Method development kit for the Orbitrap Astral MS family

Max Hoek, Arne Kreutzmann, Kyle Le Huray, Sophia Steigerwald, Hamish Stewart, Johannes Petzoldt, Eugen Damoc, Christian Hock
Thermo Fisher Scientific, Hanna-Kunath-Straβe 11, Bremen, Germany, 28199

Abstract

Purpose: Real-time interaction with the Thermo Scientific™ Orbitrap™ Astral™ mass spectrometer during acquisitions while maintaining high scan rates and sensitivity, enabling smart acquisition strategies.

Methods: Open and language independent technologies to facilitate a connection to the mass spectrometer.

Results: Update mass list tables and filters based on received spectrum data to enhance method acquisition, with low latency.

Introduction

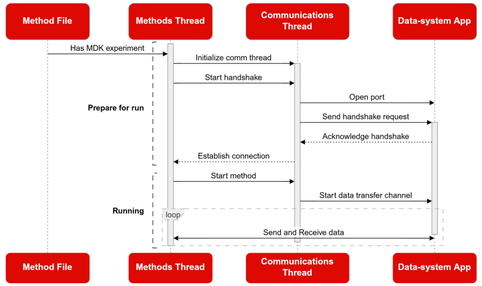
The Thermo Scientific™ Orbitrap™ Astral™ MS platform enables high-throughput analysis with deep proteome coverage and accurate and precise quantitation. These performance characteristics are achieved through a high degree of parallelization of ion packages, allowing the instrument to sustain very high repetition rates. To support this architecture, the instrument control software has been highly streamlined, making traditional scan-level control mechanisms, such as the instrument Application Programming Interface (iAPI)(1), challenging to implement on the Orbitrap Astral MS platform.

We introduce a new Method Development Kit (MDK) for the Orbitrap Astral MS platform, facilitating real-time interaction with the instrument. The MDK enables development of exciting features. It aims to provide a low barrier of entry to enable users with limited programming knowledge to write their own smart acquisition strategies.

The MDK uses language independent technologies used in the data transfer allowing applications to be written in any programming language on any operating system. All code snippets on this poster are written in Python. Once the MDK is released, we aim to provide reference implementations for multiple programming languages.

Figure 1. MDK connection schematic overview.

Figure 2. Connection initialization sequence diagram.



Setting up the connection

A new bi-directional connection between the mass spectrometer and a user application is established for each method run that uses the MDK (Figure 1). The MDK can connect to any endpoint on the network, it could connect to a computer that has more powerful processing capabilities that the standard windows PC.

Upon receiving a method using MDK, the instrument will open a port. A handshake is performed where each keys are exchanged, following the elliptic curve security model. Upon success an encrypted connection is established. Otherwise, the method is canceled. When the run starts, a data transfer channel is opened that allows sending and receiving of data. At the end of the run, an “end of acquisition” message is sent which contains a summary of the run. Finally, the communications channels is closed (Figure 2).

Receiving spectrum data

When configuring the method, specific scan nodes can be marked to send their data through the MDK. Only spectra from these nodes are transmitted, giving control over which data is received. For example, a user can opt to only receive Orbitrap full scan spectra and not Astral MS^2 spectra, reducing the data load by >99%. Data is sent as soon as signal processing is finished and may be sent out-of-order, due to the parallelized processing of spectra. The application can even receive scan data even before that data is written to the raw file. Because the data has the Apache Arrow format it can be easily converted into a data frame by the user application as shown in the code snippet (Figure 3).

Figure 3. Code snippet showing receipt of spectrum data.
The snippet was taken from the MDK Python reference implementation.

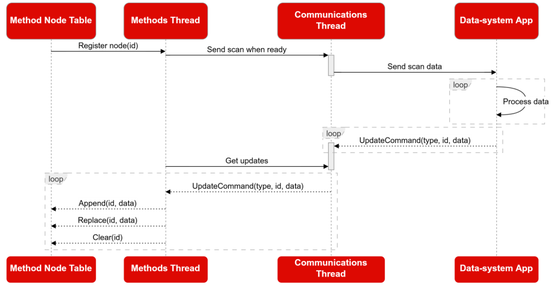
```python
def receive_message(connection: mdk.Connection) -> mdk.Message:
    """Receive a message from the instrument and return it as a structured object."""
    match connection.socket.recv_multipart(zmq.NOBLOCK):  # message type: tuple[bytes, ...]
        case (b'spectrum', meta_info, spectrum_data):
            source = pa.BufferReader(spectrum_data)
            reader = pa.ipc.RecordBatchStreamReader(source)
            return mdk.SpectrumData(spectrum=reader.read_next_batch(), settings=mdk.decode_scan_meta_info(meta_info))
        # Error handling and handling for message type 'complete' (indicating the end of acquisition) not shown


def print_spectrum(data: mdk.SpectrumData) -> None:
    print(data.settings.scan_number, data.settings.analyzer_type)  # Print scan settings metadata. Many more fields are available
    df = pl.from_arrow(data.spectrum)  # Use polars or pd.DataFrame.from_arrow(data.spectrum) for a Pandas dataframe
    print(df.head())
```

Sending updates

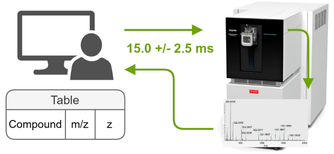
In the method, the user also specifies which inclusion/exclusion/trigger/target (tSIM/tMS^2) tables should be dynamically controlled through the MDK. The user application sends “append”, “replace”, and “clear” commands to these tables to update them. During execution at the start of every cycle, all user commands are processed, and tables are updated (Figure 4). This provides versatile control over the acquisition without directly interacting with scan queuing. As a result, the maximum delay for command processing is proportional to the cycle time of the method.

Figure 4. Data flow sequence diagram.



Sending updates

Figure 5. Latency benchmark.

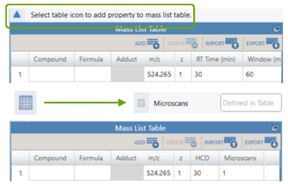


Measurements were performed to determine the latency. The user application was set up with a loop to send a single target for a tSIM scan and time when the corresponding spectrum was received (Figure 5). Measurements showed that the round-trip response time was 15.0 +/- 2.5 ms. This encompasses the user application sending a target, the instrument scheduling, recording, processing, and subsequently sending back the spectrum, all in just 15 ms. Scan settings were configured to give a high scan rate on the instrument. Results may vary based on injection time and spectral density. Also, for Orbitrap scans eFT processing takes significantly longer than Astral scan processing.

Which parameters are available for updates is entirely controlled through the method editor. Like regular Mass List Tables every parameter that is specified as table value appears as a column, and each of these columns is included in the MDK update command (Figure 6).

During the connection handshake the instrument will send the exact schema of every table that can receive update commands along with the exact column names and data types in each table.

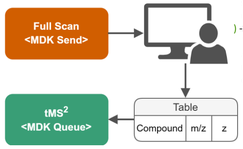
Figure 6. Selecting the MDK parameters



Example user applications

DDA-like method code example

Figure 7. MDK DDA-like method example.




```python
def example_dda_acquisition(
    spectrum: pa.RecordBatch,
    settings: mdk.ScanMetaInfo,
    node_schema: dict[str, pa.Schema]
) -> tuple[str, pa.Table]:
    """Process the received spectrum data and create a DDA-like Top150 method."""
    # Add a compound name with the master scan that is added in the raw file.
    name_expr = pl.lit(f'MDK Master Scan {settings.scan_number}', dtype=pl.Utf8)
    candidates = (
        pl.from_arrow(spectrum).lazy()
        .filter(pl.col('charge').is_between(2, 5))  # Filter for charge states 2 - 5
        .top_k(k=150, by='intensity')  # Select TopN 150 most intense peaks
        .with_columns(name_expr.alias('compound_name'))
        .select(pl.Schema(node_schema['TMS2_ID']).names())  # Select relevant columns.
        .collect().to_arrow()  # Collect the candidates and convert to Apache Arrow
    )
    return TMS2_ID, candidates
```

Figure 7 shows an example of how one could configure a simple DDA-like acquisition to run with the MDK. Of course, DDA is already natively supported in the method editor, this serves as a basic example to demonstrate the MDK. In the method editor the Full Scan is marked with “MDK Send” to indicate that scans from this node will be sent to the user application. Instead of a ddMS^2 node, a tMS^2 node is added and marked as “MDK Queue”. This signifies that candidates sent to this node end up in a queue and are scanned only once. While running the Full Scan data is processed using the code snippet on the right. The resulting candidates are sent back using the “Replace” command type, which results in a scan sequence that mimics a DDA method using top 150 and a charge state filter. This example illustrates the low barrier of entry to write an MDK program. Altogether entire user application can be written in as little as ~50 lines of code.

Hybrid DIA

A regular DIA method can be converted to a hybrid DIA method with minimal modifications (Figure 8). The DIA scans are sent to the user application which then compares them to a list of fragment ions. Spectra matching these fragment ions are added to the tMS^2 node using the MDK “Append” command. By specifying retention time windows, the application can control when these targets are scanned.

Figure 9. MDK CE retriggering.

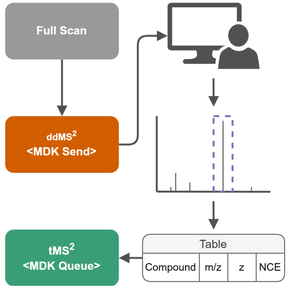
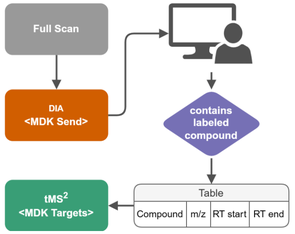


Figure 8. MDK Hybrid DIA.



Collision energy retriggering

The typical normalized collisional energy (NCE) gives good fragmentation for a broad range of precursors. However, occasionally a precursor is either under- or over-fragmented. With the MDK the ddMS^2 scans can be sent to the user application which analyzes the region around the isolation window, shown by dashed purple box (Figure 9). If poor fragmentation is determined, the precursor can be retriggered with an adjusted NCE. Optimizing the NCE in this way results in fewer unidentified precursors throughout the experiment.

Conclusions

- Real-time interaction with methods during acquisition.
- Fast response time for sending and receiving updates
- Development kit enabling smart acquisition strategies.
- Language independent technologies that provide flexibility for users.
- Reference implementations in multiple languages to handle the connection details.

References

1. Instrument application programming interface (iAPI) https://github.com/thermofisherlsms/iapi.

Acknowledgements and disclosure of interests

The authors would like to thank the many colleagues from Thermo Fisher Scientific and collaborators around the world who made this work possible. All authors are employees of Thermo Fisher Scientific, a manufacturer of scientific instruments who funded this work.

Trademarks/licensing

Learn more at thermofisher.com/orbitrapastralzoom