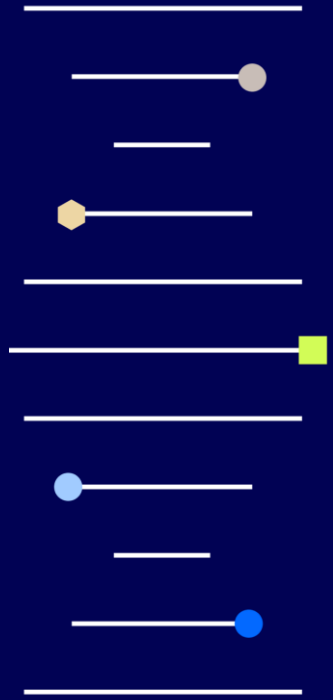
Advancing data standardization via Allotropy

The Allotropy Open-Source Library for Instrument Data Conversion into ASM



Bioanalytical Use Case Enabled by Allotropy Convertor Catalog

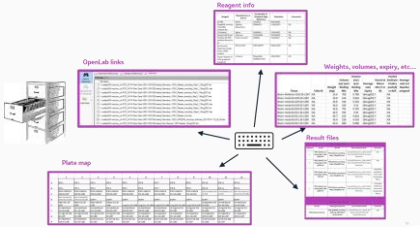
Vaccine Bioanalysis: Reimagining E2E workflows to accelerate innovation

FROM

Sample Management

Assay Execution

Data Reporting



TO

Automated sample management with an integrated tracking system

Integrated robotics for routine assay execution

Integrated, end-to-end data management

Vaccines Bioanalytical Solution

- Vaxbaam is being built using the Benchling Connect platform to enable end-to-end assay data management, data connectivity across upstream and downstream systems. Intent to reuse data to drive future insights.
- Leverage the Allotropy library of ASM converters (~50 in total) in the solution by the end of Q2 2025. Approximately 40% of these are available now or very soon (see below).

✓*Biotek Cytation 7*
✓*TapeStation*
✓*Biotek Cytation 5*
✓*Quant Studio 7 Pro*
✓*Quant Studio 5*
✓*Bio-Plex 200 Luminex*
✓*Nucleocounter NC200*
✓*Immunospot Analyzer S6U Universal*
✓*FLEXMAP 3D READER*
✓*Iris Reader*

✓*Sector S 600*
✓*Ensight Multimode Plate Reader*
✓*Lunatic*
✓*QuantStudio 7 Flex*
✓*Vi-CELL XR Automated Cell Viability Analyzer*
✓*Qiagen*
✓*Qiacuity 8, PCR system*
✓*Absolute Q*
✓*Genesys 150uv scanning*
✓*Cellaca MXFL2*

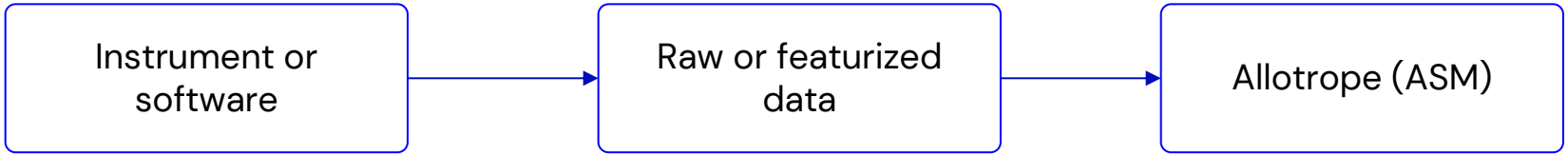
Today, you will learn how to leverage & contribute to Allotropy!

Allotropy Overview

Supported ASM Schemas

Contributing a Connector

Allotropy is a python library to convert instrument data to ASM



Allotropy coverage is growing, to help drive ASM standardization

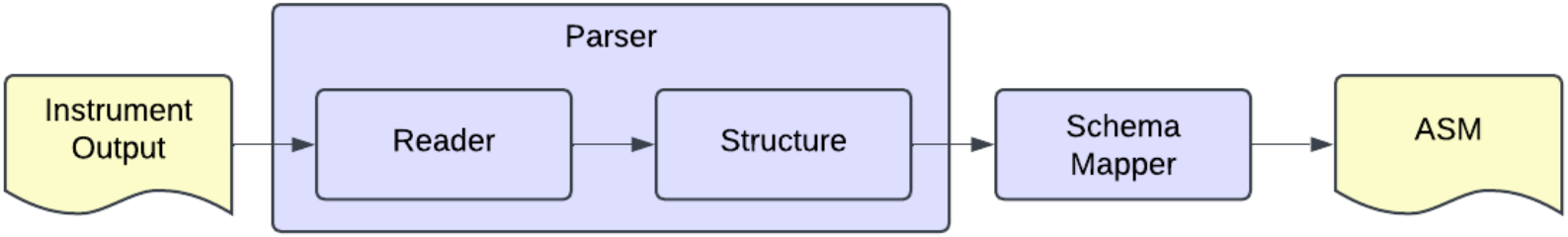
8 ASM schemas

34 parsers

Over 100 supported instruments

ASM schema	Parsers
Cell Counting	6
Electrophoresis	1
Multianalyte Profiling	2
Plate Reader	10
Solution Analyzer	3
Spectrophotometry	7
dPCR	2
qPRC	3

Contributing a connector includes parser & schema mapping



Reader

- Reads in raw file
- Stores in structured manner - typically pandas dataframes
- Parsers for common patterns in util library

Structure

- Organizes raw data into a logical structure, e.g. metadata, plate data, calculated data.
- Populates simple dataclasses defined by schema mapper

Schema Mapper

- One mapper per ASM schema
- Accepts python dataclasses to populate ASM model
- Enables simple upgrade to new schema version

Allotropy has tools to make building connectors to ASM easy

Boilerplate code automation
start a new parser in seconds

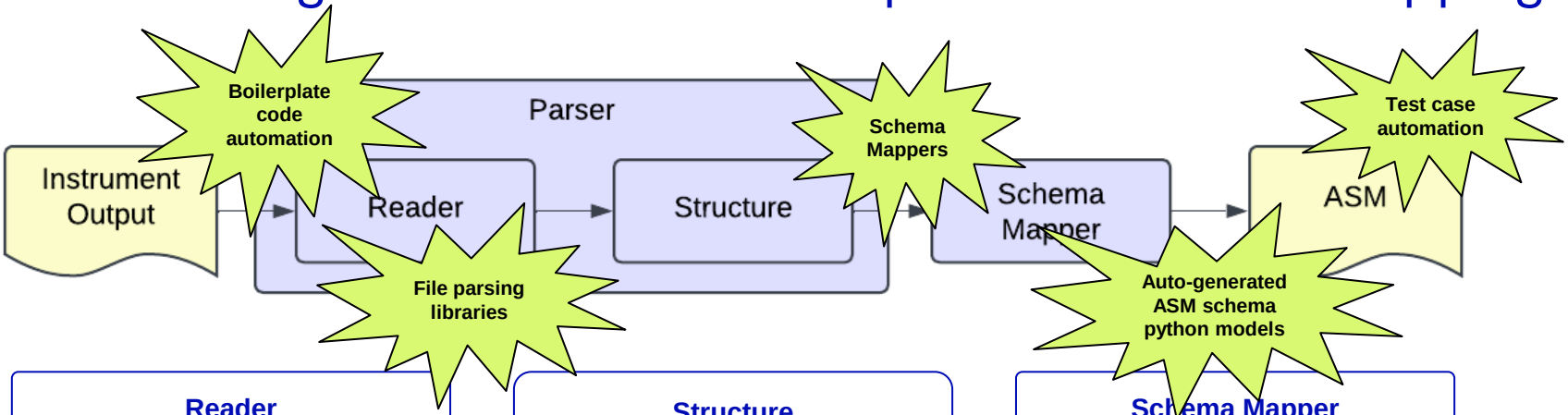
File parsing libraries
standard methods for common formats

Auto-generated ASM schema python models
makes populating ASM easy

Schema Mappers
makes populating ASM version agnostic

Test case automation
add a test case with no code

Contributing a connector includes parser & schema mapping



Reader

- Use a library

Structure

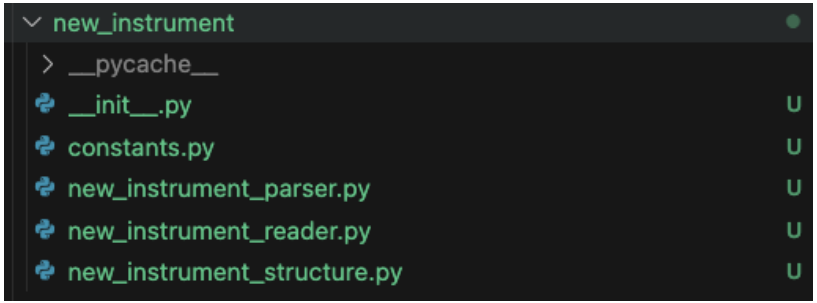
- Organizes raw data into a logical structure, e.g. metadata, plate data, calculated data.
- Populates simple dataclasses defined by schema mapper

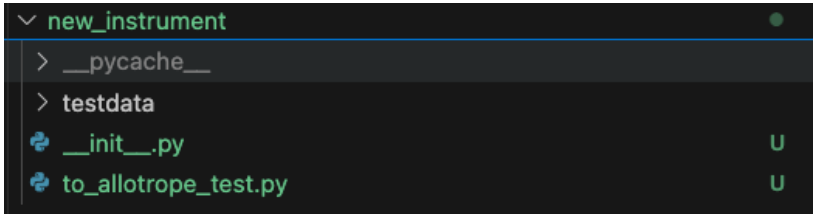
Schema Mapper

- Already built

Allotropy has a command to create all boilerplate for a new parser

```
hatch run scripts:create-parser "New Instrument" "2024/06/plate-reader"
```






```python
from allotropy.allotrope.models.adm.plate_reader.rec._2024._06.plate_reader import (
    Model,
)
from allotropy.allotrope.schema_mappers.adm.plate_reader.rec._2024._06.plate_reader import (
    Data,
    Mapper,
)
from allotropy.named_file_contents import NamedFileContents
from allotropy.parsers.new_instrument.constants import DISPLAY_NAME
from allotropy.parsers.new_instrument.new_instrument_reader import (
    NewInstrumentReader,
)
from allotropy.parsers.new_instrument.new_instrument_structure import (
    create_measurement_groups,
    create_metadata,
)
from allotropy.parsers.release_state import ReleaseState
from allotropy.parsers.utils.pandas import map_rows
from allotropy.parsers.vendor_parser import VendorParser


class NewInstrumentParser(VendorParser[Data, Model]):
    DISPLAY_NAME = DISPLAY_NAME
    RELEASE_STATE = ReleaseState.WORKING_DRAFT
    SUPPORTED_EXTENSIONS = NewInstrumentReader.SUPPORTED_EXTENSIONS
    SCHEMA_MAPPER = Mapper

    def create_data(self, named_file_contents: NamedFileContents) -> Data:
        reader = NewInstrumentReader.read(named_file_contents)
        return Data(
            create_metadata(reader.header, named_file_contents.original_file_path),
            map_rows(reader.data, create_measurement_groups)
        )
```

Allotropy has a growing library of utilities for parsing input files

Section reader

```
##BLOCKS= 12
Plate:  BNCH_69983542_96w   1.3 TimeFormat  Endpoint    Absorbance  Raw FALSE   1
    Temperature(¡C) A1  A2  A3  A4  A5  A6  A7  A8  A9  A10 A11 A12 B1  B2  B3  B4  B5  B6
    0   1.328   0.387   -1.422  -0.446  -0.721  -0.22   -1.902  0.181   0.943   1.387   -0

    0   -1.21525    1.21775 0.60375 0.89175 0.57575 1.36875 0.28075 1.60475 2.37175 2.1237

    0   -1.941625   -2.396625   -0.759625   -2.145625   -0.880625   -1.461625   0.501375

        A1  A2  A3  A4  A5  A6  A7  A8  A9  A10 A11 A12 B1  B2  B3  B4  B5  B6  B7  B8  B9
        4.5012906556530714  0.770153809936019363 -0.48583478500045846    0.559570352689411
~End
Group: PositiveControl
Sample  Well    Concentration   Values  MeanValue   Std.Dev.    CV% WellPlateName
CONTROL_74801   A1      4.501   1.064   1.534   144.129 BNCH_69983542_96w
    A2      0.770               BNCH_69983542_96w
    B1      -0.130              BNCH_69983542_96w
    B2      0.745               BNCH_69983542_96w
    C1      1.255               BNCH_69983542_96w
    C2      -0.055              BNCH_69983542_96w
    D1      1.582               BNCH_69983542_96w
    D2      -0.154              BNCH_69983542_96w

Group Column    Formula Name    Formula Precision   Notation
1   Sample  !SampleNames    3 decimal places    Numeric
2   Well    !WellIDs    3 decimal places    Numeric
3   Concentration   !SampleDescriptor   3 decimal places    Numeric
4   Values  !WellValues 3 decimal places    Numeric
5   MeanValue   Average(!WellValues)    3 decimal places    Numeric
6   Std.Dev.    Stdev(!WellValues)  3 decimal places    Numeric
7   CV% Cv(!WellValues) 3 decimal places    Numeric
8   WellPlateName   !WellPlateName  3 decimal places    Numeric
```

```
for sub_reader in BlockList._iter_blocks_reader(reader):
    if sub_reader.match("^Group"):
        if "WellPlateName" in assert_not_none(
            sub_reader.get_line(sub_reader.current_line + 1),
            msg="Unable to get columns from group block",
        ):
            group_blocks.append(GroupBlock.create(sub_reader))
    elif sub_reader.match("^Plate"):
        header_series = PlateBlock.read_header(sub_reader)
        plate_block_cls = PlateBlock.get_plate_block_cls(header_series)
        header = plate_block_cls.parse_header(header_series)

        export_format_to_data_format = {
            ExportFormat.TIME_FORMAT.value: TimeData,
            ExportFormat.PLATE_FORMAT.value: PlateData,
        }
        data_format: type[TimeData] | type[PlateData] = get_key_or_error(
            "export format", header.export_format, export_format_to_data_format
        )
        block_data = data_format.create(sub_reader, header)

        plate_blocks[header.name] = plate_block_cls(
            block_type="Plate",
            header=header,
            block_data=block_data,
        )
    elif not sub_reader.match("^Note"):
        msg = f"Expected block '{sub_reader.get()}' to start with Group, Plate or Note."
        raise AllotropeConversionError(msg)
```

Pandas row accessors

```
Measurement(
    measurement_identifier=random_uuid_str(),
    timestamp=data[str, "Analysis date/time"],
    sample_identifier=data[str, "Sample ID"],
    cell_type_processing_method=data.get(str, "Cell type"),
    minimum_cell_diameter_setting=data.get(float, "Minimum Diameter (µm)"),
    maximum_cell_diameter_setting=data.get(float, "Maximum Diameter (µm)"),
    cell_density_dilution_factor=data.get(float, "Dilution"),
    viability=data[float, "Viability (%)"],
    viable_cell_density=data[float, "Viable (x10^6) cells/mL"],
    total_cell_count=total_cell_count,
    total_cell_density=data.get(float, "Total (x10^6) cells/mL"),
    average_total_cell_diameter=data.get(float, "Average diameter (µm)"),
    average_live_cell_diameter=data.get(
        float, "Average viable diameter (µm)"
    ),
    viable_cell_count=viable_cell_count,
    average_total_cell_circularity=data.get(float, "Average circularity"),
    average_viable_cell_circularity=data.get(
        float, "Average viable circularity"
    ),
)
```

XML reader

```
analyst=get_val_from_xml(environment, "Experimenter"),
analytical_method_identifier=get_val_from_xml_or_none(
    file_information, "Assay"
),
data_system_instance_identifier=get_val_from_xml(environment, "Computer"),
```

Allotropy has a tool to generate python models for ASM schemas

```python
@dataclass(kw_only=True)
class PlateReaderDocumentItem:
    measurement_aggregate_document: MeasurementAggregateDocument
    analyst: TStringValue | None = None
    electronic_project_record: ElectronicProjectRecord | None = None
    submitter: TStringValue | None = None


@dataclass(kw_only=True)
class PlateReaderAggregateDocument:
    plate_reader_document: list[PlateReaderDocumentItem]
    analysis_sequence_document: AnalysisSequenceDocument | None = None
    calculated_data_aggregate_document: CalculatedDataAggregateDocument | None = None
    custom_information_aggregate_document: CustomInformationAggregateDocument | None = (
        None
    )
    data_system_document: DataSystemDocument | None = None
    device_system_document: DeviceSystemDocument | None = None
    electronic_project_record: ElectronicProjectRecord | None = None
    electronic_signature_aggregate_document: ElectronicSignatureAggregateDocument | None = (
        None
    )
    processed_data_aggregate_document: ProcessedDataAggregateDocument | None = None
    statistics_aggregate_document: StatisticsAggregateDocument | None = None


@dataclass(kw_only=True)
class Model:
    field_asm_manifest: AdmCoreREC202409ManifestSchema | str
    plate_reader_aggregate_document: PlateReaderAggregateDocument | None = None
```

```python
@dataclass(kw_only=True)
class MeasurementDocument:
    device_control_aggregate_document: DeviceControlAggregateDocument
    measurement_identifier: TStringValue
    sample_document: SampleDocument
    absorbance: TQuantityValueMilliAbsorbanceUnit | None = None
    calculated_data_aggregate_document: CalculatedDataAggregateDocument | None = None
    custom_information_aggregate_document: CustomInformationAggregateDocument | None = (
        None
    )
    detection_type: TStringValue | None = None
    electronic_project_record: ElectronicProjectRecord | None = None
    error_aggregate_document: ErrorAggregateDocument | None = None
    image_aggregate_document: ImageAggregateDocument | None = None
    measurement_time: TDateTimeStampValue | None = None
    processed_data_aggregate_document: ProcessedDataAggregateDocument | None = None
    statistics_aggregate_document: StatisticsAggregateDocument | None = None
    compartment_temperature: TQuantityValueDegreeCelsius | None = None
    mass_concentration: TQuantityValuePicogramPerMilliliter | None = None
    electropherogram_data_cube: TDatacube | None = None
    chromatogram_data_cube: TDatacube | None = None
    processed_data_document: ProcessedDataDocument | None = None
    absorption_profile_data_cube: TDatacube | None = None
    fluorescence: TQuantityValueRelativeFluorescenceUnit | None = None
    fluorescence_emission_profile_data_cube: TDatacube | None = None
    luminescence: TQuantityValueCountsPerSecond | TQuantityValueRelativeLightUnit | None = (
        None
    )
    luminescence_profile_data_cube: TDatacube | None = None
```

```
hatch run scripts:download-schema <url>
```

```
hatch run scripts:generate-schemas
```

Allotropy has a schema mappers to make populating ASM easy

```python
@dataclass(frozen=True)
class Measurement:
    # Measurement metadata
    type_: MeasurementType
    device_type: str
    identifier: str
    sample_identifier: str
    location_identifier: str

    # Optional metadata
    well_plate_identifier: str | None = None
    detection_type: str | None = None
    sample_role_type: SampleRoleType | None = None

    # Measurements
    absorbance: float | None = None
    fluorescence: float | None = None
    luminescence: float | None = None

    # Settings
    detector_wavelength_setting: float | None = None
    detector_bandwidth_setting: float | None = None
    excitation_wavelength_setting: float | None = None
    excitation_bandwidth_setting: float | None = None
    wavelength_filter_cutoff_setting: float | None = None
    detector_distance_setting: float | None = None
    scan_position_setting: ScanPositionSettingPlateReader | None = None
    detector_gain_setting: str | None = None
    detector_carriage_speed: str | None = None
    compartment_temperature: float | None = None
    number_of_averages: float | None = None
```

- Removes need for dev to create nested document
- One schema mapper per ASM schema - used by all parsers
- Enables upgrading schema version without changing parsers

```python
def _get_ultraviolet_absorbance_measurement_document(
    self, measurement: Measurement
) -> MeasurementDocument:
    return MeasurementDocument(
        measurement_identifier=measurement.identifier,
        sample_document=self._get_sample_document(measurement),
        device_control_aggregate_document=DeviceControlAggregateDocument(
            device_control_document=[
                DeviceControlDocumentItem(
                    device_type=measurement.device_type,
                    detection_type=measurement.detection_type,
                    detector_wavelength_setting=quantity_or_none(
                        TQuantityValueNanometer,
                        measurement.detector_wavelength_setting,
                    ),
                    number_of_averages=quantity_or_none(
                        TQuantityValueNumber, measurement.number_of_averages
                    ),
                    detector_carriage_speed_setting=measurement.detector_carriage_speed,
                    detector_gain_setting=measurement.detector_gain_setting,
                    detector_distance_setting__plate_reader_=quantity_or_none(
                        TQuantityValueMillimeter,
                        measurement.detector_distance_setting,
                    ),
                )
            ]
        ),
```

Allotropy has a testing library that allows no-code test addition

```
hatch run test tests/parsers/moldev_softmax_pro/ --overwrite
```

```
hatch run test tests/parsers/moldev_softmax_pro/ --filter <test_case>
```

Automatically detects test cases in testdata/

Validates output against ASM schema

Overwrite expected output easily while iterating

~100 lines of code of non-boilerplate code to add a simple parser!

```python
import pandas as pd

from allotropy.named_file_contents import NamedFileContents
from allotropy.parsers.lines_reader import determine_encoding
from allotropy.parsers.utils.pandas import (
    df_to_series_data,
    read_csv,
    read_excel,
)


class RevvityMatrixReader:
    SUPPORTED_EXTENSIONS = "csv,xlsx"
    data: pd.DataFrame

    def __init__(self, named_file_contents: NamedFileContents) -> None:
        if named_file_contents.extension == "csv":
            contents = named_file_contents.contents.read()
            encoding = (
                determine_encoding(contents, named_file_contents.encoding)
                if isinstance(contents, bytes)
                else None
            )
            named_file_contents.contents.seek(0)
            df = read_csv(named_file_contents.contents, encoding=encoding)
        else:
            df = read_excel(named_file_contents.contents)
            # Reading a percent value (50%) in read_excel results in a decimal: 0.5
            # Detect and adjust value back to 0-100%
            first_row = df_to_series_data(df, 0)
            viability = first_row[str, "Viability"]
            if "%" not in first_row[str, "Viability"] and float(viability) < 1:
                df["Viability"] = df["Viability"] * 100
        self.data = df
```

```python
def create_measurement_group(data: SeriesData) -> MeasurementGroup:
    # This function will be called for every row in the dataset, use it to create
    # a corresponding measurement group.

    # Cell counts are measured in cells/mL, but reported in millions of cells/mL
    viable_cell_density = float(
        Decimal(data[float, "Live Cells/mL"]) / Decimal("1000000")
    )
    total_cell_density = data.get(float, "Total Cells/mL")
    if total_cell_density:
        total_cell_density = float(Decimal(total_cell_density) / Decimal("1000000"))
    dead_cell_density = data.get(float, "Dead Cells/mL")
    if dead_cell_density:
        dead_cell_density = float(Decimal(dead_cell_density) / Decimal("1000000"))

    errors = data.get(str, "Errors:", validate=SeriesData.NOT_NAN)
    return MeasurementGroup(
        measurements=[
            Measurement(
                measurement_identifier=random_uuid_str(),
                # NOTE: instrument file does not provide a timestamp, but it is required by ASM, so pass
                # EPOCH to signal no timestamp.
                timestamp=DEFAULT_EPOCH_TIMESTAMP,
                sample_identifier=data[str, "Well Name"],
                viability=data[float, "Viability"],
                total_cell_count=data.get(float, "Total Count"),
                total_cell_density=total_cell_density,
                average_total_cell_diameter=data.get(float, "Total Mean Size"),
                viable_cell_count=data.get(float, "Live Count"),
                viable_cell_density=viable_cell_density,
                average_live_cell_diameter=data.get(float, "Live Mean Size"),
                dead_cell_count=data.get(float, "Dead Count"),
                dead_cell_density=dead_cell_density,
                average_dead_cell_diameter=data.get(float, "Dead Mean Size"),
                errors=[
                    Error(error=error)
                    for error in (errors.split(",") if errors else [])
                ],
            )
        ]
    )
```

There are 5 easy ways to contribute

Connector Code

Test Data

Testing Connectors

Raise Issues

Spread the word

Thank you!

Questions?