OPEN MARKET DATA INITIATIVE

BLPAPI Use Case: Request/Response Paradigm

February 1, 2012

Bloomberg

CONTENTS

02 BPLAPI PROGRAMMING INTERFACE

02 REQUEST/RESPONSE OVERVIEW

03 SERVICES

03 C++ EXAMPLE: EQUITY INDEX CALCULATOR

BLPAPI Use Case: Request/Response Paradigm

BPLAPI PROGRAMMING INTERFACE

BLPAPI is the programming interface for Bloomberg's Desktop and Server API products as well as for Managed B-Pipe and Platform. Built on a flexible service-oriented architecture, BLPAPI supports both request/response and publish/subscribe paradigms. Request/response services are typically used to retrieve reference data such as security master information or results of financial calculations. Other applications are possible, for example submissions of trade orders; such a service can also use the subscription paradigm to receive, asynchronously, notifications of order fills.

This paper shows how to use BLPAPI for request/response services.

REQUEST/RESPONSE OVERVIEW

The following diagram shows how request/response services fit into BLPAPI:

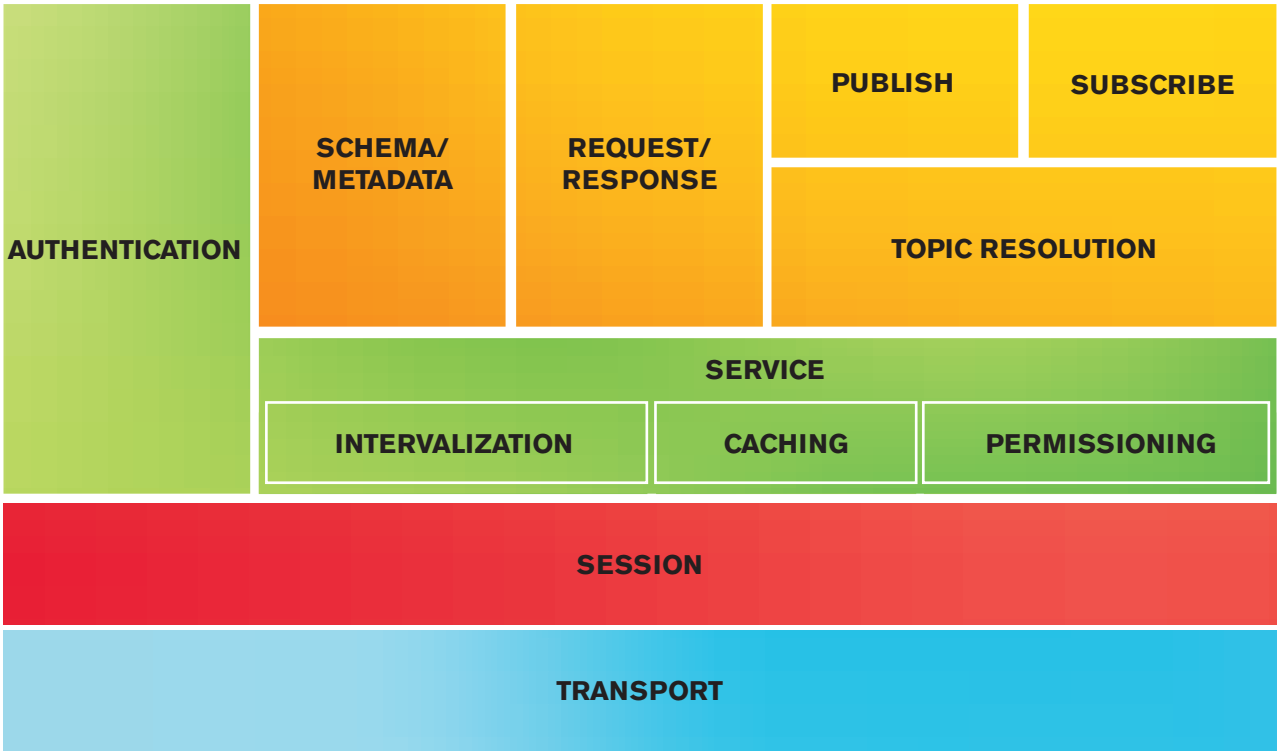

Figure 1. BLPAPI Interface Components

Although the request/response paradigm is suited for synchronous operations, BLPAPI itself supports both synchronous and asynchronous programming models. Applications can distribute requests across one or more server sessions and can flexibly dispatch response processing among BLPAPI or application threads. As detailed below, BLPAPI's service orientation means that the interface is not inherently limited to any class of request/response operations.

BLPAPI Use Case: Request/Response Paradigm

Prior to using request/response or other services, an application must create a session; in some deployments, this may require a separate authentication step. The following sequence diagram shows the session establishment process:

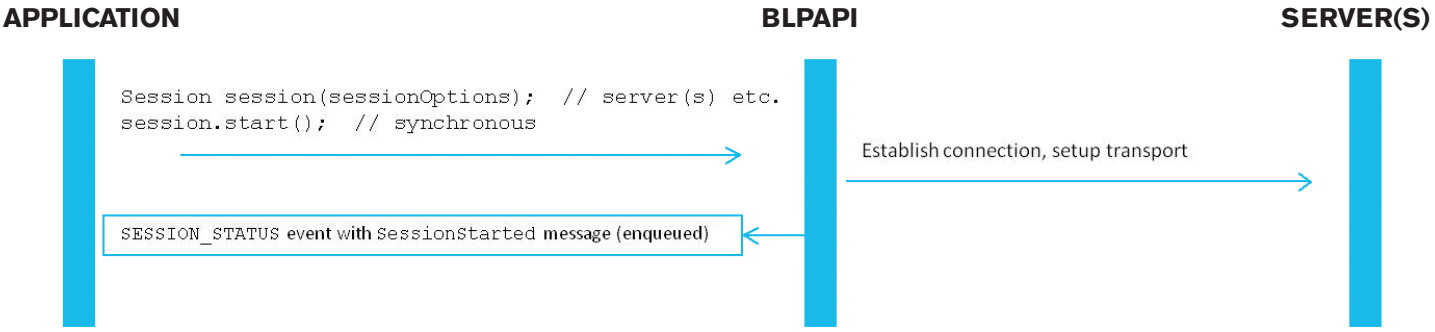

Figure 2. Session establishment (using synchronous API)

SERVICES

BLPAPI achieves its flexibility and wide applicability due to its service-oriented design. In the case of the request/response paradigm, BLPAPI's programming interfaces dictate messages conveying requests and partial, or final, responses. The structure of both request and response messages are specified dynamically through a service's schema and associated metadata. This means that BLPAPI's interfaces are not constrained to specific types of request/response operations. New operations and new services can be defined at any time, while preserving backward compatibility for existing applications.

Bloomberg's core service schemas are published in the BLPAPI Developer's Guide (http://open.bloomberg.com/pdf/blpapi-developers-guide.pdf), under unrestricted, "free use" licensing terms. For example, the schema for Bloomberg's reference data service ("//blp/refdata") expresses how security and fields are specified and how values are returned (or not, in the case of individual securities or fields that are found to be invalid by the server). This service is the subject of the programming example in the following section.

Moreover, "provider" applications can register to offer interactive services, whereby they receive client requests and create responses (again, based on a schema), which are sent through a special type of BLPAPI session.

C++ EXAMPLE: EQUITY INDEX CALCULATOR

The following C++ example provides a crude calculation of member stocks' contribution to the point change of an index. (A correct calculation can be seen on the Bloomberg terminal in {IMAP <GO>}.) The index and the member stocks are provided as command line arguments.

Code listing

First, necessary header files are included:

```
#include <blpapi_defs.h>
#include <blpapi_session.h>
#include <blpapi_event.h>
#include <blpapi_message.h>
#include <blpapi_element.h>
#include <blpapi_request.h>

#include <iostream>
#include <string>

using namespace BloombergLP::blpapi;
```

BLPAPI Use Case: Request/Response Paradigm

The business logic for this example is encapsulated in a class that outputs results based on data extracted data from BLPAPI messages for the relevant index and equity securities. Note that this engine requires the index price message be received first:

```
class IndexPointCalculator {
    double      d_indexPrice;

public:
    IndexPointCalculator() : d_indexPrice(0.0) { }

    void processIndexPrice(Message msg) {
        try {
            Element securityData = msg.getElement("securityData").getValueAsElement((int)0);
            Element fieldData = securityData.getElement("fieldData");
            d_indexPrice = fieldData.getElementAsFloat64("PX_LAST");
        } catch (...) {
            msg.print(std::cerr) << std::endl;
        }
    }

    void processEquityPrices(Message msg) {
        Element securities = msg.getElement("securityData");

        for (unsigned i = 0; i < securities.numValues(); ++i) {
            Element security = securities.getValueAsElement(i);

            std::string securityName = security.getElementAsString("security");
            Element fields = security.getElement("fieldData");

            try {
                double indexWeight = fields.getElementAsFloat64("INDX_WEIGHT") / 100.0;
                double change = fields.getElementAsFloat64("CHG_PCT_1D") / 100.0;

                // calculate member's contribution to net index point change
                double indexPoints = indexWeight * change * d_indexPrice;

                std::cout
                    << securityName << " contributed " << indexPoints << " points"
                    << std::endl;
            } catch (...) {
                // likely not a member of the index, or no price available; or invalid security
                std::cout << securityName << " N.A." << std::endl;
            }
        }
    }
};
```

BLPAPI Use Case: Request/Response Paradigm

The main routine begins with BLPAPI's simple session and service setup:

```
int main(int argc, char **argv)
{
    SessionOptions sessionOptions;  // use defaults

    Session session(sessionOptions);
    if (!session.start()) {
        std::cerr << "Failed to start session." << std::endl;
        return -1;
    }

    if (!session.openService("//blp/refdata")) {
        std::cout << "Failed to open //blp/refdata" << std::endl;
        return -1;
    }

    Service refData = session.getService("//blp/refdata");

    // application's calculation engine
    IndexPointCalculator calc;
```

To perform the calculation, we need to request the price of the index. The field identifier is PX_LAST, and the request is submitted as follows:

```
    // get the index name (first command-line argument), e.g. 'SPX'
     const std::string index(argv[1]);

     // make a request to get the current price of the index (e.g. 'SPX Index')
     {
         Request request = refData.createRequest("ReferenceDataRequest");

         // Add field to request
         Element fields = request.getElement("fields");
         fields.appendValue("PX_LAST");

         // Add security to request; we use Bloomberg symbology
         Element securities = request.getElement("securities");
         securities.appendValue(std::string(index + " Index").c_str());

         std::cout << "Sending index request: " << request << std::endl;
         session.sendRequest(request);
     }
```

BLPAPI Use Case: Request/Response Paradigm

The backend begins processing the request as soon as it is received. While the application could submit additional requests and they would be processed in parallel, the calculation engine requires the index be received first, so we await receipt of the response for the index. (If we wished to support out-of-order receipt, an application "correlation id" could be provided with the request to distinguish the responses.)

```
bool done = false;
while (!done) {
    Event event = session.nextEvent();

    switch (event.eventType()) {
        case Event::PARTIAL_RESPONSE:
        case Event::RESPONSE: {
            MessageIterator msgIter(event);

            while (msgIter.next()) {
                Message msg = msgIter.message();

                if (msg.asElement().hasElement("responseError")) {
                    msg.print(std::cerr) << std::endl;
                    continue;
                }

                calc.processIndexPrice(msg);
                done = (Event::RESPONSE == event.eventType());
            }
        } break;

        default: {
            // dump other events, which convey session/service status
            MessageIterator msgIter(event);
            while (msgIter.next()) {
                Message msg = msgIter.message();
                msg.print(std::cout) << std::endl;
            }
        } break;
    }
}
```

BLPAPI Use Case: Request/Response Paradigm

We use Bloomberg's data fields to obtain weights within the index and the relative price change of each equity (specified as command-line arguments), as shown below:

```
    // for each member stock, get its weight in the index and the
    // percentage change on day of the stock price
    {
        Request request = refData.createRequest("ReferenceDataRequest");

        // Add fields to request
        Element fields = request.getElement("fields");
        fields.appendValue("INDX_WEIGHT");  // % weight within specified index
        fields.appendValue("CHG_PCT_1D");   // % price change on day

        // Add stocks to request (second and subsequent command-line arguments)
        Element securities = request.getElement("securities");
        for (int i = 2; i < argc; ++i) {
            securities.appendValue(argv[i]);  // e.g. "IBM US Equity", "AAPL US Equity", etc.
        }

        // Provide index name as a calculation override
        Element overrides = request.getElement("overrides");
        Element override1 = overrides.appendElement();
        override1.setElement("fieldId", "REL_INDEX");
        override1.setElement("value", index.c_str());

        std::cout << "Sending securities request: " << request << std::endl;
        session.sendRequest(request);
    }
```

The code to process the index responses is the same, except that we call `processEquityPrices`.

Sample Output

Sample output follows:

```
Sending index request: ReferenceDataRequest = { fields[] = PX_LAST securities[] = INDU Index  }
SessionConnectionUp = {
    server = 127.0.0.1:8194
}

SessionStarted = {
}

ServiceOpened = {
    serviceName = //blp/refdata
}

Sending securities request: ReferenceDataRequest = { fields[] = INDX_WEIGHT CHG_
PCT_1D securities[] = XOM US Equity MMM US Equity IBM US Equity overrides[] = overrides =
{ fieldId = REL_INDEX value = INDU  } }

XOM US Equity contributed -13.7708 points
MMM US Equity contributed -9.61895 points
IBM US Equity contributed -9.26617 points
```

To learn more about the Bloomberg Open Market Data Initiative, visit open.bloomberg.com. Questions and comments about BLPAPI can be sent to open-tech@bloomberg.net. Questions and comments about BSYM can be sent to bsym@bloomberg.net.

open.bloomberg.com

New York	**London**	**Frankfurt**	**San Francisco**
+1 212 318 2000	**+44 20 7330 7500**	**+ 49 69 9204 1210**	**+1 415 912 2960**
Hong Kong	**Sao Paulo**	**Singapore**	**Tokyo**
+852 2977 6000	**+55 11 3048 4500**	**+65 6212 1000**	**+81 3 3201 8900**