TRADING TECHNOLOGIES

DRIVING AUTOSPREADER STRATEGY ENGINE WITH TT API 7.17.X

VERSION 7.17.X

DOCUMENT VERSION 7.17.X.DV1 3/5/14

LEGAL

Driving Autospreader Strategy Engine with TT API

Overview

TT's Autospreader Strategy Engine (Autospreader SE) is the industry-leading tool for defining, managing, and executing synthetic spreads on a single market or across multiple exchanges. Traders can use X_TRADER® to define synthetic spreads and subsequently launch and trade them on an Autospreader SE server in close proximity to exchanges and TT gateways.

Traders can also programmatically define, launch, and trade synthetic spreads on an Autospreader SE server using TT API, Trading Technologies' next-generation application programming interface (API). This paper demonstrates how to create custom applications that automate your spread trading using TT API to drive Autospreader SE.

Business Case

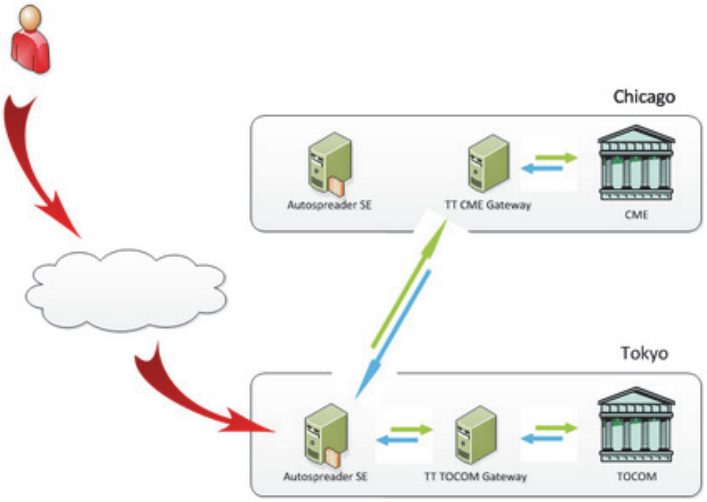
Autospreader SE is built on the highly scalable, multi-threaded architecture of TT's Strategy Engine family. This means synthetic spreads are launched, quoted, and filled in microseconds. Whether you spread contracts on a single market or multiple geographically separated markets, Autospreader SE ensures you capitalize on the superior technology and performance of TT's high-performance proximity servers.

TTNET™, TT's fully managed hosting solution, provides Autospreader SE servers in data centers regionally co-located to the exchanges in Chicago, New Jersey, São Paulo, London, Frankfurt, Singapore, Tokyo, and Sydney. These Autospreader SE servers utilize non-coalesced price feeds from TT's high-speed gateways, offering unparalleled performance.

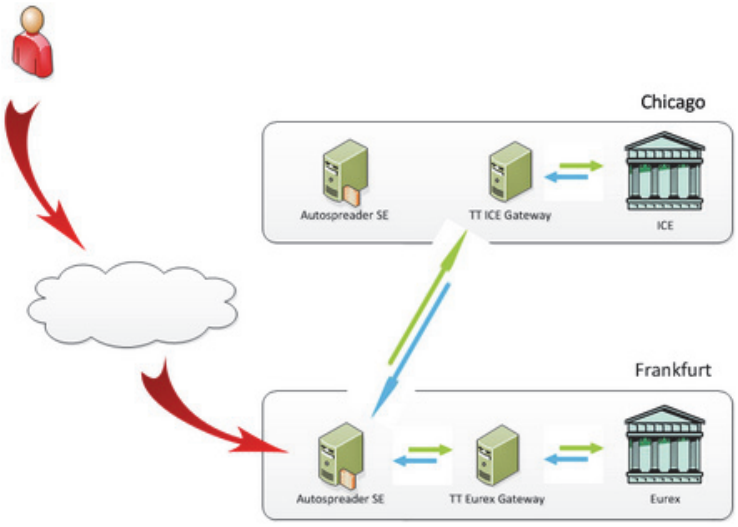
Autospreader SE is also available in environments hosted by leading financial institutions and trading organizations worldwide. Consult www.tradingtechnologies.com/customers to locate a TT distribution partner in your area.

When deciding on which Autospreader SE server to launch your spread, TT recommends that you choose one that is geographically closest to the exchange associated with the contract that you are quoting to achieve the best performance.

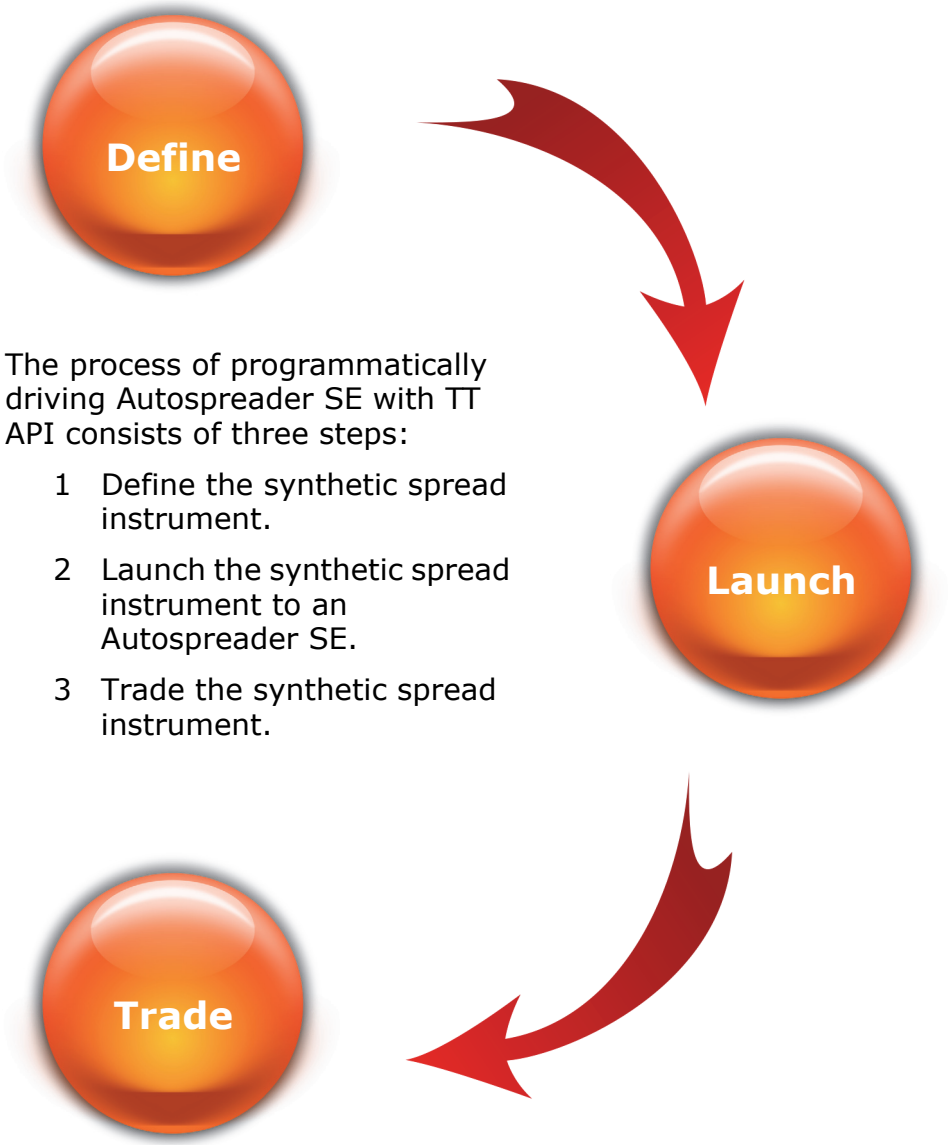
For example, if you want to spread CME gold futures against TOCOM gold futures and plan to quote the TOCOM gold futures, launching the spread to an Autospreader SE server in Tokyo would provide you with the greatest advantage.



Or if you want to spread Eurex EURO STOXX Index futures against ICE Russell Index futures and plan to quote the Eurex EURO STOXX Index futures, launching the spread to an Autospreader SE server in Frankfurt would be most advantageous.



By using TT, you can trade virtually any implied spread anywhere in the world with microsecond performance from wherever you might be.

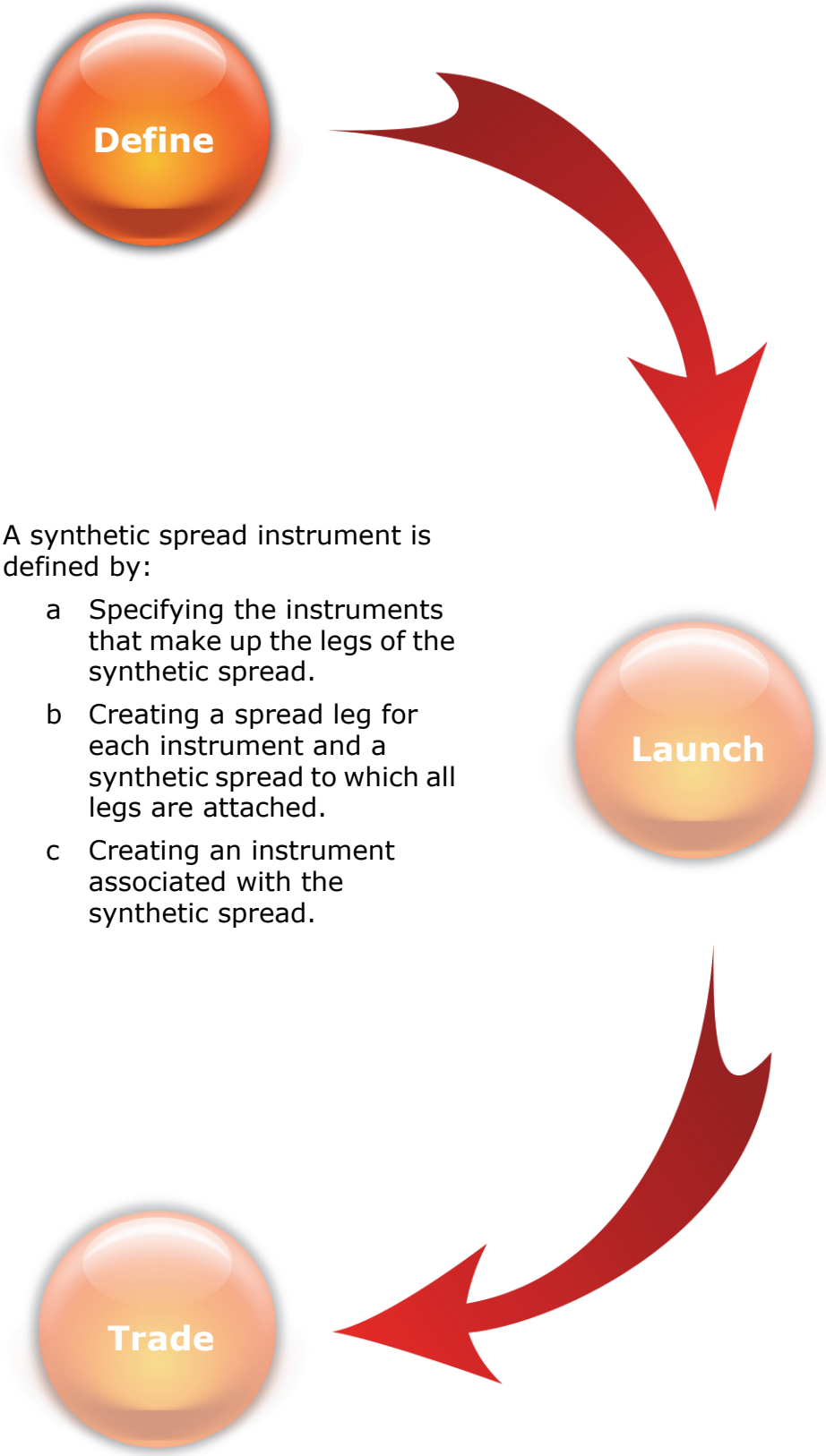


The process of programmatically driving Autospreader SE with TT API consists of three steps:

1. Define the synthetic spread instrument.
2. Launch the synthetic spread instrument to an Autospreader SE.
3. Trade the synthetic spread instrument.

Define the Synthetic Spread Instrument

Overview



A synthetic spread instrument is defined by:

a Specifying the instruments that make up the legs of the synthetic spread.

b Creating a spread leg for each instrument and a synthetic spread to which all legs are attached.

c Creating an instrument associated with the synthetic spread.

Specifying the Instruments that Make Up the Legs of the Synthetic Spread

Instruments cannot be created directly. Instead, a request must be submitted to find an instrument. The `InstrumentLookupSubscription` class is specifically designed to facilitate this request. The following C# code sample demonstrates how it is used to find an instrument using the contract specifications.

```
public void subscribeForInstrument()
{
  InstrumentLookupSubscription instrLookupReq = new
    InstrumentLookupSubscription(apiInstance.Session, Dispatcher.Current,
      new ProductKey(MarketKey.Cme, ProductType.Future, "GE"),
      "Jun14");
  instrLookupReq.Update += new
    EventHandler<InstrumentLookupSubscriptionEventArgs>(instrLookupReq_Update);
  instrLookupReq.Start();
}

public void instrLookupReq_Update(object sender,
  InstrumentLookupSubscriptionEventArgs e)
{
  if (e.Instrument != null && e.Error == null)
  {
    Console.WriteLine("Found: " + e.Instrument.Name);
  }
    else if (e.IsFinal)
  {
    Console.WriteLine("Cannot find instrument: " + e.Error.Message);
  }
}
```

Alternatively, the lookup can be done by specifying the identifier passed to TT API applications when drag-and-drop of instruments is done from X_TRADER.

```
public void Form1_DragDrop(object sender, DragEventArgs e)
{
  // If the Drop-data contains at least one contract, ...
  if (e.Data.HasInstrumentKeys())
  {
    foreach (InstrumentKey ik in e.Data.GetInstrumentKeys())
    {
      // Begin an instrument subscription
      InstrumentLookupSubscription instrLookupReq = new
        InstrumentLookupSubscription(apiInstance.Session, Dispatcher.Current, ik);
      instrLookupReq.Update += new
        EventHandler<InstrumentLookupSubscriptionEventArgs>(
          instrLookupReq_Update);
      instrLookupReq.Start();
    }
  }
}
```

Creating a Spread Leg for Each Instrument and a Synthetic Spread to Which All Legs Are Attached

The following C# code sample demonstrates the creation of two `SpreadLegDetails` objects using the `Instrument` objects provided to the method.

After `SpreadLegDetails` objects are created, the properties of each are set. A `SpreadDetails` object is then created, its properties are set, and the `SpreadLegDetails` objects are added to the `SpreadDetails` object.

```
public void createSpreadDetails(Instrument instr1, Instrument instr2,
   int orderFeedIndex1, int orderFeedIndex2)
{
   // Create the spread legs from the supplied Instrument objects
   SpreadLegDetails sld1 = new SpreadLegDetails(instr1,
      instr1.GetValidOrderFeeds()[orderFeedIndex1].ConnectionKey);
   SpreadLegDetails sld2 = new SpreadLegDetails(instr2,
      instr2.GetValidOrderFeeds()[orderFeedIndex2].ConnectionKey);

   // Define the spread leg properties
   sld1.SpreadRatio = 1;
   sld1.PriceMultiplier = 1;
   sld1.CustomerName = "<Default>";
   sld2.SpreadRatio = -1;
   sld2.PriceMultiplier = -1;
   sld2.CustomerName = "<Default>";

   // Create the SpreadDetails object
   SpreadDetails sd = new SpreadDetails();

   // Define the spread properties
   sd.Name = "My Spread";

   // Add the spread legs to the spread
   sd.Legs.Append(sld1);
   sd.Legs.Append(sld2);
}
```

Creating an Instrument Associated with the Synthetic Spread

Once the creation of the `SpreadDetails` object is complete, the final step is to create a corresponding `AutospreaderInstrument` object. This is done by creating an instance of the `CreateAutospreaderInstrumentRequest` class, passing in a reference to the `SpreadDetails` object. Since this request is asynchronous, an event handler must be provided which will be called when the creation process is complete.

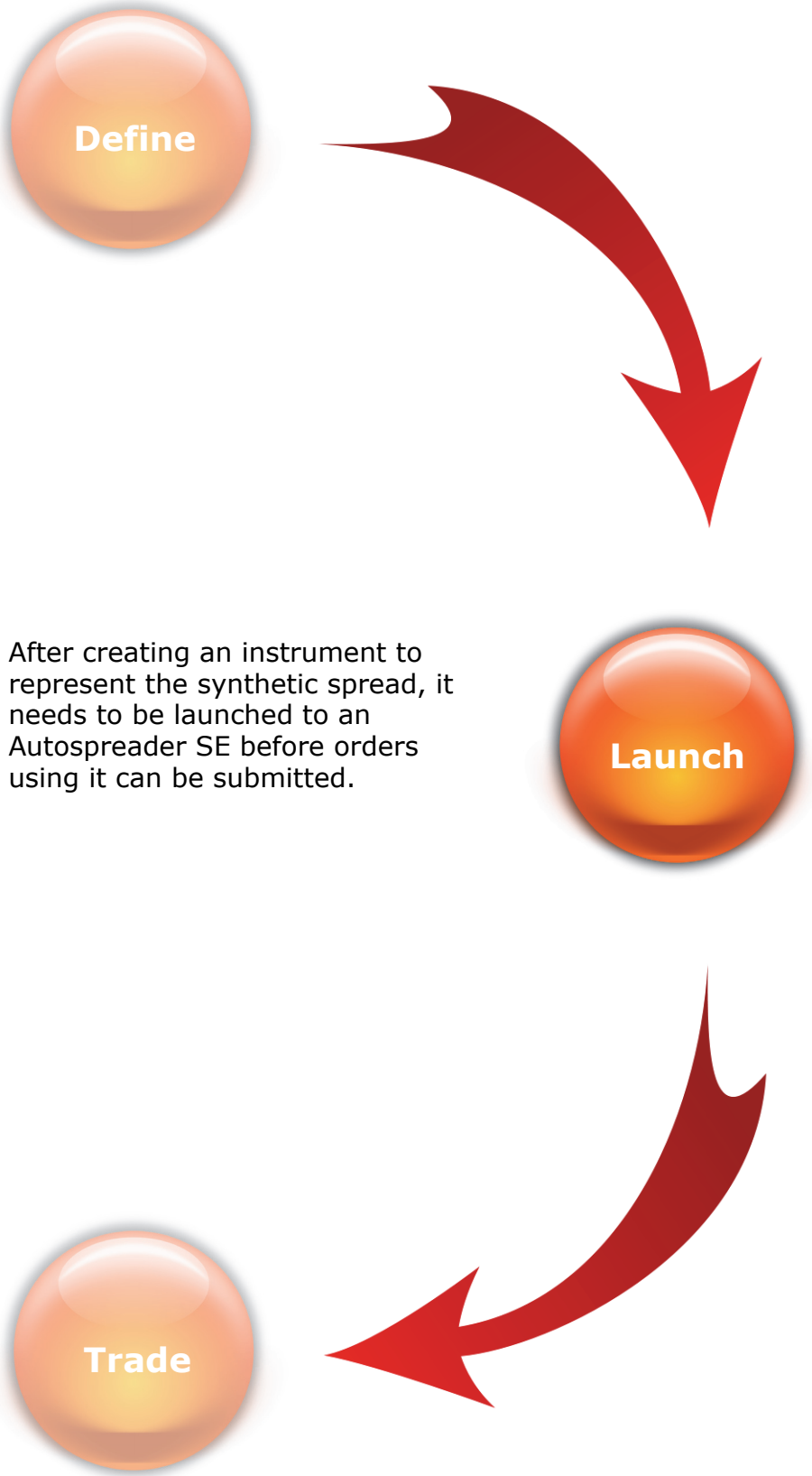
```
public void createAutospreaderInstrument(SpreadDetails sd)
{
  // Create an Instrument corresponding to the synthetic spread
  CreateAutospreaderInstrumentRequest casReq = new
    CreateAutospreaderInstrumentRequest(apiInstance.Session,
      Dispatcher.Current, sd);

  casReq.Completed += new EventHandler<CreateAutospreaderInstrumentRequestEventArgs>
    (casReq_Completed);
  casReq.Submit();
}

public void casReq_Completed(object sender,
  CreateAutospreaderInstrumentRequestEventArgs e)
{
  if (e.Error == null)
  {
    if (e.Instrument != null)
    {
      // Autospreader Instrument creation succeeded
    }
  }
  else
  {
    // Autospreader Instrument creation failed
    Console.WriteLine("Autospreader Instrument creation failed: " +
      e.Error.Message);
  }
}
```

Launch the Synthetic Spread Instrument to an Autospreader SE

Overview



After creating an instrument to represent the synthetic spread, it needs to be launched to an Autospreader SE before orders using it can be submitted.

Launching a Spread to Autospreader SE

An `AutospreaderInstrument` object must be launched to an Autospreader SE server before orders that use the synthetic instrument can be submitted. The `AutospreaderInstrument` class contains a method which returns a list of its valid order feeds. This list represents the Autospreader SE servers to which a user has access.

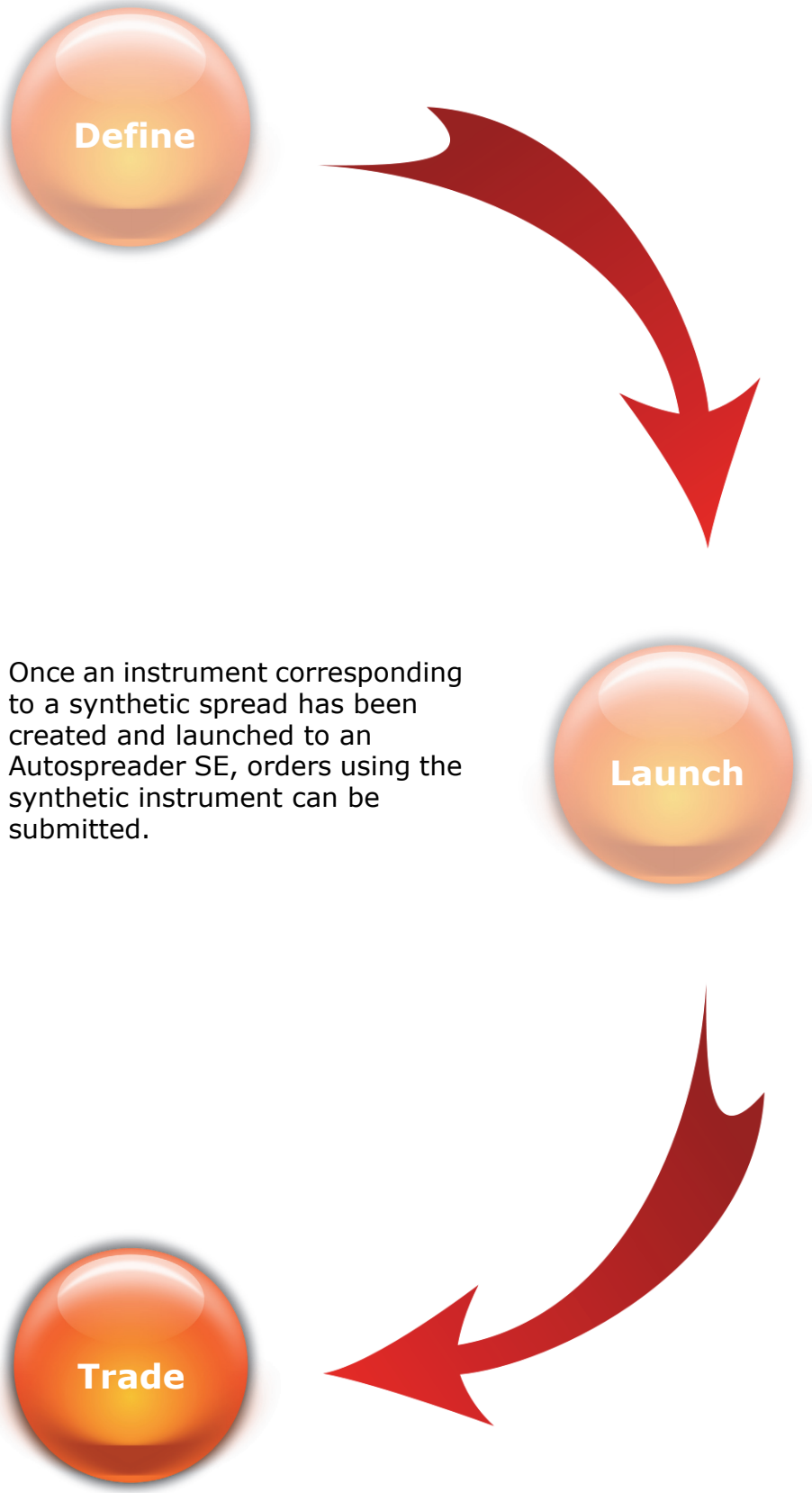
The following C# code sample demonstrates the launching of a given `AutospreaderInstrument` instance to a specific Autospreader SE server.

```
public bool launchASInst(AutospreaderInstrument instr, int orderFeedIndex1)
{
  asInstr.TradableStatusChanged += new
    EventHandler<TradableStatusChangedEventArgs>(asInstr_TradableStatusChanged);
  if (asInstr.LaunchToOrderFeed(asInstr.GetValidOrderFeeds()[orderFeedIndex])
    == LaunchReturnCode.Success)
  {
    // launch request was successful
    return true;
  }
  else
  {
    // launch request was unsuccessful
    return false;
  }
}

void Instrument_TradableStatusChanged(object sender,
  TradableStatusChangedEventArgs e)
{
  if (e.Value == true)
  {
    // launch was successful
  }
}
```

Trade the Synthetic Spread Instrument

Overview



Once an instrument corresponding to a synthetic spread has been created and launched to an Autospreader SE, orders using the synthetic instrument can be submitted.

Submitting a Synthetic Spread Order

Submission of synthetic spread orders to an Autospreader SE is done by creating an `AutospreaderSyntheticOrderProfile` instance, setting its properties, and passing it to the `SendOrder` method of the `Session` class.

The following C# code sample demonstrates the use of the `AutospreaderSyntheticOrderProfile` class to submit a synthetic spread order.

```
public void submitAutospreaderOrder(AutospreaderInstrument asInstr,
   string aseName, BuySell bs, int qty, Price p)
{
   AutospreaderSyntheticOrderProfile profile = new AutospreaderSyntheticOrderProfile(
        asInstr.GetValidGateways()[aseName], asInstr);

    profile.QuantityToWork = Quantity.FromInt(asInstr, qty);
    profile.OrderType = OrderType.Limit;
    profile.BuySell = bs;
    profile.LimitPrice = p;

    if (apiInstance.Session.SendOrder(profile))
    {
        Console.WriteLine("Order submission successful, TT order key = {0}",
            profile.SiteOrderKey);
    }
    else
    {
        Console.WriteLine("Order submission unsuccessful: {0}",
            profile.RoutingStatus.Message);
    }
}
```

You can also subscribe to prices for synthetic spread instruments using the same mechanism that is provided for exchange-native instruments.

```
public void subscribeForMarketData(Instrument instr)
{
   // Subscribe to Inside Market Data
   PriceSubscription ps = new PriceSubscription(instr, Dispatcher.Current);
   ps.Settings = new PriceSubscriptionSettings(PriceSubscriptionType.InsideMarket);
   ps.FieldsUpdated += new FieldsUpdatedEventHandler(ps_FieldsUpdated);
   ps.Start();
}

void ps_FieldsUpdated(object sender, FieldsUpdatedEventArgs e)
{
 // Inside market update received
}
```

Tracking Synthetic Spread Order Events

The `ASInstrumentTradeSubscription` class is specially designed to allow developers to track all events corresponding to a particular synthetic spread instrument. This is done by subscribing to events that will be fired by TT API as the parent synthetic spread order and the child exchange-native orders are entered, updated, filled, and canceled as demonstrated in the following C# code sample.

```
private ASInstrumentTradeSubscription asts;

public void createASTradeSubscription(AutospreaderInstrument instr)
{
   asts = new ASInstrumentTradeSubscription(apiInstance.Session,
     Dispatcher.Current, instr);

   asts.OrderAdded += new EventHandler<OrderAddedEventArgs>(asts_OrderAdded);
   asts.OrderUpdated += new EventHandler<OrderUpdatedEventArgs>(asts_OrderUpdated);
   asts.OrderFilled += new EventHandler<OrderFilledEventArgs>(asts_OrderFilled);
   asts.OrderDeleted += new EventHandler<OrderDeletedEventArgs>(asts_OrderDeleted);
   asts.Start();
}

public void asts_OrderAdded(object sender, OrderAddedEventArgs e)
{
}
public void asts_OrderUpdated(object sender, OrderUpdatedEventArgs e)
{
}
public void asts_OrderFilled(object sender, OrderFilledEventArgs e)
{
}
public void asts_OrderDeleted(object sender, OrderDeletedEventArgs e)
{
}
```

The **`ASInstrumentTradeSubscription`** class contains a list of references to the synthetic spread orders corresponding to the synthetic spread instrument. These references can be updated and subsequently resubmitted to update the orders.

```
public void updateAutospreaderOrderPrice(Price p, string ttOrderKey)
{
    AutospreaderSyntheticOrderProfile asop = (AutospreaderSyntheticOrderProfile)
        asts.Orders[ttOrderKey].GetOrderProfile();
    asop.LimitPrice = p;
    asop.Action = OrderAction.Change;

    if (!apiInstance.Session.SendOrder(asop))
    {
        Console.WriteLine("Send Order failed : {0}", asop.RoutingStatus.Message);
    }
}
```

Conclusion

So whether the Autospreader SE server is located in TTNET, TT's fully managed hosting solution, or in a company's own TT environment, TT API can be employed to programmatically drive synthetic spread orders. Give your custom strategies an edge with TT API, available at no additional cost to X_TRADER Pro users.