Odin

Release v21.01

Bharath Keshavamurthy

Mar 05, 2021

CONTENTS:

1 e2e 1

1.1 Forge module 1

1.2 Raven module 9

1.3 Realms module 17

1.4 RxRealm module 18

1.5 TxRealm module 19

2 Indices and tables 21

Python Module Index 23

Index 25

CHAPTER

ONE

E2E

1.1 Forge module

This script defines the necessary utilities for Project Odin–such as, logging parameters, decorator methods, representation callees, message validations, message parsers, etc.

Author: Bharath Keshavamurthy <bkeshava@purdue.edu>

Organization: School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN.

class `Forge.`**Controller**

Bases: `abc.ABC`

An abstract class for XXRealm Python Controllers

abstract property kafka_client

The Kafka client

Returns: The KafkaClient associated with this XXRealm Python Controller

abstract property realm

The realm-type of this controller implementation

Returns: The realm-type (member of RealmTypes enumeration) of this controller implementation

abstract property registration_key

The registration key of this controller implementation obtained post-registration from the Centralized Realms Python Controller

Returns: The registration key of this controller implementation: Set via a callback from an exposed method

abstract property serial_comm

The serial communication interface

Returns: The uC-XXRealm Python Controller serial communication interface configuration (SerialCommConfig)

abstract property setup_handler

The setup handler

Returns: The SetupHandler associated with this XXRealm Python Controller

abstract start`()` → None

Start the control operations at the XXRealm Python Controllers For example, At the TxRealm Python Controller (v21.01 ControllerImpl): uC-based servo control trigger via serial–post rotation angle estimation; and

At the RxRealm Python Controller (v21.01 ControllerImpl): uC-based servo control trigger via serial–post rotation angle estimation.

abstract property uid

The unique identifier of this controller implementation

Returns: The unique string identifier of this controller implementation (Picks up the UID during the initial installation configuration of this XXRealm Python Controller by Docker)

exception `Forge.`**InvalidControllerConfiguration**

Bases: `Exception`

Invalid Controller Configuration Error

This exception is raised when invalid configurations are received for the XXRealm Python Controller that is to be registered with the Centralized Realms Python Controller.

exception `Forge.`**InvalidDictError**

Bases: `Exception`

Invalid Dict Error

This exception is raised when an invalid/unsupported Python dictionary is provided in order to pack it into a Python dataclass instance.

class `Forge.`**KafkaClient**(*config:* Raven.KafkaConfig)

Bases: `object`

A Kafka client encapsulation

get_consumer(*topic:* Raven.KafkaTopics) → kafka.consumer.group.KafkaConsumer

Get Kafka consumer for the specified topic

Parameters **topic** – The KafkaTopics enumeration member–the consumer for which is to be returned

Returns: The KafkaConsumer instance associated with the given topic

get_kafka_api_impl_pair(*topic:* Raven.KafkaTopics) → *Raven.KafkaAPIImplPair*

Get Kafka producer/consumer API implementation pair for the specified topic

Parameters **topic** – The KafkaTopics enumeration member for which the producer-consumer pair is to be returned

Returns: The producer-consumer pair associated with the specified topic

get_producer(*topic:* Raven.KafkaTopics) → kafka.producer.kafka.KafkaProducer

Get Kafka producer for the specified topic

Parameters **topic** – The KafkaTopics enumeration member–the producer for which is to be returned

Returns: The KafkaProducer instance associated with the given topic

set_consumer(*topic:* Raven.KafkaTopics, *consumer: kafka.consumer.group.KafkaConsumer*) → None

Set Kafka consumer for the specified topic

Parameters

- **topic** – The KafkaTopics enumeration member for which the consumer is to be set
- **consumer** – The KafkaConsumer instance that is being registered with the given topic for this KafkaClient instance

set_kafka_api_impl_pair(*topic: Raven.KafkaTopics*, *producer: kafka.producer.kafka.KafkaProducer*, *consumer: kafka.consumer.group.KafkaConsumer*) → None

Set Kafka producer/consumer API implementation pair for the specified topic

Parameters

- **topic** – The KafkaTopics enumeration member for which the producer-consumer pair is to be set
- **producer** – The KafkaProducer instance constituting one-half of the KafkaAPIImplPair
- **consumer** – The KafkaConsumer instance constituting one-half of the KafkaAPIImplPair

set_producer(*topic: Raven.KafkaTopics*, *producer: kafka.producer.kafka.KafkaProducer*) → None

Set Kafka producer for the specified topic

Parameters

- **topic** – The KafkaTopics enumeration member for which the producer is to be set
- **producer** – The KafkaProducer instance that is being registered with the given topic for this KafkaClient instance

exception Forge.**KafkaClientNotRegisteredError**

Bases: `IndexError`

Kafka Client Not-Registered Error

This exception is raised when a connection creation request is placed by an unregistered KafkaClient to the KafkaConnectionFactory.

class Forge.**KafkaConnectionFactory**

Bases: `object`

A Singleton class to create connections (producers/consumers) for Kafka clients

create_connection(*registration_number: int*, *api: Raven.KafkaAPIs*, *topics: Tuple[Raven.KafkaTopics]*) → None

Create API-specific connections for registered Kafka clients–with respect to the given topics

Parameters

- **registration_number** – The KafkaClient's registration number
- **api** – The API implementation instance that is to be created for this client
- **topics** – The tuple of KafkaTopics for which API implementations are to be associated w.r.t the KafkaClient indexed by the provided registration number

Raises

- **NotImplementedError** – Method or function hasn't been implemented yet
- ***KafkaClientNotRegisteredError*** – This exception is raised when a connection creation request is placed by an unregistered KafkaClient to the KafkaConnectionFactory.

de_register(*registration_number: int*) → None

De-register the Kafka client indexed by its registration number

Parameters **registration_number** – Unregister the KafkaClient using this argument

deregister(*client: Forge.KafkaClient*) → None

De-register the given Kafka client

Parameters **client** – The KafkaClient that is to be unregistered from this connection factory

force_register(*client: Forge.KafkaClient*, *registration_number: int = - 1*) → int
Use this method to force re-registration of your Kafka client

Parameters

- **client** – The KafkaClient that is to be registered with this connection factory
- **registration_number** – If this argument is specified, remove this indexed KafkaClient from the registry, and re-register it

Returns: The new registration number

static get_factory()
Instance access method for this Singleton

Returns: The ONE and ONLY instance of this Kafka connection factory

get_registration_number(*client: Forge.KafkaClient*) → int
Get the registration number for the given KafkaClient

Parameters **client** – The KafkaClient whose registration number is to be returned

Returns: The registration number of the specified Kafka client

register(*client: Forge.KafkaClient*) → int
Register the Kafka client with this connection factory–and, return its registration number

Parameters **client** – The KafkaClient that is to be registered with this connection factory

Returns: The client's registration number post-registration

exception Forge.**KafkaConnectionFactoryInstantiationError**
Bases: Exception

Kafka Connection Factory Instantiation Error

This exception is raised when the instantiation of the Singleton KafkaConnectionFactory instance FAILS.

exception Forge.**KafkaConsumptionError**(**args*, ***kwargs*)
Bases: Exception

Kafka Consumption Error

This exception is raised when something went wrong while consuming from the specified Kafka topic.

DEPRECATION NOTE: This exception is no longer necessary due to a re-design which involves a better way to handle
consumption errors.

exception Forge.**KafkaProductionError**
Bases: Exception

Kafka Production Error

This exception is raised when something went wrong while publishing the given message to the specified Kafka topic.

exception Forge.**KafkaUnknownConnectionError**
Bases: Exception

Kafka Unknown Connection Error

This exception is raised when production/consumption is initiated for a KafkaTopic without creating its associated connections in the KafkaConnectionFactory.

Forge.**LOGGING_DATE_TIME_FORMAT = '%Y-%m-%dT%H:%M:%S'**
The rpyc.utils.server.ThreadedServer properties defined as a namedtuple utility

exception Forge.**NMEAValidationError**

Bases: `Exception`

NMEA Validation Error

This exception is raised when the NMEA validation of the GPS data received over the serial communication interface between the uC and the XXRealm Python Controllers FAILS.

Forge.**REALMS_PORT_ENVIRONMENT_VARIABLE = 'REALMS_PORT'**

Logging

Forge.**REALMS_THREADED_SERVER_DETAILS**

Decorators

alias of `Forge.RealmsThreadedServerDetails`

exception Forge.**RealmControllerNotRegisteredError**

Bases: `Exception`

Realm Controller Not Registered Error

This exception is raised when accesses are made to core methods in an unregistered XXRealm Python Controller.

exception Forge.**RealmsInstantiationError**

Bases: `Exception`

Realms Instantiation Error

This exception is raised when the instantiation of the Singleton Realms instance FAILS.

exception Forge.**RealmsStartupPipelineExecutionError**

Bases: `Exception`

Realms Startup Pipeline Execution Error

This exception is raised when an error has occurred during the execution of the startup pipeline in the Centralized Realms Python Controller. The error could be due to the following sequenced reasons (note the persistence of error): a. Unsupported platform encountered: The Centralized Realms Python Controller can only be run on Linux, b. A valid, single properties file could not be found for either Zookeeper or the Kafka server, c. Zookeeper startup failed, d. Kafka server startup failed, or e. Kafka topic creation failed.

class Forge.**SetupHandler**

Bases: `abc.ABC`

An abstract class definition for configuration & setup handling w.r.t the XXRealm Python Controllers

abstract setup(*mandates*) → None

Start the Python Controller's setup tasks For example, At the TxRealm Python Controller's setup handler (v21.01 SetupHandlerImpl): Parse the NMEA GPS data from the uC (received over serial) into a GPSEvent instance, publish the JSON represented GPSEvent as an ODIN_GPS_EVENT <module.__name__ (TxRealm), GPSEvent.json_repr> to the ODIN_TX_GPS_EVENTS Kafka topic, and simultaneously subscribe to the ODIN_RX_GPS_EVENTS Kafka topic–parse the consumed JSON ODIN_GPS_EVENTs <module.__name__ (RxRealm), GPSEvent.json_repr> from the Rx into the GPSEvent dataclass for use in the TxRealm Python Controller (v21.01 ControllerImpl); and

At the RxRealm Python Controller's setup handler (v21.01 SetupHandlerImpl): Parse the NMEA GPS data from the uC (received over serial) into a GPSEvent instance, publish the JSON represented GPSEvent as an ODIN_GPS_EVENT <module.__name__ (RxRealm), GPSEvent.json_repr> to the ODIN_TX_GPS_EVENTS Kafka topic, and simultaneously subscribe to the ODIN_RX_GPS_EVENTS Kafka topic–parse the consumed JSON ODIN_GPS_EVENTs <module.__name__ (TxRealm), GPSEvent.json_repr> from the Tx into the GPSEvent dataclass for use in the RxRealm Python Controller (v21.01 ControllerImpl).

NOTE: The Tx in v21.01 is fixed (Mobility.IMMOBILE): So, hard-code the Tx position in the Tx uC control code, and post the positional data in the agreed-upon NMEA contract on the uC-TxRealm Python Controller serial interface.

Parameters **`mandates`** – The XXRealm Python Controller's Kafka MOM mandates

`Forge.`**`TOPIC_NAME_SEPARATOR = '_'`**

note here that these are set during installation via the Docker CLI

Type The environment variables (system properties) associated with the various modules in Project Odin

`exception` `Forge.`**`ThreadedServerNotConfiguredError`**

Bases: `Exception`

Threaded Server Not Configured Error

This exception is raised when either the hostname or the port is not configured as environment variables (or system properties) on the Centralized Realms Python Controller's platform.

`Forge.`**`accepts`**(**types*)

A decorator for function arg type checking

Parameters **`*types`** – The supported data types that are to be verified against the arguments provided to a method that is decorated with this routine.

Returns: check_accepts assertion

`Forge.`**`comm_fetch_publish`**(**args*, ***kwargs*) → None

A utility method to fetch Comm data and publish it to the associated Kafka topic

Parameters

- **`*args`** – Non-keyword arguments
- **`**kwargs`** – Keyword arguments

Raises **`NotImplementedError`** – Method or function hasn't been implemented yet.

`Forge.`**`comm_subscribe`**(**args*, ***kwargs*) → None

A utility method to subscribe to the Kafka topic associated with Comm data in Project Odin

Parameters

- **`*args`** – Non-keyword arguments
- **`**kwargs`** – Keyword arguments

Raises **`NotImplementedError`** – Method or function hasn't been implemented yet.

`Forge.`**`connect`**(*funcs*)

A pipeline creation method

Parameters **`funcs`** – The routines to be pipelined in the order in which they are provided

Returns: A wrapper method that encapsulates the provided routines (in the order in which they are provided) in a a data/functional pipeline

`Forge.`**`ctrl_fetch_publish`**(**args*, ***kwargs*) → None

A utility method to fetch Control data and publish it to the associated Kafka topic

Parameters

- **`*args`** – Non-keyword arguments
- **`**kwargs`** – Keyword arguments

Raises **NotImplementedError** – Method or function hasn't been implemented yet.

Forge.**ctrl_subscribe**(**args*, ***kwargs*) → None

A utility method to subscribe to the Kafka topic associated with Control data in Project Odin

Parameters

- ***args** – Non-keyword arguments
- ****kwargs** – Keyword arguments

Raises **NotImplementedError** – Method or function hasn't been implemented yet.

Forge.**deprecated**(*func*)

This is a decorator which can be used to mark functions as deprecated

Parameters **func** – The method (decorated with @deprecated) against which this deprecation check is made

Returns: Warnings for deprecation outlined in new_func

Forge.**get_basic_logging**() → Dict[str, Any]

Get the basic logging configurations for all the components in Project Odin

Returns: A Python dictionary encapsulating the basic, common configurations employed for logging in almost every major component in Project Odin

Forge.**get_file_name**(*kafka_api:* Raven.KafkaAPIs, *seq_number: int*) → str

Get the event log file name in accordance with the Kafka API reference and the specified event sequence number

Parameters

- **kafka_api** – The KafkaAPIs enumeration member for which the event log file name is to be returned
- **seq_number** – The event sequence number that is incorporated into the log file's name

Returns: The event log file name w.r.t the Kafka API reference and the specified event sequence number

Forge.**gps_fetch_publish**(*mobility:* Raven.Mobility, *serial_comm:* Raven.SerialCommConfig, *kafka_client:* Forge.KafkaClient, *kafka_topic:* Raven.KafkaTopics, *gps_event:* Raven.GPSEvent, *logger: logging.Logger*, *lock: _thread.allocate_lock*) → None

A utility method to connect to the uC's Serial port, extract the NMEA GPS data, process it into a GPSEvent, and publish it to the Apache Kafka Message Oriented Middleware (MOM) framework on the specified topic.

DESIGN_NOTE: Although a lot of the checks in this method may seem gratuitous, they are necessary because this is a utility method and external calls to this method may be totally wild and non-conforming as Project Odin scales.

Parameters

- **mobility** – The Mobility enumeration member in order to determine if we need indefinite GPSEvent publishes
- **serial_comm** – The uC-XXRealm Python Controller serial communication interface configuration
- **kafka_client** – The KafkaClient instance whose KafkaProducer is used for the required publishes
- **kafka_topic** – The KafkaTopics enumeration member to which the KafkaProducer publishes processed GPS data
- **gps_event** – The GPSEvent dataclass instance of the caller that is to be populated in an indefinite thread

- **logger** – The logger instance passed down by the caller for logging and module identification
- **lock** – The threading.Lock object to avoid resource access problems/race conditions

Raises

- **SerialException** – Base class for serial port related exceptions
- ***KafkaClientNotRegisteredError*** – This exception is raised when a connection creation request is placed by an unregistered KafkaClient to the KafkaConnectionFactory.
- ***KafkaUnknownConnectionError*** – This exception is raised when production/consumption is initiated for a KafkaTopic without creating its associated connections in the KafkaConnectionFactory.
- ***NMEAValidationError*** – This exception is raised when the NMEA validation of the GPS data received over the serial communication interface between the uC and the XXRealmPython Controllers FAILS.
- ***KafkaProductionError*** – This exception is raised when something went wrong while publishing the given message to the specified Kafka topic.
- **Exception** – Common base class for all non-exit exceptions.

Forge.**gps_subscribe**(*kafka_client: Forge.KafkaClient*, *kafka_topic: Raven.KafkaTopics*, *gps_event: Raven.GPSEvent*, *allowed_publishers: Tuple[str]*, *logger: logging.Logger*, *lock: _thread.allocate_lock*) → None

A utility method to subscribe to the specified Kafka topic

DESIGN_NOTE: Although a lot of the checks in this method may seem gratuitous, they are necessary because this is a utility method and external calls to this method may be totally wild and non-conforming as Project Odin scales.

Parameters

- **kafka_client** – The KafkaClient instance whose KafkaConsumer is used for the required subscriptions
- **kafka_topic** – The KafkaTopics enumeration member that is to be subscribed to by the KafkaClient's KafkaConsumer
- **gps_event** – The GPSEvent dataclass instance of the caller that is to be populated in an indefinite thread
- **allowed_publishers** – The filter which allows the KafkaConsumer on the provided KafkaTopics enumeration member to process only those messages whose keys match the publisher identifiers registered with the centralized Scheduler
- **logger** – The logger instance passed down by the caller for module identification
- **lock** – The threading.Lock object to avoid resource access problems/race conditions

Raises

- ***KafkaClientNotRegisteredError*** – This exception is raised when a connection creation request is placed by an unregistered KafkaClient to the KafkaConnectionFactory.
- ***KafkaUnknownConnectionError*** – This exception is raised when production/consumption is initiated for a KafkaTopic without creating its associated connections in the KafkaConnectionFactory.
- **Exception** – Common base class for all non-exit exceptions

Forge.**json_repr**(*dataclass_obj: dataclasses.dataclass*, *kafka_api:* Raven.KafkaAPIs, *seq_number: int*) → str

A utility method to save the JSON GPSEvent to a file (for logging) AND return the JSON-formatted string for the Kafka publish routine

Parameters

- **dataclass_obj** – The dataclass instance which is to be represented as a JSON string
- **kafka_api** – The KafkaAPI reference for event-specific log file name determination
- **seq_number** – The sequence number of event that is to be logged and represented as a JSON string

Returns: JSON representation of the provided dataclass instance

Forge.**nmea_parse**(*nmea_data: str*, *gps_event:* Raven.GPSEvent) → None

A utility method to parse NMEA data

Parameters

- **nmea_data** – The NMEA GPS string that is to be checked against the available GPSEvent dataclass instance
- **gps_event** – The available GPSEvent dataclass instance that is to be modified with the updates from the NMEA GPS string from the GPS receiver over the uC-XXRealm Python Controller serial communication interface

Forge.**nmea_validate**(*nmea_data: str*) → bool

A utility method to validate NMEA data

Parameters **nmea_data** – The NMEA GPS data string from the uC that is to be validated

Returns: True/False validation status

Forge.**pack_dict_into_dataclass**(*dict_: dict*, *dataclass_: dataclasses.dataclass*) → dataclasses.dataclass

Pack the data from a Python dictionary into a dataclass instance

Parameters

- **dict** – The dictionary to be packed into a dataclass instance
- **dataclass** – The dataclass whose instance is to be returned post-dictionary-packing

Returns: An instance of the provided dataclass reference packed with fields and values from the provided dictionary

Raises: InvalidDictError This exception is raised when an invalid/unsupported Python dictionary is provided in order to pack it into a Python dataclass instance.

1.2 Raven module

This script defines the following dataclasses used for inter-system OR inter-module communication in Project Odin:

a. ODIN_GPS_EVENT (GPSEvent contract between the TxRealm and the RxRealm Python Controllers);

b. BIFROST (SerialCommConfig for data transfer between the uC and the TxRealm/RxRealm Python Controllers); and

c. HEIMDALL (KafkaConfig for publish/subscribe between the TxRealm and RxRealm Python Controllers).

Author: Bharath Keshavamurthy <bkeshava@purdue.edu>

Organization: School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN.

class `Raven.`**`CarrierSolutionType`**`(`*value*`)`

Bases: `enum.Enum`

An enumeration listing all possible carrier solution-types supported by the GPS receiver

`FIXED_SOLUTION = 2`

`FLOAT_SOLUTION = 1`

`NO_SOLUTION = 0`

class `Raven.`**`CommEvent`**

Bases: `object`

Primary information tier in CommEvent

The CommEvent class: ODIN_COMM_EVENT communicated over the ODIN_COMM_EVENTS Kafka topic encapsulates this data object

class `Raven.`**`ControlEvent`**

Bases: `object`

Primary information tier in ControlEvent

The ControlEvent class: ODIN_CONTROL_EVENT communicated over the ODIN_CONTROL_EVENTS Kafka topic encapsulates this data object

class `Raven.`**`ControllerMandate`**`(`*data_type:* Raven.DataTypes, *production_topic:* Raven.KafkaTopics, *consumption_topic:* Raven.KafkaTopics, *production_routine:* *Any*, *consumption_routine:* *Any*, *allowed_producers: list*`)`

Bases: `object`

A Message Oriented Middleware (Kafka MOM) framework mandate for an XXRealm Python Controller in Project Odin

`allowed_producers: list`

`consumption_routine: Any`

`consumption_topic:` ***`Raven.KafkaTopics`***

`data_type:` ***`Raven.DataTypes`***

`production_routine: Any`

`production_topic:` ***`Raven.KafkaTopics`***

class `Raven.`**`DataTypes`**`(`*value*`)`

Bases: `enum.Enum`

An enumeration listing the various types of data exchanged among realms in Project Odin

`COMM = 'COMM'`

`CTRL = 'CONTROL'`

`GPS = 'GPS'`

class `Raven.`**`FixType`**`(`*value*`)`

Bases: `enum.Enum`

An enumeration listing all possible fix-types supported by the GPS receiver

DEAD_RECKONING = 1

GNSS = 4

NO_FIX = 0

THREE_DIMENSIONAL = 3

TIME_FIX = 5

TWO_DIMENSIONAL = 2

class Raven.**GPSEvent**(seq_number: int = 0, timestamp: datetime.datetime = datetime.datetime(2021, 3, 5, 19, 22, 47, 948539), is_gnss_fix_ok: bool = False, siv: int = 0, fix_type: Raven.FixType = <FixType.NO_FIX: 0>, carrier_solution_type: Raven.CarrierSolutionType = <CarrierSolutionType.NO_SOLUTION: 0>, latitude: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), longitude: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), altitude_ellipsoid: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), altitude_msl: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), speed: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), heading: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), horizontal_acc: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), vertical_acc: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), speed_acc: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), heading_acc: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), ned_north_vel: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), ned_east_vel: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), ned_down_vel: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), pdop: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), mag_acc: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), mag_dec: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), geometric_dop: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), position_dop: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0, precision=0.0, units=<Units.DIMENSIONLESS: 9>), time_dop: Raven.Member = Member(is_high_precision=False, component=0.0, main_component=0.0, high_precision_component=0.0,

Bases: object

Primary information tier in GPSEvent

The GPSEvent class: ODIN_GPS_EVENT communicated over the ODIN_GPS_EVENTS Kafka topic encapsulates this data object

altitude_ellipsoid: *Raven.Member* = Member(is_high_precision=False, component=0.0, mai

altitude_msl: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

carrier_solution_type: *Raven.CarrierSolutionType* = 0

easting_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_compo

fix_type: *Raven.FixType* = 0

geometric_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_com

heading: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_component

heading_acc: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_compo

horizontal_acc: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_co

horizontal_accuracy: *Raven.Member* = Member(is_high_precision=False, component=0.0, ma

horizontal_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_co

is_gnss_fix_ok: bool = False

latitude: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_componen

longitude: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_compone

mag_acc: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_component

mag_dec: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_component

ned_down_vel: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

ned_east_vel: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

ned_north_vel: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_com

northing_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

pdop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_component=0.

position_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

seq_number: int = 0

siv: int = 0

speed: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_component=0

speed_acc: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_compone

time_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_componen

timestamp: datetime.datetime = datetime.datetime(2021, 3, 5, 19, 22, 47, 948539)

vertical_acc: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

vertical_accuracy: *Raven.Member* = Member(is_high_precision=False, component=0.0, main

vertical_dop: *Raven.Member* = Member(is_high_precision=False, component=0.0, main_comp

class Raven.**KafkaAPIImplPair**(*producer: Optional[kafka.producer.kafka.KafkaProducer] = None, consumer: Optional[kafka.consumer.group.KafkaConsumer] = None*)

Bases: `object`

A dataclass encapsulating a standard Kafka API impl pair used in Project Odin

consumer: kafka.consumer.group.KafkaConsumer = None

producer: kafka.producer.kafka.KafkaProducer = None

class Raven.**KafkaAPIs**(*value*)

Bases: `enum.Enum`

An enumeration outlining the APIs provided by the Kafka Message Oriented Middleware (MOM) framework

CONNECTOR = 3

CONSUMER = 1

PRODUCER = 0

STREAM = 2

class Raven.**KafkaConfig**(*client_id: str = 'xxx', group_id: str = 'yyy', broker_id: int = 0, acks: bool = True, bootstrap_server_config: str = 'localhost:9093', zookeeper_config: str = 'localhost:2181', retry_backoff: float = 3.0, poll_interval: float = 3.0, commit_each_poll: bool = True, auto_commit_interval: float = 0.1, use_synch_mode: bool = True, api_version: Tuple = (0, 10), auto_offset_reset: str = 'earliest', consumer_timeout: float = 0.1*)

Bases: `object`

A dataclass defining the args associated with the Apache Kafka publish/subscribe framework between the TxRealm and RxRealm Python Controllers

acks: bool = True

api_version: Tuple = (0, 10)

auto_commit_interval: float = 0.1

auto_offset_reset: str = 'earliest'

bootstrap_server_config: str = 'localhost:9093'

broker_id: int = 0

client_id: str = 'xxx'

commit_each_poll: bool = True

consumer_timeout: float = 0.1

group_id: str = 'yyy'

poll_interval: float = 3.0

retry_backoff: float = 3.0

use_synch_mode: bool = True

zookeeper_config: str = 'localhost:2181'

class Raven.**KafkaTopicConfig**(*name: str = 'ODIN_XX_XXXX_EVENTS', partitions: int = 1, replication_factor: int = 1*)

Bases: `object`

A dataclass encapsulating all the configs needed to create a Kafka topic at the Centralized Realms Python Controller

name: str = 'ODIN_XX_XXXX_EVENTS'

partitions: int = 1

replication_factor: int = 1

class Raven.**KafkaTopics**(*value*)

Bases: enum.Enum

An enumeration listing the various topics employed in Project Odin within the Kafka publish/subscribe framework

DESIGN NOTES:

a. Publishes to Kafka topics happen based on the Realm of the publisher and the type of data being communicated;

b. Subscriptions also happen based on the realm of the subscriber and the type of data being communicated, but the key-based message filtering for actual message consumption from the subscribed topic will be done based on the allowed_publishers filter; and

c. Generally, the Kafka topic to which a XXRealm Python Controller subscribes to belongs to the opposite Realm.

ODIN_CONTROL_EVENTS = KafkaTopicConfig(name='ODIN_CONTROL_EVENTS', partitions=3, repli

ODIN_RX_COMM_EVENTS = KafkaTopicConfig(name='ODIN_RX_COMM_SAMPLES', partitions=3, repl

ODIN_RX_GPS_EVENTS = KafkaTopicConfig(name='ODIN_RX_GPS_EVENTS', partitions=3, replica

ODIN_TX_COMM_EVENTS = KafkaTopicConfig(name='ODIN_TX_COMM_SAMPLES', partitions=3, repl

ODIN_TX_GPS_EVENTS = KafkaTopicConfig(name='ODIN_TX_GPS_EVENTS', partitions=3, replica

Raven.**MOM_PROPERTIES**

alias of Raven.MOMProperties

Raven.**MOM_ROUTINE_PAIR**

Enumerations and Dataclasses relevant to the Kafka Message Oriented Middleware (MOM) framework's API calls within XXRealm Python Controllers

alias of Raven.MOMRoutinePair

class Raven.**Member**(*is_high_precision: bool = False*, *component: float = 0.0*, *main_component: float = 0.0*, *high_precision_component: float = 0.0*, *precision: float = 0.0*, *units: Raven.Units = <Units.DIMENSIONLESS: 9>*)

Bases: object

Secondary information tier in GPSEvent

A core member which encapsulates highly specific details about latitude, longitude, altitude, speed, heading, and other components in the primary information tier of GPSEvent

component: float = 0.0

high_precision_component: float = 0.0

is_high_precision: bool = False

main_component: float = 0.0

precision: float = 0.0

units: ***Raven.Units*** **= 9**

class Raven.**Mobility**(*value*)

Bases: `enum.Enum`

An enumeration listing the mobility configurations of the Tx in Project Odin

CONTINUOUS_DRONE = 2

CONTINUOUS_ROVER = 1

DISCONTINUOUS_DRONE = 4

DISCONTINUOUS_ROVER = 3

IMMOBILE = 0

Raven.**PIPELINE_INTERNAL_CAPSULE**

alias of `Raven.PipelineInternalCapsule`

class Raven.**RealmTypes**(*value*)

Bases: `enum.Enum`

An enumeration listing the various supported Realm Types (Node Types) in Project Odin

AGGREGATOR = 'Agg'

GATEWAY = 'Gw'

RECEPTION = 'Rx'

REPEATER = 'Rp'

TRANSMISSION = 'Tx'

class Raven.**SerialCommConfig**(*id: str = 'xxx'*, *is_wireless: bool = False*, *port: str = 'COMx'*, *baud_rate: int = 9600*, *timeout: float = 0.1*, *sleep_duration: float = 0.1*)

Bases: `object`

A dataclass defining the args associated with the serial communication interface between the uC and TxRealm/RxRealm Python Controllers

baud_rate: int = 9600

id: str = 'xxx'

is_wireless: bool = False

port: str = 'COMx'

sleep_duration: float = 0.1

timeout: float = 0.1

class Raven.**Units**(*value*)

Bases: `enum.Enum`

An enumeration listing all possible units for GPSEvent members

CENTIMETERS = 1

DEGREES = 3

DIMENSIONLESS = 9

FEET = 7

INCHES = 6

METERS = 0

MILLIMETERS = 2

MINUTES = 4

SECONDS = 5

YARDS = 8

1.3 Realms module

This script details the operations at the Centralized Python Controller that handles:

a. The registration and de-registration of XXRealm Python Controllers–along with their operational access rights authentication and subscription filtration tasks;

b. The control operations related to the global frameworks employed in Project Odin, i.e., Zookeeper startup, Kafka server startup, Kafka topics creation, Zookeeper status check, Kafka server status check, etc.

DESIGN NOTE: For optimal deployment, this Centralized Realms Python Controller is restricted to run only on Linux.

Author: Bharath Keshavamurthy <bkeshava@purdue.edu>

Organization: School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN.

class `Realms.`**Realms**

Bases: `object`

A Singleton class encapsulating the centralized control operations for Project Odin

async exposed_controllers_count() → int

Get the number of registered XXRealm Python Controllers in this deployment of Project Odin

Returns: the number of registered XXRealm Python Controllers in this deployment of Project Odin

async exposed_get_allowed_publishers(*controller_uid: str*) → List[str]

Get the allowed publishers for an XXRealm Python Controller referenced by its UID

Parameters **controller_uid** – The unique identifier of the XXRealm Python Controller

Returns: The allowed publishers for an XXRealm Python Controller referenced by its UID

async exposed_get_association(*data_type:* Raven.DataTypes) → Any

Get the Kafka API routine association for the specified data_type

Parameters **data_type** – The DataType enumeration member corresponding to which a Kafka API routine in Project Odin is to be returned

Returns: The Kafka API routine in Project Odin corresponding to the provided DataType enumeration member

async static exposed_get_topic(*controller:* Forge.Controller, *data_type:* Raven.DataTypes) → *Raven.KafkaTopics*

Get the Kafka topic for the XXRealm Python Controller associated with the provided registration_key

Parameters

- **controller** – The registered controller implementation requesting a Kafka topic for its publishes

- **data_type** – The type of data being published by this registered referenced XXRealm Python Controller

Returns: The Kafka topic for the registered referenced XXRealm Python Controller's publishes of the specified
data_type

Raises **XXRealmPythonControllerNotRegisteredError** – This exception is raised when accesses are made to core methods in an unregistered XXRealm Python Controller.

async exposed_register(*controller: Forge.Controller*, *data_type_associations: dict*, *callback: Optional[Any] = None*) → None

Register a controller implementation instance with this Centralized Realms Python Controller

DESIGN NOTE: The scalability of this code is unlimited horizontally within each realm, while there is no need
for vertical scalability because the number of realms will be a very small number (max 5).

Parameters

- **controller** – The controller implementation instance to be registered
- **data_type_associations** – The data type associations in Project Odin (these are global and cannot be overwritten during registration)
- **callback** – The callback routine in the XXRealm Python Controller which will be triggered post-registration

Raises ***InvalidControllerConfiguration*** – This exception is raised when invalid configurations are received for the XXRealm Python Controller that is to be registered.

static get_realms()

Instance access method for this Singleton

Returns: The ONE and ONLY instance of this Realms Controller

1.4 RxRealm module

This script describes the operations performed by the controller at Rx side of the Odin channel statistics measurement campaign. The operations performed by this controller include:

a. Receive the custom NMEA-formatted location messages from the microcontroller over the USB/BT serial COM port (SerialUSB/SerialBT): (The Rx uC-based (RxController.ino) operations are detailed below)

1. Use the uC-GPS parser library to parse the standard NMEA GPS messages and obtain the time, latitude, longitude, altitude, heading, velocity, and other metadata metrics from the getter-setter methods within this library; and

2. Using these extracted data values construct a custom NMEA-formatted message (start with $ and the ordered data values are comma-separated) and publish it to the USB/BT serial COM port within the Rx uC-based code (SerialUSB/SerialBT)

b. Parse the received custom NMEA-formatted location messages to populate a JSON data object which would next be encapsulated in an ODIN_GPS_EVENT; and

c. Publish the created ODIN_GPS_EVENT to the Kafka topic ODIN_GPS_EVENTS.

DESIGN NOTE: The RTCM correction stream is sent to the GPS module from an NTRIP client (Lefebure Windows Client) over another Bluetooth link with the same BT module.

Author: Bharath Keshavamurthy <bkeshava@purdue.edu>

Organization: School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN.

1.5 TxRealm module

This script describes the operations of the controller at the Tx side of the Odin channel statistics measurement campaign. The operations performed by this controller include:

1. Subscribe to the ODIN_GPS_EVENTS Kafka topic to receive the JSON-based ODIN_GPS_EVENTs (Rx location msgs);

2. Parse these JSON ODIN_GPS_EVENTs to extract the (time, latitude, longitude, altitude, attitude, . . .) “data object” collection corresponding to the Rx; and

3. Determine the rotation_angle (the angle which the Tx should turn to w.r.t the home_plate) and publish it (with a timestamp) to the USB/BT serial monitor COM port of the microcontroller (SerialUSB/SerialBT). Note that the timestamps might be redundant–but, are necessary for post-operation analyses of system delays/timing synchronization.

DESIGN NOTE: The Tx is stationary (fixed at a mount-point) in this version of Odin (v21.01).

Author: Bharath Keshavamurthy <bkeshava@purdue.edu>

Organization: School of Electrical & Computer Engineering, Purdue University, West Lafayette, IN.

class TxRealm.**TxController**

Bases: *Forge.Controller*

The Tx controller class (v21.01)

static data_type_associations() → Dict[*Raven.DataTypes*, Raven.MOMRoutinePair]

The data type associations (these are global and cannot be overwritten during registration)

Returns: The data_type-publish/subscribe routine associations

property kafka_client

The Kafka client getter method

Returns: KafkaClient

property realm

The realm-type getter method

Returns: RealmTypes.TRANSMISSION

property registration_key

The registration key getter method

Returns: The registration key of this TxRealm Python Controller post-registration with the Centralized Realms Python Controller

Raises XXRealmPythonControllerNotRegisteredError – This exception is raised when accesses are made to core methods in an unregistered XXRealm Python Controller.

property serial_comm

The serial communication interface getter method

Returns: SerialCommConfig

property setup_handler

The setup handler getter method

Returns: SetupHandler

start() → None

Start the control operations: Rotation angle estimation, and post the angle to serial for uC-based servo control

property uid

The UID getter method

Returns: The unique identifier (UID) of this TxRealm Python Controller

class TxRealm.**TxSetupHandler**(*mobility:* Raven.Mobility, *serial_comm:* Raven.SerialCommConfig, *kafka_config:* Raven.KafkaConfig)

Bases: *Forge.SetupHandler*

The configuration details and configuration setup tasks of the Tx rotating platform

setup(*mandates*) → None

Start the TxRealm Python Controller's setup tasks

Parameters mandates – A collection of ControllerMandates for the Kafka MOM API calls

Raises NotImplementedError – Method or function hasn't been implemented yet.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

Forge, 1

r

Raven, 9
Realms, 17
RxRealm, 18

t

TxRealm, 19

INDEX

A

accepts() (*in module Forge*), 6

acks (*Raven.KafkaConfig attribute*), 14

AGGREGATOR (*Raven.RealmTypes attribute*), 16

allowed_producers (*Raven.ControllerMandate attribute*), 10

altitude_ellipsoid (*Raven.GPSEvent attribute*), 13

altitude_msl (*Raven.GPSEvent attribute*), 13

api_version (*Raven.KafkaConfig attribute*), 14

auto_commit_interval (*Raven.KafkaConfig attribute*), 14

auto_offset_reset (*Raven.KafkaConfig attribute*), 14

B

baud_rate (*Raven.SerialCommConfig attribute*), 16

bootstrap_server_config (*Raven.KafkaConfig attribute*), 14

broker_id (*Raven.KafkaConfig attribute*), 14

C

carrier_solution_type (*Raven.GPSEvent attribute*), 13

CarrierSolutionType (*class in Raven*), 10

CENTIMETERS (*Raven.Units attribute*), 16

client_id (*Raven.KafkaConfig attribute*), 14

COMM (*Raven.DataTypes attribute*), 10

comm_fetch_publish() (*in module Forge*), 6

comm_subscribe() (*in module Forge*), 6

CommEvent (*class in Raven*), 10

commit_each_poll (*Raven.KafkaConfig attribute*), 14

component (*Raven.Member attribute*), 15

connect() (*in module Forge*), 6

CONNECTOR (*Raven.KafkaAPIs attribute*), 14

consumer (*Raven.KafkaAPIImplPair attribute*), 14

CONSUMER (*Raven.KafkaAPIs attribute*), 14

consumer_timeout (*Raven.KafkaConfig attribute*), 14

consumption_routine (*Raven.ControllerMandate attribute*), 10

consumption_topic (*Raven.ControllerMandate attribute*), 10

CONTINUOUS_DRONE (*Raven.Mobility attribute*), 16

CONTINUOUS_ROVER (*Raven.Mobility attribute*), 16

ControlEvent (*class in Raven*), 10

Controller (*class in Forge*), 1

ControllerMandate (*class in Raven*), 10

create_connection() (*Forge.KafkaConnectionFactory method*), 3

CTRL (*Raven.DataTypes attribute*), 10

ctrl_fetch_publish() (*in module Forge*), 6

ctrl_subscribe() (*in module Forge*), 7

D

data_type (*Raven.ControllerMandate attribute*), 10

data_type_associations() (*TxRealm.TxController static method*), 19

DataTypes (*class in Raven*), 10

de_register() (*Forge.KafkaConnectionFactory method*), 3

DEAD_RECKONING (*Raven.FixType attribute*), 11

DEGREES (*Raven.Units attribute*), 16

deprecated() (*in module Forge*), 7

deregister() (*Forge.KafkaConnectionFactory method*), 3

DIMENSIONLESS (*Raven.Units attribute*), 16

DISCONTINUOUS_DRONE (*Raven.Mobility attribute*), 16

DISCONTINUOUS_ROVER (*Raven.Mobility attribute*), 16

E

easting_dop (*Raven.GPSEvent attribute*), 13

exposed_controllers_count() (*Realms.Realms method*), 17

exposed_get_allowed_publishers() (*Realms.Realms method*), 17

exposed_get_association() (*Realms.Realms method*), 17

exposed_get_topic() (*Realms.Realms static method*), 17 exposed_register() (*Realms.Realms method*), 18

F

FEET (*Raven.Units attribute*), 16
fix_type (*Raven.GPSEvent attribute*), 13
FIXED_SOLUTION (*Raven.CarrierSolutionType attribute*), 10
FixType (*class in Raven*), 10
FLOAT_SOLUTION (*Raven.CarrierSolutionType attribute*), 10
force_register() (*Forge.KafkaConnectionFactory method*), 3
Forge
  module, 1

G

GATEWAY (*Raven.RealmTypes attribute*), 16
geometric_dop (*Raven.GPSEvent attribute*), 13
get_basic_logging() (*in module Forge*), 7
get_consumer() (*Forge.KafkaClient method*), 2
get_factory() (*Forge.KafkaConnectionFactory static method*), 4
get_file_name() (*in module Forge*), 7
get_kafka_api_impl_pair() (*Forge.KafkaClient method*), 2
get_producer() (*Forge.KafkaClient method*), 2
get_realms() (*Realms.Realms static method*), 18
get_registration_number() (*Forge.KafkaConnectionFactory method*), 4
GNSS (*Raven.FixType attribute*), 11
GPS (*Raven.DataTypes attribute*), 10
gps_fetch_publish() (*in module Forge*), 7
gps_subscribe() (*in module Forge*), 8
GPSEvent (*class in Raven*), 11
group_id (*Raven.KafkaConfig attribute*), 14

H

heading (*Raven.GPSEvent attribute*), 13
heading_acc (*Raven.GPSEvent attribute*), 13
high_precision_component (*Raven.Member attribute*), 15
horizontal_acc (*Raven.GPSEvent attribute*), 13
horizontal_accuracy (*Raven.GPSEvent attribute*), 13
horizontal_dop (*Raven.GPSEvent attribute*), 13

I

id (*Raven.SerialCommConfig attribute*), 16
IMMOBILE (*Raven.Mobility attribute*), 16
INCHES (*Raven.Units attribute*), 16
InvalidControllerConfiguration, 2
InvalidDictError, 2
is_gnss_fix_ok (*Raven.GPSEvent attribute*), 13
is_high_precision (*Raven.Member attribute*), 15
is_wireless (*Raven.SerialCommConfig attribute*), 16

J

json_repr() (*in module Forge*), 8

K

kafka_client() (*Forge.Controller property*), 1
kafka_client() (*TxRealm.TxController property*), 19
KafkaAPIImplPair (*class in Raven*), 13
KafkaAPIs (*class in Raven*), 14
KafkaClient (*class in Forge*), 2
KafkaClientNotRegisteredError, 3
KafkaConfig (*class in Raven*), 14
KafkaConnectionFactory (*class in Forge*), 3
KafkaConnectionFactoryInstantiationError, 4
KafkaConsumptionError, 4
KafkaProductionError, 4
KafkaTopicConfig (*class in Raven*), 14
KafkaTopics (*class in Raven*), 15
KafkaUnknownConnectionError, 4

L

latitude (*Raven.GPSEvent attribute*), 13
LOGGING_DATE_TIME_FORMAT (*in module Forge*), 4
longitude (*Raven.GPSEvent attribute*), 13

M

mag_acc (*Raven.GPSEvent attribute*), 13
mag_dec (*Raven.GPSEvent attribute*), 13
main_component (*Raven.Member attribute*), 15
Member (*class in Raven*), 15
METERS (*Raven.Units attribute*), 16
MILLIMETERS (*Raven.Units attribute*), 16
MINUTES (*Raven.Units attribute*), 17
Mobility (*class in Raven*), 15
module
  Forge, 1
  Raven, 9
  Realms, 17
  RxRealm, 18
  TxRealm, 19
MOM_PROPERTIES (*in module Raven*), 15
MOM_ROUTINE_PAIR (*in module Raven*), 15

N

name (*Raven.KafkaTopicConfig attribute*), 15
ned_down_vel (*Raven.GPSEvent attribute*), 13
ned_east_vel (*Raven.GPSEvent attribute*), 13
ned_north_vel (*Raven.GPSEvent attribute*), 13
nmea_parse() (*in module Forge*), 9

nmea_validate() (*in module Forge*), 9
NMEAValidationError, 4
NO_FIX (*Raven.FixType attribute*), 11
NO_SOLUTION (*Raven.CarrierSolutionType attribute*), 10
northing_dop (*Raven.GPSEvent attribute*), 13

O

ODIN_CONTROL_EVENTS (*Raven.KafkaTopics attribute*), 15
ODIN_RX_COMM_EVENTS (*Raven.KafkaTopics attribute*), 15
ODIN_RX_GPS_EVENTS (*Raven.KafkaTopics attribute*), 15
ODIN_TX_COMM_EVENTS (*Raven.KafkaTopics attribute*), 15
ODIN_TX_GPS_EVENTS (*Raven.KafkaTopics attribute*), 15

P

pack_dict_into_dataclass() (*in module Forge*), 9
partitions (*Raven.KafkaTopicConfig attribute*), 15
pdop (*Raven.GPSEvent attribute*), 13
PIPELINE_INTERNAL_CAPSULE (*in module Raven*), 16
poll_interval (*Raven.KafkaConfig attribute*), 14
port (*Raven.SerialCommConfig attribute*), 16
position_dop (*Raven.GPSEvent attribute*), 13
precision (*Raven.Member attribute*), 15
producer (*Raven.KafkaAPIImplPair attribute*), 14
PRODUCER (*Raven.KafkaAPIs attribute*), 14
production_routine (*Raven.ControllerMandate attribute*), 10
production_topic (*Raven.ControllerMandate attribute*), 10

R

Raven
    module, 9
realm() (*Forge.Controller property*), 1
realm() (*TxRealm.TxController property*), 19
RealmControllerNotRegisteredError, 5
Realms
    module, 17
Realms (*class in Realms*), 17
REALMS_PORT_ENVIRONMENT_VARIABLE (*in module Forge*), 5
REALMS_THREADED_SERVER_DETAILS (*in module Forge*), 5
RealmsInstantiationError, 5
RealmsStartupPipelineExecutionError, 5
RealmTypes (*class in Raven*), 16
RECEPTION (*Raven.RealmTypes attribute*), 16
register() (*Forge.KafkaConnectionFactory method*), 4
registration_key() (*Forge.Controller property*), 1
registration_key() (*TxRealm.TxController property*), 19
REPEATER (*Raven.RealmTypes attribute*), 16
replication_factor (*Raven.KafkaTopicConfig attribute*), 15
retry_backoff (*Raven.KafkaConfig attribute*), 14
RxRealm
    module, 18

S

SECONDS (*Raven.Units attribute*), 17
seq_number (*Raven.GPSEvent attribute*), 13
serial_comm() (*Forge.Controller property*), 1
serial_comm() (*TxRealm.TxController property*), 19
SerialCommConfig (*class in Raven*), 16
set_consumer() (*Forge.KafkaClient method*), 2
set_kafka_api_impl_pair() (*Forge.KafkaClient method*), 2
set_producer() (*Forge.KafkaClient method*), 3
setup() (*Forge.SetupHandler method*), 5
setup() (*TxRealm.TxSetupHandler method*), 20
setup_handler() (*Forge.Controller property*), 1
setup_handler() (*TxRealm.TxController property*), 20
SetupHandler (*class in Forge*), 5
siv (*Raven.GPSEvent attribute*), 13
sleep_duration (*Raven.SerialCommConfig attribute*), 16
speed (*Raven.GPSEvent attribute*), 13
speed_acc (*Raven.GPSEvent attribute*), 13
start() (*Forge.Controller method*), 1
start() (*TxRealm.TxController method*), 20
STREAM (*Raven.KafkaAPIs attribute*), 14

T

ThreadedServerNotConfiguredError, 6
THREE_DIMENSIONAL (*Raven.FixType attribute*), 11
time_dop (*Raven.GPSEvent attribute*), 13
TIME_FIX (*Raven.FixType attribute*), 11
timeout (*Raven.SerialCommConfig attribute*), 16
timestamp (*Raven.GPSEvent attribute*), 13
TOPIC_NAME_SEPARATOR (*in module Forge*), 6
TRANSMISSION (*Raven.RealmTypes attribute*), 16
TWO_DIMENSIONAL (*Raven.FixType attribute*), 11
TxController (*class in TxRealm*), 19
TxRealm
    module, 19
TxSetupHandler (*class in TxRealm*), 20

U

uid() (*Forge.Controller property*), 2

uid() (*TxRealm.TxController property*), 20
Units (*class in Raven*), 16
units (*Raven.Member attribute*), 15
use_synch_mode (*Raven.KafkaConfig attribute*), 14

V

vertical_acc (*Raven.GPSEvent attribute*), 13
vertical_accuracy (*Raven.GPSEvent attribute*), 13
vertical_dop (*Raven.GPSEvent attribute*), 13

Y

YARDS (*Raven.Units attribute*), 17

Z

zookeeper_config (*Raven.KafkaConfig attribute*), 14