Introduction

To The

SmartQuant System Architecture



[1 Introduction 3](#_Toc437191737)

[1.1 Other SmartQuant Documentation 3](#_Toc437191738)

[1.2 Goals of This Document 3](#_Toc437191739)

[1.2.1 Screenshot – IDE 5](#_Toc437191740)

[1.2.2 Screenshot – DataCenter (DC) 5](#_Toc437191741)

[1.2.3 Screenshot – CATS 6](#_Toc437191742)

[1.3 Intended Audience 7](#_Toc437191743)

[1.4 For Novice Traders 7](#_Toc437191744)

[1.5 For Skilled Individual Traders 11](#_Toc437191745)

[1.6 For Hedge Fund Groups 12](#_Toc437191746)

[1.7 For Institutional Trading Groups 13](#_Toc437191747)

[2 Typical System Architectures and Data Flows 14](#_Toc437191748)

[2.1 Typical Tasks for Strategy Development 15](#_Toc437191749)

[2.2 Diagram – SmartQuant Framework API 15](#_Toc437191750)

[2.3 Diagram – SmartQuant Database Architecture 16](#_Toc437191751)

[2.4 Diagram – SmartQuant System Architecture 17](#_Toc437191752)

[2.5 Diagram – DataCenter Importing Historical Market Data 18](#_Toc437191753)

[2.6 Diagram – IDE Importing Historical Market Data 20](#_Toc437191754)

[2.7 Diagram – IDE Importing Historical Data from ESignal 21](#_Toc437191755)

[2.8 Diagram – DataCenter Capturing Real Time Market Data 22](#_Toc437191756)

[2.9 Diagram – IDE Capturing Real Time Data from ESignal or IB 22](#_Toc437191757)

[2.10 Diagram – IDE Importing Market Data from DataCenter 24](#_Toc437191758)

[2.11 Diagram – IDE Back Testing a Strategy 25](#_Toc437191759)

[2.12 Diagram – IDE Live Trading a Strategy 26](#_Toc437191760)

[2.13 Diagram – CATS Live Trading a Strategy 27](#_Toc437191761)

[3 SmartQuant System Building Blocks 28](#_Toc437191762)

[3.1 SmartQuant Database Design 28](#_Toc437191763)

[3.2 SmartQuant C# Framework Assemblies 29](#_Toc437191764)

[3.3 The DC (Data Center) Application 31](#_Toc437191765)

[3.4 The IDE (Integrated Development Environment) Application 33](#_Toc437191766)

[3.4.1 Strategy Framework Choices 34](#_Toc437191767)

[3.4.2 The ATS Strategy Framework 34](#_Toc437191768)

[3.4.3 The MetaStrategy Framework 35](#_Toc437191769)

[3.4.4 Project Level and Strategy Level Components 37](#_Toc437191770)

[3.4.5 Normal and Cross Components 37](#_Toc437191771)

[3.4.6 Unknown Components 38](#_Toc437191772)

[3.5 The CATS (Canned Automated Trading System) Application 39](#_Toc437191773)

[4 External Providers to the SmartQuant System 40](#_Toc437191774)

[4.1 Instrument Providers and Configuration 41](#_Toc437191775)

[4.2 Historical Market Data Providers 42](#_Toc437191776)

[4.3 Real Time Market Data Providers 43](#_Toc437191777)

[4.4 Trade Execution Providers (Brokers) 43](#_Toc437191778)

[5 Summary and Conclusion 44](#_Toc437191779)

# Introduction

SmartQuant is a system for designing and executing computerized quantitative trading strategies. It is a very flexible and powerful system, so it contains many different software components and an associated degree of complexity.

For new users, identifying the various pieces of the system and understanding their respective functions can be a confusing task.

This document helps new users to identify the major pieces of the SmartQuant system, and how to arrange the system components to accomplish typical user goals such as importing historical market data, experimenting with existing strategies, or developing and executing a custom automated trading strategy.

## Other SmartQuant Documentation

Here is a list of other SmartQuant documents. If you’re a new SmartQuant user, you might consider reading the documentation in the following order to maximize your learning speed.

1. **System Architecture Manual** (this one). This manual introduces readers to the system terminology, to major system components, and to various system configurations for solving typical user goals.
2. **Getting Started Manual**. This manual shows users how to use the IDE to configure and execute a simulation of a simple strategy on predefined market data stored in the SmartQuant database. Users can choose a financial instrument, run a strategy on some historical market data for that instrument, and then inspect the simulation results.
3. **User’s Guide Manual**. This manual digs deeper into both the IDE and Framework API, and shows users how to construct and execute user-defined strategies on user-specified financial instruments. A variety of common tasks and procedures are explained.
4. **Code Recipes Manual**. This manual contains code examples that show how to accomplish common user goals with the SmartQuant framework API.

## Goals of This Document

This document describes the high-level architecture of the SmartQuant framework system for developing computerized quantitative trading strategies. There are 4 major SmartQuant products in the SmartQuant system:

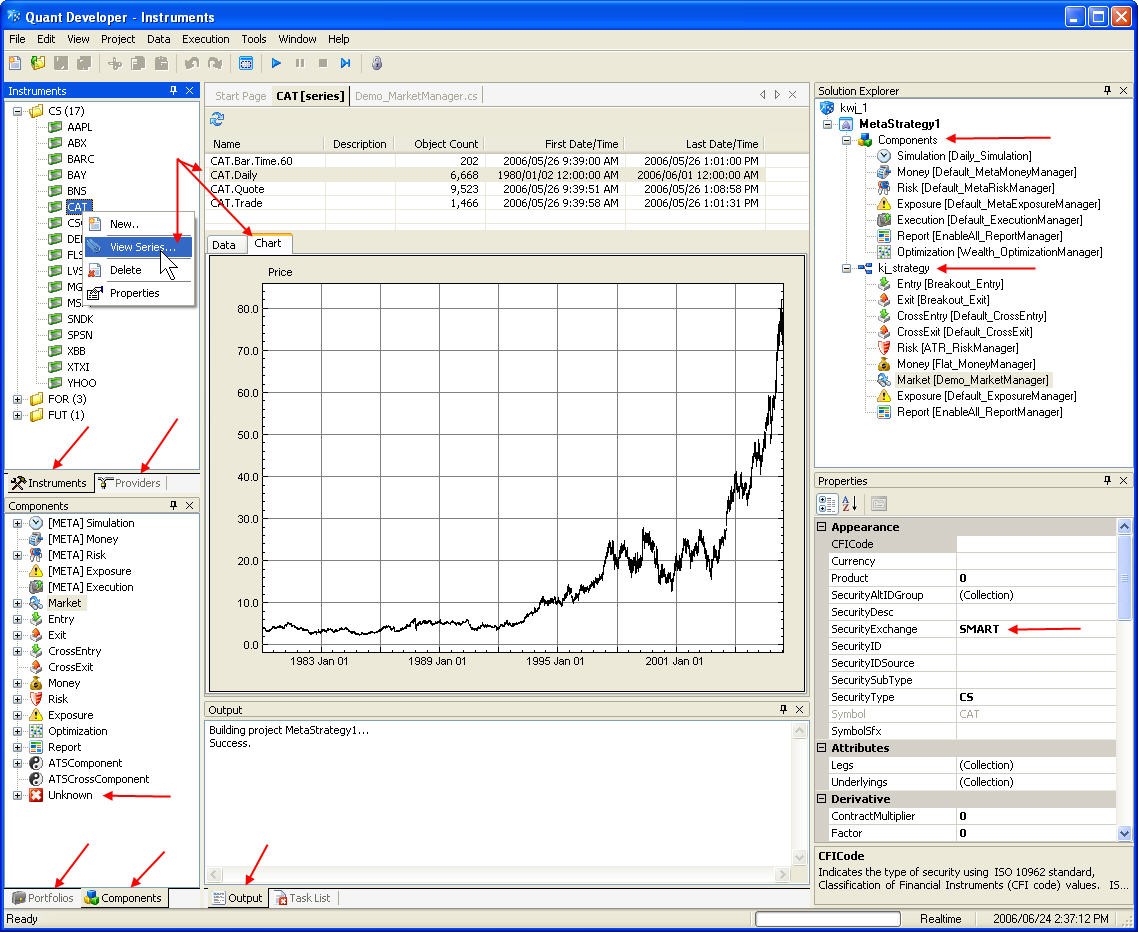
1. **SmartQuant C# Framework API,** a set of .NET assemblies that provide an industrialstrength API (Application Programming Interface) for implementing automated quantitative trading systems. The framework API is the core of the SmartQuant system. All other SmartQuant products are built on top of this API. You can also build your own (completely custom) applications on top of this API, since you can use the API just like any other API in the .NET environment. Any .NET language (such as Basic, C++) can be used with the framework assemblies.
2. **QuantDeveloper IDE (IDE)**, an integrated development environment for developing, testing, and optimizing quantitative trading strategies. This is the place where you work to create and test your own trading strategies.
3. **CATS (CATS)**, a Canned Automated Trading Strategy application for executing completed trading strategies in a standalone, non-IDE execution environment. This application can accept precompiled trading strategy components that have been exported from the IDE, and then execute the strategies in a more-or-less unattended manner.
4. **DataCenter (DC)**, an interactive software application for capturing dynamic market data from data providers (24x7, with no reboots) in a high-performance, centralized database. DataCenter can then provide historical market data for use by individuals who want to browse and download specific market data into their local QuantDeveloper IDE databases. The DataCenter can also import market data from a variety of other static sources, such as text files, CSV files, NYSE TAQ (Trade And Quote) and TAQ2 CDROM database formats, and can execute C# data maintenance programs of arbitrary complexity, to help make your data maintenance tasks considerably easier. In summary, DataCenter can combine hundreds of gigabytes data from multiple different sources into a single centralized, easily accessible, easily maintained database.

The three SmartQuant application programs (IDE, CATS, DC) are priced separately. All of them include the framework API. You do not need all three applications in order to write and run your own computerized trading strategy. You only need the IDE at a minimum, and can add the DataCenter and CATS components later, as required.

### Screenshot – IDE

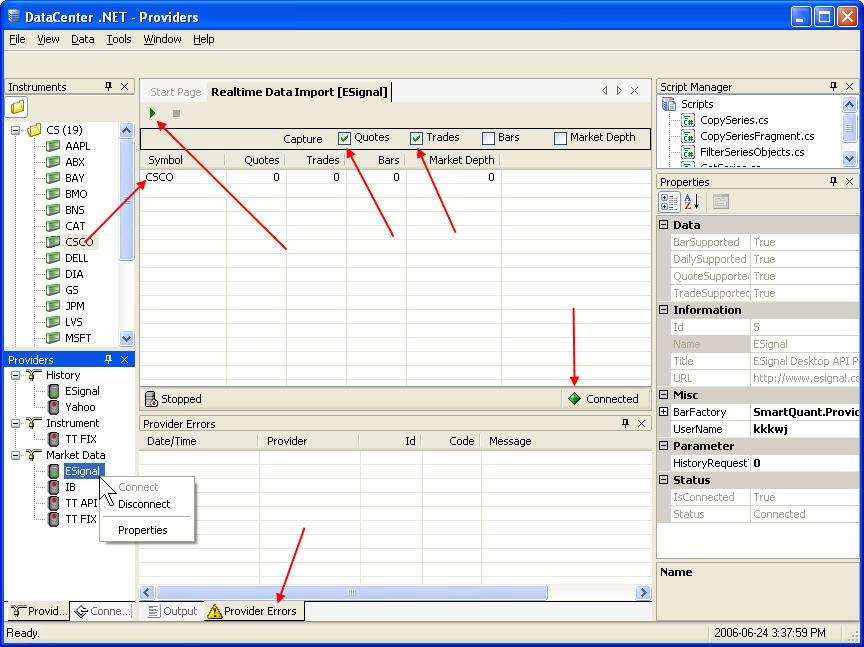
Here are screenshots of the IDE, DC, and CATS applications. More explanations of these systems are provided later in this document.

First, here is the IDE. (Zoom your document viewer for a larger view of the screenshots.) The little red arrows highlight features that are described later in this document. The little red arrows merely point at interesting screenshot features—there is no text annotation associated with any of the little red arrows.



### Screenshot – DataCenter (DC)

Next, here is a screenshot of the DataCenter (DC). As before, the red arrows highlight features that are described later in this document.



In the bottom left of the screenshot above is the Providers panel, which shows a list of historical market data providers (ESignal and Yahoo shown), Instrument definition providers, and real time market data providers (ESignal, Interactive Brokers, Trading Technologies API and FIX interfaces shown).

SmartQuant supports other providers too, but they are not shown in the screenshot above because the providers have not been enabled in the DataCenter configuration settings. To make other supported providers show up in the Providers panel on the SmartQuant IDE, Datacenter, or CATS screens, simply enable the desired providers on the Tools / Configuration / Providers tab.

### Screenshot – CATS

Finally, here is a picture of the CATS automated trading system. The IDE can export an ATS strategy component in the form of a “canned plugin” that can be loaded into the CATS system framework. Later, you can choose among various strategies on the CATS strategy menu (as shown below) to run a strategy that interests you. The picture below shows the CATS chart screen after CATS has run a canned moving average crossover strategy on a Cisco data series.



## Intended Audience

This document is intended as an orientation document for new users of SmartQuant products, so it only covers basic automated trading system concepts, large architectural building blocks, and the most typical user trading scenarios. More advanced information can be found in other SmartQuant documentation.

## For Novice Traders

The SmartQuant system is probably quite advanced for novice market traders, because it contains software representations of many kinds of advanced trading *instruments*, trading orders, and *technical indicators* that will not be familiar to novice traders or non-trader programmers. A financial *instrument* is something that carries financial value, such a stock, a bond, a futures contract, or a stock option. A *technical indicator* is a mathematical, formula-calculated value that indicates some interesting property of the underlying financial instrument. For example, one technical indicator is a simple 10-day moving average of daily closing price.

But since this document must have a simple starting point for all audiences, here are some basic ideas about trading and computerized trading systems in general.

**Charting Software versus Computerized Trading Systems:** It is worthwhile to start by describing the (big) differences between advanced stock charting programs and the SmartQuant quantitative trading system framework. Generally speaking, advanced charting programs are selfcontained, non-extensible applications that make it easy for people to create and apply simple-tomedium trading strategies to market data. Trading strategies in charting programs are always expressed in the charting program language, which is usually some kind of scripting language that has convenient expressions for trade and quote bars, moving averages, and technical indicators.

In contrast, SmartQuant is a C# programming environment that is modeled after Microsoft Visual Studio. All trading strategies and data maintenance scripts are written in C#, which is a fullfeatured programming language that is not as limited as the typical scripting languages that are used in charting programs). C# is a general purpose language, whereas chart scripting languages are primarily designed with chart program operations in mind. In contrast, SmartQuant enables you to create and run extensible, industrial-strength, fully computerized, quantitative trading systems of arbitrary complexity.

The most advanced stock charting programs can do four major things: (1) read live or historical market data, (2) chart and display the data along with various technical indicators, and (3) execute your favorite trading strategy rules (in either live or back testing modes) to show you when trades should occur, and (4) report various statistics on profits made, distribution of winning and losing trades, and other strategy statistics. These standalone programs are a user-friendly way to get computers to help you to execute trading strategies. Perhaps their biggest limitation for advanced users is that trading strategies must be composed from various built-in indicators and rule logic expressions that are resident in the program. For example, there is no hope of having such programs call one of your own custom code libraries written in a full blown development environment such as Microsoft Visual studio. And there is no hope of programming heavily customized indicators, instrument correlation matrices, or advanced financial decision making logic in the languages provided by such stock charting programs.

In contrast, SmartQuant is a C# programming environment and application framework (IDE, sample programs, libraries) for building industrial-strength computerized quantitative trading applications. While SmartQuant does not have the user-friendliness and convenience of good stock charting programs, it has far more functional depth for people who want to implement their own customized trading applications with custom GUI interfaces, like hedge funds or institutional trading departments might do.

**Trading System Goals:** The main goal of a computerized trading system is to interact with computerized broker APIs (Application Programming Interfaces) to accomplish goals that are important to humans. Not all human goals are concerned with the obvious goal of making money in the financial market. Some other possible goals are to write software applications to reduce numeric calculation complexity for traders, or to reduce the effort required to balance offsetting trade positions for traders, or to make it easier to hedge positions using trading futures, or to bring more analytical power to bear on the problem of analyzing correlations between price movements of various financial instruments, or even to be an interesting hobby for individual programmer / traders. (Having said this, of course, all of these goals are within the general scope of trading in the market to increase profits or to reduce the risk of losses.)

Ideally, a computerized trading system should demand a minimum of work from human operators, provide a maximum of accuracy in trading and record keeping, provide a maximum of risk reduction where possible, and finally, to earn a maximum amount of money (or at least not lose it) in a minimum amount of time.

**Good Trading Strategies are Difficult to Create:** But make no mistake—it is very difficult to create an automated trading system that delivers all of these qualities. The market itself is so complex that it can deliver large variations in price volatility, trend directions, random price movements, and timing of various events. This enormous variation in dimensions and magnitudes makes it very difficult for a computer program to understand what is really going on, and to do something useful in response to what the market is doing.

Computer programs are generally good at doing the same simple things over and over again at high speeds, but they are not widely known for their inherent ability to successfully analyze and respond to complex market behaviors. As you can appreciate, since it is even difficult for skilled humans with years of experience to consistently make money in the financial market, writing a computerized trading system that has an equivalent level of skill and that can produce equivalent financial profits is not a simple matter.

**Complexity of Choices in Strategies:** For computer programmers, the number of binary alternatives involved in trading is interesting. You can enter the market or exit the market. You can use equity ownership or not; options or not; futures or not. You can trade stocks or not; fixed incomes or not; currencies or not. You can enter a trade, or not. When you enter, you can buy or sell. The next price goes up, or down. There is a trend, or not. The trend (if there is one) goes up, or down. The trend (if there is one) starts, or stops. Either the general market direction is strongly correlated with your financial instrument, or it is not. The list of interesting choices to make in a trading program can be a very long one!

**Hardware Resources:** Turning now to technical resources and methods, you need at least one computer to create and run a computerized trading system. For example, all SmartQuant system components can run side by side on a single computer. But a word to the wise—you might want to use multiple computers to isolate your development machine (which you might have to reboot during market hours) from your “production” machine. Separation of development machines from production machines is a good idea because you don’t want to reboot production machines that are in the middle of executing advanced trades that might lose you money if the trades are not entered or exited properly, in the right sequence, precisely at the right times.

**Algorithms:** The algorithms used by computerized trading systems all involve the analysis of incoming market information so that the system can decide what it should trade. The incoming information would almost certainly include information about particular financial instruments, such as their recent price and volume information. Incoming information might also include information about the market itself, such as the NYSE exchange volume, market volatility indicators, sector index movements, advance/decline ratios, put/call ratios, sentiment and contrarian indicators, or other information that is either generated by, or managed by, brokerages or stock exchanges.

**Technical Indicators:** The technical indicators used by trading systems all help the computer to analyze the current market in order to decide how to trade one or more financial instruments. Many tens of technical indicators have been invented for the analysis of financial instruments and markets. Indicators can be used by themselves, in combinations with other indicators, in particular market situations (such as trend vs. no trend), or in any other way that you can imagine. There are monetary economic indicators (that measure price or dollar movements), sentiment indicators (that measure the sentiment of people in the trading industry), momentum indicators (that measure trends in price, volume), leading indicators (that try to predict price movements), lagging indicators (that follow past price movements), and other kinds of indicators that are useful in particular situations.

**Reference Books:** Probably hundreds of books have been written about technical analysis of the stock, option, futures, currency, and commodities markets. Some of the indicators in SmartQuant products were programmed using formulas from the popular book *Technical Analysis from A to Z*, by Steven Achelis. Another reputable book is *Technical Analysis of the Financial Markets* by John Murphy. One thing is for sure—if you’re going to write yourself a computerized trading application, then you’ll certainly have to know how technical indicators work.

**Built-In Technical Indicators:** The SmartQuant IDE ships with about 20 indicators preprogrammed for you, so that you can use them in your own trading strategies. For example, the set of preprogrammed indicators includes leading indicators such as Stochastics indicators, as well as lagging indicators such as Simple and Exponential Moving Averages.

**Strategy Development Process:**  When you develop a new trading strategy, you normally follow a strategy development process that is much like a traditional iterative software development process. That is, you code a new strategy idea or improvement, test it out (“backtest” it) on historical market data, refine the code, test again, and iterate until you are happy with the results. Then you switch market data, and test the strategy algorithm on a different financial instrument (fast growth stock versus slow moving market index), on a different price range (cheap stock or expensive stock), on a different market period (trend or not), or on other meaningful test cases.

**Conventional Strategy Wisdom:** Conventional wisdom is that simpler strategies (especially those based on moving average crossover indicators) tend to work better under a wider variety of conditions than do complex strategies that are based on complex technical indicators. One possible reason for this is that combinations of complex, sensitive indicators might be too hard to control and manage in volatile market conditions. The indicators might be perturbed so much that they become useless, and give off unwanted trading signals at the worst possible times. A second aspect of strategy wisdom is that you should not tune (optimize) your strategy parameters too closely to any particular set of back testing market data. If you do, then your strategy will work well on your test data, but might work much *less* well on fresh new market data.

**MACD Moving Average Method:** It is worth saying that the popular MACD moving average crossover method is one of the most predictable strategies that you can use (MACD means Moving Average Convergence Divergence). The MACD method will certainly put you on the right side of market trends that are long enough and strong enough and steep enough in comparison to the length of the moving averages that you choose. So for novice strategy developers, the moving average crossover technique is a good place to start. The SmartQuant system documentation includes code for a simple MACD crossover trading strategy, as well as sample historical market data. If you run the MACD example on some historical market data that contains good trends, the strategy will show a profit.

But unfortunately, the MACD method does not work well in all market conditions. When the market is flat and trendless, MACD will throw off many useless and unwanted trading signals as the two moving averages cross back and forth on any price movement of consequential size. In a practical sense, the constant crossovers that occur will “whipsaw” your position from long to short to long again, with lots of commissions but no profit in between flips.

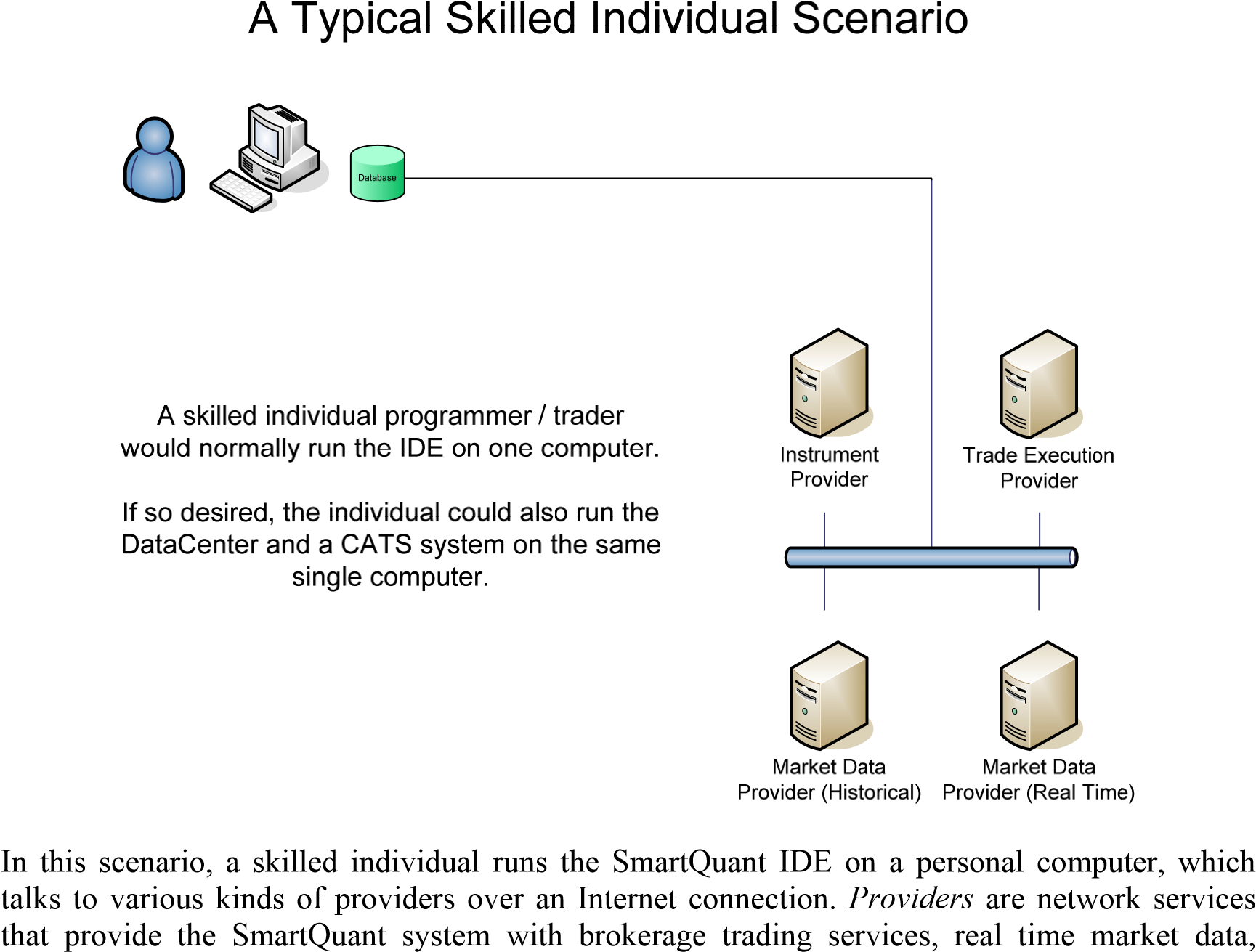
**Summary of Concepts:** Here is a list of trading system concepts we discussed above:

* the goals of a trading system,
* the difficulty of creating a good strategy for all market conditions,
* the computer hardware required,
* the general algorithm that is used,
* the technical indicators that can be used,
* two good technical analysis reference books that can be used,
* the iterative strategy development process that is used, and
* a mention of the popular MACD moving average indicator strategy.

Now that you have a basic understanding of computerized trading system concepts, we can now move on to introducing the main SmartQuant system, and common configurations of components for solving typical trading system goals for skilled individuals, for small hedge fund operations, and for large institutional trading groups.

## For Skilled Individual Traders

In this document, we define *skilled individuals* as people who understand both trading and programming, and who want to improve their computerized trading infrastructure so they can trade like a hedge fund. You don’t have to be either rich or a genius in order to run a computerized trading system just like a real hedge fund—all you need is a personal computer, some patience, and some knowledge about trading and programming. Here is a picture of how a skilled individual might configure a SmartQuant system to do computerized trading.



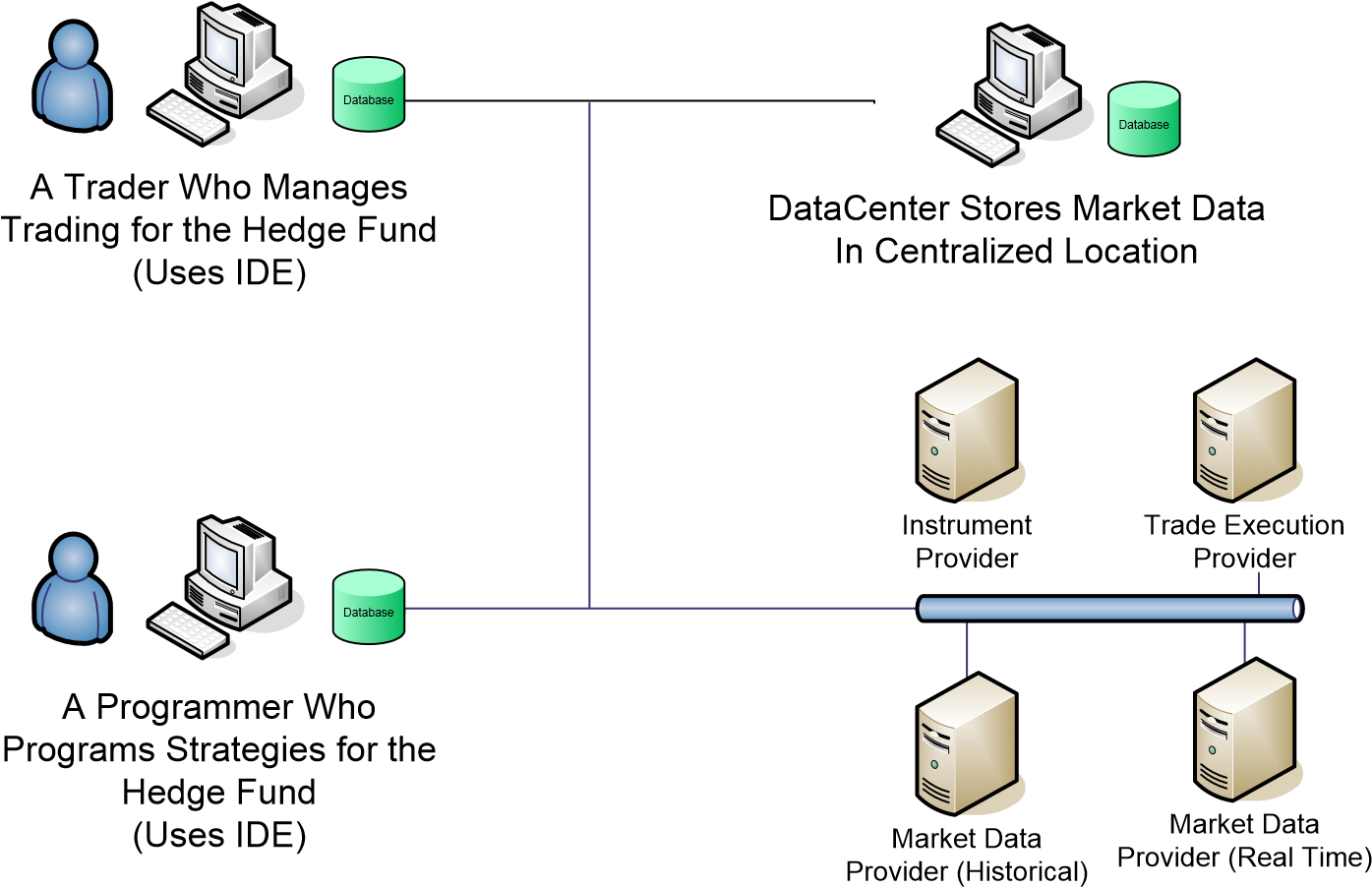
historical market data, or in some cases, with automated lists of financial instruments available from a brokerage provider. Here are some examples of providers:

**Yahoo.com** is a popular provider for historical daily prices for US stocks, for the past year. **ESignal.com** is a provider of real time and historical market data, delivered through an ESignal application program that displays data, news, and charts. **IB (Interactive Brokers)** is a brokerage service that provides trade execution services through a TWS (Traders Workstation) application for trading and charting.  **TT (Trading Technologies)** is a brokerage service that provides trade execution services through an X\_TRADER application program for trading and charting.

## For Hedge Fund Groups

A typical startup hedge fund organization is composed of multiple people who have different responsibilities, and of multiple computers that have different responsibilities. Here is a picture of how SmartQuant might be used by a small startup hedge fund. The picture shows two computers running the SmartQuant IDE, plus a third computer running a SmartQuant DataCenter application that captures real time and historical market data in a central database for later use by IDE strategy developers.

A Typical Small Hedge Fund Scenario

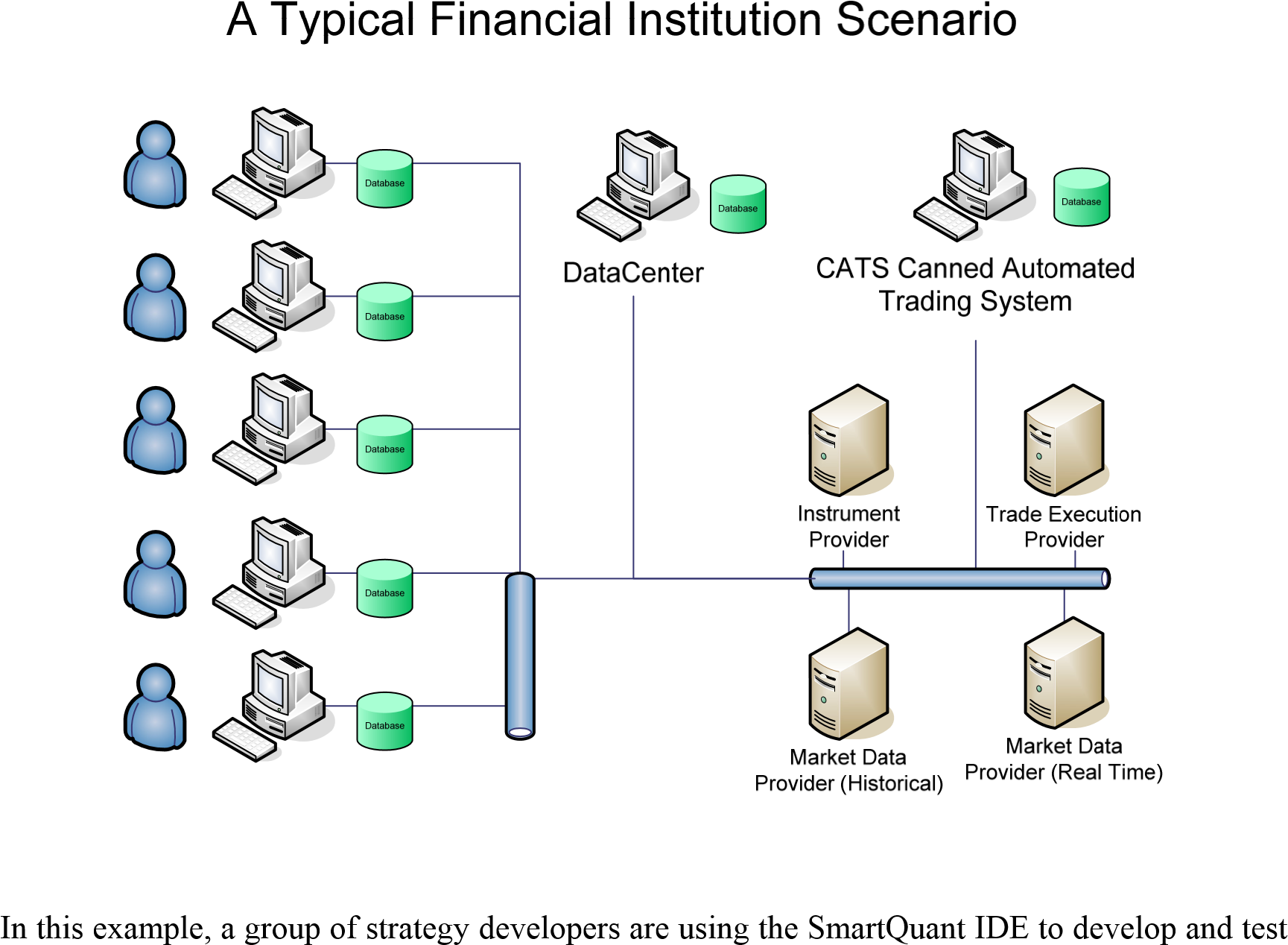


In this scenario, you can imagine that one person would be responsible for data management (collection of real time tick data and historical market data from market data providers), another person would be responsible for trading knowledge, and still another person (a programmer) would be responsible for programming the trading strategies specified by the trader.

This scenario shows the use of only two SmartQuant products—the IDE for strategy development and execution, and the DataCenter for capturing real time and historical market data in a centralized location. But the hedge fund could also use the SmartQuant CATS (Canned Automated Trading System) application to continuously run a completed trading strategy if desired, rather than trying to run live trading strategies out of an IDE development environment. The SmartQuant system is flexible enough to support the structures and goals of many different kinds of trading organizations.

## For Institutional Trading Groups

Large institutional trading groups would likely use the most complex SmartQuant systems. Here is a picture that shows how SmartQuant might be configured to fit the organizational structure of a trading department within a large financial institution.



new trading strategies. They download interesting sequences of market price data to their IDE machines from the centralized database of market data maintained by the DC (DataCenter).

The DataCenter application receives and stores real time quote and trade data from a market provider such as ESignal or TT (Trading Technologies) into its central SmartQuant database. This configuration enables IDE strategy developers to use the same data for back testing and debugging as their strategies used in live testing mode.

Finally, this configuration shows a production CATS system that continuously runs completed trading strategies on a separate computer that is not subject to reboots or unstable development environments that are commonly associated with software development machines. Completed strategies can be compiled and exported from the IDE environment into the CATS environment.

# Typical System Architectures and Data Flows

First, this manual shows a global architecture of all major system building blocks. Then second, after you have an understanding of the major building blocks, this manual shows you how to arrange the system building blocks into configurations that can solve common user tasks.

This manual is intended for three typical kinds of human organizations—for skilled individual traders/programmers, for small hedge funds, and for institutional trading groups. All three kinds of organizations can (and do) use exactly the same SmartQuant products.

The system configurations for all three scenarios are also very similar. The main difference is that hedge funds and institutions are likely to have both more people and more SmartQuant products in their environments. For example, a skilled individual trader might only use the SmartQuant IDE (one product), while institutional trading groups would likely use all three SmartQuant products (IDE, DataCenter, and CATS Canned Automated Trading System).

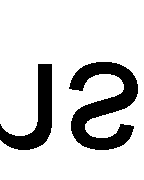
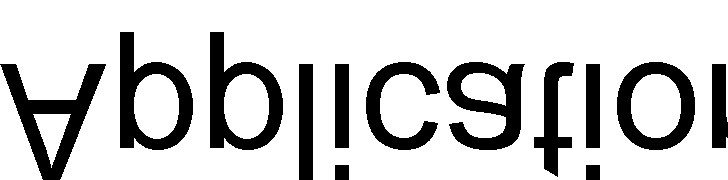
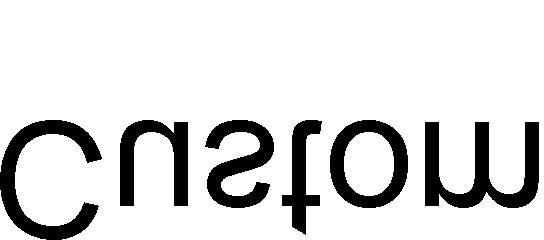
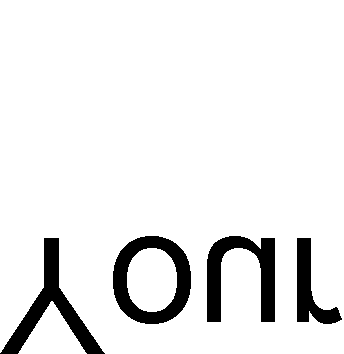
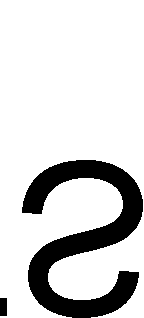
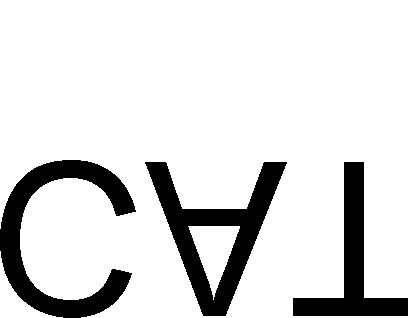
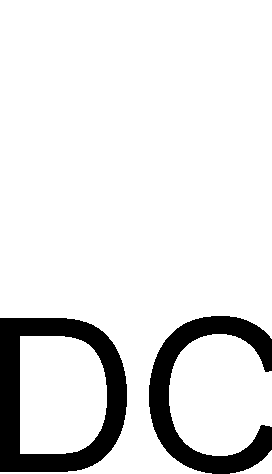
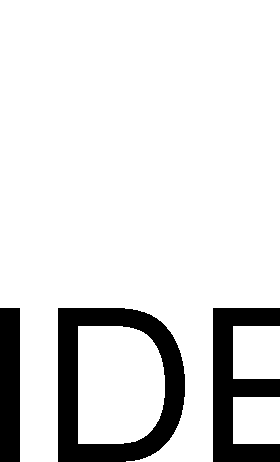
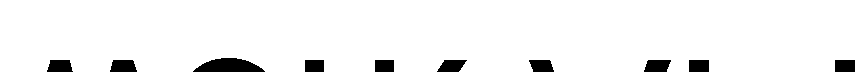
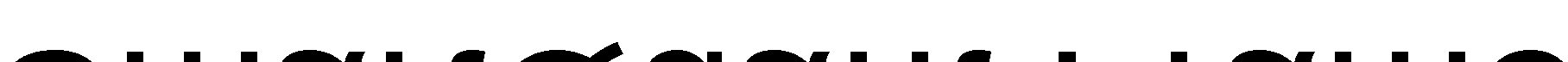
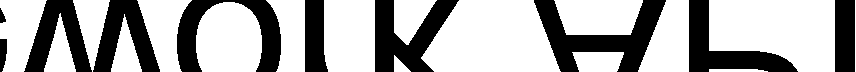
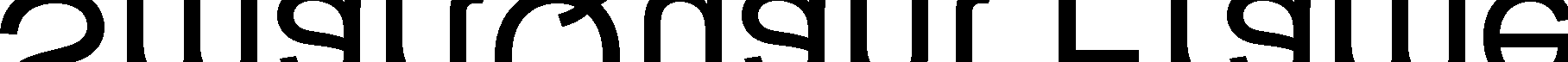
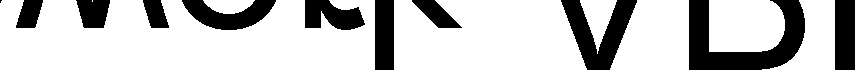
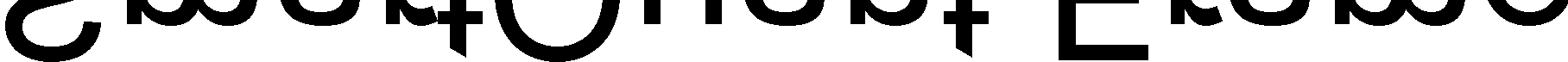
## Typical Tasks for Strategy Development

SmartQuant supports various user trading tasks that are necessary for developing and testing quantitative trading strategies. For example, some of the system arrangements that are discussed in the sections below support tasks such as these:

* Capturing market data from external data providers into a centralized DataCenter (DC) database for use by a team of people within a trading group, then
* Importing the captured data from a DataCenter (DC) database into a local SmartQuant IDE database for strategy development, then
* Testing a candidate strategy in back-testing mode using the imported market data from the local SmartQuant IDE database, and then
* Running a completed strategy in live-trading mode using live market data that is imported directly into the IDE where the executing strategy can use it to issue trading orders to an external trading execution provider.

## Diagram – SmartQuant Framework API

Here is a picture that shows how the three SmartQuant applications (IDE, DC, CATS) are built on top of the SmartQuant Framework API. A fourth application box has been drawn to illustrate how your custom applications could also be built on top of the SmartQuant Framework API.

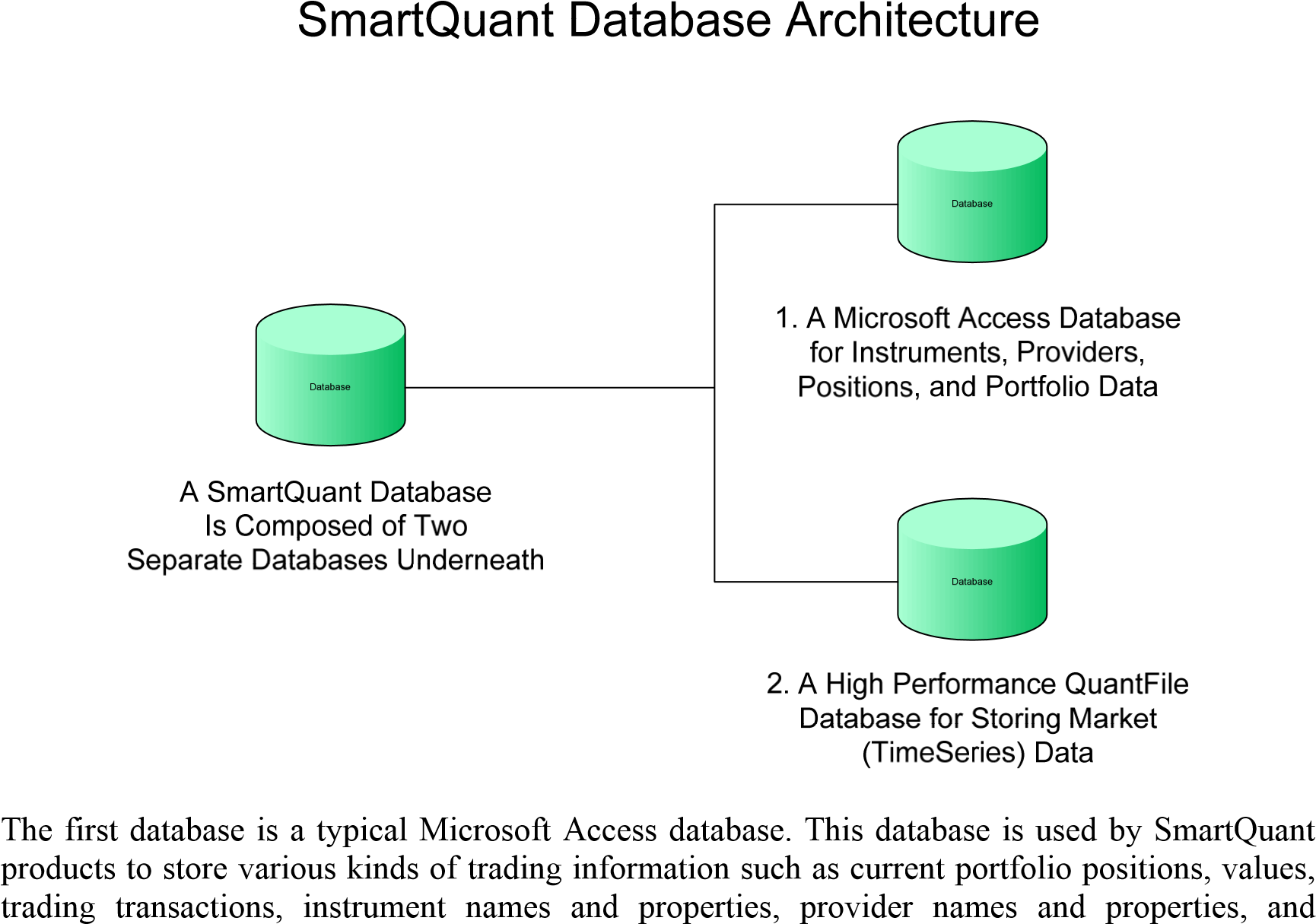


The real power of the SmartQuant system is contained within the framework API, which provides an extensive, well-designed software framework for working with computerized quantitative trading concepts, financial objects, portfolios, and trading order management operations.

The framework API objects (C# classes) and interfaces are based on many years of programming experience with computerized trading systems. This means the framework can help you to guide your software development work down more productive paths that use software objects and operations that effectively cover a large universe of possible financial trading strategies and operations.

## Diagram – SmartQuant Database Architecture

A special high performance database is at the heart of the SmartQuant system. Each SmartQuant software product uses a *SmartQuant Database* that is composed of two separate databases under the hood, as shown in the picture below.



strategy information and parameters. (Actually three physical Access databases are used—one for instrument definitions, one for portfolio transactions, and one for trade orders and executions.)

The second database is a special, high-performance database that uses *QuantFile* technology to store large amounts of financial data in a *TimeSeries* format. A *time series* is a series of data elements that are related to each other by time—for example, stock prices that vary over time.

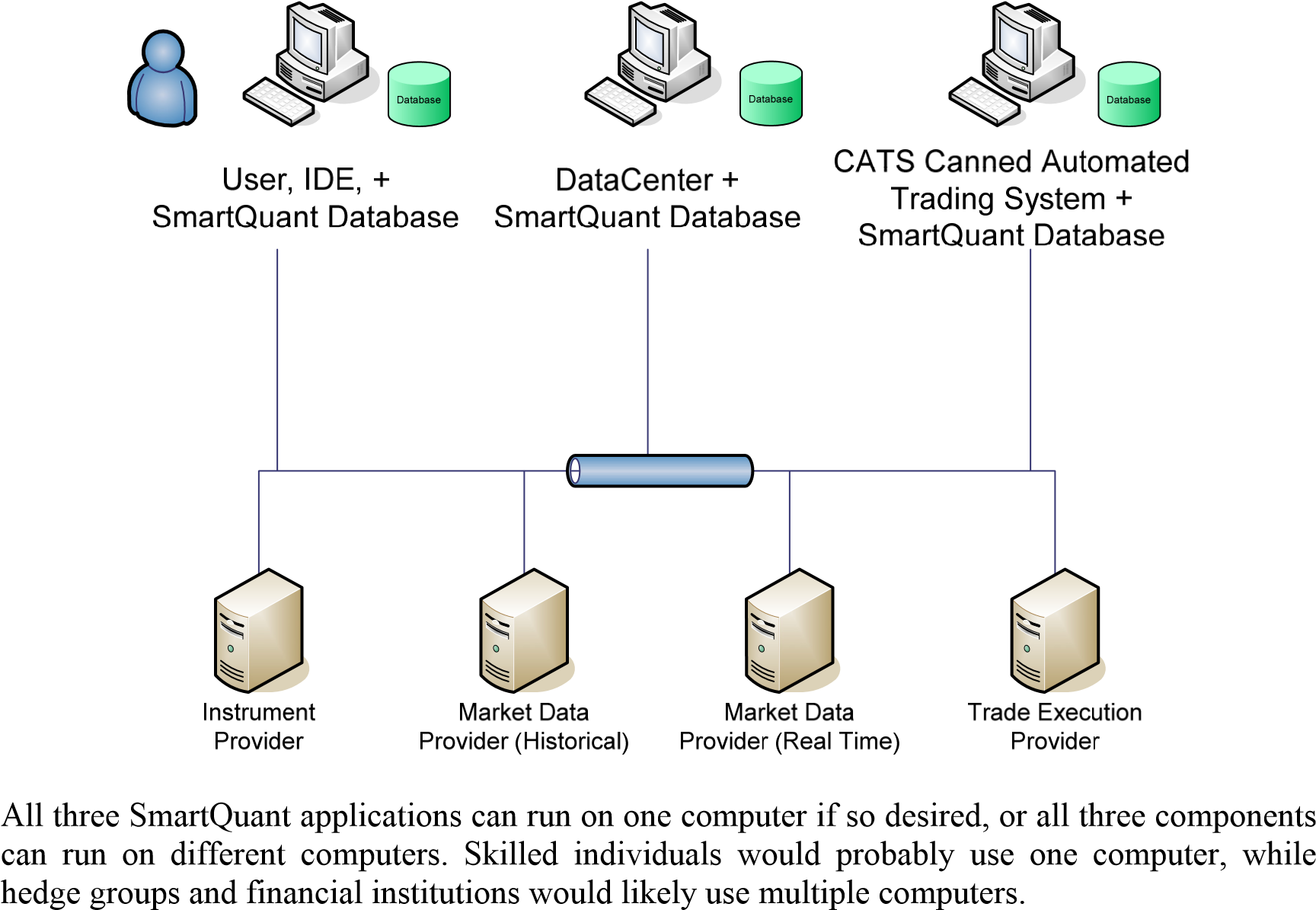
The main problem with working with time series is that there can be huge amounts of data to process—for example, a time series of 5-second price bars for a 1 week period contains tens of thousands of elements, for only one financial instrument. Since that much data won’t fit into computer memory, programs must read and write large amounts of time series data to hard disks, something that usually slows down programs quite a bit.

To work with such large amounts of market data efficiently, you need specialized file I/O software—that’s why the *QuantFile* technology was developed. *QuantFile technology* enables programs to work with large amounts of time series data as if the data was already stored in computer memory, instead of being stored on a hard disk. *QuantFile* uses compressed binary storage formats. Because of this special high-performance technology, SmartQuant products can easily provide industrial-strength speed and performance for large amounts of time series data.

## Diagram – SmartQuant System Architecture

The following picture shows the overall architecture of the SmartQuant system. In what follows, we will show a variety of subset pictures that illustrate various useful configurations of the three main SmartQuant products—the IDE (Interactive Development Environment), the DC (DataCenter), and the CATS (Canned Automated Trading System) applications.





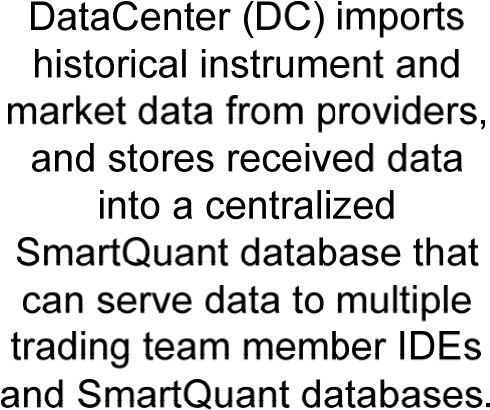
SmartQuant products understand 4 kinds of external service providers. Three of the providers are easy to understand—historical market data, real time market data, and trade execution providers. Sometimes one provider can provide multiple services. For example, ESignal provides historical and real time data (but no trade execution), and IB does real time data plus trade execution (but no historical data).

The fourth kind of provider, an Instrument Provider, is not so common. This kind of provider lets you download financial instrument definitions into your SmartQuant database, to save you the trouble of manually typing in lots of detailed instrument definitions for large numbers of instruments. IB (Interactive Brokers) and ESignal and Yahoo do not provide instruments, but other providers such as TT (Trading Technologies) do. Fortunately the DC DataCenter is also an instrument provider, so once you have defined instruments in the DC, you can download the instrument definitions into the IDE and into CATS.

## Diagram – DataCenter Importing Historical Market Data

Since you need market data to do anything interesting with SmartQuant products, we will begin by explaining how to get market data into a SmartQuant database where it can be used by strategies running under the IDE or CATS systems.







There are three ways that a program can access market data. These ways are as follows:

* The first way is to just use the fragments of historical data that are shipped with the SmartQuant software distribution (but this is uninteresting, since it is only sample data).
* The second way is to download data directly from a provider into a SmartQuant product (IDE, DC, or CATS). You can download either historical or real-time data this way.
* The third way is to move historical data from DC (DataCenter) into an IDE database. This is the usual way for back testing, since multiple people can share a centralized store of historical data, and just download the particular data they want. The DC can be configured to continuously download real time data during trading hours.

## Diagram – IDE Importing Historical Market Data

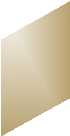
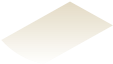
The simplest way to obtain some new data is to download some historical data from Yahoo (one of the default historical data providers in the IDE). This picture shows the data flow path.

IDE Importing Historical Market Data

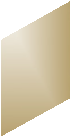
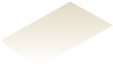


IDE imports historical instrument and market

User, IDE, + data from providers, and



Instrument



Market Data

SmartQuant Database

stores imported data in a SmartQuant database.

Provider Provider (Historical)

To download some of your favorite historical data in the IDE, you need to do two things:

1. First, define your favorite instrument in the Instruments panel, otherwise it won’t show up in the list of instruments in the download dialog. (Or for your first time, just use one of the predefined instruments such as MSFT or YHOO). To keep it simple, just specify the instrument symbol (e.g. CAT) and the exchange (e.g. NYSE). No other instrument definition data is required in the instrument properties panel.
2. Second, use this IDE menu sequence: Data / Import / History / Dialog (select Yahoo). Then select your symbols, and when you see the “Waiting…” dialog, click start. Data should download in a few seconds in normal cases.

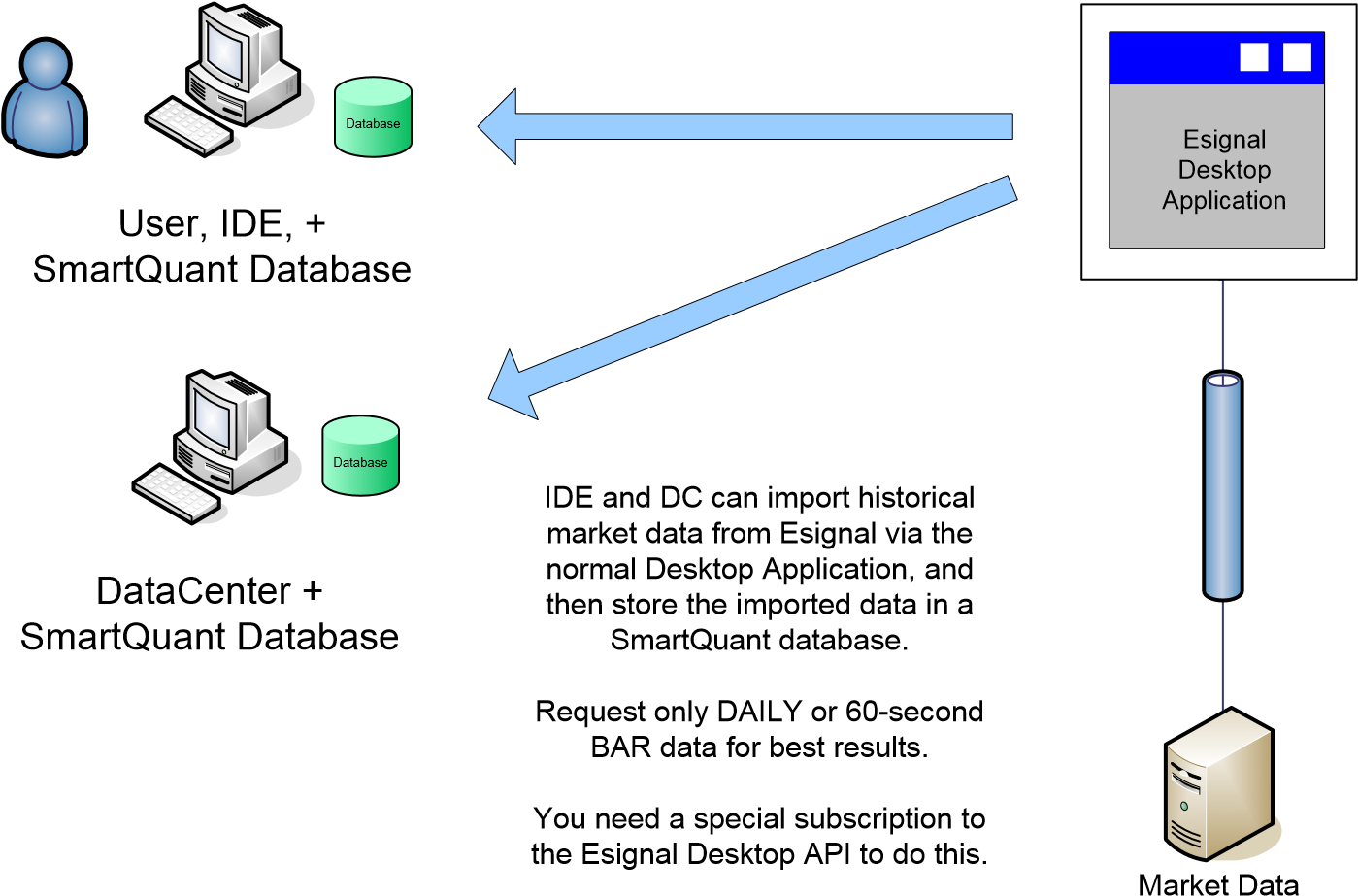
Yahoo is a free service, so it only provides daily market history for about a year. It cannot provide fine-grained 1-minute bar data like ESignal or TT, and cannot provide data for many years. Now you know how to get some new data into the IDE database. The procedure is the essentially the same for the DC and CATS systems. Define instruments and use them in the download sequence.

## Diagram – IDE Importing Historical Data from ESignal

Importing historical data from Yahoo is very simple because the IDE just contacts the Yahoo website for requested data using a simple web protocol. But if you want to get more detailed historical data, you’ll have to get it from a commercial provider through a more complex software API interface.

The picture below illustrates how to obtain market data from a historical data provider such as ESignal. Notice the data flow path has changed—now the IDE (or DC, or CATS) talks to a local desktop application, which in turn talks to the ESignal historical data servers. That way ESignal can do licensing operations, and can use a complex protocol between its desktop application and the ESignal data servers.

IDE Importing Historical Data – Esignal



Provider (Historical)

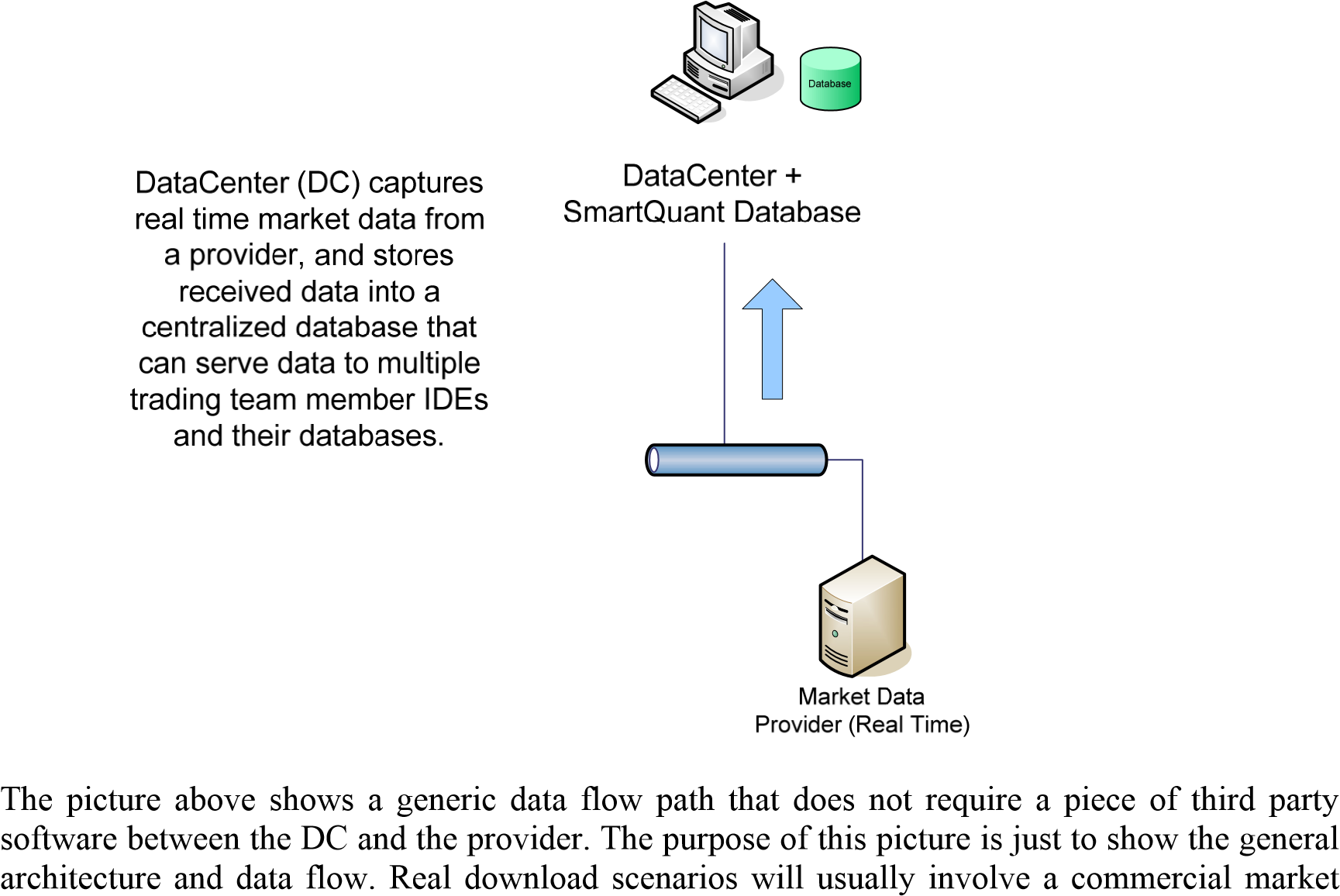
Esignal

To make this configuration work for you, use the same procedure as you did above with Yahoo, except select ESignal instead of Yahoo. You will need the ESignal desktop application installed and running on your computer before the download will work, since the desktop application is required to complete the data path. In addition, you need to pay ESignal an additional fee to add their Desktop API service to your normal ESignal subscription.

## Diagram – DataCenter Capturing Real Time Market Data

If you are part of a team of people who all use various bits of historical market data for strategy development, or if you just like to keep all your historical data in one convenient place, you can use the DataCenter to download historical or real time data into a centralized database. Then when individuals need a particular time series of data, they can download it from the DC into their IDEs or CATS trading systems.





data provider that will require a third piece of software between the DC and the provider servers, to control licensing, and to interact with the provider data servers.

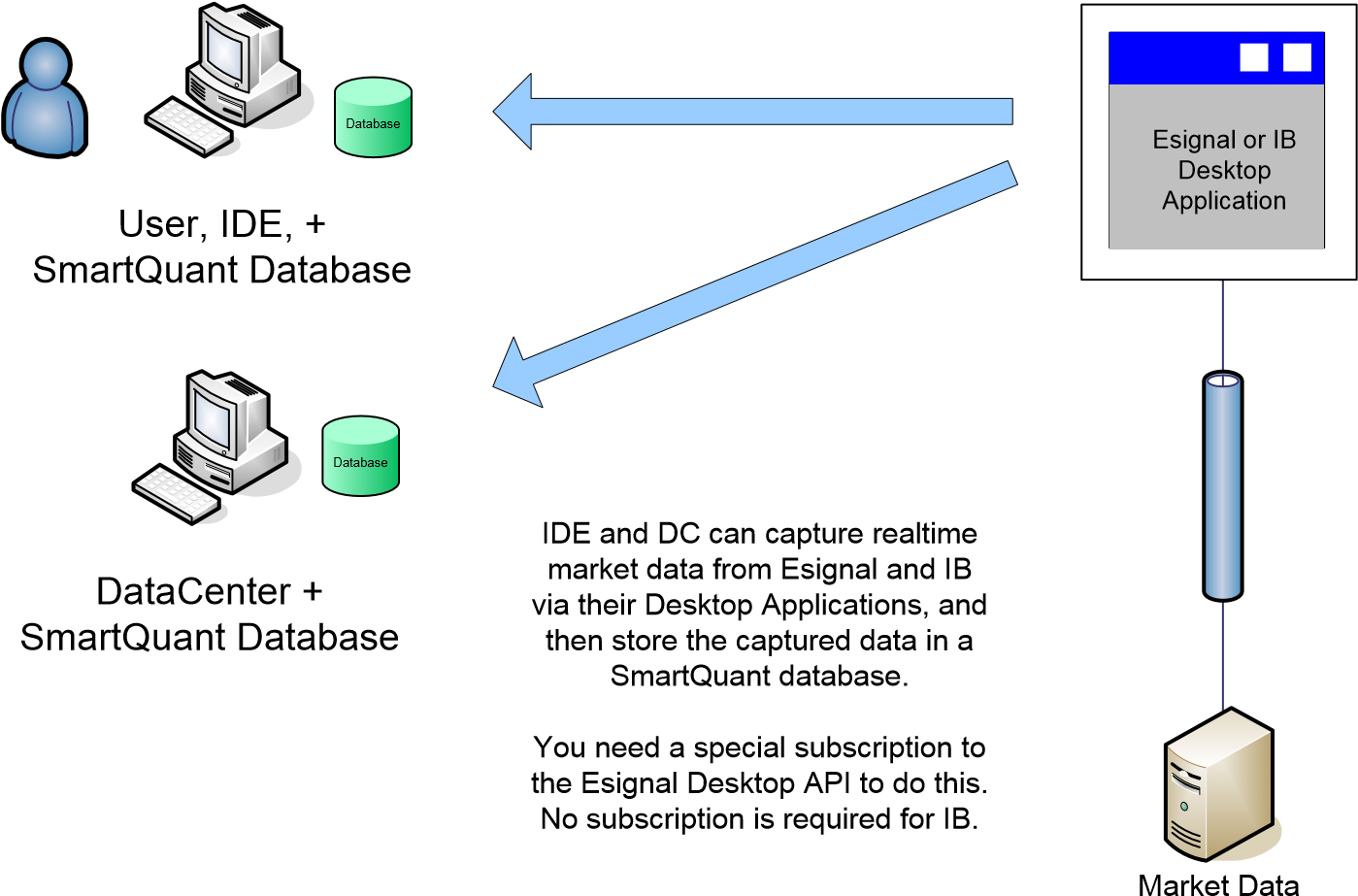
For example, the picture below shows how to capture real time market data using ESignal or IB (or some other real time provider) and a third piece of software (desktop application or library API) between a SmartQuant product and the provider servers.

## Diagram – IDE Capturing Real Time Data from ESignal or IB

This section explains in more detail how to capture real time market data from ESignal or IB (Interactive Brokers). Both of these real time providers require that their desktop applications be running to complete the data path between SmartQuant products and the provider servers.

In the case of ESignal, the ESignal desktop application must be running, and you must have a subscription to the ESignal Desktop API service. In the case of IB, the IB TWS (Traders Workstation) must be running, and no special subscription is required.

IDE Capturing Real Time Data – Esignal or IB



Provider (Real Time)

Either Esignal or IB

To download real time data into the IDE or DC or CATS, you must use a different procedure than for historical data. The real time procedure requires you to define your instruments (as usual), then connect to the provider, then drag and drop your instruments into the data collection center panel, and then finally click the little green arrow—easy to miss!—above the center panel in order to start the real time data collection.

Here are the individual steps to follow for the DataCenter. We assume you have defined your instruments already.

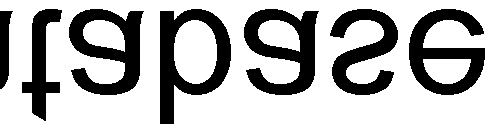
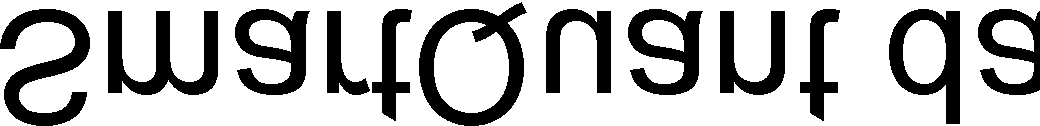
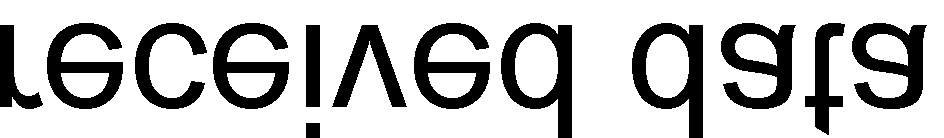
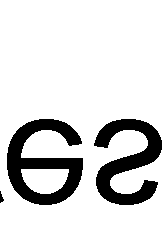
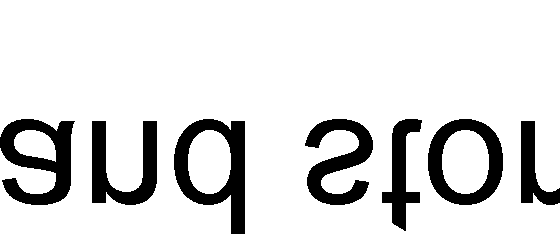
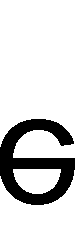
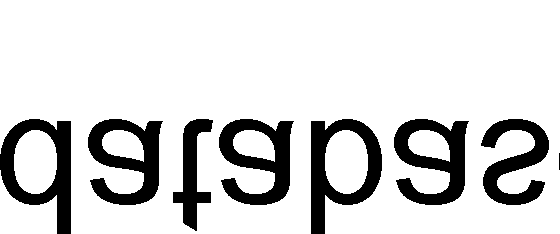
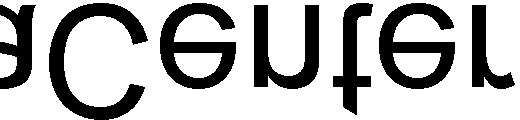
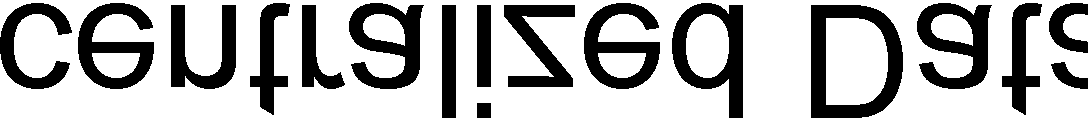
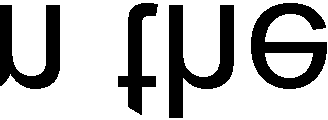
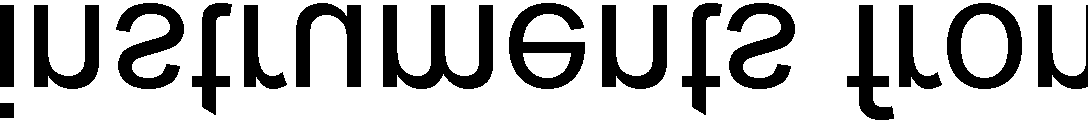
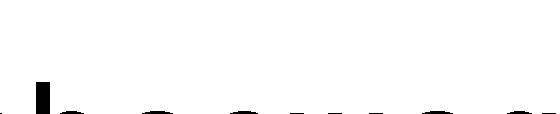
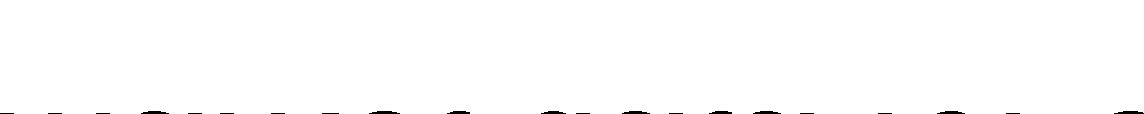
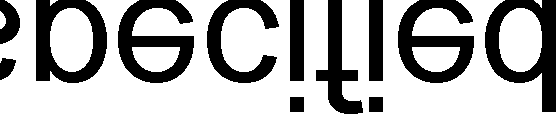
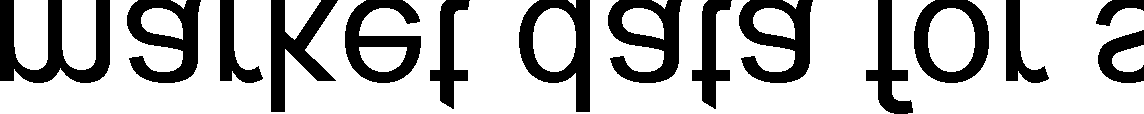
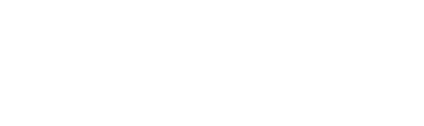
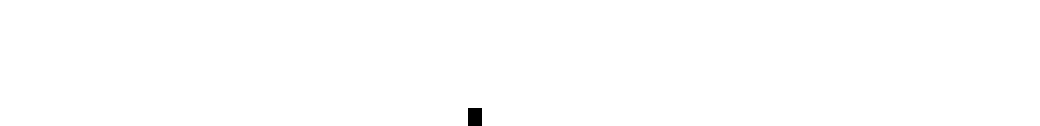
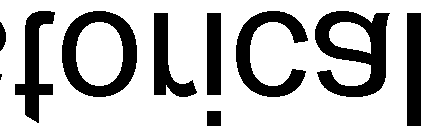
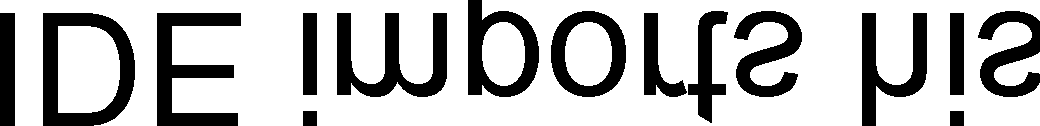
1. Connect the DC to the real time provider by clicking the provider name in the Provider panel (upper left of the IDE or lower left of DC), and configuring the right properties (such as user name) in the Properties panel (lower right).
2. You can also adjust the BarFactory settings in the properties panel. By default, the BarFactory will create 60-second bars from the provider data stream, and store the bars into the DC database. You can see the settings by clicking on the Items Collection in the BarFactory properties.
3. After you set your desired properties, right click the provider icon the Providers panel, and select Connect. Either the icon will turn green, or you will see an error message in the Provider Errors tab in the bottom center panel of the IDE or DC.
4. Drag and drop your instruments of choice into the center panel. There are several checkboxes on the panel header for data capturing options. Ensure that Quotes is checked. Later, you can make Bars (of any length 60, 300 seconds, etc) from the captured real time Quotes data.
5. Click the little green arrow above the top left of the center panel to start the real time data collection. Of course this procedure only works during exchange trading hours. Of course you need a market data subscription from the provider (ESignal) for the exchanges listed in your instrument definitions. For example, you need a provider subscription to the LSE London Stock Exchange in order to capture real time data for LSE-listed instruments.
6. You might have to wait a minute or two before you see any action, because your BarFactory settings (in the Properties panel for the Provider) might be set to produce 60second bars. This means that it will take the DC (BarFactory) sixty seconds to produce the first bar.

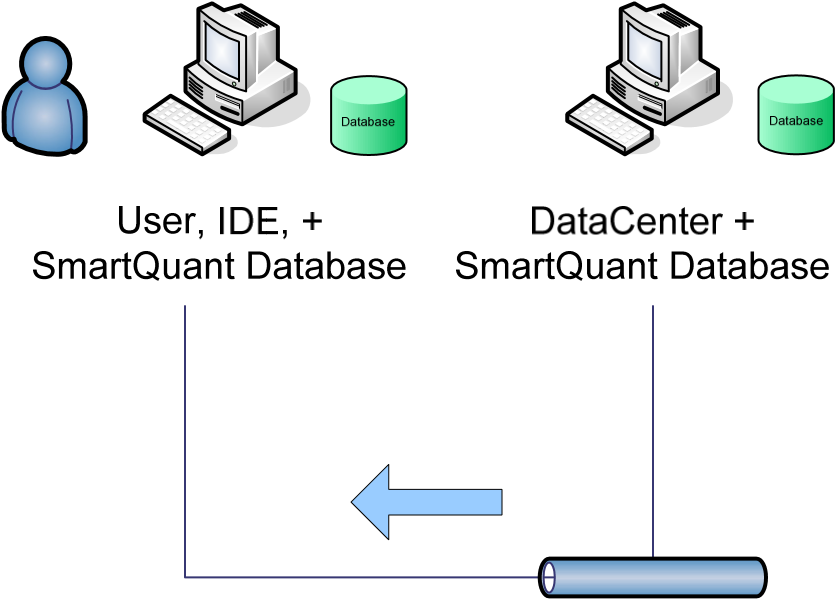
You can use the DC to capture the same data stream that IDE or CATS strategies use. This method enables you to go back and rerun the same data again to see why a strategy behaved a particular way. Then after fixing the strategy, you can retest it on the same (maybe tricky) sequence of market data to ensure that the strategy is behaving in a proper manner.

## Diagram – IDE Importing Market Data from DataCenter

Once you have captured some real time data, or imported some historical data, into the DC database, the historical DC data can be downloaded from the DC into the IDE database of strategy developers.

The picture below shows the general configuration and data flow paths. Since there are no commercial data providers involved in this configuration, no special passwords or subscriptions are required—all applications involved are SmartQuant products.





To download from the DC to an IDE:

1. Define the instruments that you are interested in for your IDE. (Even though the instrument definition exists in the DC, you must also define it in your IDE.)
2. Use the same IDE menu sequence as you did for Yahoo, except choose DC instead of Yahoo. The sequence is Data / Import / History / <wizard dialog>.

To see the data after it has been downloaded to the IDE, you have two choices:

1. Right click the instrument in the Instrument panel, and choose View Series. This shows just the series in the IDE database for that instrument.
2. Choose Data / QuantServer Explorer / and click down to see all series for all instruments in the IDE database. (QuantServer is the external name of the two-part SmartQuant database that is inside each SmartQuant product (IDE, DC, CATS).

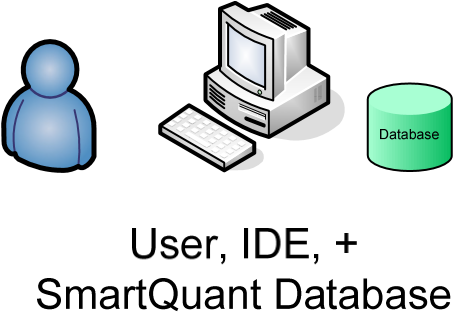
To make custom Bar data from Quote or Trade data, right click the series and choose Make Bars.

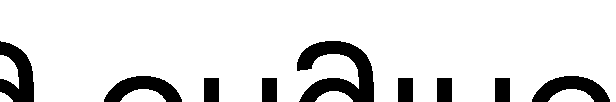
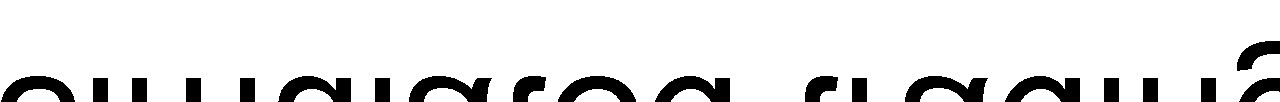
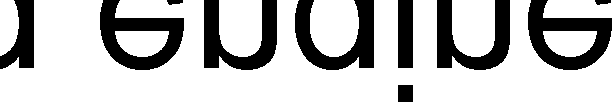
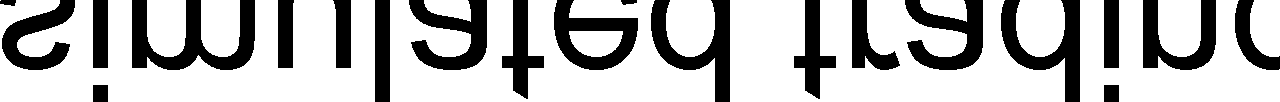
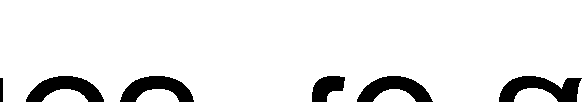
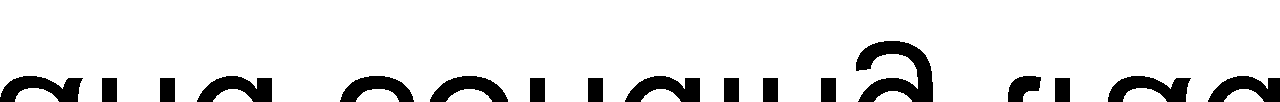
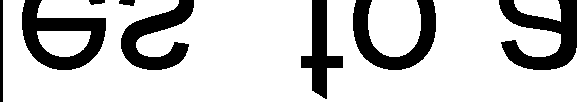
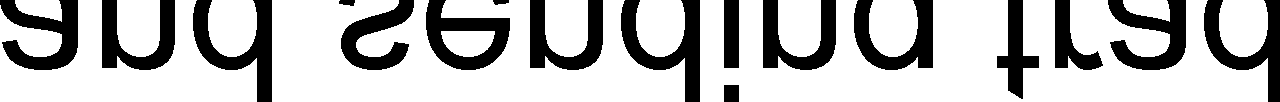
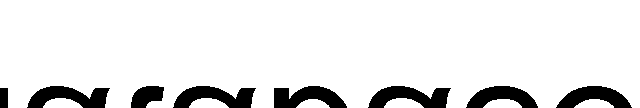
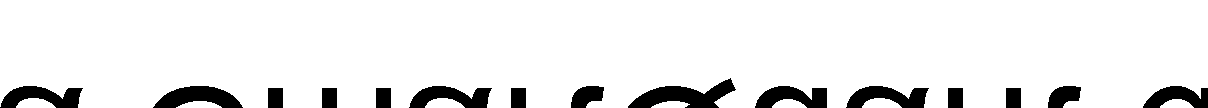
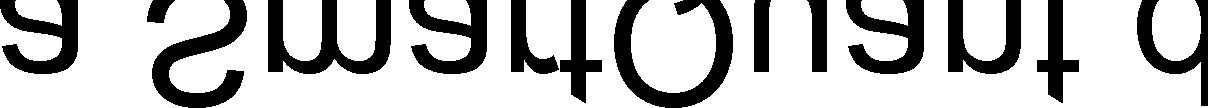
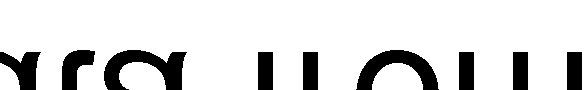
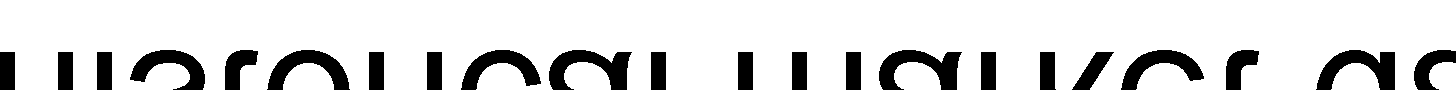
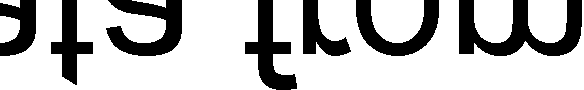
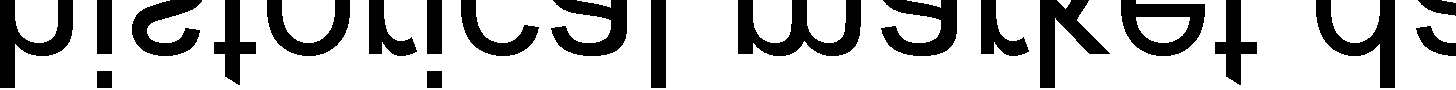
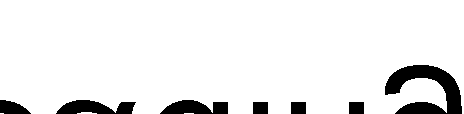
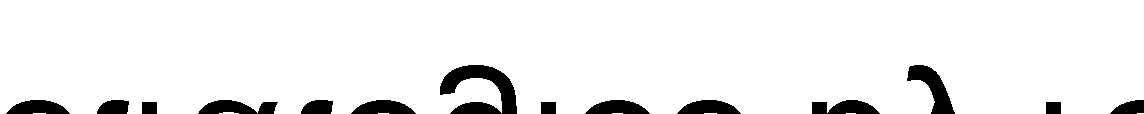
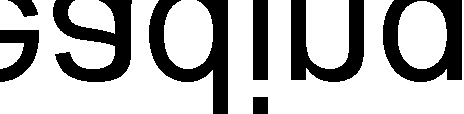
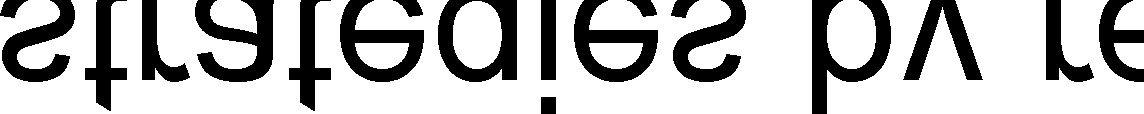
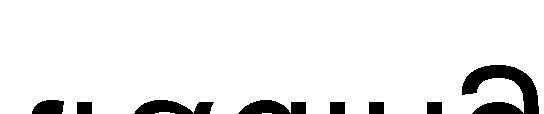
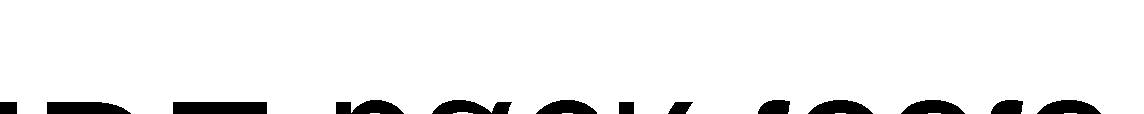
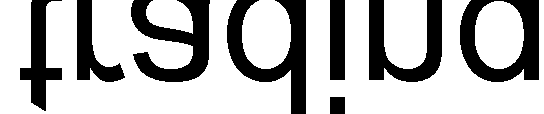
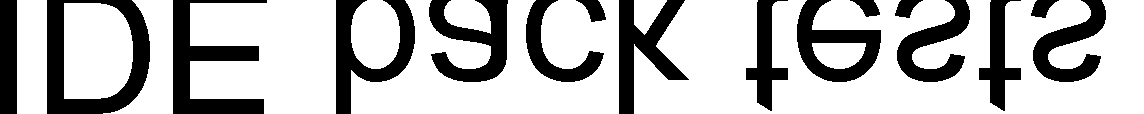
## Diagram – IDE Back Testing a Strategy

Now that you have seen some architectural configurations for getting market into the SmartQuant system, we can show some typical system configurations for back testing strategies against historical market data, and for live trading strategies against real time market data streams.

Probably the most common configuration of all is for using the IDE to develop and back test a strategy. Many interesting hours of developer time are spent in this mode, for sure.







Fortunately, this mode of operation is very simple. The IDE works against historical data that is stored in the local high performance SmartQuant database that is part of the IDE configuration.

Explanations of how to work the IDE, or of how to configure a strategy to pull a particular data time series for a particular instrument out of the database, are beyond the scope of this system architecture document. They are explained elsewhere.

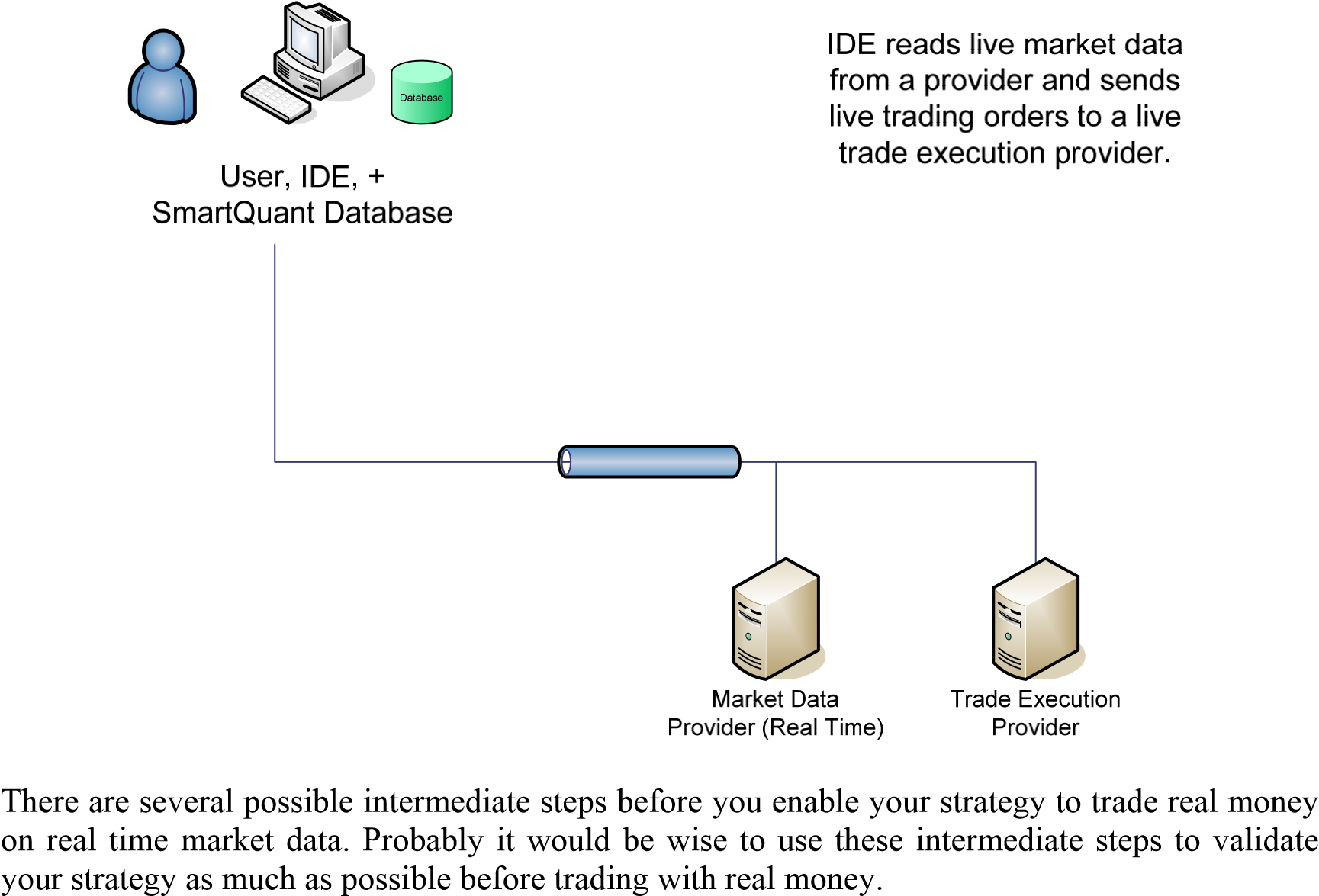
However, one useful “architectural” point can be made about how to work in this mode using Visual Studio for your main editor, instead of using the IDE editor. It is possible to point both the IDE and VS to the same set of source code files, so that you can edit files in VStudio (which has a more powerful editor), and yet run the files as part of a strategy in the IDE, so you can see your strategy performance.

The main idea is to use links in VS to link to the actual files that are used by the IDE. One method for doing this is to start a dummy project in VS, and then add IDE strategy files to the dummy project using the “Add Existing Item” right-click menu choice and the “Add as Link” option on the Add button to create the links. More detail can be found in the SmartQuant web forums, if you search for “IDE” and “Visual Studio”.

## Diagram – IDE Live Trading a Strategy

Once you have a strategy developed and are ready to run it on live data streams, the next step is hook up your IDE and strategy to a real time data provider and (optionally) a live trading execution provider. Here is a picture of the general configuration to achieve this goal.



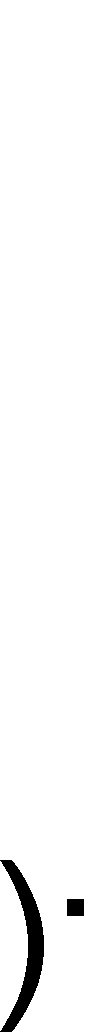
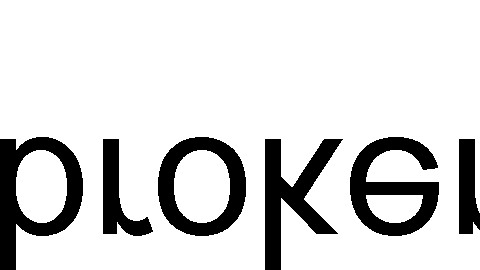
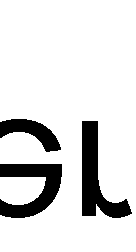
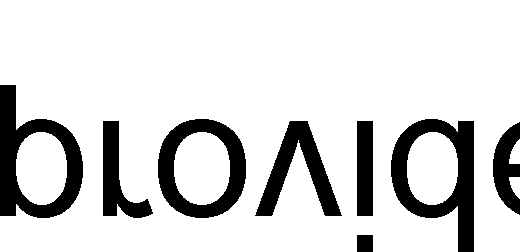
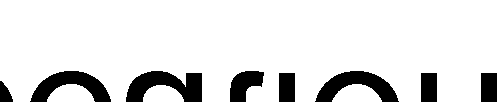
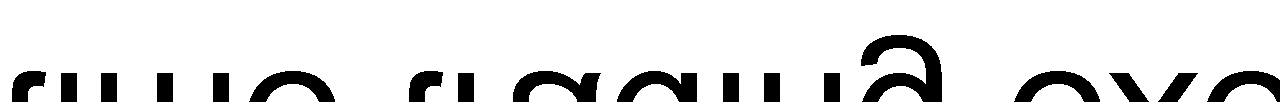
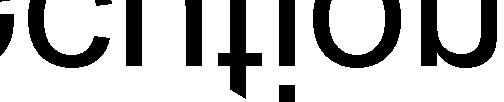
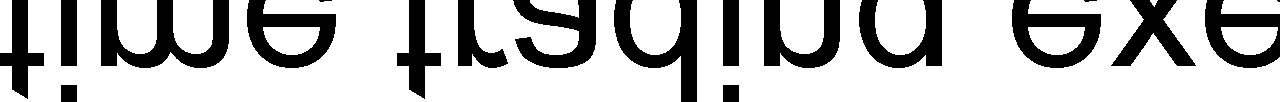
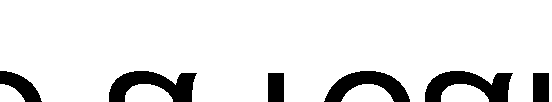
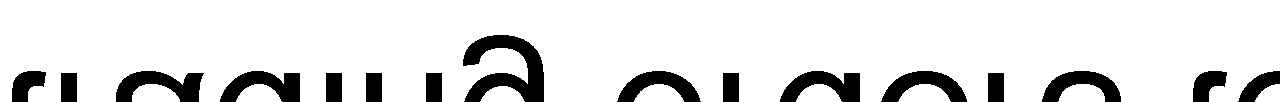
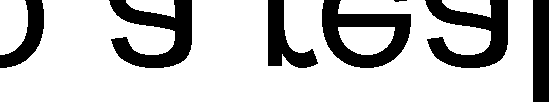
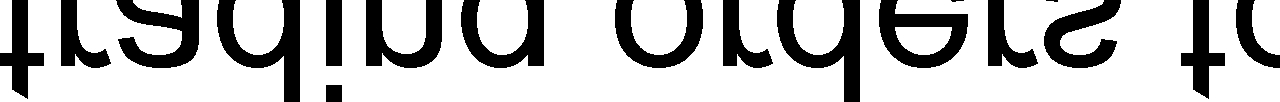
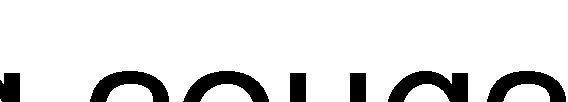
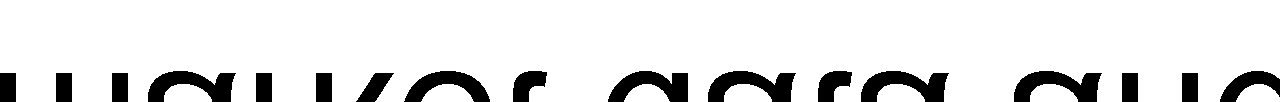
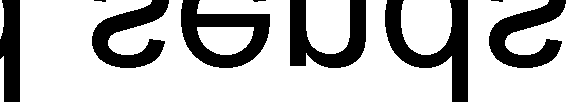
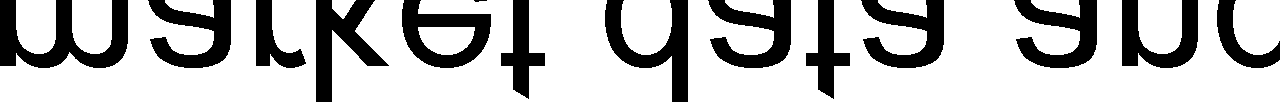
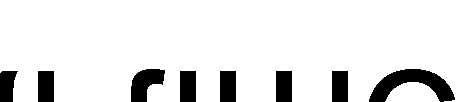
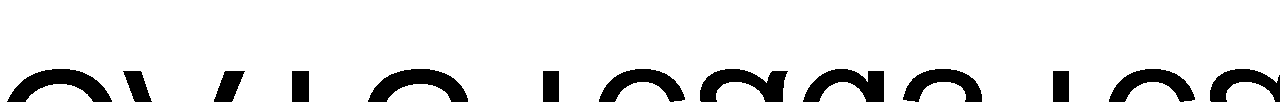
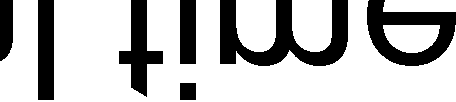
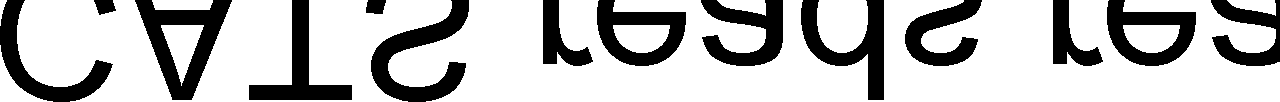


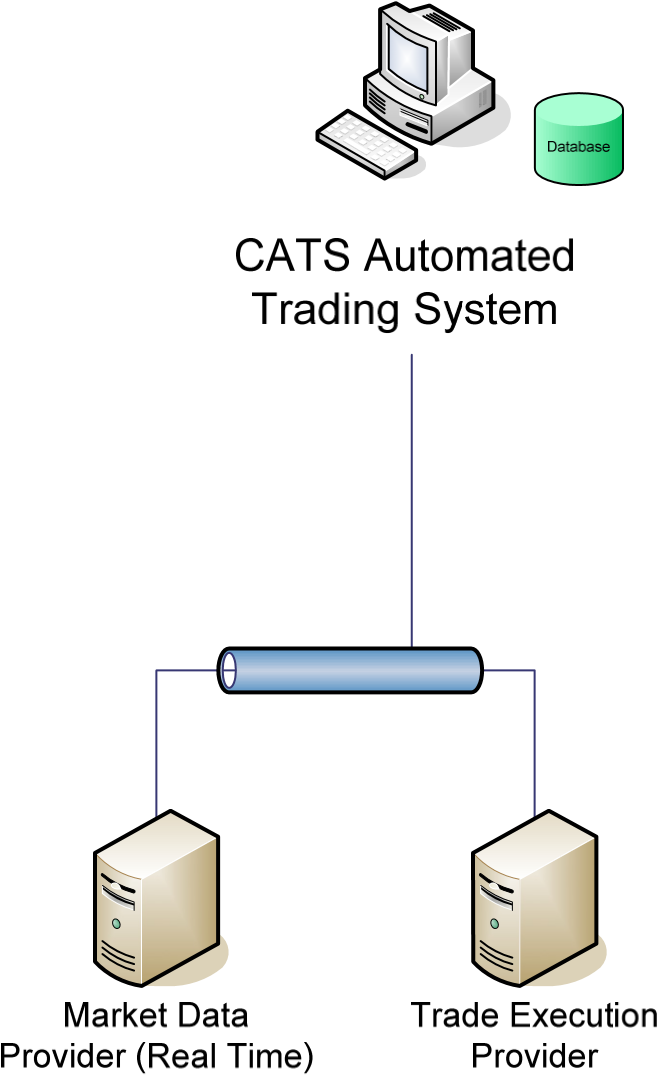
* Connect the IDE to a live market data stream, but use the IDE simulator for a trading execution engine.
* Connect the IDE to a live market data stream and a live trading execution provider (such as IB) that offers a “paper-trading” account. A paper trading account will execute all your trades in a realistic manner, recording profits and losses, but it does not trade real money.
* Connect the IDE to a live market data stream and a live trading execution engine, and trade real money. (You might do this for a minute or two early on, just to see if you can configure the software components correctly and get things working. Then you can go back to simulated trading—it is much cheaper while you are optimizing your strategies.)

## Diagram – CATS Live Trading a Strategy

The main purpose of CATS is to provide a high-reliability environment for live trading (or papertrading) previously constructed strategies that have been exported from the IDE environment. CATS is not intended to support back testing (even though a small historical database is shipped with the product so you can demo CATS without having to write your own strategy).

Live trading a strategy using a CATS system is very similar to live trading with the IDE as described above. The main difference is that you configure CATS to use real time market data and a real time trading engine, instead of using the IDE.



As before with IDE live trading, you might want to use the previously-mentioned successive intermediate steps before doing live trading with real money. The intermediate steps were these:

* Connect CATS to a live market data stream, but use the simulated trading engine in CATS to “paper trade” your strategy before using a real trading engine.
* Connect CATS to a live market data stream and a live trading execution provider (such as IB) that offers a “paper-trading” account that uses simulated money.
* Connect CATS to a live data feed, a live trading engine, and trade real money.

# SmartQuant System Building Blocks

Now that you have seen how to configure SmartQuant products to meet various data acquisition, strategy development, and trading goals, we can dig a little deeper into the design and purpose of each of the components that were shown in the network diagrams above.

## SmartQuant Database Design

Recall that each SmartQuant database has two parts—a first Microsoft Access database for storing strategy data (such as instrument properties, portfolio properties, strategy properties, trades made, and performance statistics), and a second high-performance QuantFile database for storing market data (ticks, quotes, trades, and bars for financial instruments) in compressed form.

Also recall that you can capture real time data (or import historical data) using either the IDE or DC, and then store it in the SmartQuant database associated with the IDE or DC. Typically you would load the DC with data obtained from an external data provider, and then load the IDE from the DataCenter (if you have purchased the DataCenter), or from an external provider.

The CATS system is different than the IDE and DC regarding its data storage behavior—the CATS system cannot store incoming market data streams into its database. Instead, the CATS system only uses the incoming market data for strategy execution, and then throws the market data away immediately after using it. There are no Data Importing menus on the CATS menu bar.

If you want to investigate why CATS made a particular trading move in a live trading scenario after the live trading period has ended, you must be sure to use the IDE or the DC to capture and store the real time market data feed that CATS is seeing during the scenario. That way you can play the captured data back later, to reproduce the live data stream to help debug the strategy.

## SmartQuant C# Framework Assemblies

The foundation of the SmartQuant framework is a large set of C# assemblies that provide classes and functions for building computerized quantitative trading systems. The programming API is quite powerful, and enables you to build complex computerized financial trading applications.

**Modeling Capabilities:** The framework libraries understand things such as computerized market data providers, trade execution providers, many kinds of financial instruments, time series of market data, trading signals, trading orders, positions, portfolios, trading risk and exposure management, price and volume charting, many kinds of technical indicators and their charting appearances, as well as framework support for trading simulations and performance analysis of trading results. The framework is also extensible, so you can implement your own C# objects.

**Operational Capabilities:** The SmartQuant framework is not just a set of passive library API interfaces. Instead, the framework implements a working “operating system” or environment for quantitative financial trading applications. For example, when you initialize (or “boot”) the framework in your code, the framework automatically load portfolios and financial instrument definitions from databases, connects to various data providers, loads strategies, and performs other useful operations that contribute to the smooth running of financial trading systems.

**Functional Capabilities:** The SmartQuant framework provides an extensive set of API interfaces and operations for modeling and testing computerized trading systems. It goes well beyond most other computerized trading system products in functionality because it has been designed from the ground up to be a framework API system with example applications (IDE, DC, CATS) built on top of the framework. For example, with some effort you could use SmartQuant to build a Tradestation or Wealth Lab equivalent application, but you could not do the reverse.

**Financial Capabilities:** The SmartQuant framework uses the FIX (Financial Information eXchange) protocol model to represent a large set of electronic transactions involving financial instruments, markets, trading order types, and communication scenarios between buyers/sellers in the financial trading domain. The FIX model is the most comprehensive industry protocol model for international real-time exchange of [securities](http://en.wikipedia.org/wiki/Security_%28finance%29) transactions in the [finance](http://en.wikipedia.org/wiki/Finance) markets.

**Event Driven Application Capabilities:** The SmartQuant framework uses an event-driven programming model to more effectively represent real-world interactions between trading system clients and broker APIs or data vendor APIs. This means that SmartQuant makes it almost trivial to switch your trading system from back testing to real-time live trading with market data providers and trade execution providers of your choice.

**Application Support Capabilities:** The SmartQuant system includes several useful programs (IDE, DC, CATS, DemoApp) that are built on top of the framework. These applications illustrate the power of the SmartQuant programming API. You can use these applications as models for your own applications. (Source code licenses are available.)

You can choose from two philosophies when you build your own trading system. The first way is to do all your development work within the provided SmartQuant applications (IDE + CATS), to end up with a convenient standalone CATS system that can live trade your strategies. The second way is to code your own completely new application using the SmartQuant framework libraries just like you would any other set of C# assemblies. Coding your own applications will obviously give you the ultimate flexibility, but of course at the cost of much more programming work.

Here is a partial list of framework libraries, with their major functional areas of expertise:

|  |  |
| --- | --- |
| SmartQuant.dll | - configuration info for the framework |
| SmartQuant.Business.dll | - calendar functions, business days, holidays |
| SmartQuant.Charting.dll | - functions for drawing bar charts, performance charts |
| SmartQuant.Data.dll | - time series arrays, order books, market depth arrays |

SmartQuant.DataCenterLib.dll - DataCenter support functions

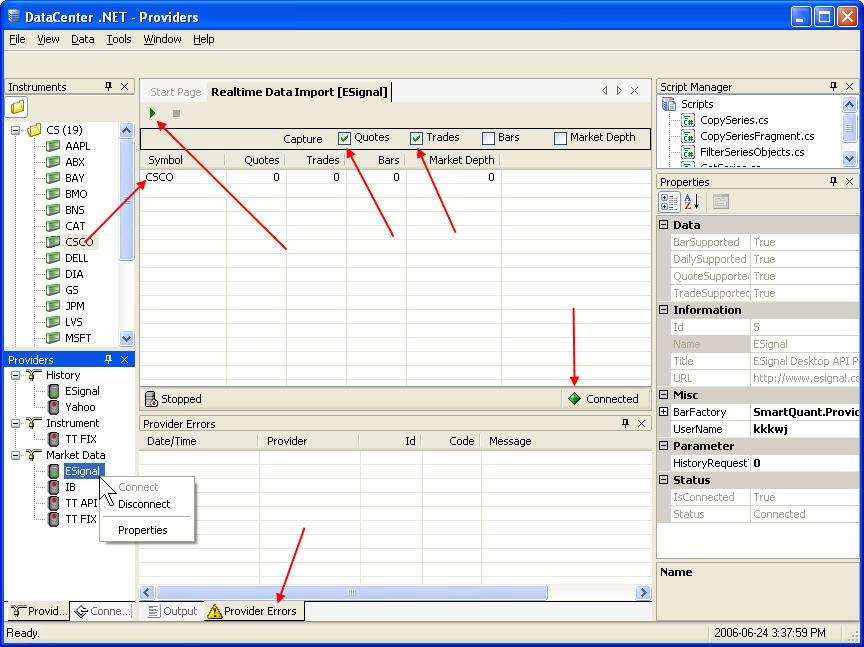
|  |  |
| --- | --- |
| SmartQuant.DC.dll | - DataCenter support functions |
| SmartQuant.Execution.dll | - order routing, order management, trade execution |
| SmartQuant.File.dll | - QuantFile high-performance file system |
| SmartQuant.FinChart.dll | - support for showing triggers, signals, on charts |
| SmartQuant.FIX.dll | - Financial Information eXchange protocol functions |
| SmartQuant.Indicators.dll | - technical indicators such as moving averages |
| SmartQuant.Instruments.dll | - financial instrument types, storage, properties |
| SmartQuant.Neural.dll | - neural network support for trading strategies |
| SmartQuant.Optimization.dll | - optimize (“tune”) strategy parameters |
| SmartQuant.Pricers.dll | - calculate “Greeks” delta, gamma, vega for instruments |
| SmartQuant.Providers.dll | - support for external providers |
| SmartQuant.Quant.dll | - financial pricing functions, options pricing |
| SmartQuant.Series.dll | - time series, double series, bar series |
| SmartQuant.Simulation.dll | - simulation engine for back testing strategies |
| SmartQuant.Testing.dll | - performance analysis of strategy test runs |
| SmartQuant.Trading.dll | - trading components entry, exit, market manager, risk mgr |
| SmartQuant.ESignal.dll | - interface to ESignal provider |
| SmartQuant.ExcelLib.dll | - interface to Excel |
| SmartQuant.IB.dll | - interface to IB Interactive Brokers |
| SmartQuant.TT.dll | - interface to TT Trading Technologies |
| SmartQuant.Yahoo.dll | - interface to Yahoo website for historical data download |

## The DC (Data Center) Application

The main purpose of the data center is to centralize and manage all historical market data in a single place for convenient use by IDE strategy developers. Data maintainers can configure DC to capture real-time market data feeds in a 24x7-no-rebooting-required mode, and then use DC data maintenance scripts (C# programs) to perform data maintenance operations in an automated, convenient way.

Here is a screenshot of the DataCenter, with red arrows to highlight interesting features. In this example, red arrows point to these things (in clockwise order):

* the drag-and-drop operation that loads instruments into the Data Import window
* the little green arrow to start data collection
* the Quotes and Trades checkboxes to capture those types of data elements
* the little green diamond connected status icon
* the Provider Errors window, in case something goes wrong
* the context menu for connecting and disconnecting DC to the ESignal provider



The DataCenter is quite useful in practice, because it is convenient to have all data reliably captured and organized in one place. Strategy developers can download just the pieces of data that they need to work on, and no more. Developers can delete irrelevant data from their IDE databases to keep their workspaces clean, and thereby reduce the amount of old data stored on their hard disks. DataCenter can be run in a “24x7 no-reboot” mode to continuously capture data.

In addition, for people who run CATS, the DataCenter makes it easy to capture the same real time data feed that CATS is running. This is very useful if you want to investigate why a CATS strategy behaved in a particular (market data) situation. To investigate, you would download the interesting market data series into an IDE, and then run the strategy on the IDE data in a stepwise fashion to see what the strategy was thinking.

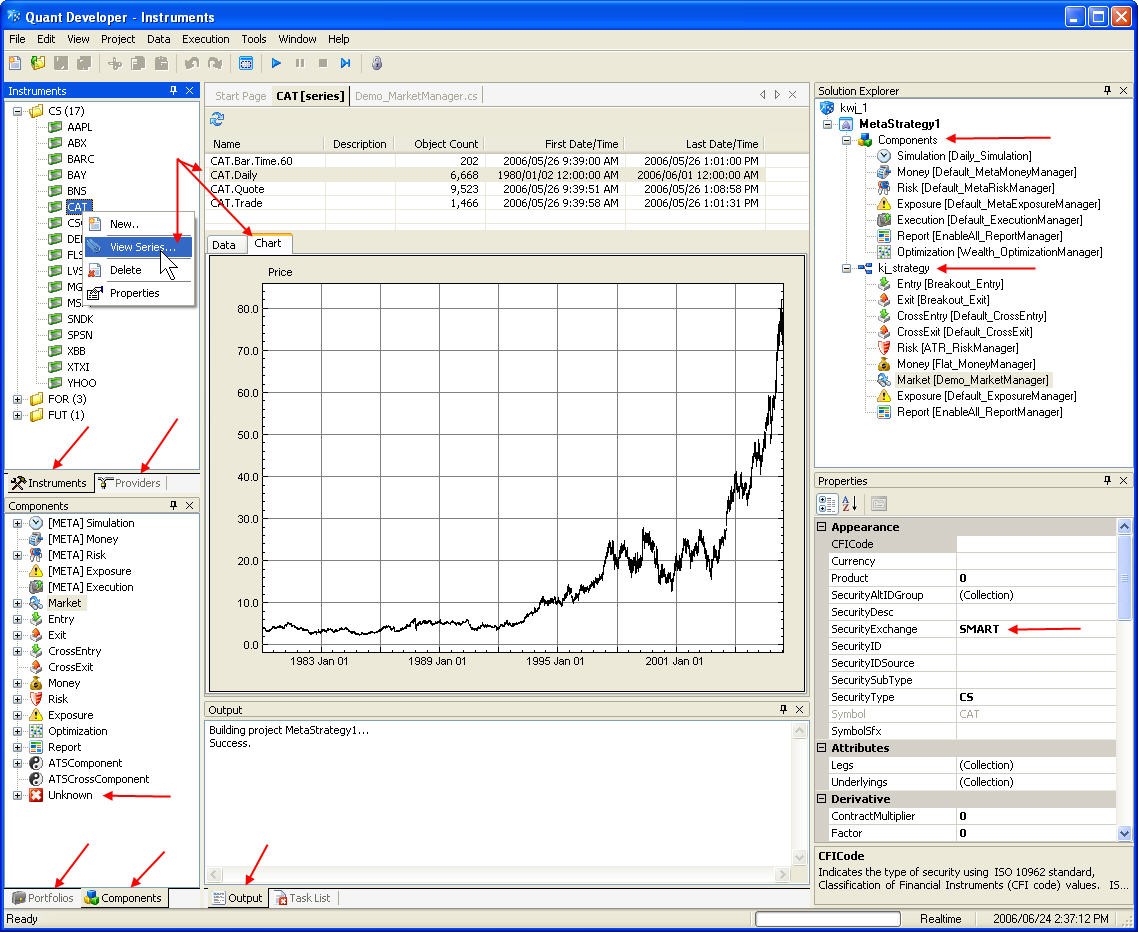
Although the DataCenter provides several checkbox options (Quotes, Trades, Bars, and Market Depth) on the header of the real time data capture window, not all of the checkbox options are relevant to all market data providers. Typically it is sufficient to check Quotes and/or Trades to collect the real time data. If your provider provides market depth, and if you have a subscription to market depth, you could also capture market depth information too.

In all cases, once data is captured, you can make your own Bar series from captured Quotes or Trades. To make your own bars for a particular time period, use the DC or IDE menu sequence Data / QuantServer Explorer to view all the series that are stored in the database, and then right click a series and choose Make Bars to make your own bar series.

## The IDE (Integrated Development Environment) Application

The main purpose of the IDE is to support the development and testing of quantitative computer trading strategies that can later be executed in a standalone, live-trading CATS environment. For this goal, the IDE provides mechanisms for defining instruments, acquiring data, creating strategies, executing simulations, and analyzing strategy performance results. The general work flow is to develop and test a strategy in the IDE, then export the compiled strategy for use in CATS, then load the strategy into CATS, and run the strategy in the automated CATS system.

Here is a picture of the IDE, with red arrows pointing to some of the interesting features that are mentioned in this document. In this example, we have right-clicked an instrument to View the available series for the instrument in the underlying database. We selected CAT (Caterpillar) and have displayed a chart of daily prices for that data series.

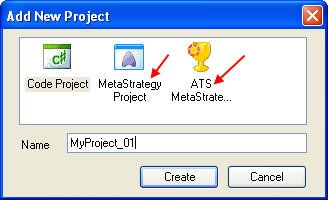


On the right, this picture shows a MetaStrategy project that has a first Components section for the meta-strategy level, and a second strategy section named kj\_strategy that contains components at the individual strategy level. The trading exchange name for CAT has been defined as SMART, which is a name used by IB TWS for their smart routing of trade orders to the best of N trading exchanges, in order to get you the best price on your trades. (Try to zoom your document viewer to expand the screenshot.)

### Strategy Framework Choices

The IDE provides several different strategy frameworks to make your development life easier. The strategy frameworks are organized around two dimensions—(1) whether you want to partition your trading system into one big component or into many separate components (Entry, Exit, Risk, Market), and (2) for the separate component approach, the number of instruments that a strategy component wants to “see” within a strategy component (Entry, Exit…).

When you create a new solution (File, New, Blank Solution) and then right click the blank solution icon in the Solution Explorer window, a small dialog pops up so you can specify whether you want to create an ATS project, or a MetaStrategy project. Here is a screenshot of the dialog, with red arrows pointing out the two choices. (Admittedly, the “ATS Metastrategy Project” is confusingly close to “MetaStrategy Project”.) The Code project choice is for “research” code projects that are not part of a strategy at all

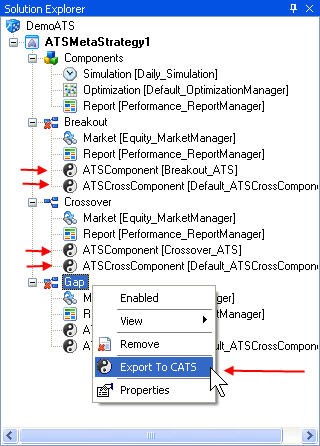


**BEWARE—by default, the IDE creates meta-strategy projects that do not work on the example data. This is because newly created projects use only Default\_XXX (that is, empty) strategy components that do nothing.** To get a new strategy project to work, you must go through the component types and select working components. This process is explained in more detail below, where this warning is given to you again. (This warning bears repeating because it can save you a few hours of time debugging working but empty “do nothing” strategy projects).

### The ATS Strategy Framework

The most complex trading systems should probably use an ATS project as the foundation of their trading system. This is because the top-level ATSComponent (inside the ATS project) combines all the Entry, Exit, Money, Risk, and Market components into one large component, so that your code can have access to all aspects of your strategy at any particular time.

Here is a screenshot showing the organization of the example DemoATS project. This ATS project has 3 strategies in it, plus the top-level section for the entire trading system. Notice that each of the three strategies (Breakout, Crossover, Gap) has four mandatory components (Market, Report, ATSComponent, and ATSCrossComponent).



Typically you would only put your strategy code in one of the two ATS components, and leave the other component blank by selecting the default version of it. Here, normal ATS components carry the code, and the ATS cross components are empty.

When you right click a strategy node (as shown above for the Gap strategy), you get a context menu that lets you enable or disable the strategy within the project, plus an option to export the strategy to a precompiled assembly for use in the CATS system.

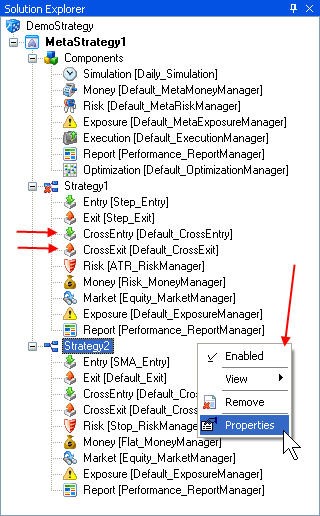
### The MetaStrategy Framework

In contrast, if you prefer a more structured and partitioned development approach, you should use a MetaStrategy project as the foundation of your trading system. A MetaStrategy project partitions various strategy functions into separate individual Entry, Exit, Risk, Money, Exposure, and Market components.

The main advantage of using a component approach is that you can easily reconfigure your strategy to try out different combinations of components. This “mix and match” capability is possible because all components of a particular type (Entry, Exit) have the same API, and therefore are interchangeable. Just right click the component type name in the Solution Explorer panel of the IDE, and choose a component instance that interests you.

MetaStrategy projects allow you to create multiple strategy projects within one MetaStrategy project framework. That way you can run different strategies against the same data stream, and collect aggregated performance statistics on the set of strategies, in addition to the performance statistics on each individual strategy. For example, you might combine a trend-following moving average strategy with a short-selling strategy based on Aroon Oscillators, all in the same MetaStrategy project.

Here is a screenshot of what a MetaStrategy looks like. Notice that you cannot export a strategy from a MetaStrategy project for use in a CATS system—there is no Export option on the rightclick context menu for component strategies. Only strategies from ATS projects can be exported for use in a CATS system. Note also the presence of Money, Risk, and Exposure managers at the meta-strategy level, to give more global control over the combined efforts of the two individual strategies.



### Project Level and Strategy Level Components

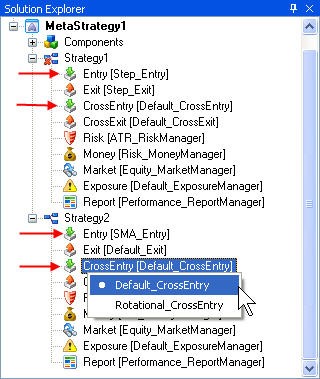
MetaStrategy project frameworks create components at both the meta-strategy level and at the strategy level. This means that in a 2-strategy project framework such as the one shown above, you will have slots for 3 Money Manager components—one to manage trade Money for each of the 2 strategies at the strategy level, and a third one to manage money for the whole portfolio at the meta-strategy level. See the fully-expanded picture of the example meta-strategy picture above, and count the money manager components (3).

For example, each strategy manager might permit half of your capital to be put into each technology instrument trade, whereas the meta-strategy money manager might disallow a trade because there can be no more than 20% of the portfolio invested in technology companies.

### Normal and Cross Components

Within each of the ATS and MetaStrategy approaches, you can choose normal components (such as Entry) or cross components (such as CrossEntry, or ATSCrossComponent). A normal component can only “see” (receive) bars for a single instrument, whereas a cross component can see (receive) bars for multiple instruments.

Here is a screenshot that shows Normal and Cross Entry components. In this example, the normal Entry components (Step Entry, SMA Entry) carry the useful code, and the Cross components are empty (Default\_CrossEntry). This picture also shows how to select components of your choice— just right click the component type, and choose a component from the drop down menu.



Technically, a new normal Entry component instance is created at strategy boot time for each instrument that is listed in the MarketManager component. This effectively limits the visibility of each component instance to the particular instrument that the component was created for. In contrast, only one CrossEntry or ATSCrossComponent is created for the whole strategy at boot time, so that single CrossEntry instance can “see’ (that is, will receive bars for) all instruments that are listed in the MarketManager component.

Programmatically, it is awkward to reference instrument BB from within a component that was instantiated to receive bars for only instrument AA. So if you want to reference bars for multiple instruments from within a single component, you should probably be using a cross component to make your programming life easier.

**BEWARE—by default, the IDE creates meta-strategy projects that do not work on the example data. This is because newly created projects use only Default\_XXX (that is, empty) components that do nothing.**

So to get a new strategy project to work, you must go through the component types and select working components. To begin with, try this set of components: Daily\_Simulation for the project

Simulation component, and Step\_Entry, Step\_Exit, Equity\_MarketManager, Flat\_MoneyManager, and EnableAll\_ReportManager. This combination will produce something useful to look at when you run the demo project strategies.

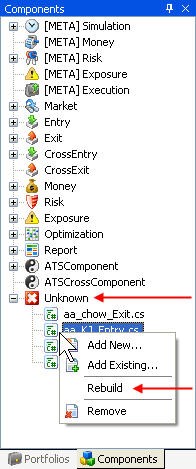
### Unknown Components

The IDE contains a component category named “Unknown” at the bottom of its component types list in the Component panel. Perhaps a better name would be “Broken Components,” because that’s what this component category contains—components that will not compile.

For example, if you right click the Entry category to create a new entry component, and then make a coding error in the new component and close its editor window before you fix the error, your component will apparently disappear into thin air. It will no longer be listed in the Entry category where it was before, with all the other Entry components.

In operation, the IDE detects broken components at boot time, and moves them into the Unknown category without telling you. So to fix the situation, you must open the Unknown category, double click the broken component, edit it to fix the code, and then right click the component filename in the Unknown category to rebuild the fixed component. If the rebuild is successful, the IDE will move the working component back into its proper category. If not, try to fix the component again.

Here is a picture of the IDE Component panel, with the Unknown components expanded, and with a right-click context menu shown. You must use the context menu Rebuild option to rebuild broken components.



## The CATS (Canned Automated Trading System) Application

The main purpose of the CATS system is to run precompiled strategies in a semi-unattended manner, in a standalone “production” CATS environment. Because CATS is not intended for strategy development, it does not store historical data in its database, and has no facilities for modifying strategy components (beyond supporting the usual strategy properties window to set strategy property values).

Instead, you use CATS as follows:

1. Export a working strategy from the IDE, for use in CATS.
2. Load the precompiled strategy component into CATS.
3. Connect CATS to a market data provider (either historical or real time).
4. Connect CATS to a trade execution provider (either simulation or live).
5. Run CATS to run the strategy.
6. Use CATS or the portfolio viewer to review strategy performance.

To repeat, the CATS system cannot store incoming market data streams into its database. Instead, the CATS system uses incoming market data for strategy execution immediately, and throws the market data away afterwards. There are no Data Importing menus on the CATS menu bar.

If you want to later investigate why CATS made a particular trading move in a live trading scenario, you must use the IDE or the DC to capture and store the real time market data feed that CATS is seeing. That way you can play the captured data back later, to help debug the strategy.

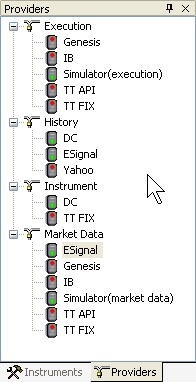
Here is a screenshot of the default CATS system, showing a choice of three loaded strategies. The results in the window are for a CATS2\_Crossover strategy run against Daily Cisco market data.



# External Providers to the SmartQuant System

The SmartQuant trading framework depends on several different kinds of service providers that provide four types of services—instrument definitions, market data feeds (historical and real time), and trade execution services.

Here is a screenshot of the IDE Providers panel, showing all four types of providers. Some of the provider icons show little green traffic lights, and some show red. Green icons indicate successful connections between the IDE and the provider.



## Instrument Providers and Configuration

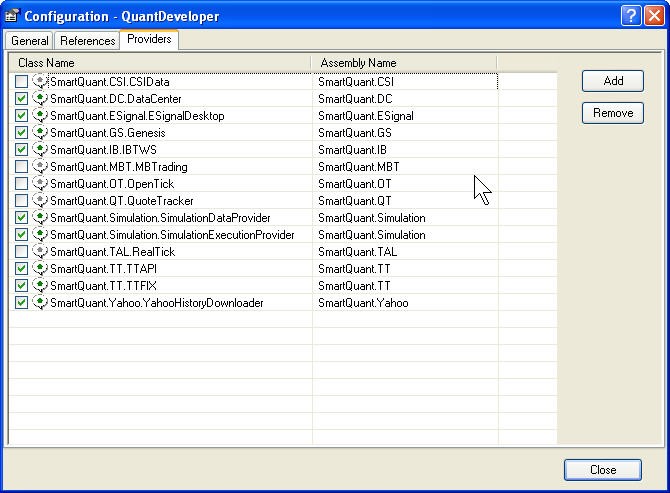
In a typical development scenario, strategy developers need to download fragments of market data for various instruments from a historical data provider (such as from the DataCenter). But often the developers must define more instruments in their IDEs before they can download from the DataCenter. Since it is tedious to repeatedly define instruments, SmartQuant products can download instrument definitions from Instrument Providers, just like downloading Bars from market data providers.

To download instrument definitions from the DataCenter into an IDE, connect the IDE to the DC by right clicking the DC icon in the Instrument section of the Providers window. Once the icon turns green to indicate connection, right click the icon and choose Security Definitions to download the definitions that interest you.

To download instrument definitions from the TT FIX provider, follow the same procedure. Connect the IDE to the TT FIX provider (be sure that your user name and provider properties are set correctly in the properties panel), and then right click the icon and choose Security Definitions, as before. You will need appropriate TT desktop API software and appropriate download permissions (subscriptions) in order to download instrument definitions from this commercial data provider.

If you generally use the same set of instruments all the time, you probably will not use Instrument providers very much. Once you manually define your favorite few instruments in the IDE, DC, and CATS, there will be no more work to do.

Here is a screenshot of the Tools / Configuration / Providers tab that shows a list of currently supported external providers. You can enable or disable providers that interest you by checking or unchecking the associated checkboxes in this dialog. This dialog is available in all SmartQuant applications (IDE, DC, and CATS).



In case your favorite provider is not listed in the configuration dialog, it is also possible to add support for a new provider by programming a new provider module in any .NET language that suits you. The SmartQuant system is thereby extensible in this way.

## Historical Market Data Providers

The main purpose of historical data providers is to make it easy for you to obtain historical market data for back testing your strategies. On the surface, it can look like historical day is pretty simple technically—it’s just the daily OHLCV (open high low close volume) data for an instrument, isn’t it? Well, kind of. There are some complexities to consider.

First, there are different kinds of historical data. Trade data specifies the time/price/volume of actual trades. Quote data specifies time/bid/ask/size of quotes, not trades. Bars provide OHLCV data for particular time periods, such as 60-second bars, or 300-second (5 minute) bars.

Second, good historical data has been cleaned up to remove bad or missing ticks, to ensure that data series are properly started and ended on particular market trading hour boundaries.

Third, good historical data can also be adjusted (normalized) for stock splits, currency devaluations, and other conditions that can affect the relationships between adjacent trades, quotes, or bars.

For reasons like these, good historical data is not as simple as it might first seem to be. The good news for most strategy developers is that they will usually not need to worry about the finegrained effects of historically accurate and conditioned data. Instead, most strategy developers can work with recent historical data of reasonable (but not perfect) quality.

SmartQuant currently supports three historical market data providers by default—Yahoo (daily for one year), ESignal, and TT, so only those providers will show up in the Providers panel by default. However, you can enable other providers by checking them in the configuration dialog, as explained above. Finally, it is also possible to write your own API interface code to other historical data providers.

If you use ESignal for historical data, you should probably request 60-second BAR data only— forum postings indicate that trying to download Trade or Quote data from ESignal will cause SmartQuant products to hang while waiting for a response.

## Real Time Market Data Providers

The main purpose of real time market data providers is to make it easy for your strategies to receive accurate market data in a timely manner. For strategies that want to respond immediately to developing market conditions, receiving data delayed by 20 minutes does no good at all.

There are two main ways of getting real time market data—from your trading execution provider (such as IB TWS), or from a separate third party market data provider (such as ESignal).

Recall that the IB Traders Workstation provides a free real time data feed over an API to SmartQuant products, as part of the normal TWS software that you get with your IB trading account.

In contrast, ESignal (because it is not a trading engine) charges you a subscription fee for both the normal ESignal market data feeds for charting in the ESignal application, plus a separate fee for the Desktop API service that delivers historical and real time data to SmartQuant products.

## Trade Execution Providers (Brokers)

The main purpose of trade execution providers (brokers) is obviously to execute trading orders that are generated by your automated trading system. Be aware that you should have very reliable Internet connectivity to your trade execution provider, lest your Internet service goes down in the middle of a delicate series of trades.

Currently Genesis Securities, IB (Interactive Brokers), and TT (Trading Technologies) are the only trading providers supported in the SmartQuant software distribution (2.1.5). As other providers are supported, they will appear in the Execution section of the IDE Providers panel.

# Summary and Conclusion

Hopefully after reading this document, you now have a much better understanding of the major structural elements and typical configurations of the SmartQuant framework, and how it can be effectively used to develop industrial-strength quantitative automated trading systems.

The main goal of this document was to identify and describe the main structural elements of the SmartQuant system, and the typical scenarios in which the configurations would be used. Typical human scenarios that were mentioned included how SmartQuant products could fit the trading system requirements of skilled individuals, small hedge funds, and large financial institutions.

Typical external providers such as Yahoo, ESignal, IB (Interactive Brokers), and TT (Trading Technologies) were discussed, along with descriptions of their major functions, subscriptions required, and use in typical operations such as download of market data, or trade executions.

Three main SmartQuant products were discussed—the IDE, the DataCenter, and the CATS system for executing ATS canned strategies that were exported from the IDE.

The following structural elements of the SmartQuant system were discussed:

* The two-part SmartQuant high-performance database
* Instruments and Providers,
* Strategy *project* types (ATS and MetaStrategy) and *component* types (Entry, Exit)
* Normal components versus Cross components,
* Project-level components versus Strategy-level components, and
* Various structural elements highlighted by red arrows on screenshots from the IDE, DC, and CATS applications.

The following typical system operations (and “gotchas”) were discussed:

* downloading historical and real time market data,
* making custom bar series from historical data,
* creating new strategy projects (ATS or MetaStrategy types),
* replacing the default “do nothing” components in a new project with ones that work,
* back testing (IDE) or live trading (IDE, CATS) a strategy.

Hopefully this short summary of concepts will help you to review what you’ve learned from this document, and help you to save many hours of time as you learn to use the SmartQuant system. Good luck, and good trading.