Change Guide: B-PIPE Macro Indicators Service

1. High-Level Overview and Impact

Bloomberg is retiring the legacy //blp/economic-data service and replacing it with the new, more efficient //blp/macro-indicators service. It represents **an architectural shift in how data is delivered.**

The core change is the move from a **stateless** to a **stateful** data consumption model.

- **Old Model:** Every real-time message (HeadlineEconomicEvent) was self-contained, including both the event value and all descriptive reference data (e.g., DESCRIPTION). This was simple for identification of messages but inefficient, as the same static information was sent repeatedly.
- **New Model:** The data is **now split for efficiency.** Static reference data is sent once in a MacroReferenceData message at the start of a subscription. Subsequent real-time updates are sent as lightweight MacroHeadlineEvent messages containing only the dynamic values that change.

Here is a dif example of message change:

This change requires the client application to become stateful—it must now receive, store, and maintain the initial reference data. This stored "context" must then be used to enrich the incoming real-time events to form a complete business object. The critical link between the reference data and its corresponding events is the **CorrelationId** that the client application provides during the subscription request.

2. Before vs. After Comparison

Feature	Old System (//blp/economic-data)	New System (//blp/macro-indicators)
Service Name	//blp/economic-data	//blp/macro-indicators
Subscription String	Included a "topic" segment (e.g., /headline-actuals/).	"Topic" segment is removed. (e.g., /ticker/...), see “Subscription path example” row below
Subscription path example	//blp/economic-data / headline-actuals / ticker / NFP TCH Index	//blp/macro-indicators / ticker / NFP TCH Index
Data Model	**Stateless**: Each message was a complete, self-contained object.	**Stateful**: Data is split into MacroReferenceData (static) and MacroHeadlineEvent (dynamic).
Required Logic	Each message could be processed independently.	Client **must** cache reference data to interpret real-time events.
Pairing Mechanism	Not required; messages were complete.	The client-provided CorrelationId is the essential link between reference data and event data.
Key Data Type	VALUE.SINGLE was delivered as a **String**.	VALUE.SINGLE is now delivered as a **Number** (Integer/Float).
Event Filtering	Subscription "topic" provided a server-side pre-filter.	A single subscription returns all event types (ACTUAL, ESTIMATE, etc.), requiring client-side filtering.

---

3. Implementation Plan: C++ (BlpConn Connector)

The guiding principle for the BlpConn C++ library is to remain a **stateless, high-performance "shovel."** Its responsibility is to facilitate communication with the B-PIPE SDK and efficiently forward raw data to the Go application. It should **not** implement any caching or state management logic.

C++ Tasks:

1. Update Configuration Handling

The config.json file and any internal defaults reference the old service name.

- **Why:** The application must connect to the new service endpoint to receive data.
- **Action:** Update all configurations and documentation to use //blp/macro-indicators as the service name.

2. Modify Subscription String Construction

The old logic used a subscription_type (e.g., HeadlineActuals) to build the full subscription string. This concept is now obsolete for string construction.

- **Why:** The new service uses a simplified, topic-less subscription format.
- **Action:** Modify the internal C++ logic that builds the subscription string for the BLPAPI. This logic must now construct the string in the new format (//blp/macro-indicators/<topic_type>/<topic>), ignoring the old subscription_type member.

3. Update FlatBuffers Schema

The data schema must be updated to support the new message types and structures from Bloomberg.

- **Why:** Without an updated schema, the library cannot serialize or deserialize the new data formats from the //blp/macro-indicators service.
- **Action:** Edit the ./fb/blpconn.fbs schema file. Define the new message types: MacroReferenceData and MacroHeadlineEvent. Ensure their fields accurately reflect the new API specification, including the addition of new fields (like RELEVANCE_VALUE) and the removal of fields from MacroHeadlineEvent. The VALUE.SINGLE field must be updated to a numeric type.
- **Action:** Regenerate the C++ and Go FlatBuffers bindings by running the make command in the fb directory.

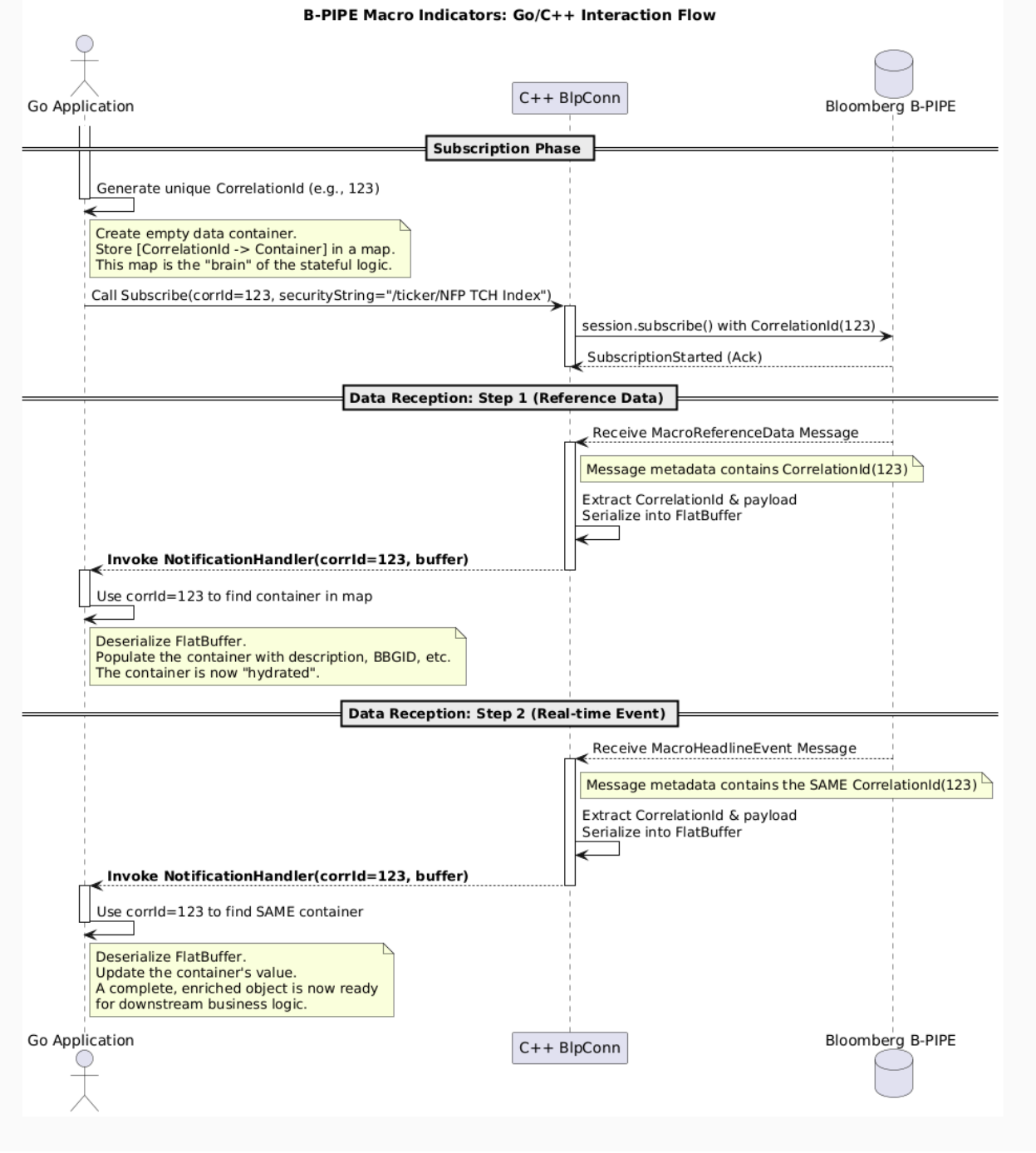
4. Ensure correlation_id Propagation

The correlation_id is the critical link that the Go application will rely on to pair messages.

- **Why**: This is the most critical task. The Go application is now completely dependent on the CorrelationId to link the initial MacroReferenceData (the context) with all subsequent MacroHeadlineEvent messages (the values). A failure to forward the CorrelationId for even a single message will break the data reassembly process on the Go side, rendering the data useless.

- **Action**: Verify and guarantee that the correlation_id from the Bloomberg Message metadata is correctly captured and included in the FlatBuffer payload for all message types passed to the Go NotificationHandler callback. This must be tested for MacroReferenceData, MacroHeadlineEvent, and any other relevant messages. This data contract is the foundation of the new architecture.

Sequence Diagram



Acceptance Criteria

Category	Acceptance Criterion
1. Build & Schema	The FlatBuffers schema (`.fbs` file) has been updated to define the new `MacroReferenceData` and `MacroHeadlineEvent` message types, and the entire library (including new bindings) compiles successfully into a static library (`.a`).
2. Connection & Subscription	The library can successfully open the `//blp/macro-indicators` service and establish a subscription using the new, simplified subscription string format (e.g., `/ticker/<identifier>`), correctly ignoring the old `subscription_type` field.
3. Event Handling (New Message Types)	The internal C++ `EventHandler` correctly identifies and dispatches incoming B-PIPE messages of type `MacroReferenceData`.
4. Event Handling (New Message Types)	The internal C++ `EventHandler` correctly identifies and dispatches incoming B-PIPE messages of type `MacroHeadlineEvent`.
5. Data Extraction (CorrelationId)	For **all** relevant messages (including `MacroReferenceData` and `MacroHeadlineEvent`), the `CorrelationId` is correctly extracted from the B-PIPE `Message` metadata.
6. Data Extraction (Payload)	For a `MacroReferenceData` message, all required static fields (e.g., `DESCRIPTION`, `ID_BB_GLOBAL`) are correctly extracted from the payload.
7. Data Extraction (Payload)	For a `MacroHeadlineEvent` message, all required dynamic fields (e.g., `EVENT_TYPE`, `VALUE`) are correctly extracted, and the `VALUE.SINGLE` field is correctly handled as a **numeric type**.

8. Serialization	Extracted data from both `MacroReferenceData` and `MacroHeadlineEvent` messages is correctly serialized into the corresponding FlatBuffers binary format.
9. Callback Invocation	The Go `NotificationHandler` callback is successfully invoked for both `MacroReferenceData` and `MacroHeadlineEvent` messages.
10. Callback Contract	The callback to Go correctly provides both the **`CorrelationId`** and the **serialized FlatBuffer** payload, ensuring the Go layer has the necessary information to link the messages.
11. Architectural Compliance	The C++ library remains **entirely stateless**, with no internal caching or storage of `MacroReferenceData` or any other application-level state.
12. Testing	Unit tests have been created or updated to verify the new parsing, serialization, and `CorrelationId` propagation logic for the `MacroReferenceData` and `MacroHeadlineEvent` message paths.

Additional Questions Asked To Bloomberg

Green - my questions to Bloomberg
Blue - answers from Bloomberg

1. Service and Subscription String Format:

Our understanding is that the service name is now //blp/macro-indicators and that subscription strings are simplified. For example, a subscription to /headline-actuals/ticker/NFP TCH Index on the old service is now simply /ticker/NFP TCH Index on the new service, with the "topic" segment removed. Is this correct?

This is correct. We are moving from a //service/topic/identifier syntax to a simplified //service/identifier subscription model. The topic that used to identify actuals, surveys and calendars are no longer needed in the new macro-indicators service.

2. Data Message Separation (Stateful Logic):

We understand the data model has been split. Static, descriptive data (like DESCRIPTION, ID_BB_GLOBAL) is now delivered in an initial MacroReferenceData message, while subsequent real-time updates (the VALUE) arrive in separate, lightweight MacroHeadlineEvent messages. Is this correct?

Yes, this is correct.

3. Pairing Mechanism (CorrelationId):

To link the MacroReferenceData with its corresponding MacroHeadlineEvent messages, our understanding is that we must use the CorrelationId that we provide when making the subscription. This CorrelationId will be present on the metadata of every message for that stream, allowing us to associate the real-time value with the correct cached reference data. Is this the correct and intended pairing mechanism?

Yes, the CorrelationId that is provided upon creating the subscription is carried through the message object of each message for that stream. This is the intended mechanism for pairing real-time values with their corresponding cached data, and it aligns with the behavior defined in BLPAPI.

Additional information is also available in:
https://developer.bloomberg.com/portal/apis/blpapi?chapterId=5405&entityType=document#blpapi-api_concepts-subscriptions-subscription_strings-subscription_string_components-correlation_id

4. Event Type Filtering:

Because subscription topics are gone, we assume a single subscription (e.g., to /ticker/NFP TCH Index) will now deliver all event types for that indicator (ACTUAL, REVISION and especially ESTIMATE, CALENDAR) on a single stream. This means we are now responsible for fully filtering these EVENT_TYPEs on our client-side application. Is

this assumption correct? If so, can you please provide us with example messages form ESTIMATE and CALENDAR type of events?

Yes, that is correct. Here are some examples:

EVENT_TYPE = CALENDAR


```
{
        "MacroCalendarEvent":
        {
                "EVENT_TYPE":"CALENDAR",
                "EVENT_SUBTYPE":"UPDATE",
                "OBSERVATION_PERIOD":"Jul",
                "ECO_RELEASE_DT":
                {
                        "DATETIME":"2025-08-25T10:00:00.0
                        00-04:00"
                },
                "EVENT_ID":2161832,
                "RELEASE_STATUS":"RELEASED",
                "RELEVANCE_VALUE":52.381
        }

}
```

```
{
        "MacroCalendarEvent":
        {
                "EVENT_TYPE":"CALENDAR",
                "EVENT_SUBTYPE":"NEW",
                "OBSERVATION_PERIOD":"Aug 22",
                "ECO_RELEASE_DT":
                {
                        "DATETIME":"2025-08-25T16:00:00.0
                        00-04:00"
                },
                "EVENT_ID":2217584,
                "RELEASE_STATUS":"SCHEDULED",
                "RELEASE_VALUE":34.2342
        }

}
```


EVENT_TYPE = ESTIMATE


```
{
        "MacroHeadlineEvent":
        {
                "EVENT_TYPE":"ESTIMATE",
                "EVENT_SUBTYPE":"NEW",
                "EVENT_ID":2142643,
                "RELEASE_VALUE":75.5102,
                "OBSERVATION_PERIOD":"Aug",
                "ECO_RELEASE_DT":
                {
                        "DATETIME":"2025-09-11T08:30:00.000-04:00"
                },
                "VALUE":
                {
```

```
                "DISTRIBUTION":
                {
                        "LOW":3,
                        "HIGH":3.1,
                        "MEDIAN":3.1,
                        "AVERAGE":3.07333333333333,
                        "STANDARD_DEVIATION":0.0377123616632826,
                        "NUMBER":3
                }
        }
    }

}


 {
        "MacroHeadlineEvent":
        {
                "EVENT_TYPE":"ESTIMATE",
                "EVENT_SUBTYPE":"UPDATE",
                "EVENT_ID":2151223,
                "RELEVANCE_VALUE":98.6395,
                "OBSERVATION_PERIOD":"Aug 23",
                "ECO_RELEASE_DT":
                {
                        "DATETIME":"2025-08-28T08:30:00.0
                        00-04:00"
                },
                "VALUE":
                {
                        "DISTRIBUTION":
                        {
                                "LOW":221,
                                "HIGH":238,
                                "MEDIAN":230,
                                "AVERAGE":230.733805555
                                556,
                                "STANDARD_DEVIATION":3
                                .60216349973985,
                                "NUMBER":36
                        }
```

```
                }
            }

}
```

5. Data Type Change in VALUE Field:

Our analysis of sample messages indicates that the data type of the VALUE.SINGLE field within a MacroHeadlineEvent has changed from a String in the old service to a Number (Integer/Float) in the new service. Can you confirm this data type change is intentional and consistent? What is the final data type of this change?

Yes, this change was intentional. One of the most frequent requests from our beta partners was to enforce strong typing for numeric values to improve processing efficiency. Number of estimates and event IDs are provided in int32, and the rest of the numerical values like actual release and survey distributions (including calculated fields) are in float64 as they may contain decimal points.

4. Implementation Plan: Go Application Orchestrator

The guiding principle for the Go application is to become the **owner of all stateful logic.** It is responsible for orchestrating subscriptions, managing the cache of reference data, and combining the split messages into complete, usable business objects.

Go Tasks:

1. Define New Data Structures

- **Why:** The old data structures are insufficient as they assume a single, self-contained message. A new structure is needed to hold the combined data from the two new message types.
- **Action:** Create a new Go data structure that will serve as the complete, stateful representation of an economic indicator. This struct must have fields to hold information from both MacroReferenceData (e.g., Description, BBGID) and MacroHeadlineEvent (e.g., LatestValue, EventType). It should also include an internal state field (e.g., a boolean flag) to track whether its reference data has been received.

2. Implement State Management (Caching)

This is the core architectural change.

- **Why:** Because real-time events no longer contain descriptive information, this information must be stored by the application.
- **Action:** Implement an application-level map or other dictionary-like structure to manage the state of all active subscriptions. This structure will link the correlation_id of each subscription to the corresponding complete data structure defined in the previous step.

3. Update Subscription Workflow

The process for initiating a subscription must now include setting up the state for that subscription.

- **Why:** The application must be ready to receive and store the MacroReferenceData before the first real-time event arrives.
- **Action:** Modify the subscription logic. Before calling ctx.Subscribe, your application must:
    1. Generate a new, unique correlation_id.
    2. Create a new, empty instance of your complete data structure.
    3. Store this instance in your state management map, using the new correlation_id as the key.
    4. Call ctx.Subscribe with the new service name, the simplified topic string, and the correlation_id you just generated and stored.

4. Re-architect the NotificationHandler Callback

The observer callback is where the split messages are reassembled into a meaningful whole.

- **Why:** This is the entry point for all data from BlpConn, and it must now handle the logic of enriching lightweight events with cached context.
- **Action:** Modify your NotificationHandler function to implement the following logic:
    1. Upon receiving data, extract the correlation_id provided by the C++ layer.
    2. Use this correlation_id to look up the correct data structure from your state management map.
    3. Deserialize the binary buffer to determine the message type.
    4. Implement logic to handle the different message types:
        - If it is MacroReferenceData, populate the static, descriptive fields in your stored data structure and mark it as "hydrated."
        - If it is MacroHeadlineEvent, update the dynamic, real-time fields in that same data structure.
    5. Once an event is processed, your business logic should operate on the complete, enriched data structure from your map, not on the raw incoming message.
- **Action:** The old subscription topics provided a server-side filter. Since that is gone, implement client-side filtering logic within the handler to process only the EVENT_TYPEs you care about (e.g., ACTUAL, REVISION) and explicitly ignore any others that are not needed.