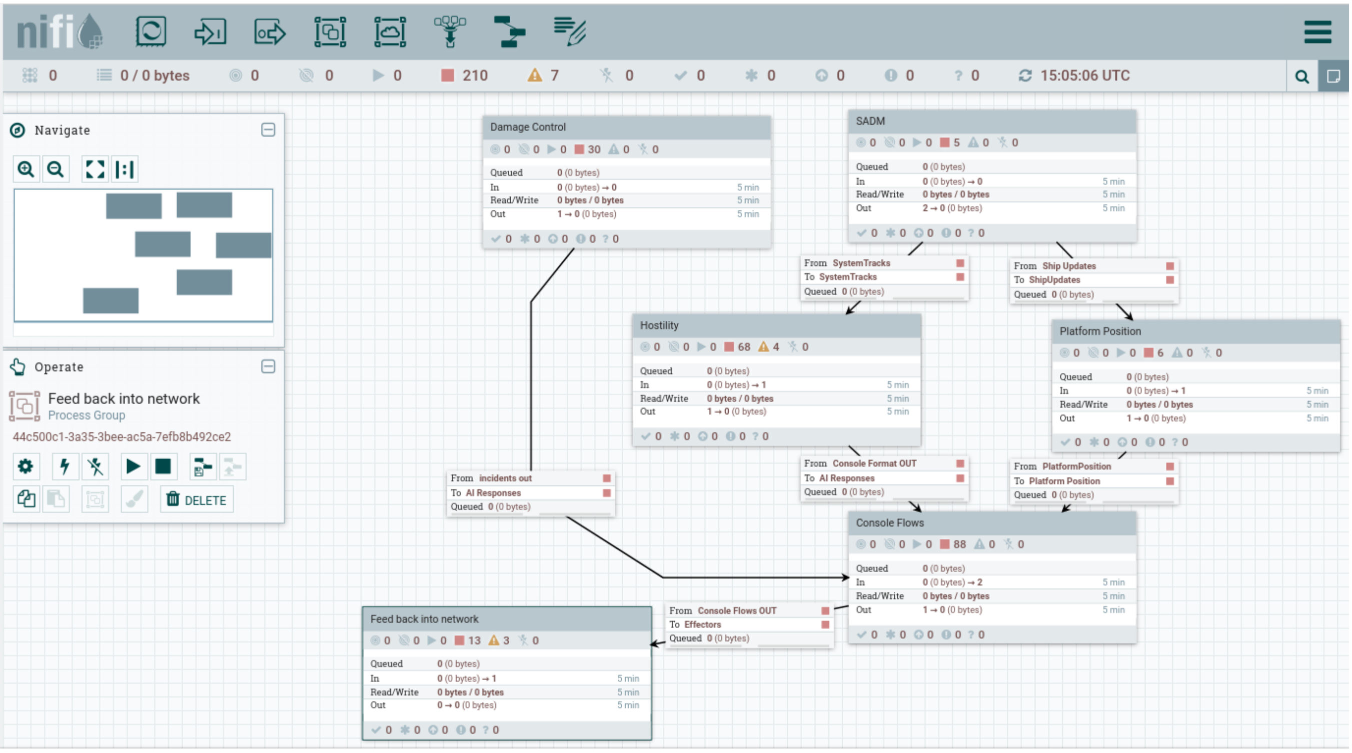


Figure 14 - Example NiFi Dataflow

New processors are added to the dataflow by dragging the “Processor” icon onto the canvas and selecting the desired processor type. Existing processors are configured by right clicking on the processor and selecting “Configure”. Processors are connected together by hovering over a processor and dragging the connector icon to the destination processor. Individual processors are stopped and started by right clicking on them and selecting Stop/Start. An entire dataflow is stopped and started by right clicking on the canvas and selecting Stop/Start.

NiFi writes its flow configuration to `<nifi installation>/conf/flow.xml.gz`. Once a flow has been modified, this file should be added to the nifi git repository.

Refer to the NiFi documentation referenced in Table 2 for further information on amending and controlling data flows.

8 MongoDB

The MongoDB ISAIN database is used to store the output from the connected AI applications and any user actions performed in the ISAIN Console.

A collection exists for each domain, e.g. damage control, hostility, etc, with AI output being stored in the relevant domain collection. HITL and HOTL user interactions are stored in the interactions collection. All AI output, including HOOTL, and user interactions are stored in an event log collection.

All collections are currently de-normalised, however this may be subject to change in a later release.

8.1 Admin UI

MongoDB Compass Community is the admin UI used to visualise data within the ISAIN database. The connection URL for a default local installation is `mongodb://localhost:27017`.

8.2 Collections

8.2.1 Domains

The structure of a domain collection is shown in Figure 15.

```
ai: String // name of the AI
automationMethod: String, // enum (HITL, HOTL, HOOTL)
ttc: Integer // seconds from epoch time stamp
dataSpecifics: Object // values specific to the domain
    location: String
    condition: String
    action: String
justification: Object // values specific to AI
    priority: Integer
    description: String
timeRequested: Integer // epoch timestamp of when request was made (timeReceived is set by flow)
```

Figure 15 - Example Domain collection structure

8.2.2 Interactions

The structure of the interactions collection is shown in Figure 16.

```
interactionMethod: String // enum (CONFIRM, VETO)
timeInteracted: Integer // epoch time stamp of interaction
aiResponse: Object // the domain object associated with interaction
```

Figure 16 - Effected Responses collection structure

8.2.3 Event Log

The structure of the event log collection is shown in Figure 17.

```
eventTime: Integer
eventType: String // enum (RESPONSE, INTERACTION)
event: Object // the domain object associated with event- domain response or user interaction
```

Figure 17 - Event Log collection structure

8.2.4 Tabs

The structure of the tabs collection is shown Figure 18.

```
eventTime: Integer
eventType: String // enum (RESPONSE, INTERACTION)
event: Object // the domain object associated with event- domain response or user interaction
```

Figure 18 - Event Log collection structure

8.2.5 Reference Data

The structure of the reference data collection is shown in Figure 19.

```
eventTime: Integer
eventType: String // enum (RESPONSE, INTERACTION)
event: Object // the domain object associated with event- domain response or user interaction
```

Figure 19 - Event Log collection structure

9 ApacheDS

ApacheDS is used as the ISAIN LDAP server. This contains the users and roles that are permitted to access the ISAIN Console. Users and roles can be added/removed/edited through Apache Directory Studio, and a default set of users and roles are declared in an LDAP Data Interchange Format (LDIF) file stored within the apache_ds git repository. LDIF files can be uploaded into ApacheDS using Apache Directory Studio, or via 3rd party command line tools.

Refer to the ApacheDS and Apache Directory Studio documentation referenced in Table 2 for further information.

10 Console

10.1 Overview

The ISAIN Console consists of a Spring Boot server component which serves an Angular web app client component to the user’s web browser. User authentication and authorisation is performed through the use of LDAP and JSON Web Tokens (JWT). ApacheDS is used as the LDAP provider, however it is possible to substitute this with a different provider such as Microsoft Active Directory with minimal effort.

10.2 Configuring tabs

Each tab on the console displays the data associated with a given domain, be it hostility, damage control, etc. Whenever an AI is added to the network that will be giving responses in a new domain, a new tab must be configured to handle and display this domain. This configuration is defined by adding a new row to the MongoDB **tabs** collection. The configuration is in the form of a JSON object describing the attributes required in order to render the tab. An example of this configuration is shown in Figure 20.

```
{
    "id": "DAMAGE_CONTROL",
    "title": "Damage Control",
    "domain": "damageControl",
    "exclusive": true,
    "headers": [{
        "displayName": "Condition",
        "key": "condition",
        "selectable": true,
        "aiOptions": {
            "IBIS": [
                "RUNNING",
                "FAULTY",
                "NO POWER"
            ]
        }
    }, {
        "displayName": "Action",
        "key": "action",
        "selectable": true,
        "aiOptions": {
            "IBIS": [
                "MONITOR",
                "INVESTIGATE",
                "RESET FUSE",
                "EXTINGUISH"
            ]
        }
    }],
    "selectorKey": "location",
    "authorities: []
}
```

Figure 20 - Example Tab configuration

If the tabs collection is empty, for instance after a new deployment, the application is bundled with a default tab configuration which is defined within `tabs.json` and the **tabs** collection is populated with this file when the collection is queried and found to be empty. `tabs.json` is stored within the console git repository.

To create a new tab, it is easiest to copy an existing entry and modify it accordingly therefore ensuring all of the required fields have been set. The required fields and their purpose are:

- “id” – a string that identifies the tab. Must be unique, with no whitespace, but can take any form otherwise. Convention is that the “id” is equal to the “title”, but capitalised and using an _ for any whitespace.

- “title” – the title of the tab, as it appears on the console.
- “domain” – the domain of the tab. This must match the domain specified in the NiFi flow configuration to ensure the correct data is displayed on the console.
- “exclusive” – should be set to either true or false. If true, then confirming one response from an AI for this domain will automatically veto any other responses in the same group. If false, the operator can confirm as many responses in a group as they wish.
- “headers” – this contains an array of header objects, which describe how the data specific to the chosen domain should be interpreted and displayed on the console. The attributes associated with a header are as follows:
    - “displayName” – the name that the column will have for this header.
    - “key” – the key of the associated data found in the data specifics for this domain. Must match the key set in the NiFi configuration.
    - “selectable” – this attribute is optional, and if not included, will default to false. If false, then the operator will not be able to add a custom value for this data, whereas if true, the operator will have the option to pick a custom option.
    - “aiOptions” – this attribute is only required if “selectable” is set to true. This attribute is a list of possible options for this data, grouped by the AI the options belong to. An example configuration for this option is shown in Figure 21.

```
"headers": [{
    "displayName": "Hostility",
    "key": "hostility",
    "selectable": true,
    "aiOptions": {
        "SYCOIEA": [
            "FRIENDLY",
            "NUETRAL",
            "SUSPECT",
            "HOSTILE"
        ],
        "STARTLE": [
            "CONCERN",
            "NO CONCERN",
            "THREAT"
        ],
    }
}]
```

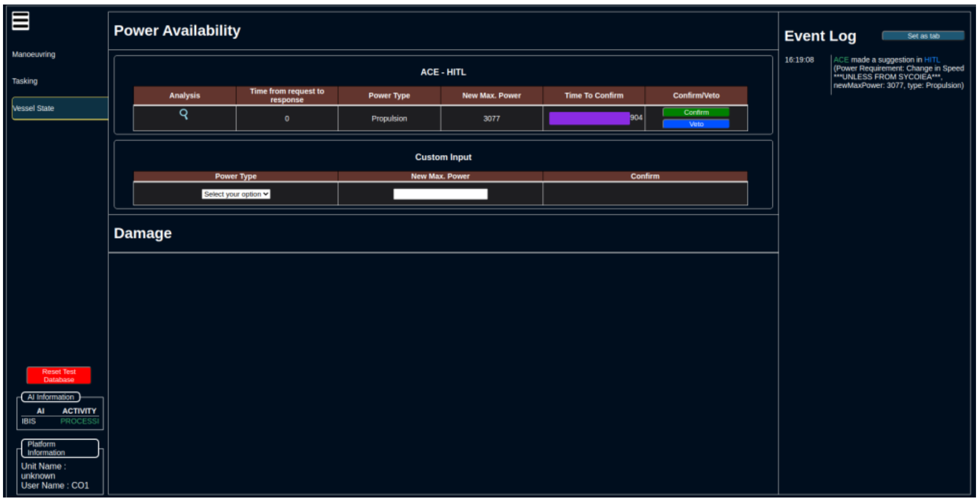
Figure 21 - Example headers configuration

- “selectorKey” – the selectorKey refers to an attribute specific to the domain that the responses will be grouped by. The value must match the attribute, including the same case, for the grouping to work correctly
- “sortSelector” – if set to true, the selector for this tab on the console will be sorted numerically, in ascending order.
- “authorities” – an array of the groups that an operator must be a part of to access this tab, as defined in the LDAP configuration. If left empty, any operator will be able to access the tab. Each role should be wrapped in quotes “” and will be of the form ROLE_{LDAP-GROUP-NAME}, all in upper-case. Figure 22 shows an example of the format, if the groups configured in the LDAP server are ‘officers’ and ‘airdomain’.

```
"authorities": [
    "ROLE_OFFICERS",
    "ROLE_AIRDOMAIN"
]
```

Figure 22 - Example authorities configuration

When adding a new tab to the database, the web browser page will need to be refreshed in order to display the new tab. If configured correctly, the new tab should appear and when data is sent to the domain, it should be displayed as expected. If there are issues with data not being displayed, then check the tabs configured and ensure they are valid. Numerous tools are available online that can verify the syntactic correctness of JSON, for example https://jsonlint.com/, which can be used to help if any issues persist.



ACE - HITL

Analysis	Time from request to response	Power Type	New Max. Power	Time To Confirm	Confirm/Veto
	0	Propulsion	3077	904	Confirm Veto

Custom Input

Power Type	New Max. Power	Confirm
Select your option		

16:19:08 ACE made a suggestion in HITL (Power Requirement: Change in Speed ***UNLESS FROM SYCOIEA***, newMaxPower: 3077, type: Propulsion)

AI	ACTIVITY
IBIS	PROCESSI

Unit Name : unknown
User Name : CO1

Figure 23 - Example Console Screen

10.3 Configuring Platform Specific Data

Platform specific data is stored with the MongoDB **referenceData** collection. At the time of writing, the only platform specific data in use is the platform name which is displayed in the bottom left corner of the screen. If the platform name has not been set, for instance after a new deployment, a new “platformName”: “unknown” entry is added to the **referenceData** collection when the collection is queried and found to be empty. An example **referenceData** entry is show in Figure 24.

```
{
    "platformName": "HMS Dragon"
}
```

Figure 24 - Example Reference Data entry

11 Build

11.1 Build Process

The build process for each of the source code repositories is facilitated using Apache Maven and Jenkins. For continuous integration, Jenkins pipelines are used via a Jenkinsfile which is stored within the Git repository alongside the source code, thus allowing the Jenkins job's configuration to be maintained under source control.

11.1.1 DDS Libraries

In order to build the contents of the dds_datatypes repository, OpenSplice Community Edition must be installed. The OpenSplice installation directory is specified within the dds_datatypes pom.xml files as `/opt/opensplice/HDE/x86_64.linux`.

11.1.2 3rd Party Maven Dependencies

In order to build the contents of the dds_datatypes and nifi_dds_processors repositories, the jars from the OpenSplice installations need to be uploaded into the Maven repository as the following artifacts:

- artifactId: opensplice, groupId: dcpssaj5, version: 6.9

In order to build the contents of the nifi_isain_processors_os repository, the SADM jars, which will need to be requested as GFx from Dstl, need to be uploaded into the Maven repository as the following artefacts:

- artifactId: SADMCommon, groupId: SADMCommon, version: 1.0.0
- artifactId: SADMMsgLib, groupId: SADMMsgLib, version: 1.0.0
- artifactId: SADMSocketClient, groupId: SADMSocketClient, version: 1.0.0
- artifactId: SADMSocketCommon, groupId: SADMSocketCommon, version: 1.0.0

All other 3rd party maven dependencies are available from Maven Central.

11.2 Branching Strategy

The feature branching strategy is used, coupled with an unstable develop branch, sometimes referred to as an integration branch. New features and bug fixes are committed to their own branch, with pull requests created when they are ready to be merged into the develop branch. Only once the changes have been reviewed and approved, is the feature branch merged into the develop branch. Due to the possibility of two or more subsequent feature branches being merged into the develop branch causing a build failure, the develop branch is considered to be unstable. Only when the develop branch builds successfully is it merged into the master branch and a candidate release build is performed from the master branch. The master branch should always build successfully.

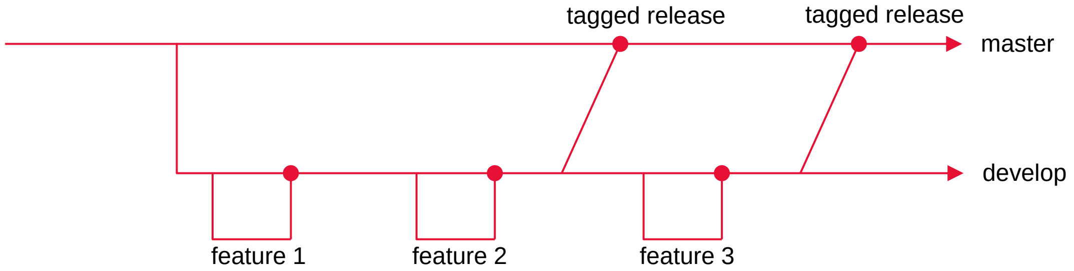


Figure 25 - Branching Strategy

11.3 Releases

Releases are performed through Jenkins from the stable master branch. Each release is given a unique version number, with the corresponding commit in the Git repository tagged with the version number. The built artefacts (e.g. jar files, Docker images, etc) are published to the appropriate internal repository.

11.4 Versioning

Version numbers of released artefacts are of the form YYYYMMDD.<build number>, where YYYYMMDD is the date of the commit to the Git repository and <build number> is the distinct build number for that commit day, indexed from 1, e.g. 20200130.2 for the second release on the 30th January 2020.

12 Deployment

12.1 Supported Operating Systems

ISAIN has been tested on Ubuntu 18.04 LTS and Centos 7.6.1810 for the server side components, and Google Chrome 80.0 or higher on Windows 10, Ubuntu 18.04 LTS and Centos 7.6.1810 for the ISAIN Console UI.

12.2 ISAIN Components

Deployment of ISAIN is performed by running docker-compose. A docker-compose file is stored within the deployment git repository which starts single instances of NiFi, MongoDB , Zookeeper, ApacheDS, the ISAIN Console and each of the Elastic Stack components (Filebeat, Logstash, Elasticsearch and Kibana). A separate Zookeeper instance is started instead of using the instance bundled in NiFi in order to facilitate possible future clustering of NiFi using Docker Swarm.

Volume mounts are specified within the docker-compose file for the runtime data directories to enable persisting of data once a docker container has been removed, the configuration directories to enable overriding of default configuration files within the containers, and the log directories to enable the collating of logs with Elastic Stack.

Host networking is specified within the docker-compose file and is a requirement in order to enable DDS multicasting, a requirement for the PublishDDS and ConsumeDDS processors. Note that user namespaces must not be enabled within the host machine's Docker configuration as this is incompatible with host networking.

12.3 Supporting Tools

12.3.1 Apache Directory Studio

Apache Directory Studio is used to administer the ApacheDS LDAP server. Since Apache Directory Studio has a native UI as opposed to a web interface, it not possible to include it with the docker-compose file and must therefore be installed separately. Note that it is a Java application and therefore Java must be installed as a pre-requisite.

12.3.2 MongoDB Compass Community

MongoDB Compass Community is used to administer MongoDB. Since MongoDB Compass Community has a native UI as opposed to a web interface, it not possible to include it with the docker-compose file and must therefore be installed separately.

APPENDICES

Appendix A – Glossary

Acronym	Description
AI	Artificial Intelligence
API	Application Programming Interface
DDS	Data Distribution Service
Dstl	Defence Science and Technology Laboratory
HITL	Human In The Loop
HOOTL	Human Out Of The Loop
HOTL	Human On The Loop
IDL	Interface Description Language
ISAIN	Intelligent Ship Artificial Intelligence Network
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
JWT	JSON Web Tokens
LDAP	Lightweight Directory Access Protocol
LDIF	LDAP Data Interchange Format
OACS	Open Architecture Combat System
OARIS	Open Architecture Radar Interface Standard
POJO	Plain Old Java Object
SADM	Ship Air Defence Model
UI	User Interface
URL	Uniform Resource Locator

Appendix B – NiFi Built-in Processors

Table 5 lists the NiFi processors that are provided “out of the box” with NiFi 1.11.2; a full description of each one can be found at https://nifi.apache.org/docs.html.

Processor Name
AttributeRollingWindow
AttributesToCSV
AttributesToJSON
Base64EncodeContent
CalculateRecordStats
CaptureChangeMySQL
CompareFuzzyHash
CompressContent
ConnectWebSocket
ConsumeAMQP
ConsumeAzureEventHub
ConsumeEWS
ConsumeGCPubSub
ConsumeIMAP
ConsumeJMS
ConsumeKafka
ConsumeKafka_0_10
ConsumeKafka_0_11
ConsumeKafka_1_0
ConsumeKafka_2_0
ConsumeKafkaRecord_0_10
ConsumeKafkaRecord_0_11
ConsumeKafkaRecord_1_0
ConsumeKafkaRecord_2_0
ConsumeMQTT
ConsumePOP3
ConsumeWindowsEventLog
ControlRate
ConvertAvroToJSON

ConvertAvroToORC
ConvertAvroToParquet
ConvertCharacterSet
ConvertExcelToCSVProcessor
ConvertJSONToSQL
ConvertRecord
CountText
CreateHadoopSequenceFile
CryptographicHashAttribute
CryptographicHashContent
DebugFlow
DeleteAzureBlobStorage
DeleteByQueryElasticsearch
DeleteDynamoDB
DeleteElasticsearch5
DeleteGCSObject
DeleteGridFS
DeleteHBaseCells
DeleteHBaseRow
DeleteHDFS
DeleteMongo
DeleteRethinkDB
DeleteS3Object
DeleteSQS
DetectDuplicate
DistributeLoad
DuplicateFlowFile
EncryptContent
EnforceOrder
EvaluateJsonPath
EvaluateXPath
EvaluateXQuery
ExecuteGroovyScript

ExecuteInfluxDBQuery
ExecuteProcess
ExecuteScript
ExecuteSparkInteractive
ExecuteSQL
ExecuteSQLRecord
ExecuteStreamCommand
ExtractAvroMetadata
ExtractCCDAAttributes
ExtractEmailAttachments
ExtractEmailHeaders
ExtractGrok
ExtractHL7Attributes
ExtractText
ExtractTNEFAttachments
FetchAzureBlobStorage
FetchDistributedMapCache
FetchElasticsearch
FetchElasticsearch5
FetchElasticsearchHttp
FetchFile
FetchFTP
FetchGCSObject
FetchGridFS
FetchHBaseRow
FetchHDFS
FetchParquet
FetchS3Object
FetchSFTP
FlattenJson
ForkRecord
FuzzyHashContent
GenerateFlowFile

GenerateTableFetch
GeoEnrichIP
GeoEnrichIPRecord
GetAzureEventHub
GetAzureQueueStorage
GetCouchbaseKey
GetDynamoDB
GetFile
GetFTP
GetHBase
GetHDFS
GetHDFSEvents
GetHDFSFileInfo
GetHDFSSequenceFile
GetHTMLElement
GetHTTP
GetIgniteCache
GetJMSQueue
GetJMSTopic
GetMongo
GetMongoRecord
GetRethinkDB
GetSFTP
GetSNMP
GetSolr
GetSplunk
GetSQS
GetTCP
GetTwitter
HandleHttpRequest
HandleHttpResponse
HashAttribute
HashContent

IdentifyMimeType
InvokeAWSGatewayApi
InvokeGRPC
InvokeHTTP
InvokeScriptedProcessor
ISPEnrichIP
JoltTransformJSON
JoltTransformRecord
JsonQueryElasticsearch
ListAzureBlobStorage
ListDatabaseTables
ListenBeats
ListenGRPC
ListenHTTP
ListenLumberjack
ListenRELP
ListenSMTP
ListenSyslog
ListenTCP
ListenTCPRecord
ListenUDP
ListenUDPRecord
ListenWebSocket
ListFile
ListFTP
ListGCSBucket
ListHDFS
ListS3
ListSFTP
LogAttribute
LogMessage
LookupAttribute
LookupRecord

MergeContent
MergeRecord
ModifyBytes
ModifyHTMLElement
MonitorActivity
MoveHDFS
Notify
ParseCEF
ParseEvtx
ParseNetflowv5
ParseSyslog
ParseSyslog5424
PartitionRecord
PostHTTP
PostSlack
PublishAMQP
PublishGCPubSub
PublishJMS
PublishKafka
PublishKafka_0_10
PublishKafka_0_11
PublishKafka_1_0
PublishKafka_2_0
PublishKafkaRecord_0_10
PublishKafkaRecord_0_11
PublishKafkaRecord_1_0
PublishKafkaRecord_2_0
PublishMQTT
PutAzureBlobStorage
PutAzureEventHub
PutAzureQueueStorage
PutBigQueryBatch
PutBigQueryStreaming

PutCassandraQL
PutCassandraRecord
PutCloudWatchMetric
PutCouchbaseKey
PutDatabaseRecord
PutDistributedMapCache
PutDynamoDB
PutElasticsearch
PutElasticsearch5
PutElasticsearchHttp
PutElasticsearchHttpRecord
PutElasticsearchRecord
PutEmail
PutFile
PutFTP
PutGCSObject
PutGridFS
PutHBaseCell
PutHBaseJSON
PutHBaseRecord
PutHDFS
PutHiveQL
PutHiveStreaming
PutHTMLElement
PutIgniteCache
PutInfluxDB
PutJMS
PutKinesisFirehose
PutKinesisStream
PutKudu
PutLambda
PutMongo
PutMongoRecord

PutParquet
PutRecord
PutRethinkDB
PutRiemann
PutS3Object
PutSFTP
PutSlack
PutSNS
PutSolrContentStream
PutSolrRecord
PutSplunk
PutSQL
PutSQS
PutSyslog
PutTCP
PutUDP
PutWebSocket
QueryCassandra
QueryDatabaseTable
QueryDatabaseTableRecord
QueryDNS
QueryElasticsearchHttp
QueryRecord
QuerySolr
QueryWhois
ReplaceText
ReplaceTextWithMapping
RetryFlowFile
RouteHL7
RouteOnAttribute
RouteOnContent
RouteText
RunMongoAggregation

ScanAttribute
ScanContent
ScanHBase
ScrollElasticsearchHttp
SegmentContent
SelectHiveQL
SetSNMP
SplitAvro
SplitContent
SplitJson
SplitRecord
SplitText
SplitXml
SpringContextProcessor
TagS3Object
TailFile
TransformXml
UnpackContent
UpdateAttribute
UpdateCounter
UpdateRecord
ValidateCsv
ValidateRecord
ValidateXml
Wait
YandexTranslate

Table 5 - "Out of the box" NiFi processors

Appendix C – OACS IDL Files

Table 6 lists the OACS IDL files, as specified within OACS Data Fusion IFS[1], that contain the data type definitions that have been incorporated as part of the ISAIN common data format.

IDL File
CommonTypes_Types.idl
CommonTypes_Structures.idl
SensorTrackService.idl
SensorTrackService_ADSB.idl
SensorTrackService_AIS.idl
SensorTrackService_EW.idl
SensorTrackService_IFF.idl
SystemTrackService.idl
HelperFunctionService.idl
FusionService.idl
TrackManagementService.idl

Table 6 - OACS IDL files

Appendix D – OARIS IDL Files

Table 7 lists the OARIS IDL files, as specified within Open Architecture Radar Interface Standard[2], that contain the data type definitions that have been incorporated as part of the ISAIN common data format.

IDL File
Air_Engagement_Support.idl
Clutter_Reporting.idl
Common_Types.idl
Control_Battle_Override.idl
Control_Emissions.idl
Control_Fault_Scripts.idl
Control_Recording.idl
Control_Replay.idl
Control_Simulation.idl
Coordinates_and_Positions.idl
Defint_Fault_Scripts.idl
Define_Simulation_Scenario.idl
Define_Test_Target_Scenario.idl
Delete_Sensor_Track.idl
Encyclopaedic_Support.idl
Engagement_Support.idl
Extended_Subsystem_Control.idl
Heartbeat_Signal.idl
Initiate_Track.idl
Manage_Frequency_Usage.idl
Manage_Mastership.idl
Manage_Operational_Mode.idl
Manage_Physical_Configuration.idl
Manage_Subsystem_Parameters.idl
Manage_Technical_State.idl
Manage_Tracking_Zones.idl
Manage_Transmission_Sectors.idl
Missile_Guidance.idl
Perform_Cued_Search.idl

Perform_Illumination.idl
Perform_Missile_Downlink.idl
Perform_Missile_Uplink.idl
Perform_Offline_Test.idl
Perform_Splash_Spotting.idl
Plot_Reporting.idl
Process_Target_Designation.idl
Provide_Area_with_Plot_Concentration.idl
Provide_Clutter_Assessment.idl
Provide_Health_State.idl
Provide_Interference_Reports.idl
Provide_Jammer_Assessment.idl
Provide_Nominal_Performance.idl
Provide_Performance_Assessment.idl
Provide_Projectile_Positional_Information.idl
Provide_Subsystem_Identification.idl
Receive_Encyclopaedic_Data.idl
Receive_Track_Information.idl
Recording_and_Replay.idl
Register_Interest.idl
Requests.idl
Restart.idl
Search.idl
Sensor_Control.idl
Shape_Model.idl
Shutdown.idl
Simulation_Support.idl
Startup.idl
Subsystem_Control.idl
Support_Kill_Assessment.idl
Surface_Engagement_Support.idl
System_Track.idl
Test_Target_Facility.idl

TimeBase.idl
Tracking_Control.idl
Track_Reporting.idl

Table 7 - OARIS IDL files

Appendix E – ISAINService IDL

To support integration of hostility assessment AIs, the HostilityAssessment data type shown in Figure 26 is included as part of the ISAIN common data format.

```
#ifndef ISAINSERVICEDEFVAR
#define ISAINSERVICEDEFVAR

module uk {
    module mod {
        module dstl {
            module isain {
                module ISAINService {
                  enum HostilityType
                  {
                    // SYCOIEA values
                    NO_STATEMENT,
                    PENDING,
                    UNKNOWN,
                    ASSUMED_FRIEND,
                    FRIENDLY,
                    NEUTRAL,
                    SUSPECT,
                    HOSTILE,

                    // TE2 values
                    CONCERN,
                    NO_CONCERN,
                    THREAT
                  };

                  struct HostilityAssessment
                  {
                    long trackId;
                    HostilityType hostility;
                    string justification;
                    unsigned long long timeOfValidity; // 1 unit is 100 nanoseconds
                  };

                  typedef sequence<HostilityAssessment, 1000> HostilityAssessmentType;

                  struct AgentOutput
                  {
                    string agentName;  //@Key
                    HostilityAssessmentType hostilityAssessments;
                    long long timeStamp;  //@Key
                  };
                  #pragma keylist AgentOutput agentName timeStamp
                };
            };
        };
    };
};
#endif
```

Figure 26 - ISAINService IDL