Bloomberg ENTERPRISE PRODUCTS & SOLUTIONS

APPLICATION PORTAL

Bloomberg Terminal Connect API getting started guide

v1.18 | November 30, 2015

1 Introduction

Bloomberg's Terminal Connect allows 3rd party applications to integrate with the Bloomberg Terminal. The API allows developers to launch in-panel Terminal functions and to embed and control the behavior of Bloomberg components from within their application.

2 Getting Started

2.1 Requirements

In order to use Bloomberg's Terminal Connect from a third party application the machine the application is running on must have a licensed Bloomberg Terminal and a valid user logged in. Bloomberg's Terminal Connect is currently available for use with applications that run on the Microsoft .Net platform. Therefore, an additional requirement for use of Bloomberg's Terminal Connect is for the Microsoft.Net 3.5 framework or above to be installed on the machine(s) where the application will be run. Bloomberg's Terminal Connect can be used with both Windows Forms applications as well as WPF based applications.

2.2 Installing the SDK

1. In the Bloomberg Terminal, type SDK<Go>.
2. From the SDK menu, select **Terminal SDK**.
3. Select the latest production version of the SDK from the list and click the **Install** button.

   The SDK installs in a subdirectory called **TerminalApiSdk** under the root Bloomberg application directory which is typically **c:\blp**.

   The **c:\blp\TerminalApiSdk** directory creates the following subdirectories:
   - **lib** contains the binary assemblies that your application references
   - **help** contains help files and getting started document
   - **samples** contains sample applications to demonstrate how to use the API.

2.3 About the API

Bloomberg's Terminal Connect is an asynchronous interface as it communicates with the Bloomberg Terminal in an asynchronous manner. The API interface follows the standard .Net asynchronous programming model providing both **`BeginXXX()`** and **`EndXXX()`** calls for each operation. For example there is a **`BeginCreateComponent()`** method used to initiate the creation of a Bloomberg component as well as a corresponding **`EndCreateComponent()`** method which is called after the operation is complete and returns the created component. Note that your callback method will not be called on the UI thread which means that if you need to access UI components from your callback method you will need to marshal your code onto the UI thread. Synchronous calls are also provided for convenience in cases where it is ok to block the calling thread. For GUI applications it is not recommended to make any of the synchronous operation calls from the UI thread as this can cause your application to become unresponsive to user input.

2.4 Referencing API Assemblies

To use Bloomberg's Terminal Connect in your application you will need to add a reference to the API assemblies from your project. The API assemblies can be found in the installation directory described in the "Installing the SDK" section – typically the **c:\blp\TerminalApiSdk\lib** directory. The assembly that needs to be referenced is **`Bloomberglp.TerminalApiEx.dll`**. If you are building a Windows Forms application you may also want to reference the **`Bloomberglp.Component.Forms.UI.dll`** assembly. If you are building a WPF application you may want to reference the **`Bloomberglp.Component.Wpf.UI.dll`** assembly.

3 Using the API

The following section provides some code samples demonstrating how to perform the most common operations using Bloomberg's Terminal Connect. Please contact your Bloomberg representative to obtain the latest list of Bloomberg mnemonics and components exposed through the API.

3.1 API Registration

In order to use Bloomberg's Terminal Connect from your application you must give your application assembly a strong name and the public key used to sign your application assembly must be registered with Bloomberg. Before any calls to Bloomberg's Terminal Connect can be made, your application must call the asynchronous **`BlpApi.BeginRegister / BlpApi.EndRegister()`** or the synchronous **`BlpApi.Register() method`** each time the application is run. The registration operation will establish a session with the Bloomberg Terminal that is valid for the lifetime of your running application or as long as the current user is logged in to the Bloomberg Terminal. If any API calls are made before the initial registration operation is complete or after the API has been disconnected and not re-registered then a "Registration has not completed" exception will be thrown. Figure 1 shows an example of how to register use of the API from your application:

```
using Bloomberglp.TerminalApiEx;

//begin the asynchronous Register operation
BlpApi.BeginRegister(OnRegistrationComplete,   null);

//callback method called when the Register operation is complete
void OnRegistrationComplete(IAsyncResult ar)
{
    //complete the asynchronous operation
    BlpApi.EndRegister(ar);
}
```

Figure 1: Asynchronous API Registration

You can use Bloomberg's Terminal Connect with certain restrictions, from an application that is running on a different machine than the Bloomberg Terminal. To enable this you need to call **`BlpApi.RegisterRemote() / BlpApi.BeginRegisterRemote()`** methods which accept a machine name parameter. Please note that in order for this to work the application using Bloomberg's Terminal Connect must be running under the same user account as the Bloomberg Terminal on the other machine. The restrictions for using Bloomberg's Terminal Connect from a different machine than the Bloomberg Terminal include not being able to embed components in the remote application. **Please note that UDP ports 137 and 139 and TCP port 445 must be open in order to enable remote API usage**.

```
using Bloomberglp.TerminalApiEx;

//begin the asynchronous register operation
BlpApi.BeginRegisterRemote("BloombergTerminalMachine",  OnRegistrationComplete,  null);

//callback method called when the register operation is complete
void OnRegistrationComplete(IAsyncResult ar)
{
    //complete the asynchronous operation
    BlpApi.EndRegisterRemote(ar);
}
```

Figure 2: API Registration Specifying Machine Name

3.2 Terminal Connect SDK registration process

Applications using Bloomberg's Terminal Connect must have a strong name, and the public key used to sign the application must be registered with Bloomberg. Here are the steps to sign your application and register the public key token with Bloomberg:

1. ***Generate Key Pair***. To sign an assembly with a strong name, you must have a public/private key pair. This public and private cryptographic key pair is used during compilation to create a strong-named assembly. You can create a key pair using the Strong Name tool (sn -k filename.snk) or in Visual Studio, go to project Properties -> Signing -> check-ON 'Sign the assembly' -> select <New...>.

2. ***Sign App with a Strong Name***. In Visual Studio, go to the project Properties -> Signing. Check-ON 'Sign the assembly' and select <New...> (to generate a new public/private key pair) or browse to the .snk file generated in step 1 above. Then build solution. (For additional ways to sign assembly with a strong name, see MSDN http://msdn.microsoft.com/en-us/library/xc31ft41%28v=vs.100%29.aspx).

3. ***Provide Public Key Token to Bloomberg.*** Extract the public key token using the Strong Name tool (sn.exe), with the following command line: sn -T <assemblyPath> This will produce a 16 digit hexadecimal, e.g. "578c209daf68f957". Send this along with your full company name, and UUID that it should be registered under to Rich Berk (rberk@bloomberg.net). **The following token listed 5a19fc80c7e154c2 is from the AppPortal.snk which you won't be able to use - you need to use your own snk.**

Note: The Strong Name tool (sn.exe) can be found in one of two locations:

%ProgramFiles%\Microsoft SDKs\Windows\v7.0A\bin\

or on 64bit machine:

%ProgramFiles(x86)%\Microsoft SDKs\Windows\v7.0A\bin\

3.3 Running a Function in Panel

The following example shows how to run a Bloomberg function in-panel.

```
using Bloomberglp.TerminalApiEx;

//historical spread function
var mnemonic = "HS";
//run function in panel #1
var panel = "1";
//securities for HS function
var securities = new string[2] {"IBM US Equity", "MSFT US Equity"};
//no extra properties
List<BlpProperty> properties = null;
//no tails for this function
var tails = "";

//begin the asynchronous operation
BlpTerminal.BeginRunFunction(mnemonic, panel, securities, properties, tails,
                             OnRunFunctionComplete, null);

//callback method called when the RunFunction operation is complete
void OnRunFunctionComplete(IAsyncResult ar)
{
    //complete the asynchronous operation
    BlpTerminal.EndRunFunction(ar);
}
```

Figure 3: Running a Function in Panel

3.4 Embedding Bloomberg Components

3.4.1 Windows Forms

The following example shows how to embed a Bloomberg component in a Windows Forms application.

```
using Bloomberglp.TerminalApiEx;
using Bloomberglp.Component.Forms.UI;
using System.Windows.Forms;
using System.Threading;

//capture current UI thread context - assuming call is mode from UI thread
SynchronizationContext uiContext = SynchronizationContext.Current;

//creation properties for BCHART component
var properties = new List<BlpProperty>();
properties.Add(new BlpProperty("ChartId", 2));
properties.Add(new BlpProperty("PrimarySecurity", "MSFT US Equity"));

//begin the asynchronous CreateComponent operation
//passing in current SynchronizationContext as state
BlpTerminal.BeginCreateComponent("BCHART", properties, OnComponentCreated, uiContext);

//callback method called when the CreateComponent operation is complete
void OnComponentCreated(IAsyncResult ar)
{
    //complete asynchronous operation and get returned component instance
    var component = BlpTerminal.EndCreateComponent(ar);

    //get SynchronizationContext for UI thread passed as state to BeginCreateComponent
    var uiContext = (SynchronizationContext)ar.AsyncState;

    //post request to UI thread
    uiContext.Post(EmbedComponent,  component);
}

//when this method is called it will be on UI thread and safe to create and use UI controls
void EmbedComponent(object state)
{
    //get instance of component passed in as state
    var component = (BlpComponent)state;

    //create a WindowsForms Form
    var form = new Form();

    //create an initialize the control
    var control = new BlpControl();
    control.Initialize(component);

    //set dock style of control to fill entire form area and add to form
    control.Dock = DockStyle.Fill;
    form.Controls.Add(control);
    form.Show();
}
```

Figure 4: Embedding a Bloomberg Component in a Windows Forms Application

3.4.2 WPF

The following example shows how to embed a Bloomberg component in a WPF application.

```
using Bloomberglp.TerminalApiEx;
using Bloomberglp.Component.Wpf.UI;
using System.Windows;
using System.Windows.Controls;

//creation properties for BCHART component
var properties = new List<BlpProperty>();
properties.Add(new BlpProperty("ChartId", 2));
properties.Add(new BlpProperty("PrimarySecurity", "MSFT US Equity"));

//begin the asynchronous CreateComponent operation
BlpTerminal.BeginCreateComponent("BCHART", properties, OnComponentCreated, uiContext);

//callback method called when the CreateComponent operation is complete
void OnComponentCreated(IAsyncResult ar)
{
    //complete asynchronous operation and get returned component instance
    var component = BlpTerminal.EndCreateComponent(ar);

    //post request to UI thread using WPF Dispatcher
    Dispatcher.BeginInvoke(DispatcherPriority.Normal,   (Action)delegate
    {
        //create a WPF Window
        var window = new Window();

        //create an instance of the Bloomberg WPF control
        var control = new BlpControl();

        //initialize the visual control with the Bloomberg component instance
        control.Initialize(component);

        //set control as the content for the window
        window.Content = control;

        //show the window with embedded Bloomberg component
        window.Show();
    });
 }
```

Figure 5: Embedding a Bloomberg Component in a WPF Application

3.5 Component Properties

Certain components have one or more properties whose value(s) you can get and/or set programmatically. The names of the properties available will vary depending on the type of component. Also note that for some components the order that properties are listed when creating the component and/or setting property values is important.

Example of getting a component property value:

```
using Bloomberglp.TerminalApiEx;

//create chart component
var chartComponent = BlpTerminal.CreateComponent("BCHART");

var properties = new string[] { "ChartId" };

//get the property value(s)
var result = chartComponent.GetProperties(properties);
foreach (var prop in result)
{
    Trace.WriteLine(String.Format("Name:{0}, Value:{1}", prop.Name, prop.Value));
}
```

Figure 6: Get Component Property Value

Example of setting a component property value:

```
using Bloomberglp.TerminalApiEx;

//create chart component
var chartComponent = BlpTerminal.CreateComponent("BCHART");

var properties = new List<BlpProperty>();
properties.Add(new BlpProperty("PrimarySecurity", "AAPL US Equity"));

//set the property
chartComponent.SetProperties(properties);
```

Figure 7: Set Component Property Value

3.6 Disposing Bloomberg Components

When your application is done using a Bloomberg component it should call the **`Dispose()`** method on the **`BlpComponent`** instance in order to release resources held by the component. If your application has created multiple components you can call the asynchronous **`BlpTerminal.BeginDestroyAllComponents()`** or the synchronous **`BlpTerminal.DestroyAllComponents()`** method in order to destroy all components created during the current session.

```
using Bloomberglp.TerminalApiEx;

//begin the asynchronous create component operation
BlpTerminal.BeginCreateComponent("BCHART", OnComponentCreated, null);

//callback method called when the CreateComponent operation is complete
void OnComponentCreated(IAsyncResult ar)
{
    //complete asynchronous operation and get returned component instance
    BlpComponent component = BlpTerminal.EndCreateComponent(ar);

    //call to dispose/destroy a single component instance
    component.Dispose();

    //call to dispose/destroy all components created during the current session
    BlpTerminal.BeginDestroyAllComponents(delegate(IAsyncResult   ar)
    {
        BlpTerminal.EndDestroyAllComponents(ar);
    }, null);
 }
```

Figure 8: Destroying a Bloomberg Component

```
using Bloomberglp.TerminalApiEx;

//begin the asynchronous create component operation
BlpTerminal.BeginCreateComponent("BCHART", OnComponentCreated, null);

//callback method called when the CreateComponent operation is complete
void OnComponentCreated(IAsyncResult ar)
{
    //complete asynchronous operation and get returned component instance
    BlpComponent component = BlpTerminal.EndCreateComponent(ar);

    //call to dispose/destroy all components created during the current session
    BlpTerminal.BeginDestroyAllComponents(delegate(IAsyncResult   ar)
    {
        BlpTerminal.EndDestroyAllComponents(ar);
    }, null);
 }
```

Figure 9: Destroy All Bloomberg Components

3.7 Component Groups

Bloomberg components include the concept of “groups” where multiple components can be combined into a logical grouping in which all components share a common context for loaded security/portfolio. Changing the context for the group (or one of the components in the group) changes the context for all the components in the group. For example, if you have a quote component which displays a real time stock quote and a chart component which displays a graph of the intraday stock price of a security grouped together then when you change the quote component to display a quote for a particular security, the chart component automatically changes to display the graph for the same security as the quote component.

You can get a list of all your configured groups using the following example:

```
using Bloomberglp.TerminalApiEx;

BlpTerminal.BeginGetAllGroups((AsyncCallback)delegate(IAsyncResult       ar)
{
    foreach (var group in BlpTerminal.EndGetAllGroups(ar))
    {
        Trace.WriteLine("Group: " + group);
    }
}, null);
```

Figure 10: Get List of all Configured Component Groups

You can get the value of a group context using the following example:

```
using Bloomberglp.TerminalApiEx;

BlpTerminal.BeginGetGroupContext("GroupName", (AsyncCallback)delegate(IAsyncResult ar) {
    var group = BlpTerminal.EndGetGroupContext(ar);
    Trace.WriteLine(group.name + " = " + group.value);
}
```

Figure 11: Get Group Context Value

You can change the value of a group context using the following example:

```
using Bloomberglp.TerminalApiEx;

BlpTerminal.BeginSetGroupContext("GroupName", "MSFT US Equity",
(AsyncCallback)delegate(IAsyncResult ar) {
    BlpTerminal.EndSetGroupContext(ar);
}, null);
```

Figure 12: Set Group Context Value

You can subscribe to group context change notifications from your application in order to be notified when the context of a group changes. For example your application can automatically fill out a trade order for a security as soon as the loaded security on your in-panel Bloomberg chart is changed. The following example shows how to subscribe to group context change notifications.

```
using Bloomberglp.TerminalApiEx;

// attach event handler
BlpTerminal.GroupEvent += BlpTerminal_GroupEvent;


//event handler for group events
void BlpTerminal_GroupEvent(object sender, BlpGroupEventArgs e)
{
    if (e is BlpGroupContextChangedEventArgs)
    {
        var group = ((BlpGroupContextChangedEventArgs)e).Groups[0];
        Trace.WriteLine("New context of group " + group.Name + " is " + group.Value);
    }
}
```

Figure 13: Subscribe to Group Context Change Notifications

You can unsubscribe to group context change notifications using the following example:

```
using Bloomberglp.TerminalApiEx;

// detach event handler
BlpTerminal.GroupEvent -= BlpTerminal_GroupEvent;


//event handler for group events
void BlpTerminal_GroupEvent(object sender, BlpGroupEventArgs e)
{
}
```

Figure 14: Unsubscribe to Group Context Change Notifications

The latest version of Bloomberg's Terminal Connect includes an overloaded BlpTerminal.SetGroupContext() method that accepts an additional cookie argument. If a value is given for the cookie argument then the BlpGroupContextChangedEventArgs received in BlpTerminal.GroupEvent handler will contain the cookie value specified in the corresponding SetGroupContext() call. Applications can use this cookie to correlate the SetGroupContext() call with the asynchronous group context change event that is later received. There is also a new ExternalSource property on BlpGroupContextChangedEventArgs. This will be set to true if the group context change originated from a source outside the current application.

3.8 API Events

When you register use of the API (as discussed in section 3.1) a stateful connection with the Bloomberg Terminal is established. If this connection should be broken then the **`BlpApi.Disconnected`** event will be raised. Your application can subscribe to this event to know when this connection has been lost. Some reasons the connection may be lost is if the current user logs off the Bloomberg Terminal or the Bloomberg Terminal application is closed. If the **`BlpApi.Disconnected`** event is raised any components created by your application will no longer be valid and the application must register use of the API again (as discussed in section 3.1) before making calls to the API again or a "Registration has not completed" exception will be thrown.

```
using Bloomberglp.TerminalApiEx;

//subscribe to the disconnect event
BlpApi.Disconnected += new EventHandler(BlpApi_Disconnected);


void BlpApi_Disconnected(object sender, EventArgs e)
{
    System.Diagnostics.Trace.WriteLine("Terminal API Disconnected");
}
```

Figure 15: API Disconnected Event

3.9 Determine if User is Logged in to Bloomberg Terminal

In order to use Bloomberg's Terminal Connect, the Bloomberg Terminal application must be running and a valid user must be logged in. You can check if the Terminal is logged in using the **`BlpTerminal.IsLoggedIn()`** method. This method will return true if the Terminal is logged in or false if it is not running or not logged in.

```
using Bloomberglp.TerminalApiEx;

//check if terminal is logged in
if (BlpTerminal.IsLoggedIn())
       System.Diagnostics.Trace.WriteLine("User logged in");
```

Figure 16: Determine if User is Logged in to Bloomberg Terminal

3.10 Chart Component

The `Bloomberglp.TerminalApiEx.Charts` namespace contains classes related to use of the "BCHART" charting component. The "BCHART" charting component can be embedded in third party applications and allows programmatic interaction with the chart such as adding/removing plotted lines, setting periodicity, and changing the date range.

The following example shows how to create the "BCHART" component:

```
using Bloomberglp.TerminalApiEx;
using Bloomberglp.TerminalApiEx.Charts;

//synchronous
var chartComponent = BlpTerminal.CreateComponent("BCHART") as BlpChartComponent;

//asynchronous
BlpTerminal.BeginCreateComponent("BCHART", (AsyncCallback)delegate(IAsyncResult ar) {
    var chartComponent = BlpTerminal.EndCreateComponent(ar) as BlpChartComponent;
}, null);
```

Figure 17: Creating the "BCHART" component

The following example shows how to get and set various chart component properties:

```
//get chart lines
chartComponent.BeginGetLines((AsyncCallback)delegate(IAsyncResult ar) {
   var lines = chartComponent.EndGetLines(ar);
   foreach (var line in lines) {
       Trace.WriteLine("Line Id:" + line.Id + ", Line Security:" + line.Security);
   }
}, null);

//set chart lines
var lines = new ChartLine[] {
    new ChartLine() {Security = "MSFT US Equity", DataField = "PR005"},
    new ChartLine() {Security = "AAPL US Equity", DataField = "PR005"},
};
chartComponent.BeginSetLines(lines, (AsyncCallback)delegate(IAsyncResult ar) {
    chartComponent.EndSetLines(ar);
}, null);


//get chart periodicity
Trace.WriteLine("Periodicity: " + chartComponent.GetPeriodicity());

//set chart periodicity
chartComponent.SetPeriodicity(ChartPeriodicity.DAILY);


//get chart StartDate/EndDate
Trace.WriteLine("StartDate: " + chartComponent.GetStartDate());
Trace.WriteLine("EndDate: " + chartComponent.GetEndDate());

//set chart StartDate/EndDate
chartComponent.SetStartDate(new DateTime(2014, 1, 1));
chartComponent.SetEndDate(new DateTime(2014, 12, 31));
```

Figure 18: Get and set "BCHART" component properties

3.11 Terminal Connect Guide for Installation and Demo

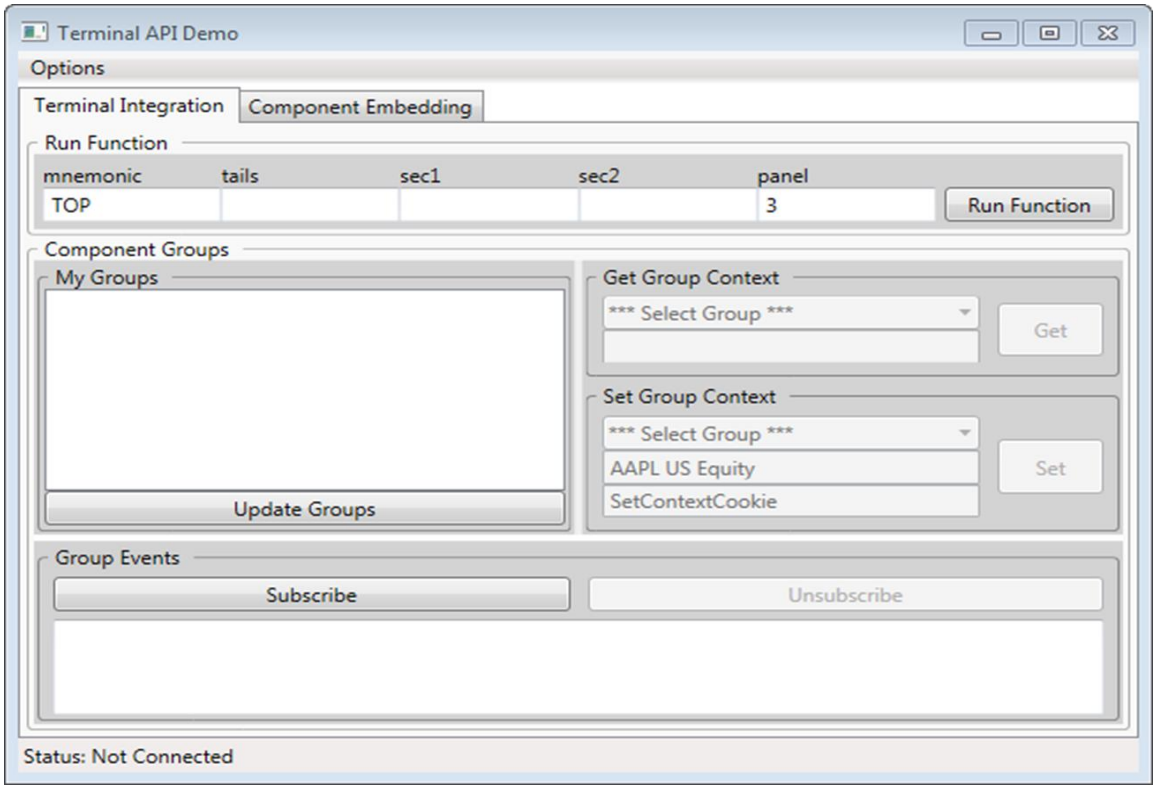
Running the Terminal API Demo App

1. In the Bloomberg Terminal, type SDK<Go>.
2. From the SDK menu, select **Terminal SDK**.
3. Select the latest production version of the SDK from the list and click the **Install** button.

   The SDK installs in a subdirectory called **TerminalApiSdk** under the root Bloomberg application directory which is typically **c:\blp**.

   The **c:\blp\TerminalApiSdk** directory creates a subdirectory called **"samples"** which contains sample applications to demonstrate how to use the API.

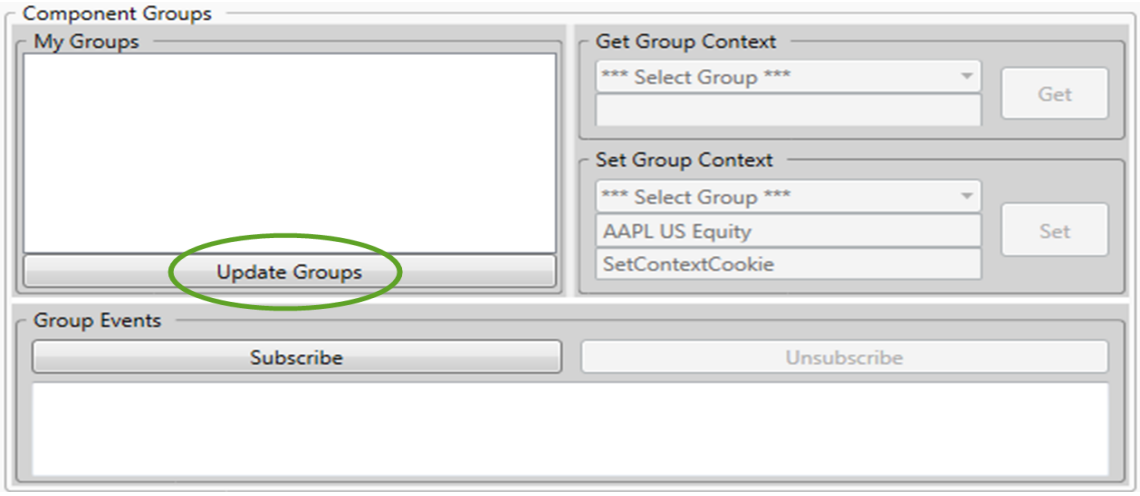
4. Select the **"samples"** directory and then choose the folder titled **"bin"**.
5. Choose the file titled **"TerminalApiSampleApp.exe"** and the following sample will be displayed.



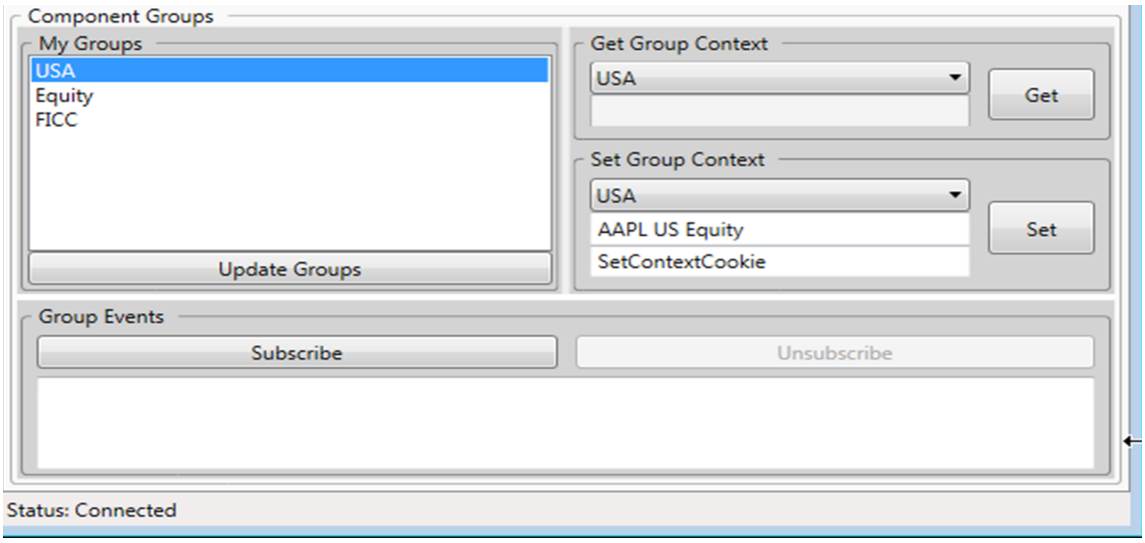
Two way Communication with Launchpad

With 2 way connectivity, Bloomberg functions can be initiated from a client's proprietary OMS, CRM, Research System or Excel. Both the Terminal and the client's Proprietary application are aware of the changes in security in the user Launchpad group. This allows a more interactive experience alongside the Bloomberg Terminal.

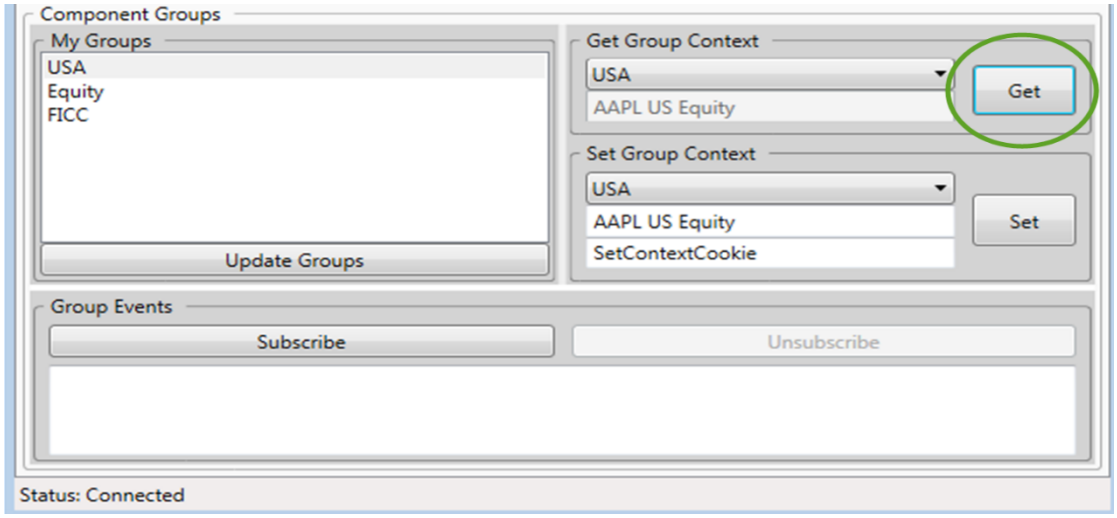
Run BLP <GO> to run an existing Launchpad group. This example will not be able to run unless a configured Launchpad group is active. Under the **Component Groups** section of the sample app click on **Update Groups**. This will display the LP groups that have been created.



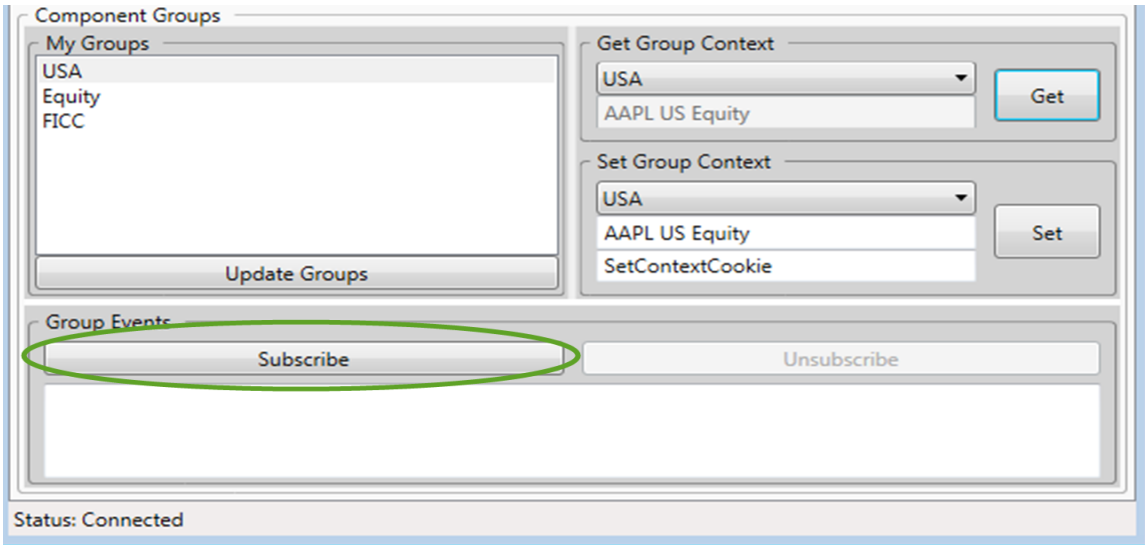
1. Pick the appropriate group under **My Groups**, clicking once on an LP Group e.g, **USA.**

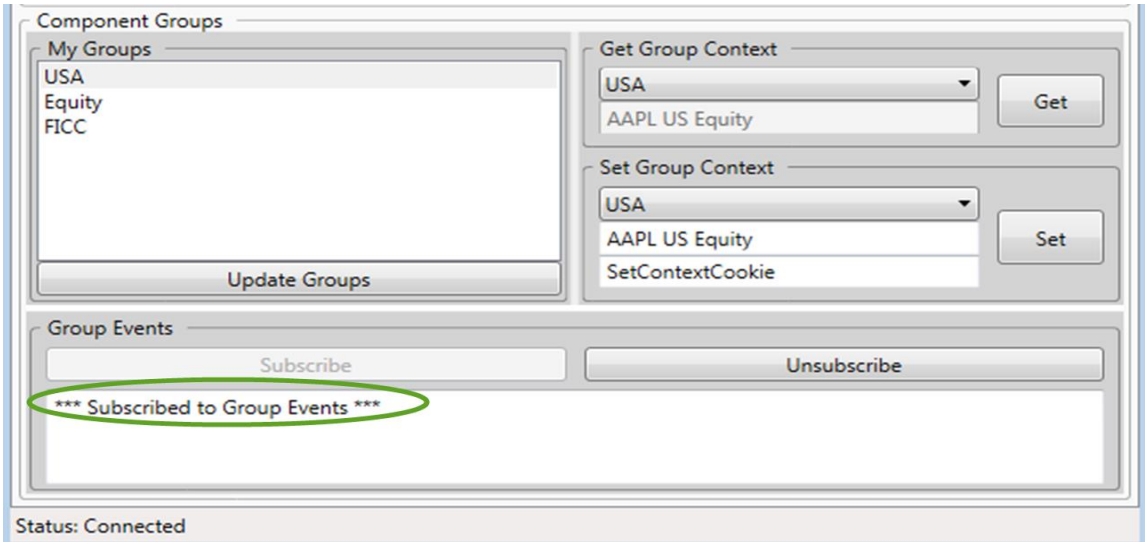


2. Under the **Get Group Context,** click **Get**. AAPL US Equity will be displayed.

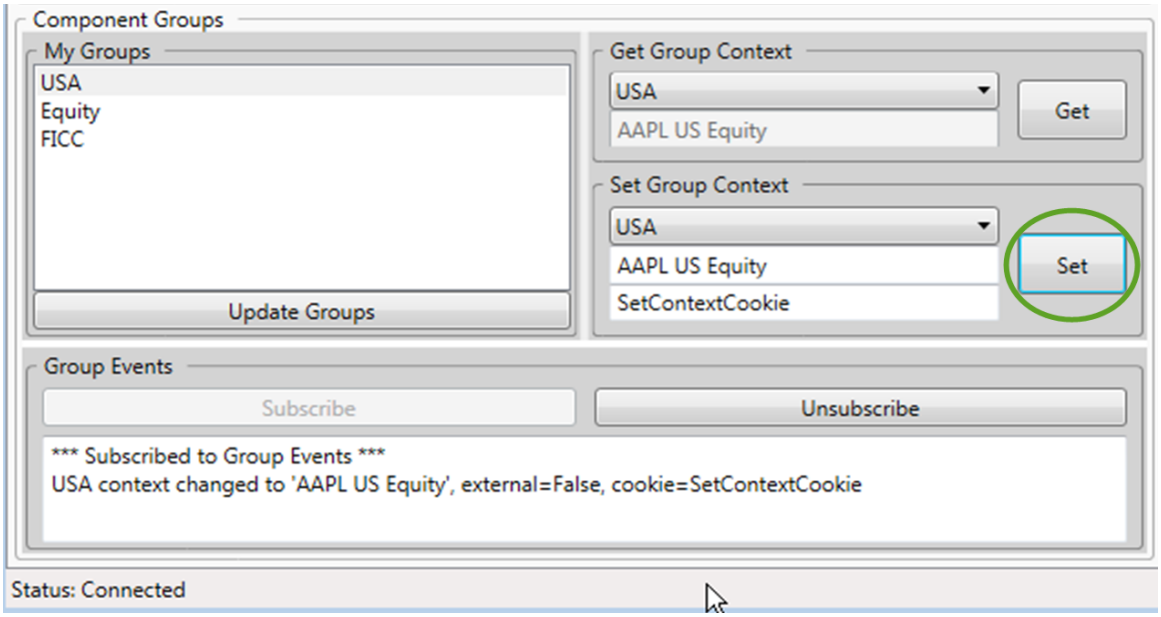


3. Under **Group Events,** click **Subscribe. ***Subscribed to Group Events*** will be displayed as the application is now aware of the active Launchpad group.





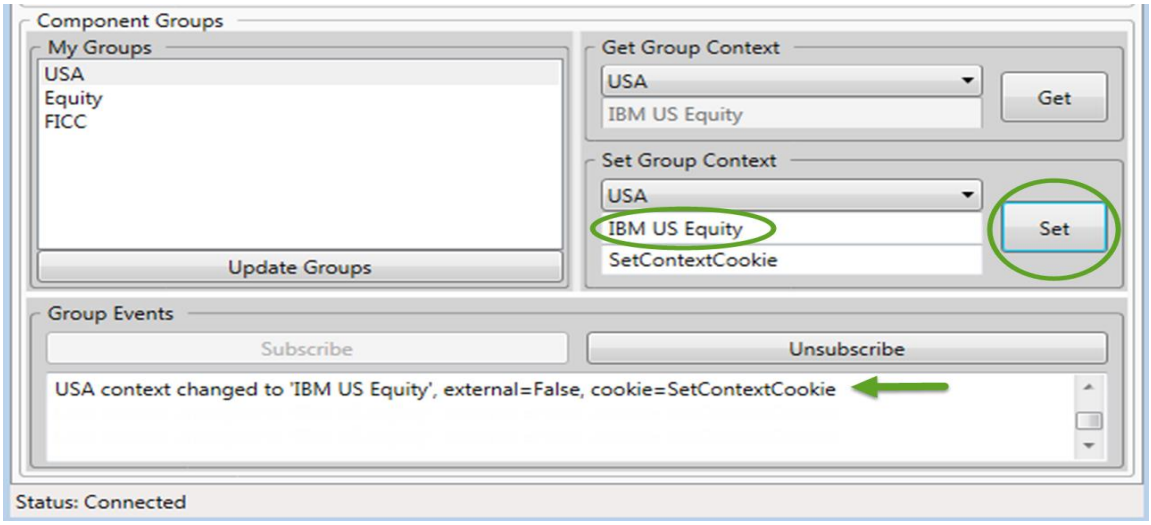
4. The Launchpad group now interacts with the application by clicking on the security in the **Set Group Context** field. AAPL US EQUITY is displayed in the field below. Click on **Set Group Context** and the Launchpad group will change to the requested security.



Setting Context

1. Using all the settings from the previous example, users can now send a security from the application to Launchpad via the **Set Group Context** field.

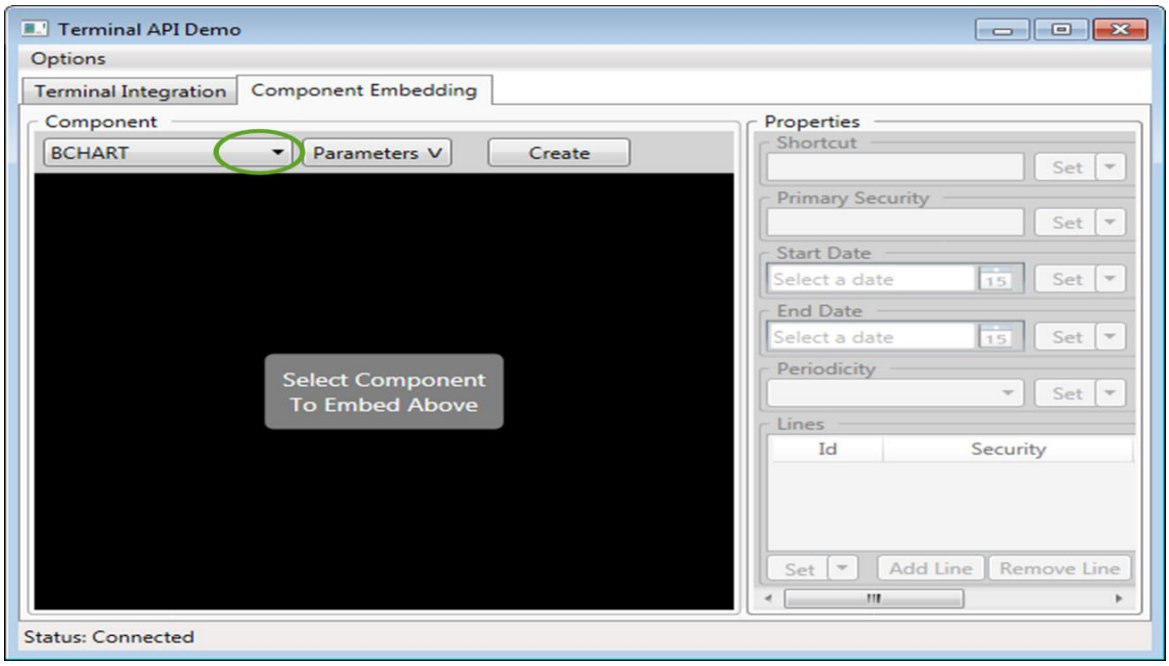
2. Click on the security in the **Set Group Context** field. Then change the security in this field to IBM US EQUITY and click on the **Set Group Context** button. The Launchpad group will change from the old security to the new one that was 'sent' from the proprietary application. The change is caught via the subscription feature in the software - see example below.



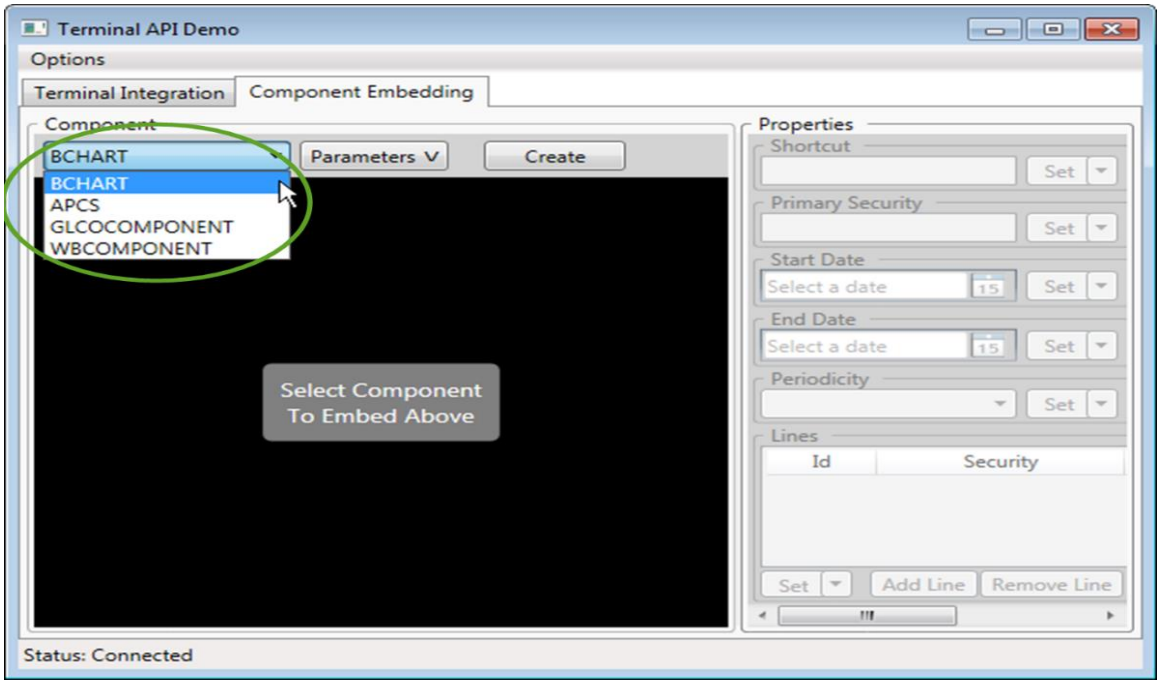
Embedding Components

1. Embedding Components allow developers to display existing Bloomberg pricing charts or graphs in their own proprietary application thus cutting down on programming time and costs - see example below.

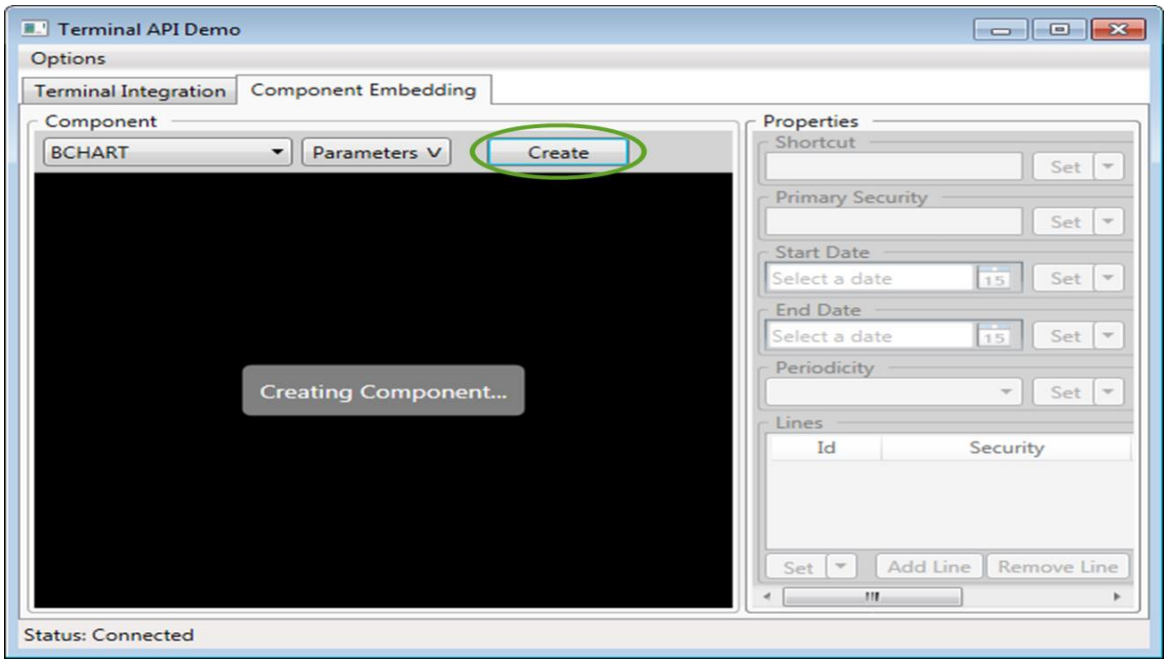
2. Click on the dropdown arrow in the **Component** section.



3. Choose one of the Component choices below.



4. Then click on the Create button and the chosen Component will be displayed for the given security.



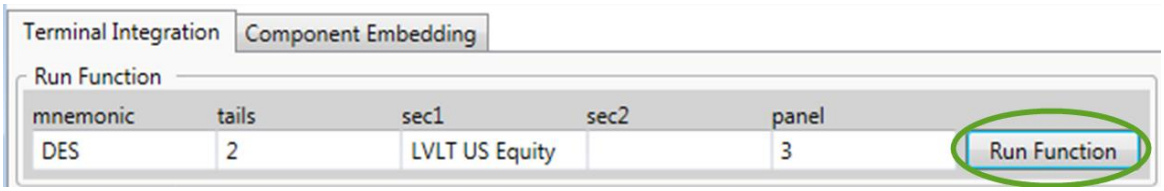
5. BCHART Component displayed below.

6. Choose the **Destroy** button to clear the selection.

7. By choosing the Component, the Launchpad group will change from the old security to the new security that was 'sent' from the application. This change is caught via the subscription feature in Terminal Connect (see detailed steps on previous pages).

Eliminating Keystrokes

1. This functionality allows users to pass commands in panel via Terminal Connect which eliminates keystrokes. In the mnemonic field type **DES**, in the tails field type **2,** in the Sec1 field type **LVLT US EQUITY** and in panel field type **3** then press the **Run Function** button.



Top function launched through example above

Currently Terminal Connect is available as:

- C# and Java SDKs available via the **SDK<GO>** function on the Bloomberg Terminal.
- In the near future an HTML5 compatible SDK will be available.
- Terminal Connect for Excel can be downloaded at **APPS BTC <GO>**.
- See **HELP MYAPP <GO>** on the Bloomberg Terminal for additional Technical documentation. For an in-depth demonstration of Terminal Connect, contact the App Portal team at terminalapi@bloomberg.net