Machina

None

Branden Ehrenreich

Table of contents

1. Welcome to the Machina documentation

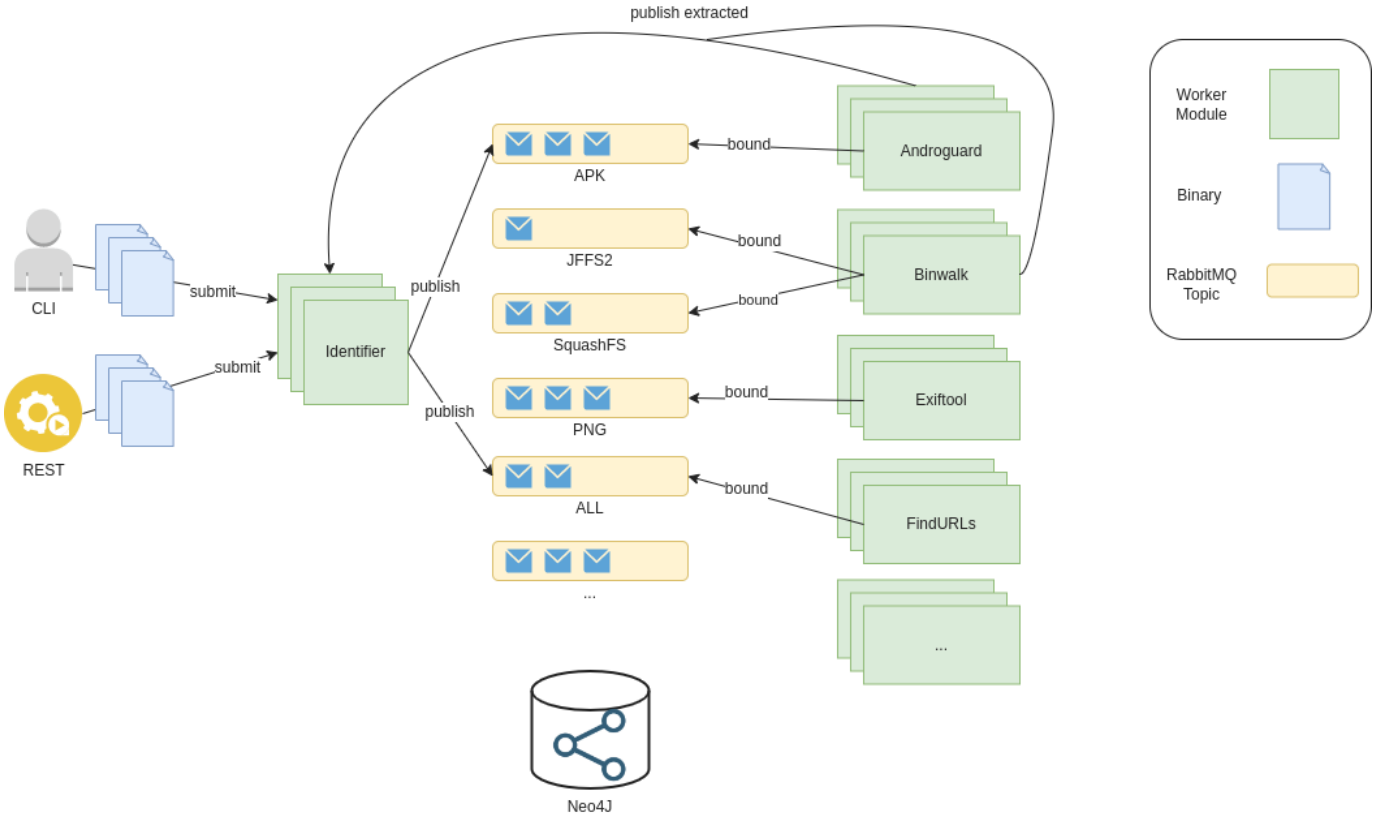
Machina is a scalable and modular analysis framework. Machina enables the rapid integration of both existing open source and novel analysis as worker modules.

Machina's key features

- **automated identification of input data** - Machina's Identifier worker module classifies input data. Optionally, a subject matter expert can provide a data type with their data if they wish
- **modular worker development** - A Worker base class provides boilerplate functionality, enabling developers to focus only on their analysis implementation
- **recursive analysis** - Worker modules may publish data back to the entrypoint of the system (Identifier). This allows for recursive extraction (firmware, archives, compressed data) of data, or re-typing of data after initial triage.
- **graph storage** - Machina uses a graph storage to store input and discovered analysis objects (as nodes), and their relations with other nodes as edges. For example: edges between nodes can represent origin ("extracted from") or similarity

Machina uses RabbitMQ for message passing into and within the system.

Each Machina Worker Module gets its own queue for direct publishing of data. More importantly, each Worker module also subscribes to topic(s) which correspond to abstract file types (e.g. an apk, pe, elf, etc..). This allows for a Worker Module to receive and handle any or all supported abstract file types.
When data is submitted to the pipeline (directly to the Identifier queue), the Identifier deteremines its type and publishes a message with the classified type as the routing key. This message is consumed by all Worker Modules that are configured to support that type. The Identifier prioritizes "detailed type" data over "mimetype" for classification.



This documentation is also available in PDF

2. Installation

2.1 System

2.1.1 Dependencies

- Tested on Ubuntu 20.04+
- git
- docker
- See Post-Installation steps to ensure Docker can be run without 'sudo'
- Ensure to install the 'docker-compose-plugin' that's described in the installation guide above

2.1.2 Production

Clone

```
git clone https://github.com/ehrenb/machina.git
```

Update/Pull

```
docker compose pull
```

> **Note**
>
> By default the 'latest' stable image versions will be used to pull. To change to a specific version, modify the 'latest' tag in the docker-compose.yml file cloned

2.1.3 Development

Clone

```
git clone --recurse-submodules https://github.com/ehrenb/machina.git &&\
    cd machina &&\
    git submodule foreach git checkout main &&\
    git submodule foreach git pull
```

Build

Build in the proper order to ensure that parent base images are built first:

```
docker compose build base-alpine base-ubuntu &&\
    docker compose build base-ghidra &&\
    docker compose build
```

2.1.4 Validate

Validate built or pulled images:

```
docker images | grep machina
```

Output:

```
behren/machina-docs                     latest          7b6f5d4fb34f   6 days ago     601MB
behren/machina-androguard               latest          de457208b6c2   7 days ago     766MB
behren/machina-binwalk                  latest          829efd9691a3   7 days ago     1.8GB
behren/machina-findurls                 latest          b09157ce035a   7 days ago     524MB
behren/machina-similarity               latest          287bc579e7c0   7 days ago     657MB
behren/machina-ssdeep                   latest          226791c470f0   7 days ago     657MB
behren/machina-identifier               latest          56d18aaeb1d4   7 days ago     446MB
behren/machina-exif                     latest          d737587a3d89   7 days ago     446MB
behren/machina-ghidra-project-creator   latest          9104987169d1   7 days ago     2.72GB
behren/machina-bz2                      latest          56262d5b591f   7 days ago     446MB
behren/machina-zip                      latest          65c8cd3611db   7 days ago     446MB
behren/machina-tar                      latest          9bb581d6779c   7 days ago     446MB
behren/machina-jar                      latest          3a05ae24f59c   7 days ago     446MB
behren/machina-gzip                     latest          2a0669038534   7 days ago     446MB
behren/machina-initializer              latest          9ae05328c766   7 days ago     446MB
behren/machina-base-ghidra              latest          8312415a7460   7 days ago     2.72GB
behren/machina-base-ubuntu              latest          679df4f10abf   7 days ago     648MB
behren/machina-base-alpine              latest          a97f8b394af8   7 days ago     446MB
```

2.2 CLI

2.2.1 Dependencies

It is recommended to set up and use a Python 3 virtual environment to isolate dependency installation. Install virtualenv.

```
virtualenv -p python3 machina-cli
source machina-cli/bin/activate
```

2.2.2 CLI

Clone

```
git clone https://github.com/ehrenb/machina.git
```

Install

Within your virtualenv:

```
cd machina/cli/
python3 setup.py install
```

2.2.3 Validate

```
python3 bin/machinacli.py --help
```

Output:

```
usage: machinacli.py [-h] [--verbose] {submit} ...

positional arguments:
{submit}
    submit       submit a file

options:
-h, --help     show this help message and exit
--verbose, -v
```

3. Administration

3.1 Start

Start the system in the background:

```
docker compose up -d
```

3.2 Scale

Scale up worker modules to support parallel analyses

```
docker compose up -d --scale identifier=2 androguardanalysis=5
```

3.3 Stop

```
docker compose down
```

3.4 System Stats

```
docker stats $(docker compose ps | awk 'NR>2 {print $1}')
```

3.5 Services

- Neo4j GUI
- http://127.0.0.1:7474
- (default) username: neo4j
- (default) password: tXOCq81bn7QfGTMJMrkQqP4J1
- RabbitMQ Management GUI
- http://127.0.0.1:15672
- (default) username: rabbitmq
- (default) password: rabbitmq

4. Usage

4.1 Submission

4.1.1 CLI

machinacli.py can be used to submit either a single file or a batch of individual files:

```
python3 bin/machinacli.py submit /usr/bin/ls /usr/bin/ps
```

Alternatively, if there are too many files to conveniently list as command line options, files can compressed or archived and submitted. Machina's decompression/unarchiving modules would then handle the unpacking and resubmission automatically to the system:

```
zip test.zip /usr/bin/ls /usr/bin/ps
python3 bin/machinacli.py submit test.zip
```

As analyses are completed, they are viewable in the Neo4J dashboard described within the 'Administration' section.

4.1.2 JSON

Optionally, you can publish your own message to the RabbitMQ Server, with the routing key set to 'Identifier' and a JSON body containing a base-64 encoded payload like the following:

```
{"data":  "<b64encoded_data>"}
```

Or, to assert a type (must be available in the 'available_types' configuration within machina/configs/types.json), provide it in the 'type' key-value pair. Providing this key forces the Identifier to skip type resolution and accept your own:

```
{
    "data": "<b64encoded_data>",
    "type": "apk"
}
```

4.2 Examples

4.2.1 Firmware (squashfs filesystem)

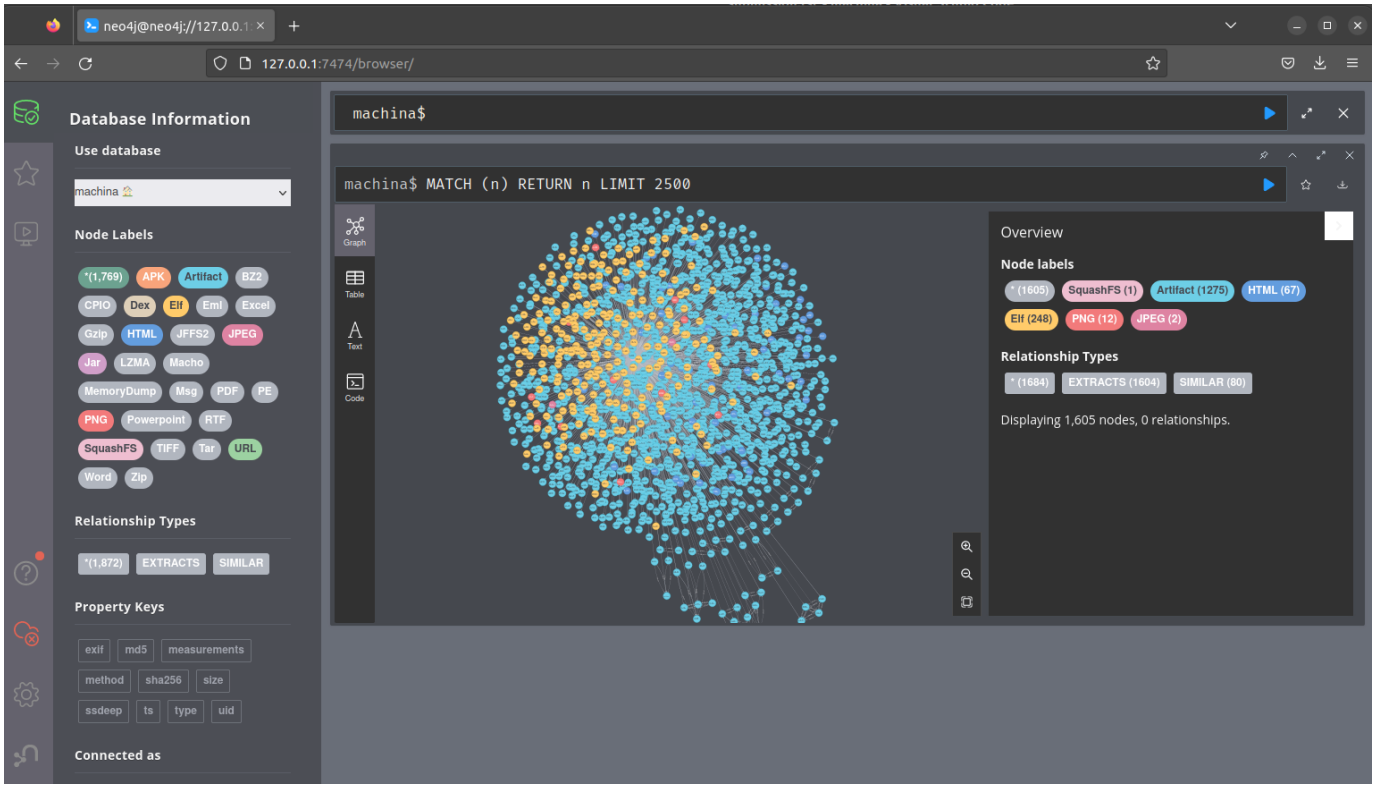
Obtain a sample of a squashfs file system:

```
wget https://github.com/ehrenb/machina-test/raw/main/data/squashfs/firmware.squashfs
```

Submit the file:

```
python3 bin/machinacli.py submit firmware.squashfs
```

After several minutes, all files in the submitted squashfs file are extracted and stored in Neo4J:



5. Workers

Name	Description	Input Types	Repository
androguard	Extract metadata from an Android Application (APK). Resubmit extracted files	apk	https://github.com/ehrenb/machina-androguard
binwalk	Unpack known firmware formats and resubmit extracted files	jffs2, squashfs, lzma, cpio	https://github.com/ehrenb/machina-binwalk
bz2	Decompress bzip2 data and resubmit extracted files	bz2	https://github.com/ehrenb/machina-bz2
exif	Extract exifdata from image files	png, jpeg, tiff	https://github.com/ehrenb/machina-exif
findurls	Extract URLs from any kind of data	all except url	https://github.com/ehrenb/machina-findurls
ghidra-project-creator	Perform default Ghidra analysis on executable files	elf, pe	https://github.com/ehrenb/machina-ghidra-project-creator
gzip	Decompress gzip data and resubmit extracted files	gz	https://github.com/ehrenb/machina-gzip
identifier	Classify input data and apply Machina type data	all	https://github.com/ehrenb/machina-identifier
jar	Detect APK from JAR format and resubmit data as APK	jar	https://github.com/ehrenb/machina-jar
similarity	Analyze previously determined SSDeep hash with a configured threshold to create a 'similarity' edge between nodes	None, invoked at interval	https://github.com/ehrenb/machina-similarity
ssdeep	Calculate SSDeep hash of file	all	https://github.com/ehrenb/machina-ssdeep
tar	Unarchive a tar file, resubmit all extracted files for analysis	tar	https://github.com/ehrenb/machina-tar
zip	Unzip a zip file, resubmit all extracted files for analysis	zip	https://github.com/ehrenb/machina-zip

6. Development

6.1 Worker development

This type of Machina worker triggers an analysis when new files are ingested into the system and tagged with a worker-compatible Machina type.

6.1.1 Dockerfile

Install any system dependencies required within your worker's Dockerfile. There are two base image options provided:

Base image options:

- behren/machina-base-alpine
- behren/machina-base-ubuntu

Dockerfile example:

```
FROM behren/machina-base-ubuntu:latest
...
RUN apt update && apt install libz-dev
...
```

6.1.2 requirements.txt

Put any Python 3 requirements required by your worker module into requirements.txt and ensure that you copy requirements.txt into the image and 'pip3 install -r requirements.txt' to install the dependencies.

6.1.3 youranalysismodule.py

This file contains the implementation of your worker module. subclass the Worker class, this will ensure your worker module has boilerplate connectivity to the database, RabbitMQ, and configurations. Choose any Machina types (see 'machina/configs/types.json') your worker module supports, or specify '*' for all. The callback function provides your analysis implementation with data that your module is configured to support. This callback function fires whenever the system identifies a compatible sample.

```
class YourAnalysisModule(Worker):
    types = ["zip"]

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self, data, properties):
        data = json.loads(data)
```

```
class YourAnalysisModule(Worker):
    types = ["*"]

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self, data, properties):
        data = json.loads(data)
```

```
class YourAnalysisModule(Worker):
    types_blacklist = ["zip"]

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self, data, properties):
        data = json.loads(data)
```

> **Note**
>
> If 'behren/machina-base-ghidra' was selected as your base, and Pythonic access to Ghidra is desired, see the Ghidra Worker Development documentation

6.1.4 YourAnalysisModule.json (schema)

This schema file provides validation constraints that are applied to data incoming to your worker module before it handles the data. The Schema name must match the class name that it belongs to (e.g. for the worker module 'AndroguardAnalysis'). This file belongs at the top level of your worker's directory, and must be copied within the Dockerfile.

Typically, since workers are handling data published by the Identifier, they inherit from the 'binary.json' schema. Additional input requirements can be specified in "properties"

```
{
    "allOf": [{ "$ref": "binary.json"}],
    "properties": {}
}
```

6.1.5 YourAnalysisModule.json (configuration)

This top-level configuration file belongs in machina/configs/workers/youranalysismodule.json. This file allows for reconfiguration without rebuilding of images or code. This file must be named after the worker class name that it corresponds to. Configuration data set in this file is made available through the worker module's 'self.config["worker"]' attribute. Log level is handled by the Worker base class to automatically adjust the subclass logging level if it is overridden in the configuration.

```
{
    "log_level": "debug",
    "hash_algorithms": ["md5", "sha256"]
}
```

```
class YourAnalysisModule(Worker):
    types = ["zip"]
...
def callback(self, data, properties):
    self.logger.info(self.config['worker']['hash_algorithms'])
```

6.1.6 Other notes

Republishing

Worker modules are not intended to create new nodes (e.g. files, binary data) in the database directly, only update node attributes or create edges (relationships). They should publish any extracted data of interest to the Identifier queue so that it re-enters the pipeline, e.g.:

```
class YourAnalysisModule(Worker):
    next_queues = ['Identifier']
    ...

    def callback(self, data, properties):
        ...
        self.publish_next(json.dumps(data)) # publish to queues configured in 'next_queues'
```

OR

```
class MyWorker(Worker):
    ...
    def callback(self, data, properties):
        ...
        self.publish(json.dumps(data), queues=['Identifier']) # publish to 'Identifier'
```

Retyping

File typing through mimetypes or file magic is not always granular enough to accurately determine a type. Sometimes it requires a bit of context, e.g. an Android APK is technically a zip file, and can only really be identified by peering into the zip and searching for common APK files. Only then can we retype the file properly as an APK. This burden should be on the Zip module to discover, not the Identifier.

The snippet below is an example of when the Zip analysis module detects that it is actually working on an APK. The Zip module resubmits most of the same data that consumed from the queue, except it manually specifies the 'type' to 'apk', which the Identifier will take at face value.

```
def callback(self, data, properties):
    ...
    body = {
        "data": data_encoded,
        "origin": {
            "ts": data['ts'],
            "md5": data['hashes']['md5'],
            "uid": data['uid'],
            "type": data['type']
        },
        'type': 'apk'
    }

    self.publish(json.dumps(data), queues=['Identifier']) # publish to 'Identifier'
```

6.2 Ghidra Worker development

6.2.1 Dockerfile

Install any additional system dependencies required within your worker's Dockerfile.

Base image options:

- behren/machina-base-ghidra

Dockerfile example:

```
FROM behren/machina-base-ghidra:latest
...
RUN apt update && apt install libz-dev
...
```

6.2.2 requirements.txt

Put any Python 3 requirements required by your worker module into requirements.txt and ensure that you copy requirements.txt into the image and 'pip3 install -r requirements.txt' to install the dependencies.

6.2.3 youranalysismodule.py

This file contains the implementation of your worker module. subclass the GhidraWorker class, this will ensure your worker module has boilerplate connectivity to the database, RabbitMQ, and configurations. Choose any Machina types (see 'machina/configs/types.json') your worker module supports, or specify '*' for all.

The callback function provides your analysis implementation with data that your module is configured to support. This callback function fires whenever the system identifies a compatible sample.

```
class YourAnalysisModule(GhidraWorker):
    types = ["elf"]

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self, data, properties):

        # resolve path
        target = self.get_binary_path(data['ts'], data['hashes']['md5'])
        self.logger.info(f"resolved path: {target}")

        self.analyze_headless(
            str(Path(target).parent),
            f'proj-{data["hashes"]["md5"]}-{self.cls_name}',
            import_files=[target]
        )
```

6.2.4 YourAnalysisModule.json (schema)

This schema file provides validation constraints that are applied to data incoming to your worker module before it handles the data. The Schema name must match the class name that it belongs to (e.g. for the worker module 'AndroguardAnalysis'). This file belongs at the top level of your worker's directory, and must be copied within the Dockerfile.

Typically, since workers are handling data published by the Identifier, they inherit from the 'binary.json' schema. Additional input requirements can be specified in "properties"

```
{
    "allOf": [{ "$ref": "binary.json"}],
    "properties": {}
}
```

6.2.5 YourAnalysisModule.json (configuration)

This top-level configuration file belongs in machina/configs/workers/youranalysismodule.json. This file allows for reconfiguration without rebuilding of images or code. This file must be named after the worker class name that it corresponds to. Configuration data set in this file is made available through the worker module's 'self.config["worker"]' attribute. Log level is handled by the Worker base class to automatically adjust the subclass logging level if it is overridden in the configuration.

```
{
    "log_level": "debug",
    "analysis_timeout_per_file": 600
}
```

```
class YourAnalysisModule(Worker):
        types = ["elf"]
    ...
    def callback(self, data, properties):
        self.logger.info(self.config['worker']['analysis_timeout_per_file'])

        # resolve path
        target = self.get_binary_path(data['ts'], data['hashes']['md5'])
        self.logger.info(f"resolved path: {target}")

        self.analyze_headless(
            str(Path(target).parent),
            f'proj-{data["hashes"]["md5"]}-{self.cls_name}',
            import_files=[target],
            analysis_timeout_per_file=self.config['worker']['analysis_timeout_per_file']
        )
```

6.3 Periodic Worker development

This type of Machina worker triggers an analysis periodically. This is useful for longer-running analysis work that isn't feasible to be run on-demand for every new ingested sample. One Periodic Worker example is SimilarityAnalysis. SimilartyAnalysis runs every hour, over all data in the graph, and updates/creates similarity relationships.

6.3.1 Dockerfile

Install any system dependencies required within your worker's Dockerfile. There are two base image options provided:

Base image options:

- behren/machina-base-alpine
- behren/machina-base-ubuntu

```
FROM behren/machina-base-ubuntu:latest
...
RUN apt update && apt install libz-dev
...
```

6.3.2 requirements.txt

Put any Python 3 requirements required by your worker module into requirements.txt and ensure that you copy requirements.txt into the image and 'pip3 install -r requirements.txt' to install the dependencies

6.3.3 youranalysismodule.py

This file contains the implementation of your worker module. Subclass the PeriodicWorker class, this will ensure your worker module has boilerplate connectivity to the database, and configurations. The callback function fires at the interval your analysis module is configured to.

```
class YourAnalysisModule(PeriodicWorker):

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self):
        self.logger.info("I'm firing!")
```

The PeriodicWorker provides some common triggers that can be used to further constrain execution at the configured interval.

For example, 'n_nodes_added_since' can be used to fire an analysis only if 1,000 new nodes of class type Elf have been added within the last 1 hour. These available triggers are documented within the machina core API.

```
from datetime import timedelta
from machina.core.models import Elf

class YourAnalysisModule(PeriodicWorker):

    def __init__(self, *args, **kwargs):
        super(YourAnalysisModule, self).__init__(*args, **kwargs)
        ...

    def callback(self):
        if self.n_nodes_added_since(
            1000,
            Elf,
            timedelta(seconds=60)
        ):
            self.logger.info("I'm firing under a special constraint!")
```

6.3.4 YourAnalysisModule.json (configuration)

This top-level configuration file belongs in machina/configs/workers/youranalysismodule.json. This file allows for reconfiguration without rebuilding of images or code. This file must be named after the worker class name that it corresponds to.

Configuration data set in this file is made available through the worker module's 'self.config["worker"]' attribute. Log level is handled by the PeriodicWorker base class to automatically adjust the subclass logging level if it is overridden in the configuration. The interval for invoking callback is also handled by the PeriodicWorker base class, and can be overridden in the configuration.

```
{
    "log_level": "debug",
    "interval": "minutely",
    "new_value": "I'm a new configuration!"
}
```

Note

Complex intervals can be set up. Under the hood, the PeriodicWorker uses rocketry for scheduling. Rocketry provides a verbose syntax for describing intervals, outlined here. This syntax can be used within the 'interval' configuration value to specify complex intervals.

```
class YourAnalysisModule(Worker):
    ...
    def callback(self):
        self.logger.info(self.config['worker']['interval'])
        self.logger.info(self.config['worker']['new_value'])
```

6.4 API

6.4.1 Machina Worker API

Analysis worker base class. Workers inheriting from this class receive data to analyze based on the chosen data type bindings

Source code in `machina/core/worker.py`

```
class Worker():
    """Analysis worker base class.  Workers inheriting from this class receive data to analyze based on the chosen data type bindings"""

    next_queues = [] # passes data to another queue in the sequence, this should allow for chaining
    types = [] # indicates what type data to bind to, this should be completed in the subclass
    types_blacklist = [] # indicates what type data NOT to bind to, this should be completed in the subclass. implies binding to all types not in this list.  cannot be combined with types

    def __init__(self):
        self.cls_name = self.__class__.__name__
        self.config = self._load_configs()
        self.schema = self._load_schema()

        # Logging
        level = logging.getLevelName(self.config['worker']['log_level'])
        logging.basicConfig(level=level, format='[*] %(message)s')
        self.logger = logging.getLogger(__name__)

        if self.types and self.types_blacklist:
            self.logger.error("both types and types_blacklist cannot be set at the same time")
            raise Exception

        # validate whitelist types
        # if *, then set types to all available types
        if self.types:
            self.logger.info(f"Validating types: {pformat(self.types)}")
            if '*' in self.types:
                self.types = self.config['types']['available_types']
            else:
                types_valid, t = self._types_valid()
                if not types_valid:
                    self.logger.error(f"{t} is not configured as a type in types.json")
                    raise Exception

        # validate black list types
        if self.types_blacklist:
            self.logger.info(f"Validating types blacklist: {pformat(self.types_blacklist)}")
            types_blacklist_valid, t = self._types_blacklist_valid()
            if not types_blacklist_valid:
                self.logger.error(f"{t} is not configured as a type in types.json, so cannot be blacklisted")
                raise Exception

            # if valid, set types to all except the ones in valid blacklist, and '*'
            self.types = [t for t in self.config['types']['available_types'] if t not in self.types_blacklist]
            # self.types.remove('*')

        # neo4j set connection
        _cfg = self.config['neo4j']
        config.DATABASE_URL = f"bolt://{_cfg['user']}:{_cfg['pass']}@{_cfg['host']}:{_cfg['port']}/{_cfg['db_name']}"

        # Initializer does no queue consumption, so
        # dont create a connection or queue for it
        if self.cls_name != 'Initializer':

            # RabbitMQ Connection info
            # note this is not thread-safe
            self.rmq_conn = self.get_rmq_conn()
            self.rmq_recv_channel = self.rmq_conn.channel()

            # reduce Pika logging level
            logging.getLogger('pika').setLevel(logging.ERROR)

            # The queue to bind to is the name of the class
            bind_queue = self.cls_name

            # Initialize an exchange
            self.rmq_recv_channel.exchange_declare('machina')

            # Initialize direct queue w/ subclass name
            self.logger.info(f"Binding to direct queue: {bind_queue}")
            self.rmq_recv_channel.queue_declare(self.cls_name, durable=True)

            # Ensure that the worker's queue is bound to the exchange
            self.rmq_recv_channel.queue_bind(exchange='machina',
                queue=self.cls_name)

            # multiple-bindings approach:
            # https://www.rabbitmq.com/tutorials/tutorial-four-python.html
            # Bind using type strings as routing_keys
            # Publish using only a routing_key should go to all queues
            # that are bound to that routing_key
            for t in self.types:
                self.logger.info(f'binding to type: {t}')
                self.rmq_recv_channel.queue_bind(exchange='machina',
                    queue=self.cls_name,
                    routing_key=t)

            self.rmq_recv_channel.basic_qos(prefetch_count=1)
            self.rmq_recv_channel.basic_consume(self.cls_name,
                on_message_callback=self._callback)

    ############################################################
    # Privates

    def _load_configs(self) -> dict:
        """load configuration files from expected path, return as dictionary

        :return: the configuration dictionary
        :rtype: dict
        """
```

```
        fdir = '/configs'

        paths_cfg_fp = Path(fdir, 'paths.json')
        with open(paths_cfg_fp, 'r') as f:
            paths_cfg = json.load(f)

        rabbitmq_cfg_fp = Path(fdir, 'rabbitmq.json')
        with open(rabbitmq_cfg_fp, 'r') as f:
            rabbitmq_cfg = json.load(f)

        neo4j_cfg_fp = Path(fdir, 'neo4j.json')
        with open(neo4j_cfg_fp, 'r') as f:
            neo4j_cfg = json.load(f)

        types_fp = Path(fdir, 'types.json')
        with open(types_fp, 'r') as f:
            types_cfg = json.load(f)

        # Base-worker configurations, will be overridden by worker-specifc
        # configurations if there is overlap
        base_worker_cfg_fp = Path(fdir, 'workers', 'Worker.json')
        with open(base_worker_cfg_fp, 'r') as f:
            worker_cfg = json.load(f)

        # Worker-specific configuration
        worker_cfg_fp = Path(fdir, 'workers', self.cls_name+'.json')
        with open(worker_cfg_fp, 'r') as f:
            worker_cfg.update(json.load(f))

        return dict(paths=paths_cfg,
                    rabbitmq=rabbitmq_cfg,
                    neo4j=neo4j_cfg,
                    types=types_cfg,
                    worker=worker_cfg)

    def _load_schema(self) -> dict:
        """automatically resolve schema name based on class name

        :return: the schema dictionary
        :rtype: dict
        """
        class_schema = Path(self.config['paths']['schemas'], self.cls_name+'.json')
        with open(class_schema, 'r') as f:
            schema_data = json.load(f)
        return schema_data

    def _callback(
        self,
        ch: pika.channel.Channel,
        method: pika.spec.Basic.Deliver,
        properties: pika.spec.BasicProperties,
        body: bytes):
        """do last-second validation before handling the callback

        :param ch: pika channel
        :type ch: pika.channel.Channel
        :param method: pike method
        :type method: pika.spec.Basic.Deliver
        :param properties: pika properties
        :type properties: pika.spec.BasicProperties
        :param body: message body
        :type body: bytes
        """
        self._validate_body(body)

        self.logger.info("entering callback")
        thread = threading.Thread(target=self.callback, args=(body, properties))
        thread.start()
        while thread.is_alive():
            self.rmq_recv_channel._connection.sleep(1.0)
        self.logger.info("exiting callback")

        self.rmq_recv_channel.basic_ack(delivery_tag=method.delivery_tag)

    def _validate_body(self, body: bytes):
        """apply subclass worker schema and validate

        :param body: message body
        :type body: bytes
        """
        self.logger.info("validating schema")
        data = json.loads(body)
        # fixed resolver to ensure base schema uri is resolved
        # e.g. https://stackoverflow.com/questions/53968770/how-to-set-up-local-file-references-in-python-jsonschema-document
        # resolver = jsonschema.RefResolver('file://{}'.format(os.path.join(self.config['paths']['schemas'], 'binary.json')), self.schema)
        resolver = jsonschema.RefResolver(f"file:{Path(self.config['paths']['schemas'], self.cls_name+'.json')}", self.schema)

        jsonschema.validate(
            instance=data,
            schema=self.schema,
            resolver=resolver)

    def _types_valid(self) -> tuple:
        """ensure that the type to bind to is configured in types.json

        :return: tuple where first element is True if all requested type bindings are valid, or False if not.  If invalid, set the second element to the first
discovered invalid type
        :rtype: tuple
        """
```

```
        for t in self.types:
            if t not in self.config['types']['available_types']:
                return False, t
        return True, None

    def _types_blacklist_valid(self) -> tuple:
        """ensure that the type to bind to is configured in types.json

        :return: tuple where first element is True if all requested type bindings are valid, or False if not.  If invalid, set the second element to the first
discovered invalid type
        :rtype: tuple
        """
        for t in self.types_blacklist:
            if t not in self.config['types']['available_types']:
                return False, t
        return True, None
    ############################################################

    ############################################################
    # RabbitMQ Helpers
    def get_rmq_conn(
        self,
        max_attempts:int=10,
        delay_seconds:int=1) -> pika.BlockingConnection:
        """get RabbitMQ connection instance

        :param max_attempts: max number of attempts to try to get the connection, defaults to 10
        :type max_attempts: int, optional
        :param delay_seconds: the delay between attempts to get the connection, defaults to 1
        :type delay_seconds: int, optional
        :return: the connection instance
        :rtype: pika.BlockingConnection
        """

        rabbitmq_user = self.config['rabbitmq']['rabbitmq_user']
        rabbitmq_password = self.config['rabbitmq']['rabbitmq_password']
        rabbitmq_host = self.config['rabbitmq']['rabbitmq_host']
        rabbitmq_port = self.config['rabbitmq']['rabbitmq_port']
        rabbitmq_heartbeat = self.config['rabbitmq']['rabbitmq_heartbeat']

        connection = None
        credentials = pika.PlainCredentials(rabbitmq_user, rabbitmq_password)
        parameters = pika.ConnectionParameters(rabbitmq_host,
            int(rabbitmq_port),
            '/',
            credentials,
            heartbeat=int(rabbitmq_heartbeat),
            socket_timeout=2)

        attempt = 0
        while attempt < max_attempts:
            try:
                connection = pika.BlockingConnection(parameters)
                break
            except pika.exceptions.AMQPConnectionError as e:
                self.logger.info(f"Attempt {attempt}/{max_attempts} to connect to RabbitMQ at {rabbitmq_host}:{rabbitmq_port}")
                self.logger.warn("Error connecting to RabbitMQ")

            attempt += 1
            time.sleep(delay_seconds)

        if not connection:
            self.logger.error('max attempts exceeded')
            sys.exit()

        return connection

    def start_consuming(self):
        """start consuming"""
        self.logger.info(f'{self.cls_name} worker started')
        try:
            self.rmq_recv_channel.start_consuming()
        except Exception as e:
            self.logger.error(e, exc_info=True)
            self.rmq_recv_channel.stop_consuming()
        self.connection.close()

    def callback(self, data: bytes, properties: pika.spec.BasicProperties):
        """callback for worker, implement in subclass

        :param data: incoming string payload
        :type data: bytes
        :param properties: message properties
        :type properties: pika.spec.BasicProperties
        :raises NotImplementedError:
        """
        raise NotImplementedError

    def publish_next(self, data: bytes):
        """publish to configured next_queues

        :param data: the data to publish
        :type data: bytes
        """
        if not self.next_queues:
            self.logger.warn('attempting to publish to next queue, but no next_queues defined in worker class')
        self.publish(data, queues=self.next_queues)

    def publish(self, data: bytes, queues: list):
        """publish directly to a list of arbitrary queues
```

```
        :param data: the data to publish
        :type data: bytes
        :param queues: the list of queue names (as strings) to publish data to
        :type queues: list
        """
        rmq_conn = self.get_rmq_conn()
        for q in queues:
            self.logger.info(f"publishing directly to {q}")
            rmq_channel = rmq_conn.channel()
            rmq_channel.basic_publish(
                exchange='machina',
                routing_key=q,
                body=data)
        rmq_conn.close()

    ############################################################

    ############################################################
    # Misc
    def get_binary_path(self, ts:str, md5:str) -> str:
        """get path to a binary on disk given a timestamp and its md5

        :param ts: the timestamp of the binary
        :type ts: str
        :param md5: the md5 of the binary
        :type md5: str
        :return: the path to the binary
        :rtype: str
        """
        binary_path = Path(self.config['paths']['binaries'], ts, md5)
        return str(binary_path)
```

`callback(data, properties)`

callback for worker, implement in subclass

Parameters:

Name	Type	Description	Default
`data`	`bytes`	incoming string payload	*required*
`properties`	`pika.spec.BasicProperties`	message properties	*required*

Raises:

Type	Description
`NotImplementedError`	

Source code in `machina/core/worker.py`

```
290   def callback(self, data: bytes, properties: pika.spec.BasicProperties):
291       """callback for worker, implement in subclass
292
293       :param data: incoming string payload
294       :type data: bytes
295       :param properties: message properties
296       :type properties: pika.spec.BasicProperties
297       :raises NotImplementedError:
298       """
299       raise NotImplementedError
```

`get_binary_path(ts, md5)`

get path to a binary on disk given a timestamp and its md5

Parameters:

Name	Type	Description	Default
`ts`	`str`	the timestamp of the binary	*required*
`md5`	`str`	the md5 of the binary	*required*

Returns:

Type	Description
`str`	the path to the binary

Source code in `machina/core/worker.py`

```
333   def get_binary_path(self, ts:str, md5:str) -> str:
334       """get path to a binary on disk given a timestamp and its md5
335
336       :param ts: the timestamp of the binary
337       :type ts: str
338       :param md5: the md5 of the binary
339       :type md5: str
340       :return: the path to the binary
341       :rtype: str
342       """
343       binary_path = Path(self.config['paths']['binaries'], ts, md5)
344       return str(binary_path)
```

`get_rmq_conn(max_attempts=10, delay_seconds=1)`

get RabbitMQ connection instance

Parameters:

Name	Type	Description	Default
`max_attempts`	`int`	max number of attempts to try to get the connection, defaults to 10	`10`
`delay_seconds`	`int`	the delay between attempts to get the connection, defaults to 1	`1`

Returns:

Type	Description
`pika.BlockingConnection`	the connection instance

Source code in `machina/core/worker.py`

```
233   def get_rmq_conn(
234       self,
235       max_attempts:int=10,
236       delay_seconds:int=1) -> pika.BlockingConnection:
237       """get RabbitMQ connection instance
238
239       :param max_attempts: max number of attempts to try to get the connection, defaults to 10
240       :type max_attempts: int, optional
241       :param delay_seconds: the delay between attempts to get the connection, defaults to 1
242       :type delay_seconds: int, optional
243       :return: the connection instance
244       :rtype: pika.BlockingConnection
245       """
246
247       rabbitmq_user = self.config['rabbitmq']['rabbitmq_user']
248       rabbitmq_password = self.config['rabbitmq']['rabbitmq_password']
249       rabbitmq_host = self.config['rabbitmq']['rabbitmq_host']
250       rabbitmq_port = self.config['rabbitmq']['rabbitmq_port']
251       rabbitmq_heartbeat = self.config['rabbitmq']['rabbitmq_heartbeat']
252
253       connection = None
254       credentials = pika.PlainCredentials(rabbitmq_user, rabbitmq_password)
255       parameters = pika.ConnectionParameters(rabbitmq_host,
256           int(rabbitmq_port),
257           '/',
258           credentials,
259           heartbeat=int(rabbitmq_heartbeat),
260           socket_timeout=2)
261
262       attempt = 0
263       while attempt < max_attempts:
264           try:
265               connection = pika.BlockingConnection(parameters)
266               break
267           except pika.exceptions.AMQPConnectionError as e:
268               self.logger.info(f"Attempt {attempt}/{max_attempts} to connect to RabbitMQ at {rabbitmq_host}:{rabbitmq_port}")
269               self.logger.warn("Error connecting to RabbitMQ")
270
271           attempt += 1
272           time.sleep(delay_seconds)
273
274       if not connection:
275           self.logger.error('max attempts exceeded')
276           sys.exit()
277
278       return connection
```

`publish(data, queues)`

publish directly to a list of arbitrary queues

Parameters:

Name	Type	Description	Default
`data`	`bytes`	the data to publish	*required*
`queues`	`list`	the list of queue names (as strings) to publish data to	*required*

Source code in `machina/core/worker.py`

```
311   def publish(self, data: bytes, queues: list):
312       """publish directly to a list of arbitrary queues
313
314       :param data: the data to publish
315       :type data: bytes
316       :param queues: the list of queue names (as strings) to publish data to
317       :type queues: list
318       """
319       rmq_conn = self.get_rmq_conn()
320       for q in queues:
321           self.logger.info(f"publishing directly to {q}")
322           rmq_channel = rmq_conn.channel()
323           rmq_channel.basic_publish(
324               exchange='machina',
325               routing_key=q,
326               body=data)
327       rmq_conn.close()
```

`publish_next(data)`

publish to configured next_queues

Parameters:

Name	Type	Description	Default
`data`	`bytes`	the data to publish	*required*

Source code in `machina/core/worker.py`

```
301   def publish_next(self, data: bytes):
302       """publish to configured next_queues
303
304       :param data: the data to publish
305       :type data: bytes
306       """
307       if not self.next_queues:
308           self.logger.warn('attempting to publish to next queue, but no next_queues defined in worker class')
309       self.publish(data, queues=self.next_queues)
```

`start_consuming()`

start consuming

Source code in `machina/core/worker.py`

```
280   def start_consuming(self):
281       """start consuming"""
282       self.logger.info(f'{self.cls_name} worker started')
283       try:
284           self.rmq_recv_channel.start_consuming()
285       except Exception as e:
286           self.logger.error(e, exc_info=True)
287           self.rmq_recv_channel.stop_consuming()
288       self.connection.close()
```

6.4.2 Machina Ghidra Worker API

Bases: `Worker`

Ghidra Analysis worker base class

Source code in `machina/core/ghidra_worker.py`

```
113
114 class GhidraWorker(Worker):
115     """Ghidra Analysis worker base class"""
116
117     def __init__(self):
118         super(GhidraWorker, self).__init__()
119
120         self.logger.info(f"GhidraWorker subclass name: {self.cls_name}")
121
122         # update worker config with GhidraWorker.json base default configurations,
123         # override with child class if specified.
124         self.config['worker'].update(self._load_ghidra_configs()['worker'])
125
126         # set GHIDRA_MAXMEM env var which is used in the patched analyzeHeadless script to control max memory
127         # os.environ['GHIDRA_MAXMEM'] = self.config['worker']['maxmem']
128
129         # update base configs
130         self.gihdra_analyzeheadless_path = Path(os.environ['GHIDRA_HOME'], 'support', 'analyzeHeadless')
131
132
133     def _load_ghidra_configs(self) -> dict:
134         """reload config again using GhidraWorker.json as default base
135
136         :return: the configuration dictionary
137         :rtype: dict
138         """
139
140         fdir = '/configs'
141
142         # Base-worker configurations, will be overridden by worker-specifc
143         # configurations if there is overlap
144         base_worker_cfg_fp = Path(fdir, 'workers', 'GhidraWorker.json')
145         with open(base_worker_cfg_fp, 'r') as f:
146             worker_cfg = json.load(f)
147
148         # Worker-specific configuration
149         worker_cfg_fp = Path(fdir, 'workers', self.cls_name+'.json')
150         with open(worker_cfg_fp, 'r') as f:
151             worker_cfg.update(json.load(f))
152
153         return dict(
154             worker=worker_cfg
155         )
156
157
158     ############################################################
    # Ghidra API wrappers
    def analyze_headless(
        self,
        project_location:str,
        project_name:str,
        import_files: List[str]=[],
        process:List[str]=[],
        pre_script:List[Tuple[str,List[str]]]=[],
        post_script:List[Tuple[str,List[str]]]=[],
        overwrite: bool=False,
        recursive: bool=False,
        read_only: bool=False,
        delete_project: bool=False,
        no_analysis: bool=False,
        analysis_timeout_per_file: int=300
        ):
        """run Ghidra's analyzeHeadless

        :param project_location: root directory of an existing project to import (if 'import' set or 'process' set), or where a new project will be created
(if 'import' set)
        :type project_location: str
        :param project_name: the name of an existing project to import (if 'import' set or 'process' set), or a new project (if 'import' set).  if the
project name includes a folder path, imports will be rooted under that folder
        :type project_name: str
        :param import_files: list of executables and/or directories containing executables to import into a project. cannot be set when 'process' is set
        :type import_files: List[str], optional
        :param process: perform processing (pre/post scripts and/or analysis) on files in the list.  if not set, all files in project_name will be analyzed.
cannot be set with 'import_files'
        :type process: List[str], optional
        :param pre_script: a list of tuples where the first element in a tuple is the pre script name (with extension), and the second element is a list of
strings containing arguments (if any) for the script. example: [ ('prescript1.java', ['-my-arg1','-my1-arg2']), ('prescript2.java', ['-my2-arg1','-my2-
arg2']) ] defaults to []
        :type pre_script: List[Tuple[str,List[str]]], optional
        :param post_script: a list of tuples where the first element in a tuple is the post script name (with extension), and the second element is a list
of strings containing arguments (if any) for the script. example: [ ('postcript1.java', ['-my-arg1','-my1-arg2']), ('postscript2.java', ['-my2-arg1','-my2-
arg2']) ] defaults to []
        :type post_script: List[Tuple[str,List[str]]], optional
        :param overwrite: overwrite any existing project files that conflict with the ones being imported.  applies only if 'import' is set, and is ignored
if 'read_only' is set. defaults to False
        :type overwrite: bool, optional
        :param recursive: enables decursive descent into directories and project sub-folders when a directory/folder has been specified in 'import' or
'process'. defaults to False
        :type recursive: bool, optional
        :param read_only: ensures imported files will not be written if 'import' is set.  also ensures any changes made when 'process' is set are discarded
        :param delete_project: delete the project once all scripts have completed.  only works on new projects created when 'import' is set. defaults to
False
        :type delete_project: bool, optional
        :param no_analysis: disables auto-analysis from running, defaults to False
        :type no_analysis: bool, optional
        :param analysis_timeout_per_file: timeout value in seconds per file analysis. defaults to 300
        :type analysis_timeout_per_file: int

        """

        if not (import_files or process) or (import_files and process):
```

```
            self.logger.error('import_files OR process must be set')
            return

        cmd = f'{self.gihdra_analyzeheadless_path} {project_location} {project_name} '

        if import_files:
            cmd += '-import ' + ' '.join(import_files) + ' '
        if process:
            cmd += '-process ' + ' '.join(process) + ' '

        for script_cfg in pre_script:
            script_name = script_cfg[0]
            cmd += f'-preScript {script_name} '
            if len(script_cfg) > 1:
                script_args = script_cfg[1]
                for script_arg in script_args:
                    cmd += f'{script_arg} '

        for script_cfg in post_script:
            script_name = script_cfg[0]
            cmd += f'-postScript {script_name} '
            if len(script_cfg) > 1:
                script_args = script_cfg[1]
                for script_arg in script_args:
                    cmd += f'{script_arg} '

        if overwrite:
            cmd += '-overwrite '
            # https://static.grumpycoder.net/pixel/support/analyzeHeadlessREADME.html#overwrite
            if not import_files:
                self.logger.warn('overwrite only applies when import_files specified')

        if recursive:
            cmd += '-recursive '

        if read_only:
            cmd += '-readOnly '

            # https://static.grumpycoder.net/pixel/support/analyzeHeadlessREADME.html#readOnly
            if import_files and overwrite:
                self.logger.warn('overwrite option ignored')

        if delete_project:
            cmd += '-deleteProject '

        if no_analysis:
            cmd += '-noanalysis '

        cmd += f'-analysisTimeoutPerFile {analysis_timeout_per_file} '

        cmd += f'-max-cpu {self.config["worker"]["max_cpu"]}'

        self.logger.debug(f"running command: {cmd}")
        p = subprocess.run(
            shlex.split(cmd),
            env={'GHIDRA_MAXMEM':self.config['worker']['maxmem']})
```

```
analyze_headless(project_location, project_name, import_files=[], process=[], pre_script=[], post_script=[], overwrite=False,
recursive=False, read_only=False, delete_project=False, no_analysis=False, analysis_timeout_per_file=300)
```

run Ghidra's analyzeHeadless

Parameters:

Name	Type	Description	Default
`project_location`	`str`	root directory of an existing project to import (if 'import' set or 'process' set), or where a new project will be created (if 'import' set)	*required*
`project_name`	`str`	the name of an existing project to import (if 'import' set or 'process' set), or a new project (if 'import' set). if the project name includes a folder path, imports will be rooted under that folder	*required*
`import_files`	`List[str]`	list of executables and/or directories containing executables to import into a project. cannot be set when 'process' is set	`[]`
`process`	`List[str]`	perform processing (pre/post scripts and/or analysis) on files in the list. if not set, all files in project_name will be analyzed. cannot be set with 'import_files'	`[]`
`pre_script`	`List[Tuple[str, List[str]]]`	a list of tuples where the first element in a tuple is the pre script name (with extension), and the second element is a list of strings containing arguments (if any) for the script. example: [('prescript1.java', ['-my-arg1','-my1-arg2']), ('prescript2.java', ['-my2-arg1','-my2-arg2'])] defaults to []	`[]`
`post_script`	`List[Tuple[str, List[str]]]`	a list of tuples where the first element in a tuple is the post script name (with extension), and the second element is a list of strings containing arguments (if any) for the script. example: [('postcript1.java', ['-my-arg1','-my1-arg2']), ('postscript2.java', ['-my2-arg1','-my2-arg2'])] defaults to []	`[]`
`overwrite`	`bool`	overwrite any existing project files that conflict with the ones being imported. applies only if 'import' is set, and is ignored if 'read_only' is set. defaults to False	`False`
`recursive`	`bool`	enables decursive descent into directories and project sub-folders when a directory/folder has been specified in 'import' or 'process'. defaults to False	`False`
`read_only`	`bool`	ensures imported files will not be written if 'import' is set. also ensures any changes made when 'process' is set are discarded	`False`
`delete_project`	`bool`	delete the project once all scripts have completed. only works on new projects created when 'import' is set. defaults to False	`False`
`no_analysis`	`bool`	disables auto-analysis from running, defaults to False	`False`
`analysis_timeout_per_file`	`int`	timeout value in seconds per file analysis. defaults to 300	`300`

Source code in `machina/core/ghidra_worker.py`

```python
def analyze_headless(
    self,
    project_location:str,
    project_name:str,
    import_files: List[str]=[],
    process:List[str]=[],
    pre_script:List[Tuple[str,List[str]]]=[],
    post_script:List[Tuple[str,List[str]]]=[],
    overwrite: bool=False,
    recursive: bool=False,
    read_only: bool=False,
    delete_project: bool=False,
    no_analysis: bool=False,
    analysis_timeout_per_file: int=300
    ):
    """run Ghidra's analyzeHeadless

    :param project_location: root directory of an existing project to import (if 'import' set or 'process' set), or where a new project will be created (if 'import' set)
    :type project_location: str
    :param project_name: the name of an existing project to import (if 'import' set or 'process' set), or a new project (if 'import' set).  if the project name includes a folder path, imports will be rooted under that folder
    :type project_name: str
    :param import_files: list of executables and/or directories containing executables to import into a project. cannot be set when 'process' is set
    :type import_files: List[str], optional
    :param process: perform processing (pre/post scripts and/or analysis) on files in the list.  if not set, all files in project_name will be analyzed. cannot be set with 'import_files'
    :type process: List[str], optional
    :param pre_script: a list of tuples where the first element in a tuple is the pre script name (with extension), and the second element is a list of strings containing arguments (if any) for the script. example: [ ('prescript1.java', ['-my-arg1','-my1-arg2']), ('prescript2.java', ['-my2-arg1','-my2-arg2']) ] defaults to []
    :type pre_script: List[Tuple[str,List[str]]], optional
    :param post_script: a list of tuples where the first element in a tuple is the post script name (with extension), and the second element is a list of strings containing arguments (if any) for the script. example: [ ('postcript1.java', ['-my-arg1','-my1-arg2']), ('postscript2.java', ['-my2-arg1','-my2-arg2']) ] defaults to []
    :type post_script: List[Tuple[str,List[str]]], optional
    :param overwrite: overwrite any existing project files that conflict with the ones being imported.  applies only if 'import' is set, and is ignored if 'read_only' is set. defaults to False
    :type overwrite: bool, optional
    :param recursive: enables decursive descent into directories and project sub-folders when a directory/folder has been specified in 'import' or 'process'. defaults to False
    :type recursive: bool, optional
    :param read_only: ensures imported files will not be written if 'import' is set.  also ensures any changes made when 'process' is set are discarded
    :param delete_project: delete the project once all scripts have completed.  only works on new projects created when 'import' is set. defaults to False
    :type delete_project: bool, optional
    :param no_analysis: disables auto-analysis from running, defaults to False
    :type no_analysis: bool, optional
    :param analysis_timeout_per_file: timeout value in seconds per file analysis. defaults to 300
    :type analysis_timeout_per_file: int

    """

    if not (import_files or process) or (import_files and process):
        self.logger.error('import_files OR process must be set')
        return

    cmd = f'{self.gihdra_analyzeheadless_path} {project_location} {project_name} '

    if import_files:
        cmd += '-import ' + ' '.join(import_files) + ' '
    if process:
        cmd += '-process ' + ' '.join(process) + ' '

    for script_cfg in pre_script:
        script_name = script_cfg[0]
        cmd += f'-preScript {script_name} '
        if len(script_cfg) > 1:
            script_args = script_cfg[1]
            for script_arg in script_args:
                cmd += f'{script_arg} '

    for script_cfg in post_script:
        script_name = script_cfg[0]
        cmd += f'-postScript {script_name} '
        if len(script_cfg) > 1:
            script_args = script_cfg[1]
            for script_arg in script_args:
                cmd += f'{script_arg} '

    if overwrite:
        cmd += '-overwrite '
        # https://static.grumpycoder.net/pixel/support/analyzeHeadlessREADME.html#overwrite
        if not import_files:
            self.logger.warn('overwrite only applies when import_files specified')

    if recursive:
        cmd += '-recursive '

    if read_only:
        cmd += '-readOnly '

        # https://static.grumpycoder.net/pixel/support/analyzeHeadlessREADME.html#readOnly
        if import_files and overwrite:
            self.logger.warn('overwrite option ignored')

    if delete_project:
        cmd += '-deleteProject '

    if no_analysis:
        cmd += '-noanalysis '
```

```
    cmd += f'-analysisTimeoutPerFile {analysis_timeout_per_file} '

    cmd += f'-max-cpu {self.config["worker"]["max_cpu"]}'

    self.logger.debug(f"running command: {cmd}")
    p = subprocess.run(
        shlex.split(cmd),
        env={'GHIDRA_MAXMEM':self.config['worker']['maxmem']})
```

6.4.3 Machina Periodic Worker API

Analysis worker base class. Workers inheriting from this class analzye based on a schedule

Source code in `machina/core/periodic_worker.py`

```python
class PeriodicWorker():
    """Analysis worker base class.  Workers inheriting from this class analzye based on a schedule"""

    def __init__(self):
        self.cls_name = self.__class__.__name__
        self.config = self._load_configs()
        self.app = Rocketry()

        # Logging
        level = logging.getLevelName(self.config['worker']['log_level'])
        logging.basicConfig(level=level, format='[*] %(message)s')
        self.logger = logging.getLogger(__name__)

        # reduce Rocketry logging level
        logging.getLogger('rocketry.scheduler').setLevel(logging.ERROR)

        # neo4j set connection
        _cfg = self.config['neo4j']
        config.DATABASE_URL = f"bolt://{_cfg['user']}:{_cfg['pass']}@{_cfg['host']}:{_cfg['port']}/{_cfg['db_name']}"

    ############################################################
    # Privates
    def _load_configs(self) -> dict:
        """load configuration files from expected path, return as dictionary

        :return: the configuration dictionary
        :rtype: dict
        """

        fdir = '/configs'

        paths_cfg_fp = Path(fdir, 'paths.json')
        with open(paths_cfg_fp, 'r') as f:
            paths_cfg = json.load(f)

        neo4j_cfg_fp = Path(fdir, 'neo4j.json')
        with open(neo4j_cfg_fp, 'r') as f:
            neo4j_cfg = json.load(f)

        types_fp = Path(fdir, 'types.json')
        with open(types_fp, 'r') as f:
            types_cfg = json.load(f)

        # Base-worker configurations, will be overridden by worker-specifc
        # configurations if there is overlap
        base_worker_cfg_fp = Path(fdir, 'workers', 'PeriodicWorker.json')
        with open(base_worker_cfg_fp, 'r') as f:
            worker_cfg = json.load(f)

        # Worker-specific configuration
        worker_cfg_fp = Path(fdir, 'workers', self.cls_name+'.json')
        with open(worker_cfg_fp, 'r') as f:
            worker_cfg.update(json.load(f))

        return dict(paths=paths_cfg,
                    neo4j=neo4j_cfg,
                    types=types_cfg,
                    worker=worker_cfg)
    ############################################################

    def start(self):
        """start running callback at interval"""
        self.logger.info(f"starting with interval: {self.config['worker']['interval']}")
        self.app.task(self.config['worker']['interval'], func=self.callback)
        self.app.run()

    def callback(self):
        """implement in subclass"""
        raise NotImplemented

    ############################################################
    # Triggers
    def n_nodes_added_since(
        n: int,
        node_cls: Type[Base],
        duration: timedelta):
        """return True if 'n' nodes of 'node_cls' type have been added within a duration
        of time, return False if either (or both) condition was not met.

        :param n: the threshold number of nodes to consider before returning True
        :type n: int
        :param node_cls: the neomodel OGM class to use for counting node instances
        :type node_cls: type[Base]
        :param duration: the datetime.timedelta object specifying the threshold duration of time
        :type duration: timedelta
        :return: True if both conditions were met, False of neither (or both) not met
        :rtype: bool
        """

        # get adjusted timestamp for provided
        # threshold duration
        now = datetime.now(timezone.utc)
        duration_ts = (now - duration)

        # filter nodes within the duration window
        nodes = node_cls.nodes.filter(ts__gte=duration_ts).order_by('ts')
        if len(nodes) >= n:
            return True
        return False
```

`callback()`

implement in subclass

Source code in `machina/core/periodic_worker.py`

```
78  def callback(self):
79      """implement in subclass"""
80      raise NotImplemented
```

`n_nodes_added_since(n, node_cls, duration)`

return True if 'n' nodes of 'node_cls' type have been added within a duration of time, return False if either (or both) condition was not met.

Parameters:

Name	Type	Description	Default
`n`	`int`	the threshold number of nodes to consider before returning True	*required*
`node_cls`	`Type[Base]`	the neomodel OGM class to use for counting node instances	*required*
`duration`	`timedelta`	the datetime.timedelta object specifying the threshold duration of time	*required*

Returns:

Type	Description
`bool`	True if both conditions were met, False of neither (or both) not met

Source code in `machina/core/periodic_worker.py`

```
84  def n_nodes_added_since(
85      n: int,
86      node_cls: Type[Base],
87      duration: timedelta):
88      """return True if 'n' nodes of 'node_cls' type have been added within a duration
89      of time, return False if either (or both) condition was not met.
90
91      :param n: the threshold number of nodes to consider before returning True
92      :type n: int
93      :param node_cls: the neomodel OGM class to use for counting node instances
94      :type node_cls: type[Base]
95      :param duration: the datetime.timedelta object specifying the threshold duration of time
96      :type duration: timedelta
97      :return: True if both conditions were met, False of neither (or both) not met
98      :rtype: bool
99      """
100
101     # get adjusted timestamp for provided
102     # threshold duration
103     now = datetime.now(timezone.utc)
104     duration_ts = (now - duration)
105
106     # filter nodes within the duration window
107     nodes = node_cls.nodes.filter(ts__gte=duration_ts).order_by('ts')
108     if len(nodes) >= n:
109         return True
110     return False
```

`start()`

start running callback at interval

Source code in `machina/core/periodic_worker.py`

```
72  def start(self):
73      """start running callback at interval"""
74      self.logger.info(f"starting with interval: {self.config['worker']['interval']}")
75      self.app.task(self.config['worker']['interval'], func=self.callback)
76      self.app.run()
```

6.4.4 Models API

Nodes

Bases: `StructuredNode`

Base node type

Source code in `machina/core/models/nodes/base.py`

```
class Base(StructuredNode):
    """Base node type"""

    __abstract_node__ = True

    # Common attributes
    uid = UniqueIdProperty()

    md5 = StringProperty(required=True)
    sha256 = StringProperty(required=True)
    size = IntegerProperty(required=True)
    ts = DateTimeProperty(required=True)
    type = StringProperty(required=True)

    ssdeep = StringProperty(default=None) # set later

    extracts = RelationshipTo('Base', 'EXTRACTS', model=Extracts)
    similar = Relationship('Base', 'SIMILAR', model=Similar)
    retyped = RelationshipTo('Base', 'RETYPED', model=Retyped)
```

Bases: `Base`

A generic artifact for unknown/untyped data

Source code in `machina/core/models/nodes/artifact.py`

```
class Artifact(Base):
    """A generic artifact for unknown/untyped data"""
    pass
```

Bases: `Base`

Source code in `machina/core/models/nodes/apk.py`

```
class APK(Base):

    # APK Attribute
    package = StringProperty()
    name = StringProperty()
    androidversion_code = StringProperty()
    androidversion_name = StringProperty()
    permissions = ArrayProperty(StringProperty())
    activities = ArrayProperty(StringProperty())
    providers = ArrayProperty(StringProperty())
    receivers = ArrayProperty(StringProperty())
    services = ArrayProperty(StringProperty())
    min_sdk_version = StringProperty()
    max_sdk_version = StringProperty()
    max_sdk_version = StringProperty()
    effective_target_sdk_version = StringProperty()
    libraries = ArrayProperty(StringProperty())
    main_activity = StringProperty()
    content_provider_uris = ArrayProperty(StringProperty())

    classes = ArrayProperty(JSONProperty())
```

Bases: `Base`

bz2 compressed file

Source code in `machina/core/models/nodes/bz2.py`

```
class BZ2(Base):
    """bz2 compressed file"""

    pass
```

Bases: `Base`

CPIO firmware

Source code in `machina/core/models/nodes/cpio.py`

```
class CPIO(Base):
    """CPIO firmware"""
    pass
```

Bases: `Base`

Android DEX file

Source code in `machina/core/models/nodes/dex.py`

```
class Dex(Base):
    """Android DEX file"""
    pass
```

Bases: `Base`

Linux ELF file

Source code in `machina/core/models/nodes/elf.py`

```
class Elf(Base):
    """Linux ELF file"""
    pass
```

Bases: `Base`

EML

Source code in `machina/core/models/nodes/eml.py`

```
class Eml(Base):
    """EML"""
    pass
```

Bases: `Base`

Excel file

Source code in `machina/core/models/nodes/excel.py`

```
class Excel(Base):
    """Excel file"""
    pass
```

Bases: `Base`

GZIP compressed file

Source code in `machina/core/models/nodes/gzip.py`

```
class Gzip(Base):
    """GZIP compressed file"""
    pass
```

Bases: `Base`

HTML file

Source code in `machina/core/models/nodes/html.py`

```
class HTML(Base):
    """HTML file"""
    pass
```

Bases: `Base`

JAR file

Source code in `machina/core/models/nodes/jar.py`

```
class Jar(Base):
    """JAR file"""
    pass
```

Bases: `Base`

JFFS2 firmware

Source code in `machina/core/models/nodes/jffs2.py`

```
class JFFS2(Base):
    """JFFS2 firmware"""
    pass
```

Bases: `Base`

JPEF Image file

Source code in `machina/core/models/nodes/jpeg.py`

```
class JPEG(Base):
    """JPEF Image file"""

    # PNG attributes
    exif = JSONProperty()
```

Bases: `Base`

LZMA compressed file

Source code in `machina/core/models/nodes/lzma.py`

```
class LZMA(Base):
    """LZMA compressed file"""
    pass
```

Bases: `Base`

MachO file

Source code in `machina/core/models/nodes/macho.py`

```
class Macho(Base):
    """MachO file"""
    pass
```

Bases: `Base`

memory dump file

Source code in `machina/core/models/nodes/memory_dump.py`

```
class MemoryDump(Base):
    """memory dump file"""
    pass
```

Bases: `Base`

MSG file

Source code in `machina/core/models/nodes/msg.py`

```
class Msg(Base):
    """MSG file"""
    pass
```

Bases: `Base`

PDF file

Source code in `machina/core/models/nodes/pdf.py`

```
class PDF(Base):
    """PDF file"""
    pass
```

Bases: `Base`

PE file

Source code in `machina/core/models/nodes/pe.py`

```
class PE(Base):
    """PE file"""
    pass
```

Bases: `Base`

Source code in `machina/core/models/nodes/png.py`

```
class PNG(Base):

    # PNG attributes
    exif = JSONProperty()
```

Bases: `Base`

Powerpoint file

Source code in `machina/core/models/nodes/powerpoint.py`

```
class Powerpoint(Base):
    """Powerpoint file"""
    pass
```

Bases: `Base`

RTF file

Source code in `machina/core/models/nodes/rtf.py`

```
class RTF(Base):
    """RTF file"""
    pass
```

Bases: `Base`

SquashFS firmware

Source code in `machina/core/models/nodes/squashfs.py`

```
class SquashFS(Base):
    """SquashFS firmware"""
    pass
```

Bases: `Base`

TAR archive file

Source code in `machina/core/models/nodes/tar.py`

```
class Tar(Base):
    """TAR archive file"""
    pass
```

Bases: `Base`

extracted URL

Source code in `machina/core/models/nodes/url.py`

```
class URL(Base):
    """extracted URL"""

    # URL Attribute
    url = StringProperty()
```

Bases: `Base`

TIFf image file

Source code in `machina/core/models/nodes/tiff.py`

```
class TIFF(Base):
    """TIFf image file"""

    # attributes
    exif = JSONProperty()
```

Bases: `Base`

Word file

Source code in `machina/core/models/nodes/word.py`

```
class Word(Base):
    """Word file"""
    pass
```

Bases: `Base`

Zip compressed file

Source code in `machina/core/models/nodes/zip.py`

```
class Zip(Base):
    """Zip compressed file"""
    pass
```

Relationships

Bases: `StructuredRel`

Base relationship

Source code in `machina/core/models/relationships/base.py`

```
class BaseRelationship(StructuredRel):
    """Base relationship"""

    ts = DateTimeProperty(default=lambda: datetime.now())
```

Bases: `BaseRelationship`

Establish a node (some binary data) as being extracted from another node

Source code in `machina/core/models/relationships/extracts.py`

```
class Extracts(BaseRelationship):
    """Establish a node (some binary data) as being
        extracted from another node"""
    label = 'extracts'

    # E.g. 'dynamic', 'static'
    method = StringProperty()
```

Bases: `BaseRelationship`

Establish a node (some binary data) as being extracted from another node

Source code in `machina/core/models/relationships/retyped.py`

```
class Retyped(BaseRelationship):
    """Establish a node (some binary data) as being
        extracted from another node"""
    pass
```

Utils

`db_ts_to_fs_fmt(ts)`

convert database timestamp to file-system formatted timestamp string

Source code in `machina/core/models/utils.py`

```
def db_ts_to_fs_fmt(ts:datetime) -> str:
    """convert database timestamp to file-system formatted timestamp string"""
    return ts.strftime("%Y%m%d%H%M%S%f")
```

`resolve_db_node_cls(resolved_type)`

resolve a OGM subclass given a resolved machina type (e.g. in types.json) if not resolved, we expect unresolved to be stored as a generic Artifact, so return that cls

Returns:

Type	Description
`str`	the type string to resolve to a class

Source code in `machina/core/models/utils.py`

```
def resolve_db_node_cls(resolved_type: str) -> Type[Base]:
    """resolve a OGM subclass given a resolved machina type (e.g. in types.json)
    if not resolved, we expect unresolved to be stored as a generic Artifact, so return that cls

    :return: the type string to resolve to a class
    :rtype: str
    """
    all_models = Base.__subclasses__()
    for c in all_models:
        # if c.element_type.lower() == resolved_type.lower():
        if c.__name__.lower() == resolved_type.lower():
            return c
    return Artifact
```