Some refactoring stuff

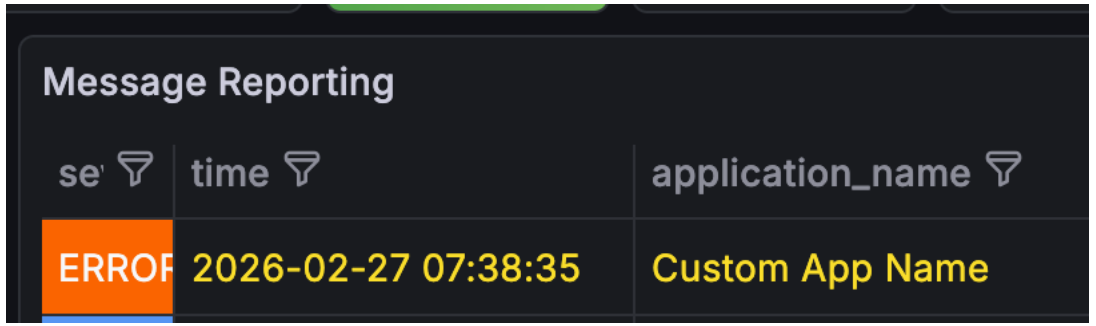
Presenter Name

Meeting Name

Date

Context

- Major features we want to add
  - Ability to configure ERS handlers based on OKS configuration
  - Ability to modify the ERS topic name to whatever the drunc session is
    - `setup_daq_ers_logger(ers_logger, "session_temp")`
  - Ability to customise what the ‘app’ name should be



Problems

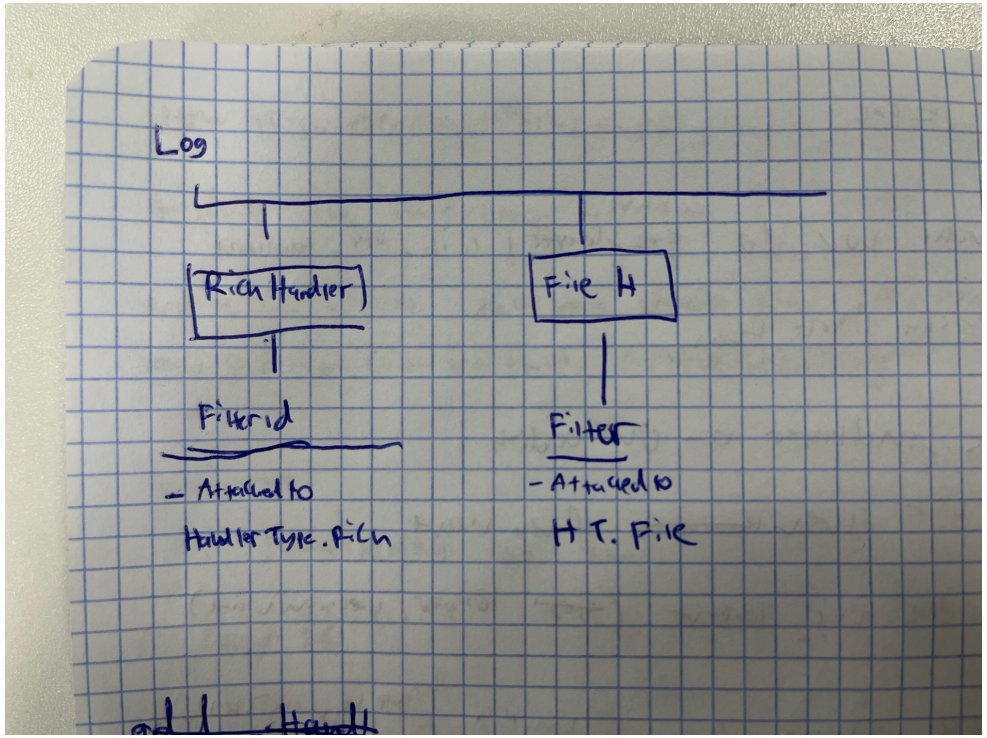
- By default, loggers will send msg to *all* handlers that exist
- We have solved that problem with `extra={HandlerType.type}`
- However, doing setup_ers_handlers will add all the ERS handlers to the logger, which means we have to add this ‘extra’ keyword to every single log message
  - Pretty cringe

- We need a way to have messages resolve correctly to a ‘default’ case
- `Log.info(“msg”) # this should resolve to`
- `Log.info(“msg”, extra=default_handlers)`

- And we want to be able to configure the default handlers to be whatever was initialised in get_daq_logger(…)

Solution

- Fallback handlers
- Logic:
  - Logger receives a record
  - Logger sends record to all Handlers
  - All handlers have Filters, which:



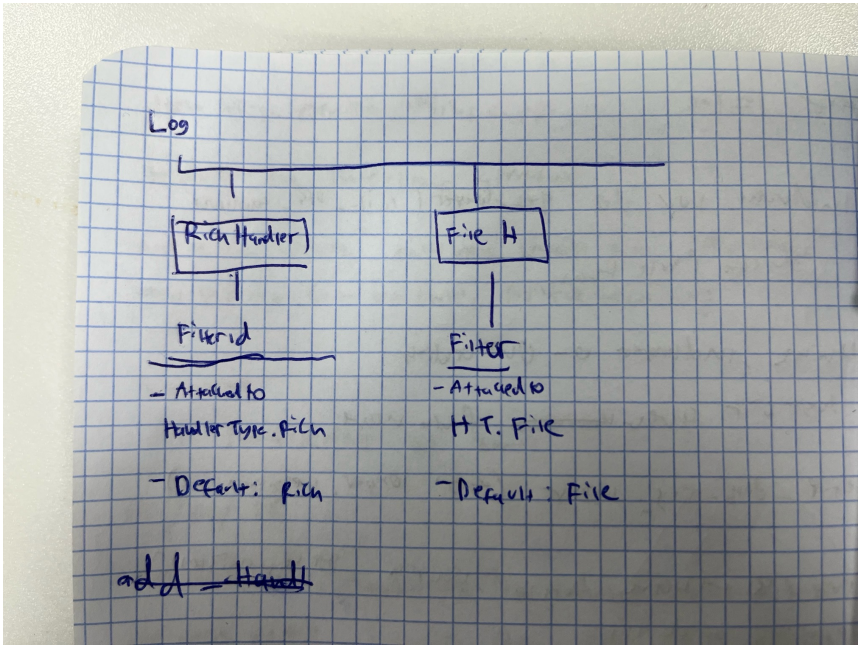
    - Check if an extra={"handlers" : ..} is supplied. If so, use that as handlers_to_check_against
    - If it doesn't, use the fallback handler it was inititialised with. Eg handlers_to_check_against=fallback_handlers

    - Transmission logic
      - If HandlerType is in handlers_to_check_against, transmit

- A bit convoluted, but I think it's the simplest way to do it

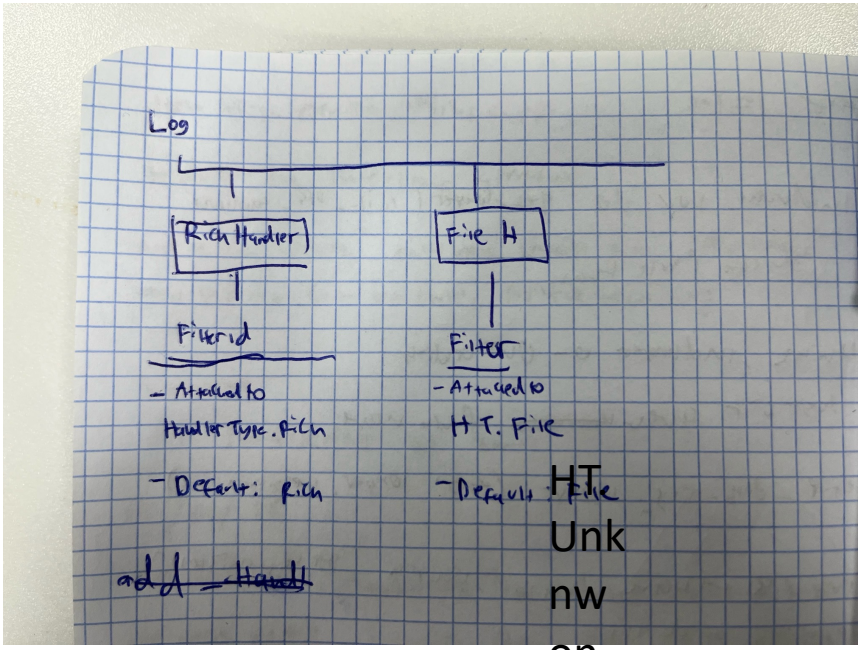
Solution

- Example:
- Log.msg("msg", extra={"handlers" : HT.Rich})
  - In the rich handler
    - Extras exist, so handlers_to_check_against = HT.Rich
    - HT.Rich is in handlers_to_check_against -> transmit
  - In file handler
    - Extras exist, so handlers_to_check_against = HT.Rich
    - HT.File is *not* in handlers_to_check_against -> transmit



Solution

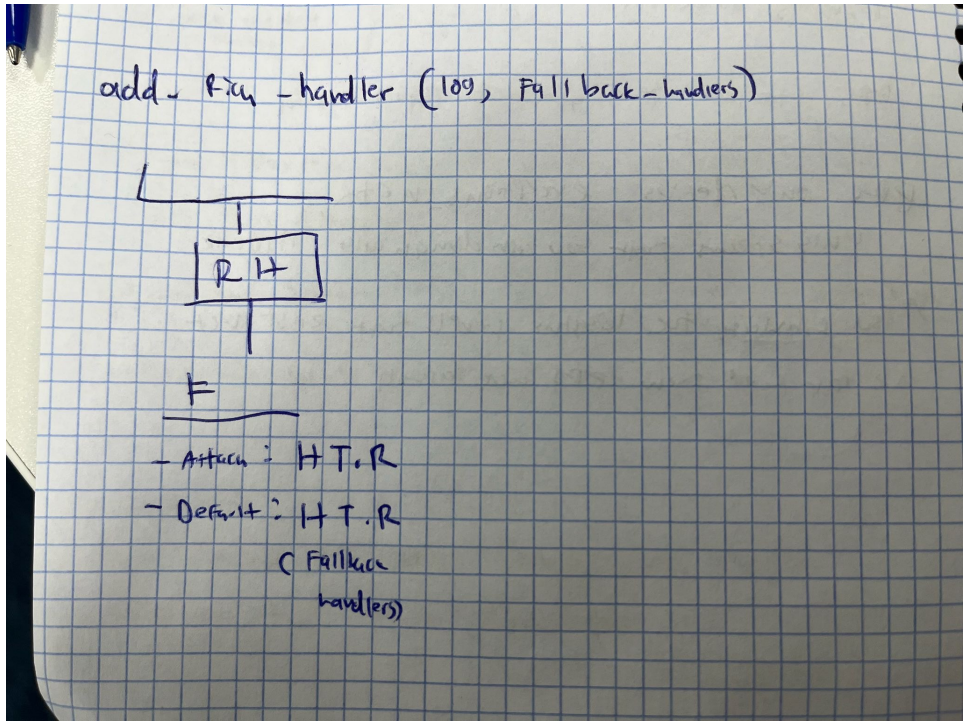
- Example:
- Log.msg(“msg”)
  - In the rich handler
    - Extras doenstexist, so handlers_to_check_against = HT.Rich
    - HT.Rich is in handlers_to_check_against -> transmit
  - In file handler
    - Extras doesn’t exist, so handlers_to_check_against = HT.Unknown
    - HT.File is in handlers_to_check_against = HT.Unknown -> doesbnt transmit
  - This is the default behaviour we want!
  - You can see if the default in File is HT.Unknown, then *by default* it will only send out as Rich



How to add a handler

- As simple as
  - Add_rich_handler(log, fallback_handlers=None)
    - If None, then it will auto add the fallback_handlers as HT.Rich
    - And same for all the other handlers

Add_rich_handler(my_log) -> defaults to fallback = HT.Rich



Refactoring something complicated

- Instead of doing this hackily, the code has been refactored to make things much easier to understand

Filtering *only* checks the 'If HandlerType is in handlers_to_check_against, transmit'

```
You, 16 hours ago | 1 author (You)
class HandleIDFilter(BaseHandlerFilter):
    """Filter class that accepts a list of 'allowed' handlers and will only fire
    if the current handler (defined by the handler_id) is within the set of
    allowed handlers.
    """
    def __init__(
        self,
        handler_id: HandlerType | list[HandlerType],
        fallback_handlers: set[HandlerType] | None = None,
        allowed_handlers_strategy: AllowedHandlersStrategy | None = None,
    ) -> None:
        """Initialises HandleIDFilter with the handler_id, to identify what
        kind of handler this filter is.
        """
        super().__init__(
            fallback_handlers = fallback_handlers,
            allowed_handlers_strategy=allowed_handlers_strategy
        )

        # Normalise handler_id to be a set
        if isinstance(handler_id, list):
            self.handler_ids = set(handler_id)
        else:
            self.handler_ids = {handler_id}

    def filter(self, record: logging.LogRecord) -> bool:
        """Identifies when a log message should be transmitted or not."""
        if not (allowed:= self.get_allowed(record)):
            return False
        return bool(self.handler_ids & allowed)
```

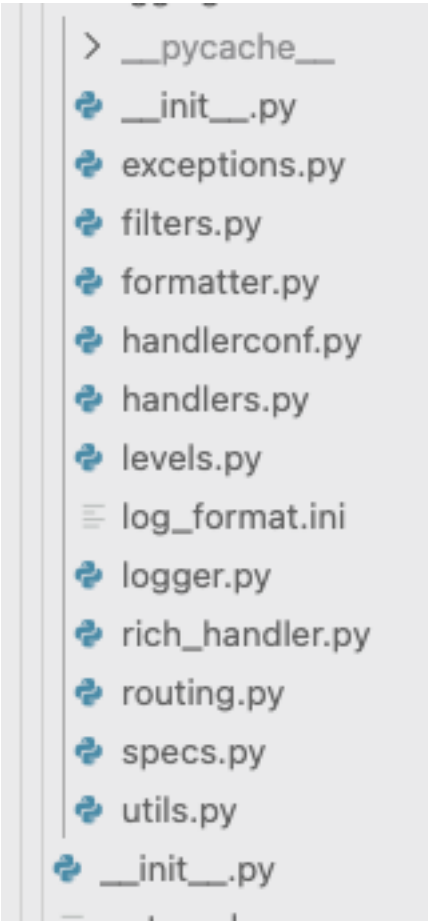
The Strategies are in routing.py, that deal with the 'is it ERS', 'does extra exist?' and gives us the handlers_to_check_against

```
You, yesterday | 1 author (You)
class StreamAwareAllowedHandlersStrategy(AllowedHandlersStrategy):
    """A dispatcher, chooses specific strategies based on record.stream"""

    def __init__(
        self,
        default_strategy : AllowedHandlersStrategy | None = None,
        ers_strategy: AllowedHandlersStrategy | None = None,
    ) -> None:
        self.default_strategy =  default_strategy or DefaultAllowedHandlerStrategy()
        self.ers_strategy = ers_strategy or ERSAllowedHandlersStrategy()

    def resolve(
        self,
        record: logging.LogRecord,
        fallback_handlers : set[Any],
        ) -> set[Any] | None:

        if getattr(record, "stream", None) == StreamType.ERS:
            return self.ers_strategy.resolve(record, fallback_handlers)
        return self.default_strategy.resolve(record, fallback_handlers)
```



Refactoring something complicated

- Instead of doing this hackily, the code has been refactored to make things much easier to understand

Filtering *only* checks the 'If HandlerType is in handlers_to_check_against, transmit'

```
You, 16 hours ago | 1 author (You)
class HandleIDFilter(BaseHandlerFilter):
    """Filter class that accepts a list of 'allowed' handlers and will only fire
    if the current handler (defined by the handler_id) is within the set of
    allowed handlers.
    """
    def __init__(
        self,
        handler_id: HandlerType | list[HandlerType],
        fallback_handlers: set[HandlerType] | None = None,
        allowed_handlers_strategy: AllowedHandlersStrategy | None = None,
    ) -> None:
        """Initialises HandleIDFilter with the handler_id, to identify what
        kind of handler this filter is.
        """
        super().__init__(
            fallback_handlers = fallback_handlers,
            allowed_handlers_strategy=allowed_handlers_strategy
        )

        # Normalise handler_id to be a set
        if isinstance(handler_id, list):
            self.handler_ids = set(handler_id)
        else:
            self.handler_ids = {handler_id}

    def filter(self, record: logging.LogRecord) -> bool:
        """Identifies when a log message should be transmitted or not."""
        if not (allowed:= self.get_allowed(record)):
            return False
        return bool(self.handler_ids & allowed)
```

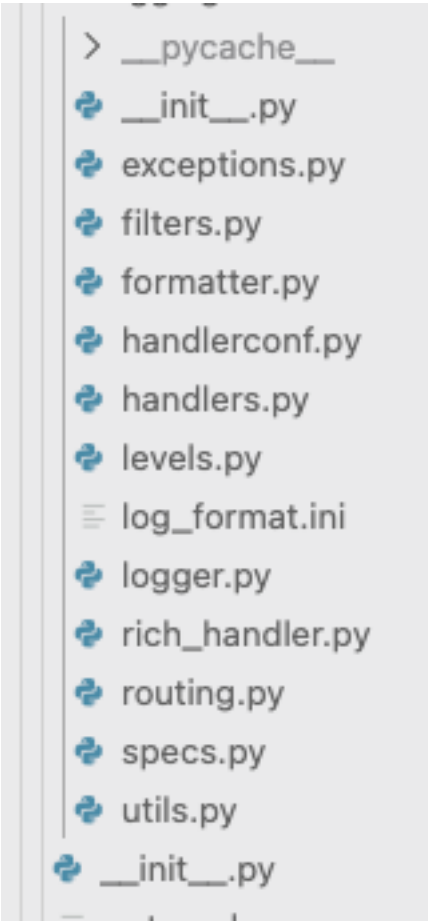
The Strategies are in routing.py, that deal with the 'is it ERS', 'does extra exist?' and gives us the handlers_to_check_against

```
You, yesterday | 1 author (You)
class StreamAwareAllowedHandlersStrategy(AllowedHandlersStrategy):
    """A dispatcher, chooses specific strategies based on record.stream"""

    def __init__(
        self,
        default_strategy : AllowedHandlersStrategy | None = None,
        ers_strategy: AllowedHandlersStrategy | None = None,
    ) -> None:
        self.default_strategy =  default_strategy or DefaultAllowedHandlerStrategy()
        self.ers_strategy = ers_strategy or ERSAllowedHandlersStrategy()

    def resolve(
        self,
        record: logging.LogRecord,
        fallback_handlers : set[Any],
        ) -> set[Any] | None:

        if getattr(record, "stream", None) == StreamType.ERS:
            return self.ers_strategy.resolve(record, fallback_handlers)
        return self.default_strategy.resolve(record, fallback_handlers)
```



Simplifying adding Handlers

Modular way of adding new handlers by defining how to build it + defining what it is (spec)

```python
def _build_rich_handler(extras: Mapping[str, Any]) -> logging.Handler:
    """Building the rich handler with any extras."""
    width = cast(int,  extras.get("width", get_width()))
    return FormattedRichHandler(width=width)

RICH_HANDLER_SPEC = HandlerSpec(
    representative_type = HandlerType.Rich, # What is it as a HandlerType
    handler_type = FormattedRichHandler, # What _is_ it? For checking to see if handler exists in the logge already
    factory=_build_rich_handler, # How to add a rich handler
    filter_handler_ids = (HandlerType.Rich,), # For HandleIDFilter default
)

def _build_file_handler(extras: Mapping[str, Any]) -> logging.Handler:
    path = cast(str | None, extras.get("path"))
    if not path:
        err_msg = "path is requiired for file handler"
        raise ValueError(err_msg)
    handler = logging.FileHandler(filename=path)
    handler.setFormatter(LoggingFormatter())

    return handler

FILE_HANDLER_SPEC = HandlerSpec(
    representative_type=HandlerType.File,
    handler_type = logging.FileHandler,
    factory = _build_file_handler,
    filter_handler_ids=(HandlerType.File,)
)
```

Some handlers require extra parameters.
Encapsulated in Extra generally (**kwargs)

Installing it in the registry , and done!

```python
HANDLER_SPEC_REGISTRY : dict[HandlerType, tuple[HandlerSpec, ...]] = {
    HandlerType.Rich: (RICH_HANDLER_SPEC,),
    HandlerType.Lstdout: (STDOUT_HANDLER_SPEC,),
    HandlerType.Lstderr: (STDERR_HANDLER_SPEC,),
    HandlerType.Stream: (STDOUT_HANDLER_SPEC,STDERR_HANDLER_SPEC),
    HandlerType.File: (FILE_HANDLER_SPEC,),
    HandlerType.Protobufstream: (ERSKAFKA_HANDLER_SPEC,)
}

def get_handler_specs(handler_type: HandlerType):
    """Get the specs defined in the registry"""
    return HANDLER_SPEC_REGISTRY.get(handler_type, tuple())

def add_handler(
    log: logging.Logger,
    handler_type: HandlerType,
    use_parent_handlers:bool,
    fallback_handler: set[HandlerType] | None,
    extras: Mapping[str, Any] | None = None,
):
```

So what changed as a user?

- Very little except for some nice new functionality, made easy thanks to the refactor!

- See logging demonstrator

- Fallback_handlers
- SetupERS

- Propagated to drunc, but untested yet

Open questions

- We now have add_handlers() as a general thing. Do we want to retire the individual add_rich_handlers()? Theyre just wrappers now

- We should consider looking at the call method of get_daq_loggers, its getting too long
- Can I try and play with the function call so its gona be more like eg. Matplotlib?
- Eg def get_daq_logger(
  - logger_name: str,
  - log_level: int | str = logging.NOTSET,
  - use_parent_handlers: bool = True,
  - **kwargs
  -)
- But anyways, everything works though!

How to test

- Protobufhandleconf in daqpytools
- That had default values as well
- Use SSH-standalone code, modify the address that's default (in drunk)
- If its still incorrect,
- Signature checking with the code
- Mypy
- Open issue 5.7.0