## **Tick Data Processor Use Cases**

**1. Client application Interaction with Tick Data Processor**

There are several methods a client application can interact with Tick Data Processor (TDP)

* 1. Direct

Tick Data Processor uses text-based data representation for sending and receiving data.  
Query request consists of one of more lines of JSON-formatted request header, followed by one or more input records separated by new line character.  
Result record set consist of one line of JSON-formatted response header followed by zero or more result records , also separated by new line character.

Client applications can successfully interact with TDP if API for TCP/IP communication, text formatting, parsing and data type conversion are offered by programming environment.

However, client code will be broken if data representation or communication protocols changes e.g. text-bases data protocol can be replaced with binary protocol to achieve better performance by optimizing record formatting and parsing.

1.2 Python extension (taqpy)

This method achieves performance benefits by using C++ to format service requests, allocate memory and parse inbound result record sets. It also hides communication protocol and data representation.

1.3 Astral API

This method exposes TDP query service to multitude of client application by offering TEST API wrapper around taqpy.  
It offers a single end point for user authentication and access to all data sources within Astral cloud.

**2. Use Cases**

We can identify following use cases

2.1 Factsheet

Current use by Enricher is limited to two functions:

1. Retrieval of quote to determine midpoint as reference price at order's arrival time
2. Calculate resting order duration measure according to algorithm that uses NBBO data

Suggested method: taqpy Python extension

Because Enricher is inaccessible outside Astral cloud its Implementation does not need access control. All what necessary is connectivity i.e. open TCP/IP address and port.

2.2 Performance Library

Suggested method: Astral API

Performance library offers set of advanced calculations based on transactional data input. Transactional data is sourced via Astral API; therefore, it makes sense to use the same method to retrieve related calculations from TDP.

2.3 Astral API Standard Data

Standard data schema can be extended to include commonly used tick data calculations, such as reference prices, to make product more useful to clients. Because calculations are standard and limited in numbers, it makes sense to store values together with standard transactional data. Presence of additional fields in database enables clients to use them in filters and aggregations.

Suggested method: taqpy Python extension while processing transactional data and store results together in the same database (same as 2.1)

2.4 Astral API extension to Standard Data

Astral API can be extended to offer client ability to create custom result fields by calling TDP functions and passing elements of transactional data as input arguments.

There is opportunity to hide some complexity by adding fields that are well understood and identified, in which case required arguments can be derived from other fields in the schema e.g. "Arrival Quote" is relevant to orders only and is the latest valid quote at the time of order arrival , therefore user might be able to simply select this field , and API implementation would pass correct arguments from the order record.

2.5 Astral API extension for TAQ

Astral API can be modified to expose TDP functionality where input records are passed via REST API rather than from results of the Standard Data query.