GOM 사용 설명서

유안타증권

목차

(아래 나열된 메뉴명 클릭 시 해당 페이지로 이동합니다)

1. GOM개요

가. 재미있는 COM 이야기

개요

COM 이 생겨난 배경과 정의에 대해 간단하게 설명합니다.

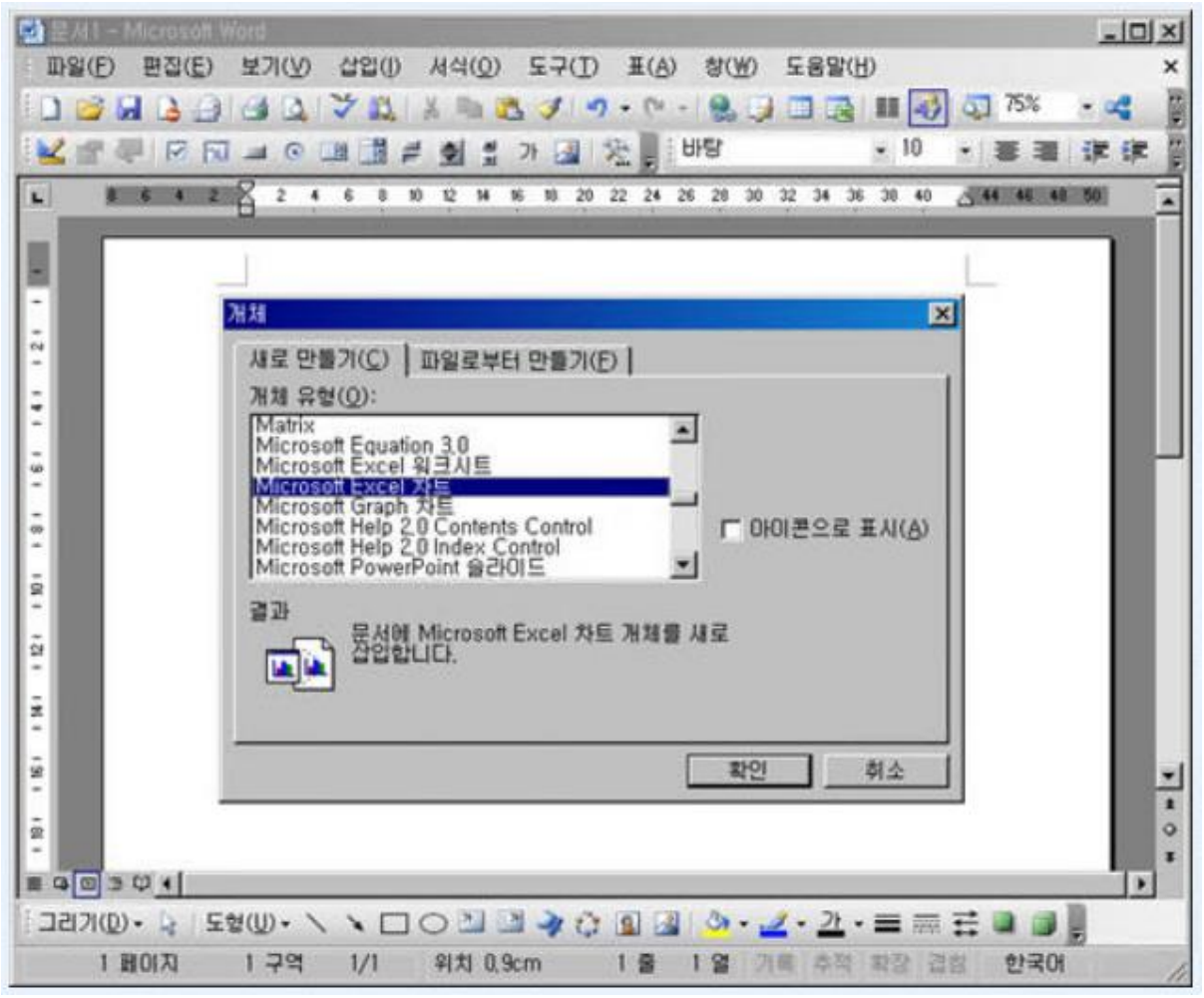
컴포넌트 프로그래밍

베이직을 아시나요? 마이크로소프트사에서는 GW-Basic, Quick-Basic 등 제품명에 Basic 을 붙여서 텍스트 기반 운영체제인 DOS(Disk-Operating System) 환경에서 사용되어 오다가 운영체제가 윈도우 환경으로 옮겨오면서 윈도우 버전인 Visual Basic 1.0 을 1991 년에 발표하였습니다. 상업용전의 베이직(BASIC:Beginner's All-purpose Symbolic Instruction Code)은 1964 년 미국 다트머스 대학에서 전산학을 전공하지 않은 비전공 학생들의 프로그래밍 교육을 목적으로 제작한 대화형 언어였습니다. MS 의 Visual Basic 역시 초보자들이 쉽게 프로그래밍을 할 수 있도록 한다는 취지는 그대로 였습니다. 이러한 Visual Basic 에서는 VBX 라는 것을 사용할 수 있었는데요. 예를 들면 옵션그릭스 계산기를 VBX 로 만들었다면 초보자는 이 VBX 를 추가하는 작업만으로 자신의 프로그램안에서 옵션그릭스 계산기를 활용할 수 있었지요. 굳이 자신도 옵션그릭스 계산기를 만들 필요가 없습니다. MS 나 또는 다른 써드파티 개발회사에서 VBX 를 개발해주면 사용하기만 하면 되었죠. 이런 VBX 가 OCX(OLE Cntrolextensions)라는 것으로 대체되었습니다. 왜 변해갔을 까요? VBX 처럼 컴포넌트화 시켜서 이를 Visual Basic 에서만이 아니라 다른 언어, 또는 엑셀이나 워드에서도 재활용성을 극대화하기 위해서입니다. 또한 기술적으로는 16bit 기준에서 32bit 기준으로 작성된다는 것이죠. Windows3.X 에서 Windows95 로 윈도우즈 플랫폼의 커다란 변화와 맞물린 겁니다.

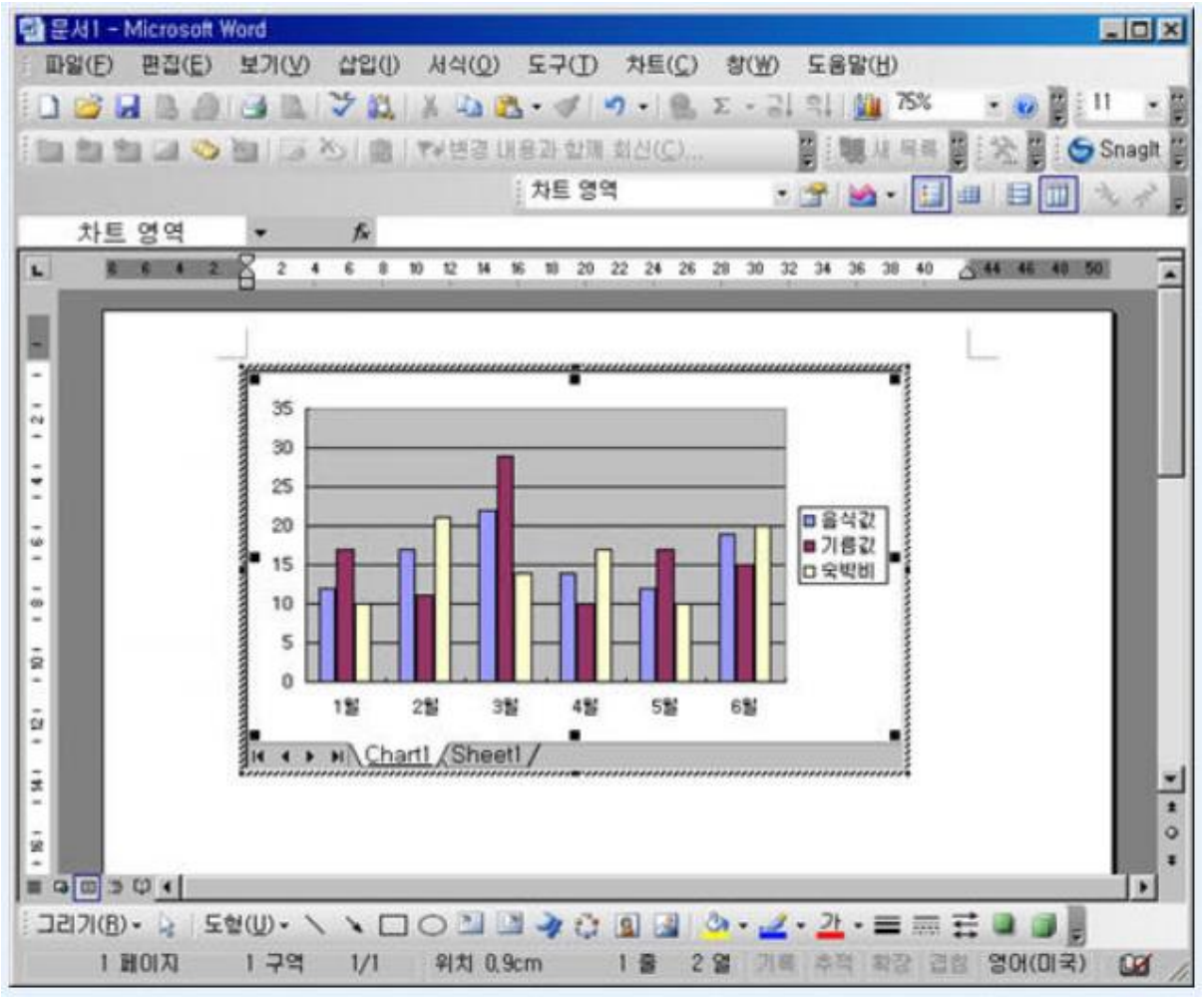
① OCX 는 무엇일까요?
이야기 되었듯이 OCX 는 기존의 16 비트의 VBX 대신 32 비트로 구성됩니다. 이 새로운 컨트롤 표준 방식은 기존에는 비주얼 베이직 프로그램에만 국한되어 있던 것을 윈도우 시스템 리소스로 서로 공유할 수 있도록 OLE 기능이 추가된 방식이라고 이해하시면 됩니다. 이처럼 OCX 가 윈도우 시스템 리소스의 공유를 기반으로 서로의 공유가 가능함으로 비주얼 베이직에서 만든 컨트롤을 다른 프로그램에서도 동일하게 사용할 수 있게 되었습니다. 자기가 필요한 부분의 컨트롤을 직접 제작하고, 다른 프로그램에서 이 컨트롤을 사용하는 것입니다. 이는 또한 윈도우뿐만이 아니라 인터넷이라는 거대한 망을 통해서도 공통으로 사용할 수 있는 길을 만들어 주는 방식이 되었습니다. 이것이 OCX 가 만들어지게 된 가장 중요한 이유라고 할 수 있습니다.

② ActiveX 란 무엇일까요?
ActiveX 는 단지 OCX 의 이름을 변경한 것에 불과합니다. 마이크로소프트에서 공식 발표한 ActiveX 라는 것은 기존의 OCX 와 별반 차이를 가지고 있지 않습니다. 다만 인터넷의 표준안에 해당하는 개념적인 의미에서 개명하면서 마이크로소프트의 목표를 알리기 위한 방법인거죠.

③ OLE(Object Linking Embedding)란?
OLE(Object Linking Embedding) : 대상이 되는 파일이나 프로그램을 연결하여 끼워 넣는 방법을 가리킵니다. 워드 프로그램에서 다른 프로그램에 있는 도표나 그림들을 개체 삽입을 통해서 끼워 넣는 방식이 OLE 의 한 대표적인 예입니다. 윈도우 3.1 로부터 표준 기능으로 제공되었습니다. OLE 에는 크게 링크와 임베딩의 기술이 있는데 이 중 링크는 원정보가 변경되면 그것을 사용한 문서에도 변경이 반영되는 것을 말하며, 임베드는 문서를 편집하는 경우 자동으로 데이터를 작성하는 프로그램이 실행되는 것을 말합니다.



[그림 1-1] MS 워드에서의 개체 삽입



[그림 1-2] MS 워드에서 엑셀챠트삽입

OLE 는 객체(Object)또는 어플리케에션이라고도 할 수 있는데 이 객체간의 데이타 교환문제를 해결하기 위한 노력의 결과물입니다. 기본적으로 어플리케이션과 어플리케이션은 각자가 독립적인 공간에서 독립적으로 실행되도록 되어 있습니다. 이러한 어플리케이션들간에 Ctrl+C 와 Ctrl+V 로 문자열을 서로 주고 받는 기술은 마법에 해당되는 일입니다. 어쨌거나 이처럼 어플리케이션간의 데이터 교환문제를 해결하기 위하여 Clipboard 와 DDE(Dynamic Data exchange)를 이용하였는데 Clipboard 는 사용하기가 간편해서 널리 사용되고 있지만, DDE 기술은 OLE 기술의 등장과 함께 점차 자취를 감추게 되었습니다. 1990 년대 후반에 나온 OLE 기술은 응용프로그램간의 Object 를 삽입하는 기술로서 현재까지도 광범위하게 사용되고 있으며 OLE 를 흡수하여 만들어진 것이 COM 입니다. COM 기술이 눈에 보이지 않는 아키텍쳐적인 성격이 강하다면, ActiveX 기술은 COM 기술을 비주얼한 환경으로 구체화시킨 것이라고 말할 수 있습니다.

④ 컴포넌트 프로그래밍

컴포넌트의 사전적인 의미는 '구성요소'를 이야기합니다. 그러면 컴포넌트 프로그래밍이란 무엇일까요? 예를 들어 자동차는 여러 가지 부품들이 모여서 하나의 자동차가 완성되듯이 프로그램의 여러 가지 부품의 역할을 하는 컴포넌트를 이용하여 하나의 프로그램을 제작하는 방법을 가리키는 것입니다. 현재와 같은 기능이 상당히 많은 프로그램을 제작할 경우 기간과 비용의 여러 가지 문제가 산재해 있기 때문에 프로그래머가 처음부터 모든 것을 제작할 수 없으므로 이런 문제들을 해결하는 방법이 바로 컴포넌트를 이용하는 프로그래밍인 것입니다. 조각조각 부품처럼 컴포넌트화하여 서로 합체하면 되니깐요. 그러기에 컴포넌트는 서로간의 상호 교류가 가능한 장점을 지니고 있습니다. 각각 다른 컴포넌트간의 교류와 하나의 컴포넌트 안에 다른 컴포넌트를 포함할 수도 있으며 외부에 있는 다른 컴포넌트와의 교류가 가능합니다. 그리고 기능이 변경될 경우에는 사용된 컴포넌트만 변경하면 구축되어져 있는 프로그램에 그 결과가 반영되는 여러 가지 장점을 가진 것이 컴포넌트를 이용한 프로그래밍의 장점인 것입니다. 컴포넌트는 단순히 재사용할 수있게 했다고 해서 컴포넌트화했다고 할 수는 없습니다. 예를 들면 DLL 에 오픈되어 있는 API 를 사용하면 DLL 을 로드만 하면 DLL 에 들어있는 기능을 재이용할 수 있지만 단순하게 이렇게 API 를 공유하는 방법은 컴포넌트라고 보기는 힘듭니다. 그 이유는 재이용 가능성이 떨어지기 때문입니다.부품화가 진행되려면 먼저 부품을 만든 이와 부품을 사용하는 이를 포함한 시장에서의 규격화가 진행되어야 합니다. 만약 그렇지않고 DLL 로 코드만 분리되는 정도라면 서브루틴을 다른파일로 저장했다는 것과 별반 다를 것이 없습니다. 컴포넌트가 되기위해서는 컴포넌트 사용자와 컨센서스가

이루어져야 합니다. 그러므로 컴포넌트라고 불리기 위해서는 MS 사가 제안하는 규격에 맞게 DLL 을 제작하거나 혹은 그러한 DLL 을 사용하여야합니다. 그 규격은 VBX, OCX, ActiveX 컴포넌트등으로 기술적 발전을 거듭하여 현재는 COM 으로 통일되었습니다. 이렇듯 오피스 프로그램끼리의 문서를 결합하기 위한 기술에서 시작하여 지금의 컴포넌트는 네트워크, 언어, 하드웨어상관없이 조립가능한 부품으로 거듭나고 있습니다. 컴포넌트의 조건 CBD 의 개념은 오브젝트 지향 프로그래밍, 즉, OOP 에서 왔습니다. 모듈설계, 캡슐화, 사양과 구현의 분리등의 OOP 의 기본개념은 CBD 에 그대로 계승되었습니다. 따라서 이 두 개념은 혼돈하기 쉽습니다만 컴포넌트는 오브젝트보다도 더 독립적이고 교환가능한 부품을 말합니다. 컴포넌트 프로그래밍에 대한 개념적인 이해를 돕기 위해서 OOP(Object Oriented Programming)에 대한 개념을 참고하세요.

COM

① COM 의 탄생

기본적으로는 멀티태스킹과 프로세스간의 통신을 위해 제안된 것들이 바로 DLL, RPC, DDE 와같은 개념입니다.

DLL(dynamic Link Library)	동적라이브러리를 이용한 프로세스간의 공유를 구현하는 방법으로 UNIX 에서 사용하는 공유라이브러리와 공유메모리와 같은 기능을 수행합니다, 현재의 윈도우를 구성하는 파일들의 상당부분을 차지하고 있으며 사용자가 직접 제작하여 사용할 수도 있습니다.
RPC(Remote Process Communication)	프로세스간의 원격호출을 구현하는 방법으로 다른 운영체제에서도 사용되고 있습니다, 요즘 분산환경기술의 발전과 더불어 개념적인 향상을 이루고 있습니다.
DDE(Dynamic Data Exchange)	프로세스간의 데이타를 동적처리를 구현하는 방법으로 기존의 제작되어 있는 프로그램의 프로세스와 새로운 응용프로그램간의 데이타 교환에 주로 사용되었습니다.

COM 에 대해서 이야기 하자면 먼저 DDE 와 OLE 를 말하지 않을 수 없습니다. DDE 는 클립보드를 사용하는 것처럼, Windows 에서 일부 데이터를 직접 교환할 수 있도록 하는 것으로 DDE 프로그래밍을 해본 경험이 있는 사람들은 알 수 있지만 윈도우에서 제공하고 있는 DDE 를 통해 데이터의 교환을 수행하면 속도와 데이터의 손실의 문제가 발생합니다. 이에 OLE 로 발전되는데 이는 OLE 서버 프로그램의 데이터를 OLE 클라이언트 프로그램으로 복사할 수 있는 것입니다. 간단하게 설명하자면 OLE 컨트롤을 저장할때 서버 애플리케이션측에 링크모드데이타가 저장되고 데이터의 링크 정보만이 삽입될 곳의 파일에 보관됩니다. 임베드 모드 데이터는 OLE 컨테이너 컨트롤측에서 관리되는데 작성된 모든 데이터는 삽입될 파일에 보관됩니다. 서버 애플리케이션에 있는 데이터는 여러 애플리케이션에서 참조할 수 있지만 컨테이너 컨트롤측의 데이타는 다른 애플리케이션에서 참조할 수 없습니다. MS 사는 Windows3.1 OLE 를 OLE2 로 업데이트하면서 OLE Automation 과 OLE Controls 의 기능을 추가했으며, Windows95 의 Shell 프로그램인 탐색기(Explorer.exe)는 OLE2 기술을 기반으로 해서 구현했습니다. 그리고 OLE Controls 는 인터넷 확장 가능하도록하면서 ActiveX Controls 로 변경합니다. 최근에 OLE 기술은 MS 사의 DNA 프로젝트와 함께 COM 으로 명명되면서 sinks, in-place activation, structured storage 등이 추가되며, Windows2000 에 가서는 COM 의 서비스와 기능을 확장한 COM+로 진화하게 됩니다.

② COM 의 구조

COM 의 일반적인 구조는 Win32 DLL 형식과 EXE 형식으로 구성됩니다. DLL 과 EXE 로 구성되기 때문에 특정언어에 국한되는 것이 아닌 언어 독립적이며, 바이너리 형태로 여러 가지 프로그램과 프로그래밍 툴에서 사용할 수 있다. COM 의 실제 사용에서는 COM 에 포함되어 있는 클래스 메서드를 실행시키기 위한 인터페이스의 사용이 가장 중요한 부분이 됩니다. 이런 COM 은 윈도우 레지스트리에서 서버로 등록되어 외부 프로그램에서 클라이언트로 COM 을 불러들여 사용할 수 있는 서버의 기능을 수행하는 것입니다.

③ COM 인터페이스

컴포넌트는 여러 객체로 구성이 됩니다. 이 컴포넌트를 사용하기 위해서는 속해있는 객체가 어떤 데이타를 가지고 있고 어떤 함수가 있는지를 알아야 합니다. COM 에 속해있는 객체마다 프로퍼티와 메쏘드를 알리기 위한 방법으로 사용자에게 인터페이스(Interface)를 제공합니다. 사용자는 사용자의 프로그램에 COM 에서 제공되는 인터페이스를 Import 하고 인터페이스를 통하여 COM 내부의 객체를 사용합니다. COM 의 개체간 통신에는 기본적으로 두 가지의 인터페이스인 IUnknown 과 IDispatch 가 사용됩니다.

IUnknown	모든 COM 개체는 IUnknown 인터페이스를 반드시 구현해야 합니다. IUnknown 인터페이스는 세 가지 기본적인 메서드인 QueryInterface, AddRef, Release 를 정의하는데 AddRef 와 Release

	메서드는 개체의 생성, 소멸을 관리하며 COM 개체 자신의 참조회수를 증가시키거나 감소시키는 기능을 가지고 있습니다. 참조 회수가 0 으로 되면 그 개체는 소멸하게 되는 것입니다. QueryInterface 는 특정 인터페이스에 관하여 그 개체에 질의를 던지는 역할을 하고 있습니다. 사용하고자 하는 컴포넌트의 인터페이스의 정보를 알아낼 수 있는데 이 메서드는 질의된 인터페이스에 대한 식별자를 인자로 받아서 그 인터페이스에 대한 간접포인터를 리턴 값으로 돌려주게 됩니다. 만약 요청된 인터페이스가 그 개체가 제공하지 않는 것이라면 에러가 발생합니다.
IDispatch	IDispatch 인터페이스는 자동화(Automation)을 지원하기 위해 IUnknown 인터페이스로부터 파생된 인터페이스입니다. 메서드나 속성에 해당하는 디스패치 식별자를 알아내어 자동화 컴포넌트에서 제공하는 속성이나 메서드에 접근하는 등의 작업을 할 수 있습니다. 초기의 바인딩을 사용하여 개체에 접근하고자 할 때에는 IUnknown 인터페이스가 사용되며, 후기의 바인딩을 사용하여 개체에 접근하고자 할 때에는 IDispatch 인터페이스가 사용됩니다.

④ Automation

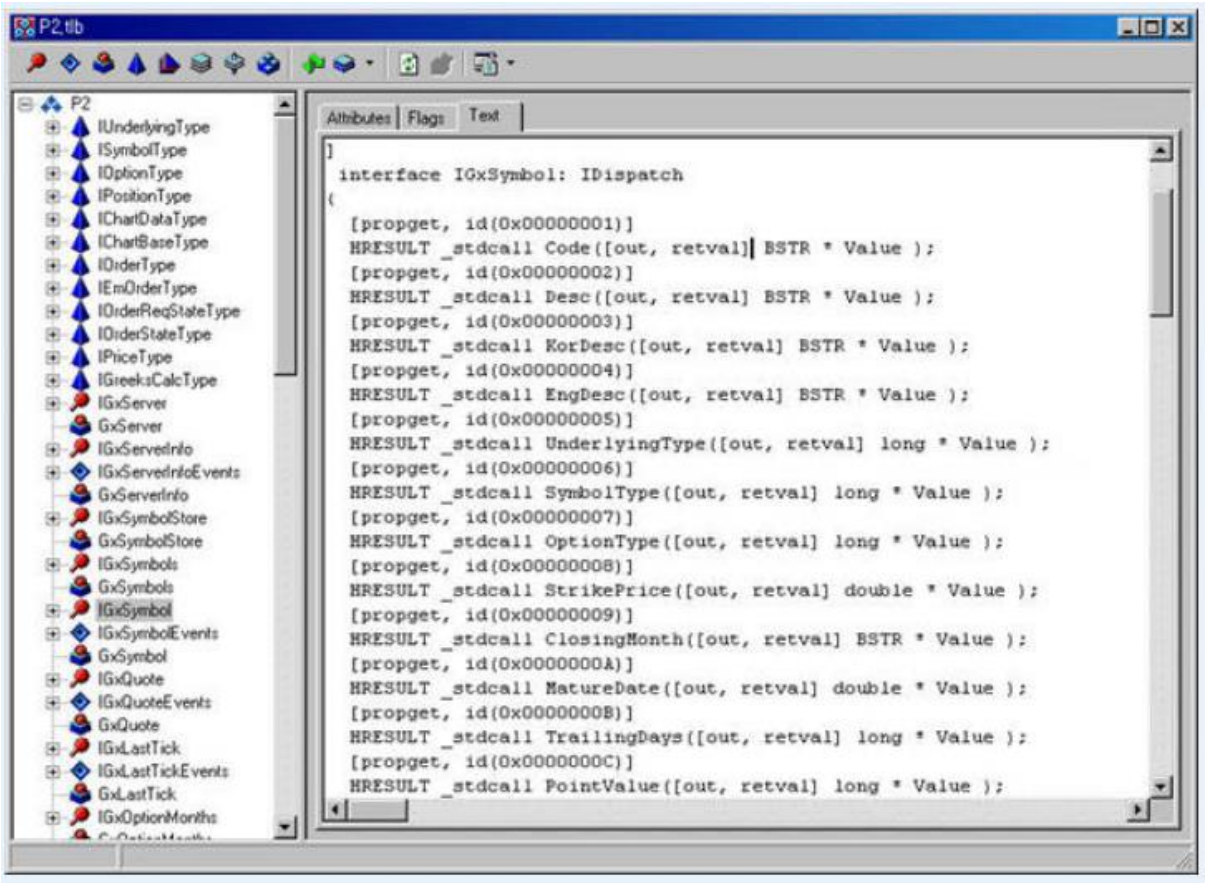
Automation 은 사용자들이 기존의 프로그램의 내용과 기능의 장점을 이용할 수 있게 하는 기술로써 인터페이스에 기반을 둔 표준 소프트웨어 아키텍처인 COM(Component Object Model)에 기반을 두고 있습니다. 그리고 Automation 은 서버와 클라이언트로 구성되는데 Automation 클라이언트를 흔히 Automation 컨트롤러라고도 합니다. Automation 서버는 서비스를 제공하고 Automation 컨트롤러는 서버에 연결하여 서버의 기능을 사용하게 됩니다. 고수 Plus 는 서비스를 제공하는 Automation 서버가 되고 여러분이 작성하시게 될 GOM 클라이언트는 Automation 컨트롤러가 됩니다.

Automation 에서 객체들의 속성과 메소드를 사용하는 방법에는 두가지 방법이 있습니다. 하나는 dispatch 인터페이스를 이용하는 것이고 하나는 vtable 을 이용하는 것입니다. 이 둘은 바인딩(binding)의 차이인데요. 바인딩은 사전적인 의미로 "동여매는 묶는"의 의미로 컴포넌트를 언제 동여묶어서 그들을 사용할 수 있는가 입니다. dispatch 인터페이스를 사용하는 후기바인딩(late binding)과 vtable 을 이용하는 초기바인딩(Early Binding)이라고 합니다.

초기바인딩(Early Binding)	vTable 바인딩, 반드시 타입라이브러리 제공 dim obj As AddBack Set obj = New AddBack obj.someMethod 인터페이스 주소는 인터페이스가 지원하는 함수들에 대한 주소값 테이블을 가리킵니다. 이 테이블을 vtable 이라고 합니다. COM 객체는 클라이언트에 객체의 vtable 를 제공함으로써 객체에 대한 직접적인 접근을 제공 합니다. 쉽게 말하자면 클라이언트를 작성하는 단계에서 컴포넌트 정보를 내포(import)하여 클라이언트 내부객체처럼 사용하는 방식을 Early Binding 이라 합니다.
후기바인딩(Late Binding)	IDispatch 인터페이스 지원 dim obj As Object Set obj = CreateObject("AddBack.AddBack.1") : obj.someMethod dispatch interface (dispinterface)는 구성 메소드와 속성은 DispID 라고 부르는 고유한 ID 로 식별 됩니다. 사용자는 IDispatch 인터페이스의 GetIDOfNames 를 통해 메소드나 속성의 DispID 를 구하고 이를 이용 Invoke 를 호출하여 메소드나 속성을 호출합니다. 이 방식을 Late Binding 이라 합니다. Late Binding 방식은 실행시 타입을 결정할 수 있는 장점이 있으나 속도가 늦어지는 결점이 있고 vtable 은 속도가 빠르고 컴파일 시 문법 체크가 가능하나 실행시 타입을 결정할 수 없는 단점이 있습니다. 그래서 Microsoft 에서는 이 두 가지 방식인 Early Binding, LateBinding 모두를 지원하도록 하는 것을 권하고 있습니다. 이러한 방식을 dual interface 라 합니다. 모든 고수 Plus 의 GOM 객체들은 이 dual interface 를 지원하고 있으므로 사용자는 사용언어가 지원한다면 Early Binding, Late Binding 방식 모두 사용할 수 있습니다.

⑤ 타입 라이브러리(Type Libraries)

타입 라이브러리는 COM 객체의 기능에 관한 정보를 제공하는 파일 입니다. 자세하게 말하면 객체를 구성하는 클래스들의 정보를 포함한 파일입니다. 타입 라이브러리는 실제 객체들을 저장하는 것이 아니고 객체들의 정보만을 저장합니다. GOM 에서도 p2.tlb 라는 타입 라이브러리를 제공하며 이 파일은 자료실에서 다운 받으실 수 있습니다. 타입 라이브러리를 보는 방법은 각 개발툴에서 제공하고 있습니다. VB 에서는 Object Browser, VC 에서는 OLE/COM Object Viewer, Delphi 는 Delphi IDE 자체에서 TLB 파일을 보실 수 있습니다.



[그림 2-1] Delphi 에서 보는 타입라이브러리

지금까지 COM 에 대한 간략한 소개로 이야기처럼 읽어보시면 됩니다. 초보자에게는 생소한 컴퓨터용어이고 어려운 개념임에는 틀림없습니다만 충분히 습득하셔서 자신의 노하우를 담은 전략을 만드시길 바랍니다. 다음은 GOM 에 대한 이야기입니다.

나. 쉬운 GOM 이야기

개요

매매하시는 분들은 보는 관점에 따라 또는 시장상황에 따라 달라지는 Factor를 바로 적용할 수 있는 전략을 구축하고자 합니다. GOM은 고수Plus의 시세/계좌/챠트등 내부데이타를 제공함으로써 자신만의 노하우를 담은 전략만들어 주문을 낼 수 있도록 도구를 지원하자는 취지에서 만들어 졌습니다. COM을 모른다고 GOM을 사용하지 못하는 것은 아닙니다. 물론 기본지식을 가지고 있다면 더 좋겠지만 기본적으로는 COM이든 COM을 기반으로 만든 GOM이든 컴포넌트의 개념상으로는 사용법만 알면 쓸 수 있게 만드는 것입니다. 개념적인 이해가 힘들더라도 사용법을 익힌다는 개념으로 따라오시고 따라하기등의 예제를 통해 실제로 작성하시면서 연습하시면 됩니다.

GOM 구조 이해하기

GOM의 구조를 이해하기에 앞서 GOM 전체 구조를 눈에 익혀 봅니다.

고수의 공개된 객체들 즉 GOM 은 위의 그림처럼 트리구조로 구조화되어 있습니다. GxServer 를 최상위객체로 하여 그 아래로 크게는 GxSymbolStore, GxTradeStore, GxChartStore 가 있고 각각의 하위구조를 갖고 있습니다. 이 구조를 눈에 익혀야 하는 이유는 다음과 같습니다. 예를 들어 계좌의 예탁금을 알고 싶다면 GxServer 객체를 얻은 후 이를 통해 이 객체의 GxTradeStore 를 얻을 수 있고 다음으로 GxAccounts 와 GxAccount 의 객체에 접근하여 해당되는 계좌의 예탁금을 비롯하여 증거금, 주문가능금액등을 가져올 수 있음을 알기 위해서 입니다. 아래의 코드상으로 이해를 해보면 ⓐ,ⓑ,ⓒ 객체를 담을 변수를 선언하고 GetObject 함수로 GOM 에 연결하여 계좌정보를 가지고 오는 코드로 간단하게 작성할 수 있습니다. 이렇듯 GOM 에 대한 트리구조를 보시면서 계좌정보를 원한다면 GxTradeStore 에게 정보를 달라고 하면 되겠구나를 이해하시면 됩니다.

```
Option Explicit
Public mobjServer As GxServer --- ⓐ 'GOM 서버의 Root Object와 연결할 객체
Public mobjTradeStore As GxTradeStore --- ⓑ 'TradeStore와 연결할 객체
```

```
Private Sub cmdExcute_Click()
Dim objAccount As GxAccount --- ⓒ' 계좌를 담을 객체

On Error GoTo DoFail
Set mobjServer = GetObject(, "P2.GxServer") 'GOM 서버의 Root Object 연결
Set mobjTradeStore = mobjServer.TradeStore 'TradeStore를 연결시켜준다.

GoTo DoSuccess
Exit Sub
```

```
'서버 연결 실패시 메세지 출력후 끝냄
DoFail:
MsgBox "서버와 연결하는데 실패 하였습니다"
Exit Sub

DoSuccess: 'Combo Box 초기화
cboAccount.Clear
'TradeStore의 계좌목록을 Combo Box에 넣어준다.
For Each objAccount In mobjServer.TradeStore.Accounts
cboAccount.AddItem objAccount.Code
Next

'기본적으로 첫번째 계좌를 선택합니다.
If cboAccount.ListCount > 0 Then
cboAccount.ListIndex = 0
End If
End Sub
Private Sub cmdStop_Click()
mobjServer = Nothing
End Sub
```

이와 같은 객체간의 관계를 이해해야 하는 것은 GOM 을 이용하는데 필수적입니다.

GOM 객체의 구성요소

GOM의 객체에는 속성(property)과 메소드(method) 그리고 이벤트(Event)로 구성되어 있습니다.

속성은 말그대로 객체의 속성을 의미하고 메소드는 객체가 어떠한 행동을 수행하게 하는 함수입니다. 또한 이벤트는 어떠한 상황이 발생되면 알려주는 기능을 합니다. 예를 들어 GxChartData 객체에는 서버로부터 챠트데이타를 요청하는 Define이라는 메소드가 있습니다. 이 Define 함수를 선물최근월물 1분봉을 100개 가지고 와라하고 호출하면 Define함수는 Symbol, Base, Period등의 속성값들을 변경하고 서버에 데이타를 달라고 요청을 하고는 제 할일을 마칩니다. GOM서버 즉 고수Plus에서는 Define의 요청을 받고 데이타를 수집하여 마침내 클라이언트에게 전달합니다. 이 때 GxChartData 객체에서는 챠트데이타가 서버로부터 도착했다라고 OnDatarefreshed라는 이벤트가 발생하게 됩니다. Define함수로 서버에 요청을 했던 클라이언트는 왔구나하고는 인지하고는 챠트자료가 준비되었다라는 Ready속성을 True로 만들고 챠트 그림을 그리게 해야지 또는 파일에 저장해야지를 작성하게 됩니다.

GOM 구조도를 보면 GxChartData를 사용하기 위해 GxChartStore 객체를 얻어야하므로 아래와 같이 선언하는 예입니다.

```
Option Explicit
'GOM Server 연결 Object
Public mobjServer As GxServer

'GxChartStore 연결 Object
Public mobjChartStore As GxChartStore
```

다음과 같이 이벤트 구독을 위해 준비하는 예입니다. 이벤트구독을 위한 자세한 절차는 따라하기의 챠트부분을 참조하시기 바랍니다.

```
Option Explicit

'여러개의 종목들에 대해 챠트 데이타의 Event를 받기 위해서 클래스로 만들어주었다.
```

```
Public WithEvents mobjChartSink As GxChartData
```

GOM의 최상위객체 얻기

MS-Excel 의 경우 사용자는 WorkSheet 객체의 하위 인터페이스를 바로 얻을 수 없고 최상위 인터페이스를 통하여 하위 객체 인터페이스를 얻습니다. 이와 같이 GOM 객체를 사용하려면 꼭 최상위 객체의 인터페이스인 IGxServer를 얻어야 합니다.

이 최상위객체를 얻기 위해서는 여러가지 방법이 있을 수 있습니다. 앞서 말씀드렸듯이 GOM은 초기바인딩(Early binding)과 후기바인딩(Late binding)을 모두 사용할 수 있는 Dual Interface를 지원합니다. 이 말은 개념적으로는 type을 프로그램 링크하는 단계에서 확정하는 방법과 실행시 타입결정을 한다는 의미이지만 사용법관점으로만 이해한다면 GOM의 객체를 얻기 위해서 타입라리브러리를 import하여 최상위객체를 얻는 방법과 VB에서의 CreateObject GetObject, 델파이의 CreateOleObject GetActiveOleObject, MFC의 CreateDispatch, WinAPI GetActiveObject등을 통하여 최상위객체를 얻는 두가지 방법을 모두 지원한다는 뜻입니다.

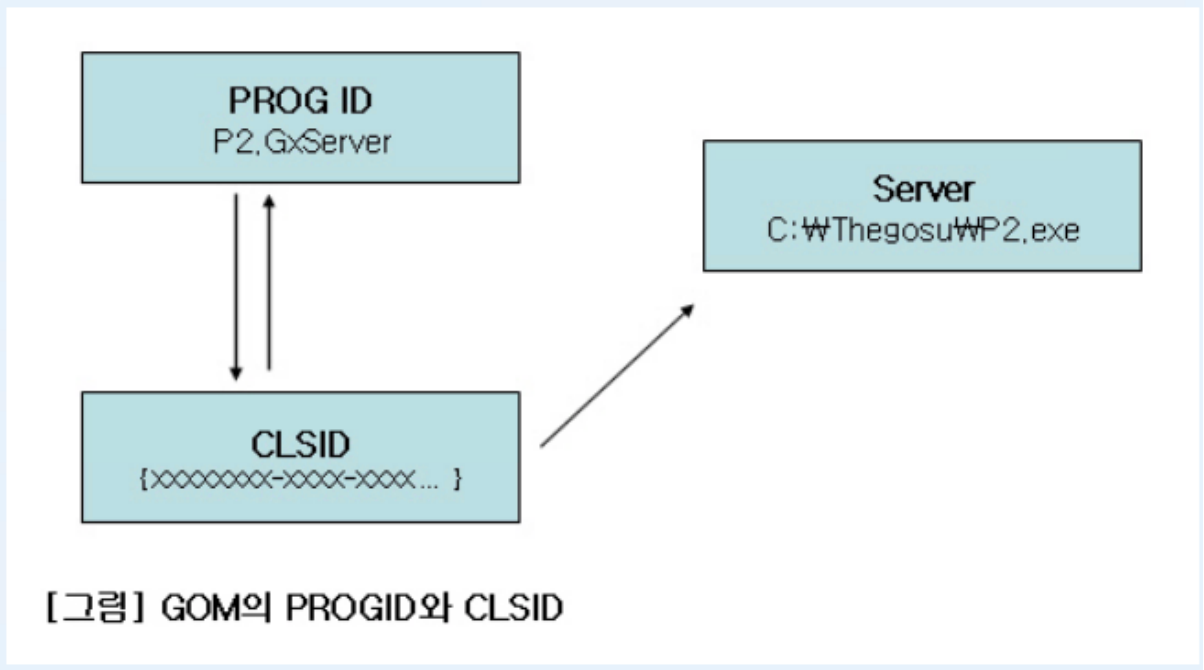
Type Library로 최상위객체 얻기

TypeLibrary Import 하기
먼저 인터페이스의 타입들의 명세서인 p2.tlb파일을 import해야만 인터페이스를 참조할 수 있습니다.
tlb파일을 import 하는 방법은 개발툴(Visual C++, Visual basic, Delphi...)마다 다르므로 객 개발툴의 도움말을 참조하시거나 [공유하기]-[각 개발언어의 클라이언트] 만들기 도움말을 참조하시면 됩니다.

Delphi Sample

```
var
i : Integer;
aAccount : IGxAccount;
aGxServer : TGxServer;
aGxTradeStore : IGxTradeStore;
aGxAccounts : IGxAccounts;
begin
aGxServer := TGxServer.Create(Self);
aGxTradeStore := aGxServer.TradeStore as IGxTradeStore;

aGxAccounts := aGxTradeStore.Accounts as IGxAccounts;

for i:=1 to aGxAccounts.Count do
begin
aAccount := aGxAccounts[i] as IGxAccount;
Memo1.Lines.Add('early : ' + aAccount.Code);
end;
end;
```

IDispatch Type의 최상위객체 얻기



[그림] GOM의 PROGID와 CLSID

PROGID
GOM Server는 Windows의 시스템 레지스트리에 모든 GOM class들을 등록 합니다. 각각의 GOM class는 GUID라는 class 식별자를 가집니다. 이를 CLSID 이라 합니다. 레지스트리에는 각각의 CLSID가 프로그래밍적인 식별자인 PROGID 와 연결되어 있습니다. GOM 사용자는 PROGID를 사용하여 GOM의 최상위 Object를 생성하거나 얻을 수 있습니다. GOM 최상위 Object의 PROGID는 'P2.GxServer' 이며 VB에서의 CreateObject GetObject, 델파이의 CreateOleObject GetActiveOleObject, MFC의 CreateDispatch, WinAPI GetActiveObject등에서 이를 PROGID를 사용 합니다.

Delphi Sample

```
var
i : Integer;
stmsg : string;
aGxServer, aAccounts, aAccount : Variant;
begin
try
aGxServer := GetActiveOleObject('P2.GxServer');
except
aGxServer := CreateOleObject('P2.GxServer');
end;

aAccounts := aGxServer.TradeStore.Accounts;
for i:=1 to aAccounts.Count do
begin
aAccount := aAccounts.Item[i];
Memo1.Lines.Add('late : ' + aAccount.Code);
end;
end;
```

대부분의 GOM 객체는 최상위객체를 생성하는 단계에서 같이 생성이 되어집니다. GxTradeStore와

GxTradeStore, GxSymbolStore, GxServerInfo를 포함하여 Collection객체의 경우는 자동 생성됩니다. 그리고 프로그램 종료 시까지 새로 추가되거나 제거 되지 않습니다.

Collection의 구성원이 되는 객체의 경우는 필요시마다 생성 시킬 수 있습니다. 예를 들어 사용자가 새롭게 주문을 낼 경우 GxOrder, GxConfirm, GxFill Object등이 GxOrders, GxConfirms, GxFills등 Collection 객체에 추가됩니다. 이는 사용자가 직접적으로 수행할 수 없고 주문을 통해서 간접적으로 생성 시킬 수 있습니다.

GxTradeStore 의 하위 GxOrderHandler Object의 경우는 IGxOrderHandler의 PutNewOrder, PutChangeOrder, PutCancelOrder등의 메소드를 통해 GxOrderHandler 하위의 GxOrderReqs Collection Object에 GxOrderReq Object를 간접 생성 시킬 수 있습니다.

GxChartStore Collection Object도 GxOrderHandler와 비슷하게 IGxChartStore의 Add 메소드를 통해 새롭게 GxChartData Object 를 생성 시킬 수 있고 Remove, RemoveByKey 등의 메소드를 통해 GxChartData Object 등을 제거 할 수도 있습니다.

Collection 객체 사용하기

Collection 객체는 구성원들을 여러개 가지고 있는 집합객체를 말합니다. GOM의 전체구조도를 보시면 초록색으로 표시한 객체들은 Collection객체입니다. 이는 기차를 연상해보시면 됩니다. 기차 한대에 1호차, 2호차...10호차까지 연결되어 있습니다. 특실도 있고 식당칸도 있고 일반실과 같이 속성은 좀 다르겠지만 기차 한칸의 공간적인 의미는 같습니다. 이렇듯 챠트 데이터를 관리하는 GxChartStore경우 1번 GxChartData, 2번 GxChartData ...n번 GxChartData를 가지고 있습니다.

Automation에서의 Collection Object 권고사항은 IEnumVariant 인터페이스를 구현하며 Add, Remove 메소드가 존재하며 _NewEnum, Item, Count 프로퍼티가 존재 해야 한다는 것입니다. 모든 GOM Collection 객체는 _NewEnum, Item, Count를 지원하며 이를 통해 VB, VBA 등에서 For - Each 문, 다른 언어에서도 for 문을 이용하여 이 집합체의 객체들을 모조리 조회할 수 있습니다. 그러나 Add, Remove 함수의 경우는 GxChartStore에서만 구현되어있습니다. 이 Collection객체의 속성들을 살펴보면 _NewEnum은 Enumeration 인터페이스인 IEnumVariant를 얻을 수 있는 속성입니다. 이는 Collection과 같은 형태의 객체는 표준화된 인터페이스를 지원하도록 되어있는데 이를 지원하는 속성이라고 보시면 됩니다. Count속성은 기차에 몇호차까지 있는지를 알 수 있는 속성입니다. 이처럼 Collection에 몇개의 구성원이 있는지를 알려줍니다. 다음으로 Item 속성은 Collection에 포함되어 있는 개별구성원의 인터페이스를 반환합니다. 기차로 다시 설명하자면 Item[1]하면 1호차의 객체인터페이스를 Item[2]는 2호차의 객체인터페이스를 반환합니다.

Item 프로퍼티에 입력 파라미터는 Variant 타입이므로 여러 타입의 입력이 가능합니다만 GOM의 모든 Collection 객체마다 허용되는 타입이 지정되어 있습니다. 모든 GOM Collection 객체는 long type을 지원하는데 long 입력값 번째의 인터페이스를 반환할 때 사용됩니다. long 타입의 경우 1 ~ (Collection 자체의 'Count' 프로퍼티값)까지 입력해야 합니다. 해당하지 않은 형식을 입력하거나 범위를 벗어나면 NULL이 반환 되므로 주의 하여야 합니다. 또한 예외로 GxSymbolStore의 경우는 long 타입이외에도 BSTR 형식도 지원합니다. Item 파라미터에 "10146"을 입력할 경우 Code 가 "10146"인 GxSymbol의 IGxSymbol 인터페이스를 반환합니다. GxStrikePrices의 경우도 long 값이외에 double값을 지원합니다. Item 파라미터에 120.50 을 입력할 경우 행사가가 120.50 인 IGxStrikePrice을 반환합니다. 그리고 Item 프로퍼티는 기본 속성 이므로 VB의 경우 SymbolStore.Item("10146") 대신 SymbolStore("10146")으로 사용 가능합니다. 이 기본 속성은 다른 Language 에서도 이용하여 코드를 간결하게 할 수 있습니다.

```
var
i : Integer;
aAccount : IGxAccount;
aGxServer : TGxServer;
aGxTradeStore : IGxTradeStore;
aGxAccounts : IGxAccounts; --- ' Collection 객체
begin
aGxServer := TGxServer.Create(Self);
aGxTradeStore := aGxServer.TradeStore as IGxTradeStore;
```

```
aGxAccounts := aGxTradeStore.Accounts as IGxAccounts;
for i:=1 to aGxAccounts.Count do --- ' 1~count만큼 item가지고 오기
begin
' aGxAccount.item[i]를 간결하게 표현
aAccount := aGxAccounts[i] as IGxAccount;
Memo1.Lines.Add('early : ' + aAccount.Code);
end;
end;
```

IDispatch형 속성 사용하기

속성에는 보통 long(정수), double(실수), BSTR(문자열)등 일반 데이터형이지만 객체가 속성이 되는 경우도 있습니다.
이때 객체속성의 경우는 IDispatch형으로 제공되므로 실제 인터페이스 타입으로 형변환하여 사용하셔야 합니다.

GxPosition

Fields			
ID: 1	IDispatch	Account	포지션의 계좌(IGxAccount)
ID: 2	IDispatch	Symbol	포지션의 종목(IGxSymbol)
ID: 3	long	Qty	최종 포지션 수량(매수: 양수, 매도: 음수)
ID: 4	long	PrevQty	전일 최종 포지션 수량
ID: 5	long	TodayQty	당일(변동) 포지션 수량
ID: 6	double	AvgPrice	평균 단가
ID: 7	double	EvalPL	평가 손익
ID: 8	long	ShortOrderQty	매도 미체결 주문 수량
ID: 9	long	LongOrderQty	매수 미체결 주문 수량
ID: 10	BSTR	LastMessage	최신 에러 정보

예를 들어 위의 GxPosition 객체를 사용하여 어떤 미결제약정에 해당하는 계좌의 증거금을 알고자 할 때 Account 속성을 참고하여 IGxAccount를 얻은 후 IGxAccount의 증거금 속성을 호출해야 합니다. 이 때 GxPosition 속성중 Account의 IDispatch 속성을 IGxAccount 타입으로 형변환하여 계좌의 증거금을 사용합니다.

다음은 델파이의 예입니다.

```
var
aGxAccount : IGxAccount;
begin
aGxAccount := FGxPosition.Account as IGxAccount;
Edit1.Text := Format('%n', [aGxAccount.Margin]);
end;
```

이벤트 수신하기

GOM 객체의 일부는 Client에 이벤트를 발생 시킵니다. 이와 같은 객체를 Connectable Object라고 부릅니다. GOM구조도에서 테두리가 빨간색으로 표시된 객체들은 이벤트 수신을 할 수 있습니다. 서버에서 참조하는 값이 변할 경우 클라이언트에 이벤트를 발생시킵니다. 클라이언트는 이벤트를 받아 이벤트의 종류에 따라 자신이 추적하고 있는 값(예:A종목의 현재가)을 갱신합니다.

클라이언트가 A종목의 현재가를 화면에 실시간으로 표시한다면 클라이언트는 A종목의 인터페이스를 찾아 현재가 속성을 호출하고 이에 대한 종목 현재가 이벤트를 요청합니다. 그 이후 서버로 부터 현재가 변경 이벤트를 수신하면 수신 시점으로 다시 A종목의 인터페이스를 통해 다시 현재가 속성을 읽습니다.

클라이언트는 이벤트를 수신하기 위해서 이벤트 수신 대상 객체인 EventSink 객체를 만들어야 합니다. 그리고 그 전에 서버에게 이벤트 요청을 해야 합니다. EventSink를 제작, Event 요청 방법은 각 Lanuage 마다 틀리므로 Language 별 클라이언트 제작 시 이를 다시 다룹니다.

GxServerInfo 사용하기

클라이언트가 COM Server 즉 고수Plus 서버와의 연결상태등을 알기 위해서는 GxServerInfo 객체를 사용해야 합니다. GxServerInfo 객체를 사용하기 위한 기본 인터페이스는 IGxServerInfo 이고 'GOM 구조도'를 보면 GxServerInfo 객체는 최상위객체의 한 단계 아래의 객체입니다. IGxServerInfo 인터페이스는 최상위 IGxServer에 'ServerInfo' 프로퍼티 호출을 통해 얻을 수 있습니다. IGxServerInfo 의 인터페이스를 얻은 후 IGxServerInfo 의 ServerDate 등의 프로퍼티를 사용할 수 있습니다.

다. GxSymbolStore

GxSymbolStore 구조

GOM 에서 종목정보를 총괄하는 GxSymbolStore 와 그 하위 객체의 사용법에 대해 알아봅니다.

클라이언트가 GOM 의 종목 정보를 사용하기 위해서는 GxSymbolStore 와 그 하위의 객체를 사용해야 합니다. GxSymbolStore 객체는 최상위 GxServer 객체의 자식으로써 IGxServer 의 'SymbolStore' 속성을 호출하여 IGxSymbolStore 를 얻을 수 있습니다.

GOM 에서 공개하는 종목은 KOSPI200, KOSPI 선물, 스프레드, 옵션, 주식 5 종목등 입니다. 언급한 종목 하나 하나가 GxSymbol 객체이고 이 모든 GxSymbol 객체를 GxSymbolStore 가 관리합니다. 한 마디로 GxSymbolStore 는 GxSymbol 들을 관리하는 Collection 객체 입니다.

GxSymbolStore 는 Collection 객체로써 For Each, For 등을 사용할 수 있습니다. Item 속성은 정수형일 경우

N 번째 GxSymbol 이 반환되고 문자열일 경우 해당 종목 코드의 GxSymbol 이 반환됩니다.

GxSymbolStore 는 전체 GxSymbol 에 대한 Collection 객체이며 Stocks, Spreads, Futures 속성은 주식, 스프레드, 선물종목을 관리하는 Collection 타입의 속성입니다. 이를 테면 선물종목만 얻기 위해서 Futures 속성값을 취하면 됩니다. 이때 반환값은 IDispatch 타입이지만 원래 타입인 IGxSymbols 로 변환하여 사용하면 됩니다. GxSymbols 는 특정 종목들에 대한 Collection 객체입니다.

옵션 종목의 경우 좀 더 복잡한 구조를 가지게 됩니다. GxSymbolStore 하위에 OptionMonths 속성으로 GxOptionMonths 를 가리킵니다. GxOptionMonths 는 GxOptionMonth 의 Collection 객체 입니다. GxOptionMonth 는 옵션 월물에 대한 정보를 가지고 있습니다. GxOptionMonth 는 StrikePrices 속성으로 GxStrikePrices 를 가리킵니다. GxStrikePrices 는 GxStrikePrice 를 가리키는 Collection 객체 입니다. GxStrikePrice 는 부모 월물의 행사가 정보를 가지고 있습니다. 그리고 Call, Put 속성으로 GxSymbol 객체를 가리킵니다.

지금까지를 종합하면 GxSymbolStore 는 GxSymbol 을 접근하는데 두 가지 방법이 있습니다. GxSymbolStore 는 모든 GxSymbol 을 관리하므로 이를 통해도 되고 GxSymbolStore 의 특정 부류의 GxSymbols 이나 옵션월물->행사가(옵션의 경우)를 통해 GxSymbol 에 접근할 수 도 있습니다.

GxSymbol 객체는 종목의 모든 정보를 가지고 있는 객체 입니다만 체결과 호가 정보는 GxSymbol 의 하위 객체에서 얻을 수 있습니다. GxSymbol 의 Quote 속성으로 GxQuote 객체를 얻을 수 있으며 종목의 호가 정보를 얻을 수 있습니다. 그리고 GxSymbol 의 LastTick 속성으로 GxLastTick 객체를 얻을 수 있으며 종목의 최종 체결 정보를 얻을 수 있습니다.

GxSymbolStore 이벤트

GOM 에서 종목 관련 이벤트는 현재가, 호가 , 체결 등 3 가지 입니다. 현재가 이벤트를 수신하려면 GxSymbol 에, 호가 이벤트를 수신하려면 GxSymbol 하위의 GxQuote 에, 체결 이벤트를 수신하려면 GxSymbol 하위의 GxLastTick 에 이벤트를 요청해야 합니다.

종목 관련 이벤트가 짧은 시간에 많은 수가 발생하면 클라이언트나 서버에 무리가 갈 수 있으므로 GxSymbolStore 에 호가, 현재가 이벤트의 경우 일정시간 간격으로만 발생시키는 방법이 있습니다. 기본적으로는 모두 사용안함으로 설정되어 있습니다만 필요 시 Quote(Price)EventFiltered, Quote(Price)EventInterval 등을 사용하여 일정시간 간격으로 이벤트 발생을 조정하십시요.

라. GxTradeStore

GxTradeStore 의 구조

거래 전반을 관리하는 GxTradeStore 와 하위 객체에 대해 알아 봅니다.

클라이언트가 거래 전반에 대한 정보와 주문 도구를 이용하기 위새서는 GxTradeStore 객체를 사용해야 합니다. GxTradeStore 는 최상위 GxServer 하위에 존재하며 GxSymbolStore 와는 같은 레벨에 존재 합니다. GxTradeStore 는 크게 두 가지로 분류가 되는데 거래 전반의 정보 객체들과 주문 도구로 분류 됩니다.

거래 전반의 정보는 GxAccounts, GxOrders, GxPositions, GxFills, GxConfirms 등이 있으며 이는 GxAccount(계좌), GxOrder(주문), GxPosition(잔고), GxFill(체결), GxConfirm(확인) 을 구성원으로 하는 Collection 객체 이며 GxTradeStore 하위에 존재 합니다.

GxAccount, GxOrder 는 하위에 다른 Collection 객체를 두는 특이한 구조입니다. GxAccount 는 하위에 GxOrders, GxPositions 를 둡니다. 이것들은 이전에 GxTradeStore 하위의 GxOrders, GxPositions 과는 분명 다른 객체 입니다. 이전에 객체는 전체 주문, 전체 잔고를 관리하는 Collection 객체 이지만 GxAccount 하위의 GxOrders, GxPositions 는 GxAccount 계좌에 해당하는 주문, 잔고를 관리하는 Collection 객체 입니다. GxOrder 도 GxAccount 와 마찬가지로 GxConfirms, GxFills Collection 객체를 하위에 두고 있습니다.

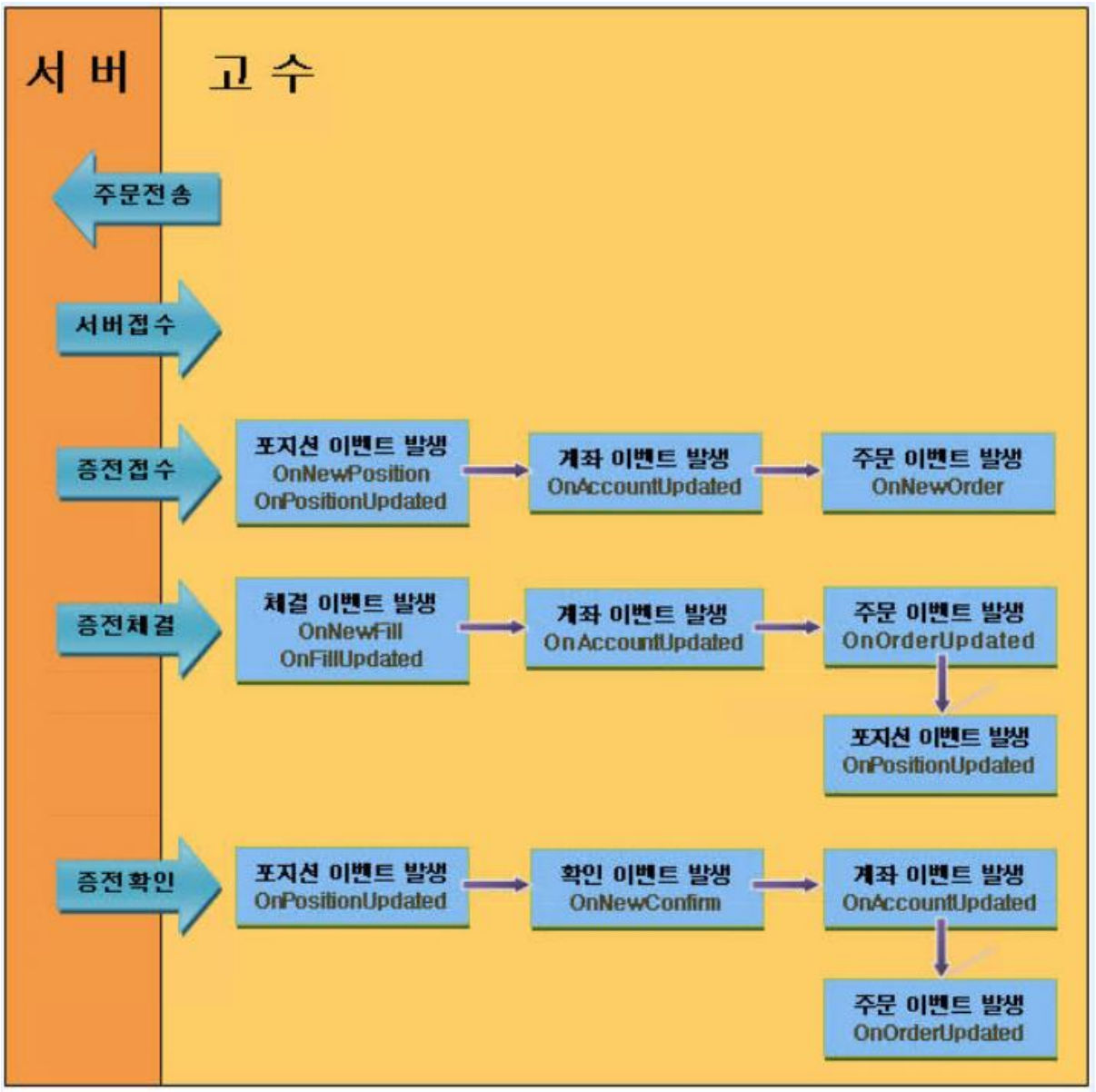
GxPosition 의 경우 'Account', 'Symbol' 속성을 가지고 있습니다. 'Account'속성은 잔고에 해당하는 GxAccount 를 가리키며 'Symbol'은 잔고에 해당하는 GxSymbol 을 가리킵니다. 이와 같은 형태는 GxConfirm, GxFill 등 자주 나타나는 형태 입니다.

주문 도구는 GxOrderHandler, GxEmHandler 두 가지가 있습니다. 이 두 가지 GxTradeStore 하위에 있으며 Collection 객체가 아닙니다. 주의할 점은 주문들이 나가기 위해서 대상 계좌가 고수 Plus 프로그램에서 '계좌 비밀번호 확인'이 되어야 합니다. 이는 고수 Plus 프로그램에 수동적으로 해 주어야 합니다. 그리고 주문 메소드 사용 후 반환되는 값이 True 인지 Null 이 아닌지 확인하는 습관을 가져야 합니다. GxOrderHandler 는 일반주문을 제외한 모든 주문 화면들이 사용하는 주문 모듈을 사용하고 GxEmHandler 는 일반주문 모듈을 사용합니다.

GxOrderHandler 는 IGxOrderHandler 인터페이스를 통해 신규, 정정, 취소 주문을 낼 수 있고 주문 추적 객체인 GxOrderReq 의 Collection 객체를 하위에 두고 있습니다. 이를 사용하여 주문의 상태를 추적할 수 있습니다.

GxEmHandler 는 긴급시의 상황을 바탕으로 만들어진 모듈을 사용하므로 주문 추적 기능은 없습니다. 단순히 전송 성공/실패를 수신 할 수 있습니다.

GxTradeStore 이벤트



GxTradeStore(주문도구 제외)의 성공적인 작업의 이벤트 발생 순서

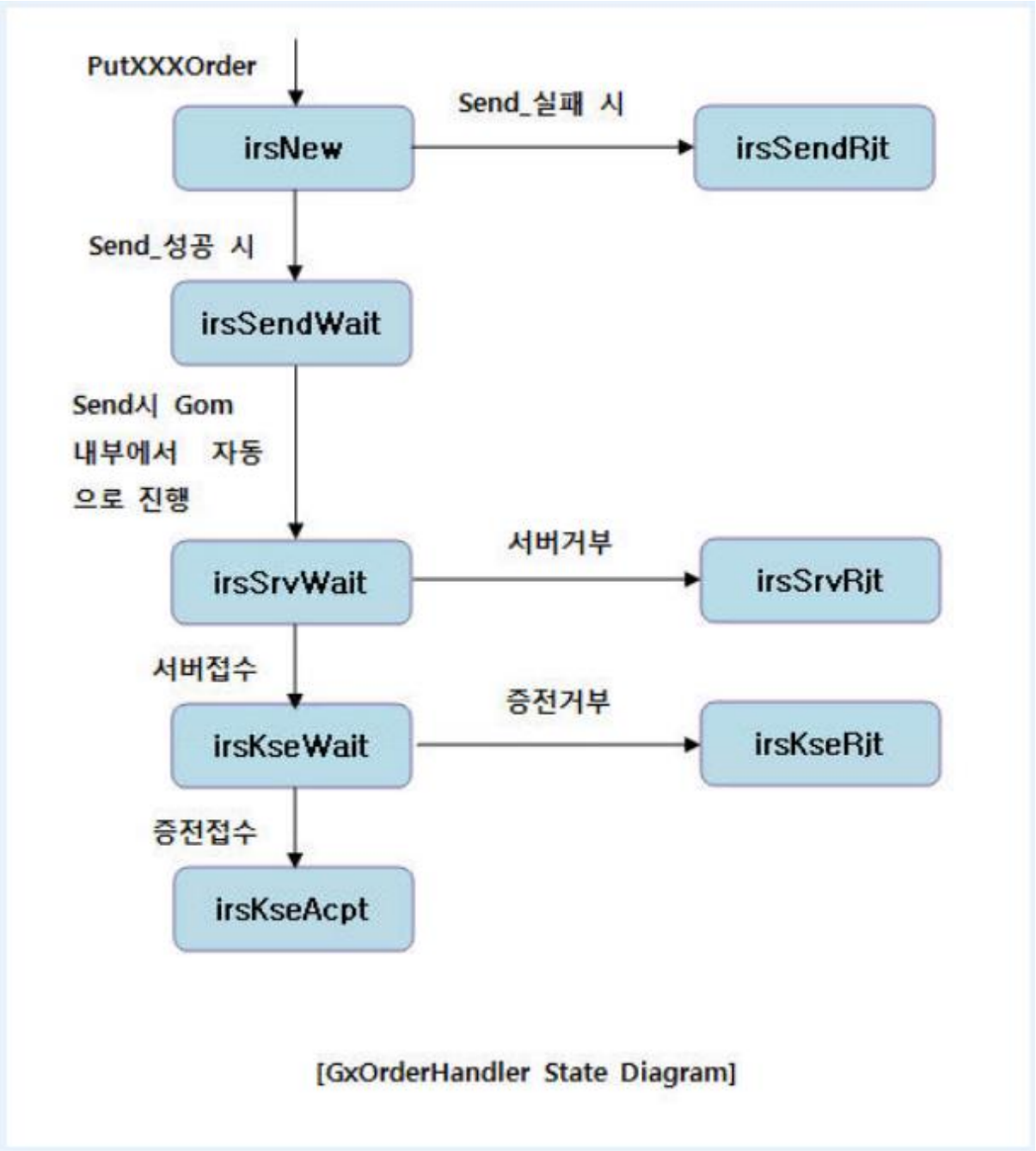
GxAccount 객체의 속성 값이 변경되었을 경우 이벤트가 발생하며 OnAccountUpdated 입니다. 이를 수신하려면 GxTradeStore 하위의 GxAccounts 객체에 이벤트 요청을 해야 합니다. GxPosition 객체가 주문을 통하여 신규로 생성이 되었을 때 OnNewPosition 이벤트가 발생합니다. 기존의 GxPosition 객체의 수량 속성등이 변경되었을 경우 OnPositionUpdated 이벤트가 발생합니다. GxPosition 의 평가손익이 변경 되었을 경우 OnEvalPLUpdated 이벤트가 발생합니다. 이 세 개의 이벤트는 GxTradeStore 하위의 GxPositions 객체에 이벤트 요청을 해야 합니다.

GxOrder 객체가 주문을 통하여 신규로 생성이 되었을 때 OnNewOrder 이벤트가 발생합니다. 기존의 GxOrder 객체가 주문 수량이 바꾸는 등의 변경이 있는 경우 OnOrderUpdated 이벤트가 발생합니다. 이 두 이벤트를 수신하려면 GxTradeStore 하위의 GxOrders 객체에 이벤트 요청을 해야 합니다.

GxFill 객체가 주문을 통한 체결 시 신규로 생성이 되었을 때 OnNewFill 이벤트가 발생합니다. GxFill 객체가 부분체결 상태에서 부분체결 상태를 유지하고 체결 수량이 변경이 변경되었을 때 OnFillUpdated 이벤트가 발생합니다. 이 두 이벤트를 수신하려면 GxTradeStore 하위의 GxFills 객체가 이벤트 요청을 해야 합니다.

GxConfirm 객체가 정정 주문을 통하여 신규로 생성이 되었을 때 OnNewConfirm 이벤트가 발생합니다. 이 이벤트를 수신하려면 GxTradeStore 하위의 GxConfirms 객체가 이벤트 요청을 해야 합니다.

GxOrderHandler 의 경우 하위의 GxOrderReq 의 상태가 변할 때 OnStateChanged 이벤트가 발생합니다. 이 이벤트를 수신하려면 GxTradeStore 하위의 GxOrderHandler 에 이벤트 요청을 해야 합니다.



[GxOrderHandler State Diagram]

GxEmHandler 의 경우 서버 접수 성공/실패를 OnEmOrderArrived 이벤트를 클라이언트에 알립니다. 이 이벤트를 수신하려면 GxTradeStore 하위의 GxEmHandler 에 이벤트 요청을 해야 합니다.

마. GxChartStore

GxChartStore 구조

모든 챠트 데이터를 관리하는 GxChartStore 객체와 그 하위 객체에 대해 알아봅니다.

GxChartStore 객체는 GxChartData 객체를 관리하는 Collection 객체 입니다. GxChartData 는 하나의 챠트 데이터를 의미하는 객체 입니다. GxChartData 는 실제 시간대별 자료를 의미하는 GxTerm 객체를 관리하는

GxTerms Collection 객체를 하위로 두고 있습니다.

GxTerm 객체는 시간대별 자료를 담고 있습니다. 예를 들어 선물 1 분 챠트 데이터의 예를 들면 선물 1 분 챠트 데이터 자체는 GxChartData 를 의미하고 9:10 분의 시/고/저/종 값은 GxTerm 을 의미합니다. 그러므로 GxChartData 는 다수의 GxTerm 을 가지는 Collection 객체인 GxTerms 를 하위로 가지는 구조인지를 쉽게 알 수 있습니다.

GxChartStore 사용

최초 사용자는 챠트 데이터를 얻기 위해 GxChartStore 에서 Add 를 통하여 새로운 GxChartData 를 생성해야 합니다. 그리고 새로운 GxChartData 에 Define 을 통하여 원하는 종목, 챠트 종류, 시간, 개수등을 설정하여 데이터를 요청합니다. GxChartData 객체 로부터 데이터가 준비 되었다는 이벤트를 받은 후 GxChartData 의 Terms 속성을 통하여 GxTerms 객체를 살펴보면 요청한 데이터가 다수의 GxTerm 구조로 쌓여 있는 것을 확인 할 수 있습니다.

GxChartStore 이벤트

모든 챠트의 이벤트 요청은 GxChartData 에 해야 합니다.

GxChartData 에 Define 을 하여 Define 의 조건에 맞게 챠트 데이터가 도착하면 OnDataRefreshed 이벤트가 발생합니다.

GxChartData 는 OnDataRefreshed 이벤트 발생 후 실시간 데이터를 받아 이에 맞게 GxTerm 을 새로 생성하거나 변경합니다. 수신된 실시간 데이터의 시간이 GxTerm 의 마지막 데이터의 시간 이내라면 마지막 GxTerm 의 값을 변경 시킵니다. 이 때 발생하는 이벤트를 OnDataUpdated 라고 합니다. 그러나 마지막 GxTerm 의 시간 범위를 초과한다면 새로운 GxTerm 을 생성해 GxTerms Collection 의 마지막에 추가 합니다. 이 때 발생하는 이벤트를 OnDataRefreshed 라고 합니다.

2. GOM 따라하기

가. GOM 시작하기

GOM 시작하기

개요

GOM 을 보다 쉽게 접근하는 것을 도와드리기 위해 기본적인 GOM 사용법을 설명해 드리겠습니다.
MFC, Vusial Basic, Delphi 등을 사용하여 GOM 을 사용할 수 도 있지만, 보다 손쉬운 접근을 위해서 여기에서는 MS-Office VBA(Visual Basic Application)를 이용하여 GOM 을 이용하는 방법에 주안점을 두어 설명 하겠습니다.

고수에서 GOM 사용하기

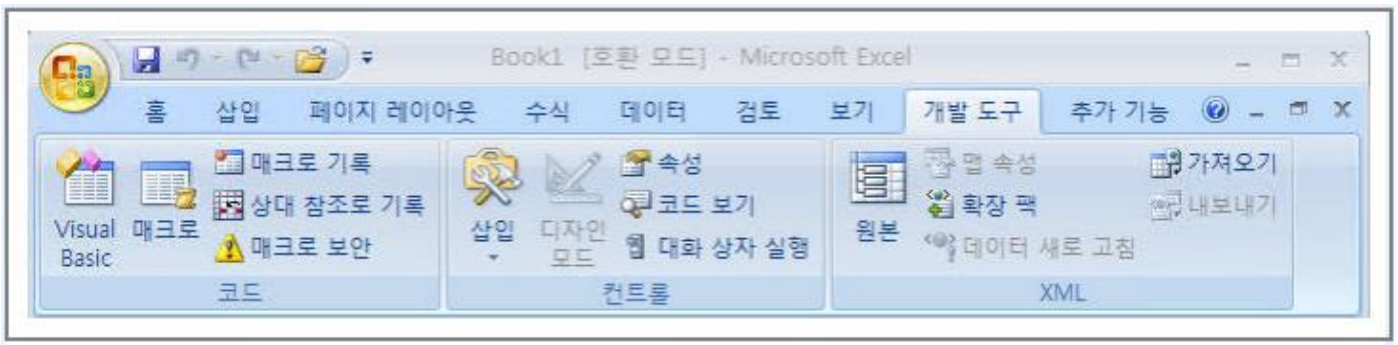
GOM 으로 작성한 클라이언트를 실행하기위해서는 GOM 서버가 필요하고 GOM 서버의 역활을 하는 것이 고수 Plus 프로그램입니다. 고수 Plus 로그인 화면에서 'GOM 사용'을 선택하고 고수 Plus 를 실행하면 초기화작업시 GOM 객체들이 생성됩니다. 따라서 GOM 을 시작하려면 고수 Plus 를 먼저 실행한 해야 GOM 을 만들기 위한 준비를 시작하는 것입니다. 고수 Plus 가 실행 되었는지 꼭 확인하세요 'GOM 사용'을 선택하지 않고 로그인을 하면 GOM 객체가 생성되지 않아서 GOM 을 이용할 수 없습니다. 만약 'GOM 사용'을 해제하고 로그인을 해서 클라이언트등에서 'CreateObject'등을 사용하면 GOM 의 최상위 객체의 생성은 가능하나 그 하위의 객체들은 생성되지 않습니다. ('GetObject'의 경우는 최상위 객체 생성도 실패합니다.')

참고로 GOM 을 사용하지 않을 때는 'GOM 사용'을 해제하고 로그인 하십시요. 'GOM 사용'을 체크하게 되면 GOM 객체가 클라이언트 연결이 없어도 차지하는 객체들의 메모리와 이들을 관리하는 CPU 로드가 있기 때문입니다.

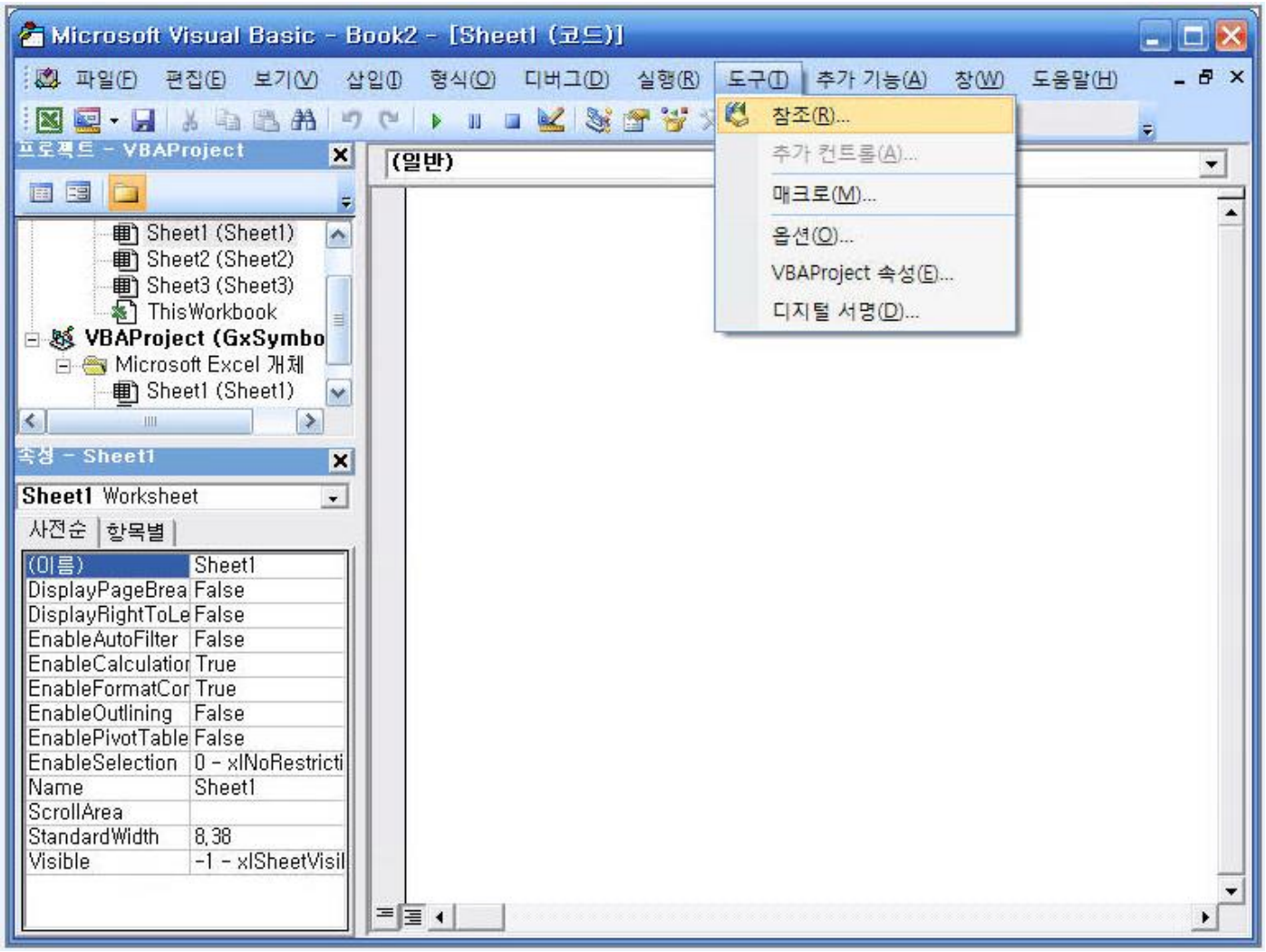
Excel 에서 Visual Basic Editor 열기

Excel 에서 GOM 을 사용하시려면 먼저 Excel 에서 Visual Basic Editor 를 열어야 합니다. Bisual Basic Editor 는 Excel 메뉴의 개발도구 항목에서 선택하실수 있습니다.

Visual Basic Project

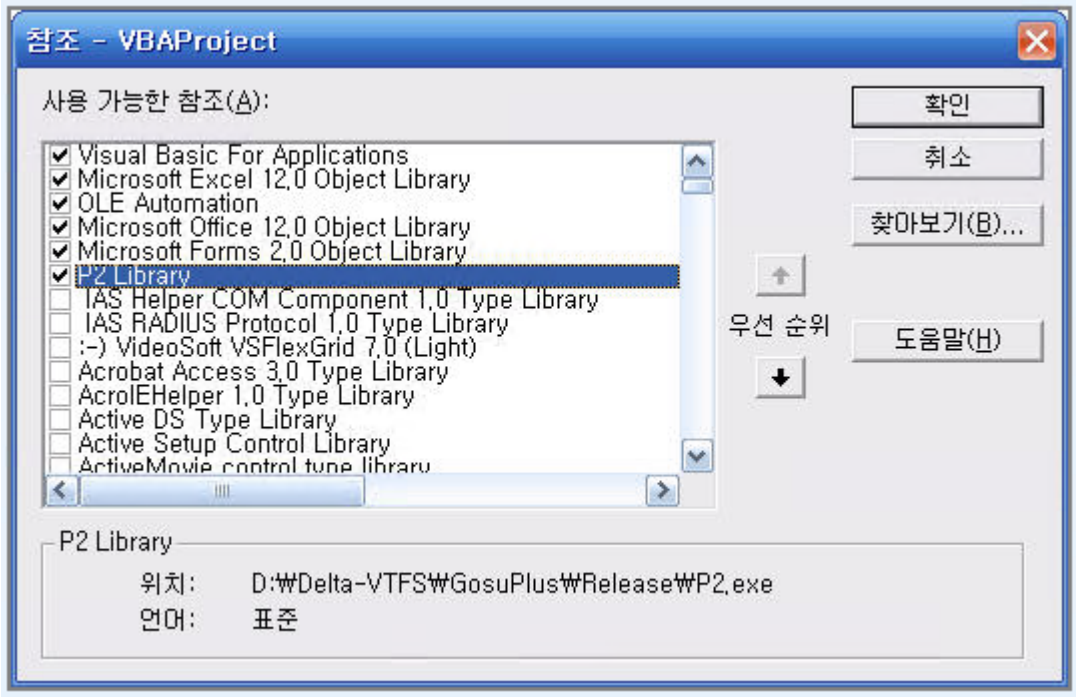


Visual Basic 에서 GOM 을 사용하시려면 GOM 타입 라이브러리를 import 해주어야 합니다.
도구>참조 메뉴를 실행하여 타입 라이브러리를 import 할 수 있습니다.



타입라이브러리

참조 화면에서는 '사용가능한 참조'항목에서 P2 Library 를 선택하고 확인 합니다.
혹시 항목에서 P2 Library 가 없다면 GOM 홈페이지 자료실에서 P2.tlb 파일을 다운로드 받아 '찾아보기'를 통해서 선택합니다.

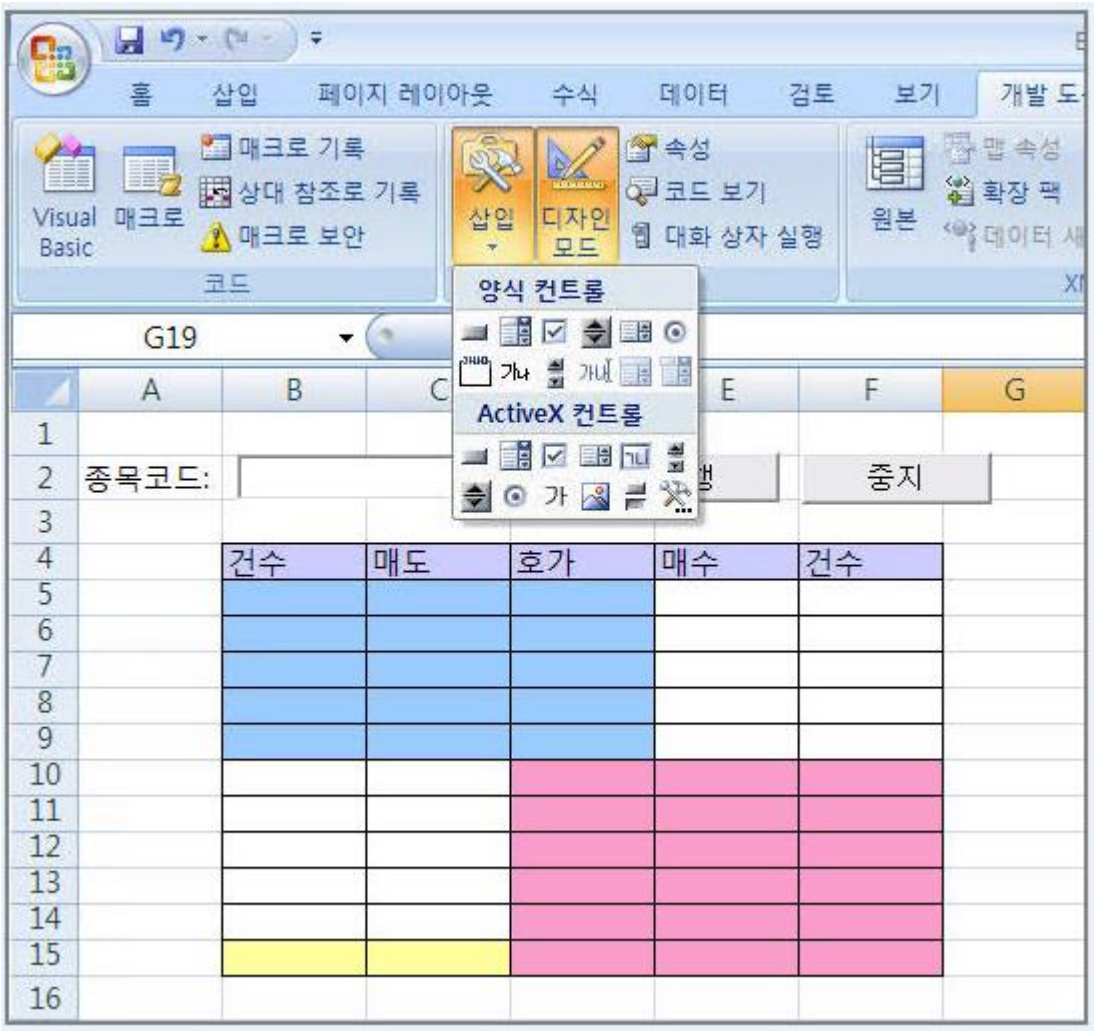


나. 현재가와 호가받기

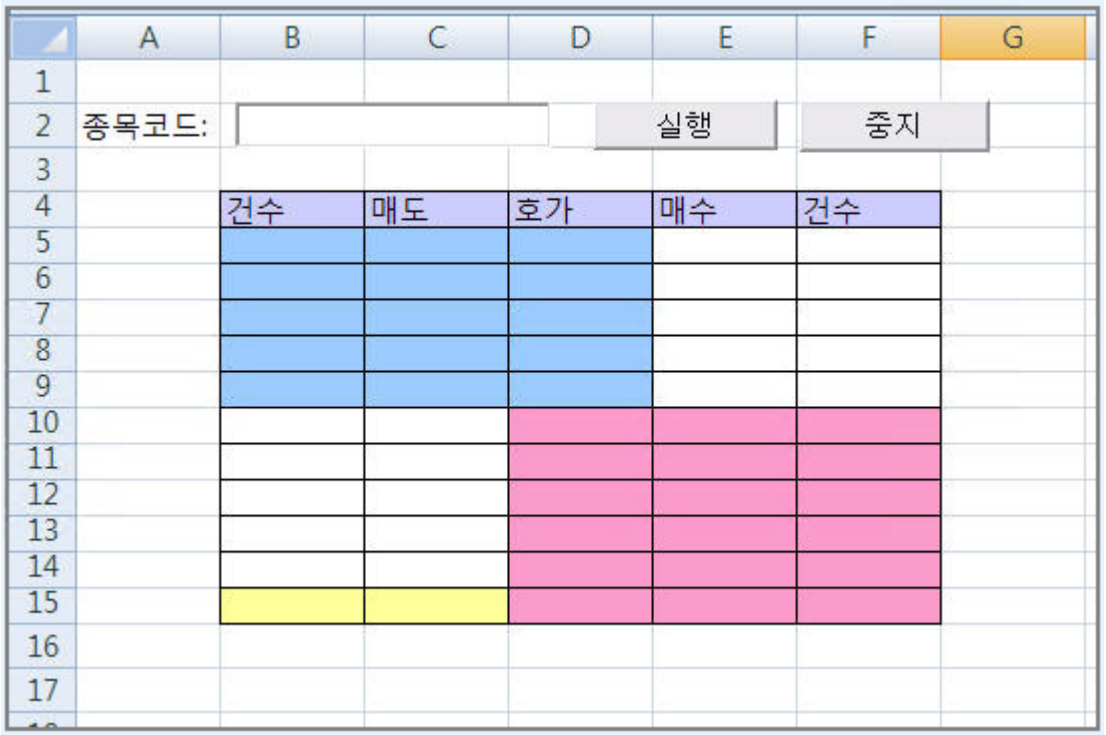
개요

여기서는 GOM 의 종목에 대한 Object 사용법을 간단하게 설명하겠습니다. 이번 예제에서는 GOM 의 종목 Object 를 사용하여 종목의 현재가와 호가를 받아서 화면에 표시하는 방법을 설명하겠습니다.

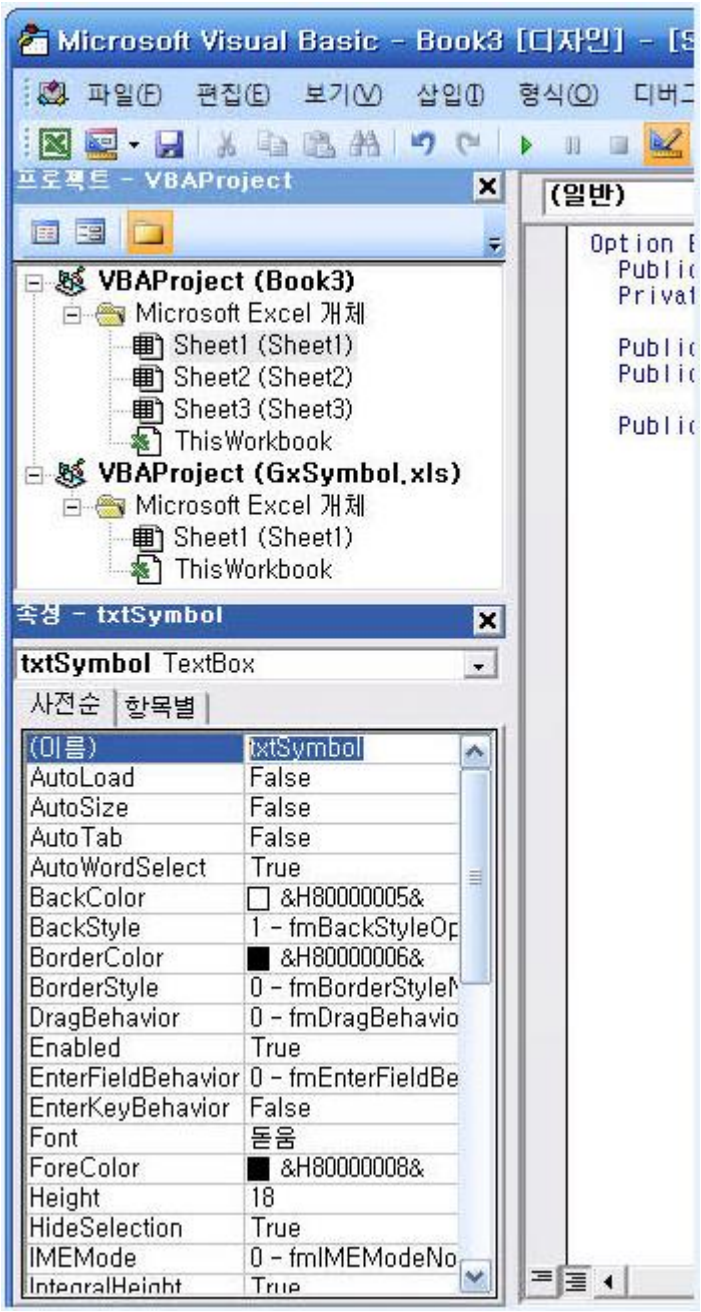
엑셀에서 화면 디자인하기



삽입 > Visual Basic 컨트롤 양식 상자에 있는 컨트롤들을 이용해서 다음 화면과 같이 종목코드를 입력할 수 있는 Edit Box 와 실행, 중지 Button 을 올려놓습니다.



Visual Basic Editor 속성창에서의 작업디자인했을 때 올려놓은 Edit Box 와 Button 들의 이름을 지정해 줍니다. 여기서 지정한 이름은 프로그램안에서 통용되는 이름으로 다른 이름들과는 구별되어야 합니다.



위에 보시는 속성 창을 이용하여 바로 전 단계에서 디자인했던 TextBox 와 Button 의 이름을 지정하여 줍니다.

Edit Box 의 이름은 txtSymbol 로 실행버튼과 중지 버튼은 각각 cmdExcute, cmdStop 으로 이름을 정해줍니다.

실행버튼과 중지버튼으로 사용할 Button 의 Caption 항목에는 각각 '실행', '중지'로 지정하여 줍니다.

Visual Basic Editor 코드창에서의 작업 Visual Basic Editor 의 코드창에서 실제 프로그래밍해야되는 작업에 대해 설명하겠습니다.
Visual Basic 에서 GOM 을 사용하시려면 GOM 타입 라이브러리를 import 해주어야 합니다. 도구>참조 메뉴를 실행하여 타입 라이브러리를 import 할 수 있습니다.

```
Option Explicit
```

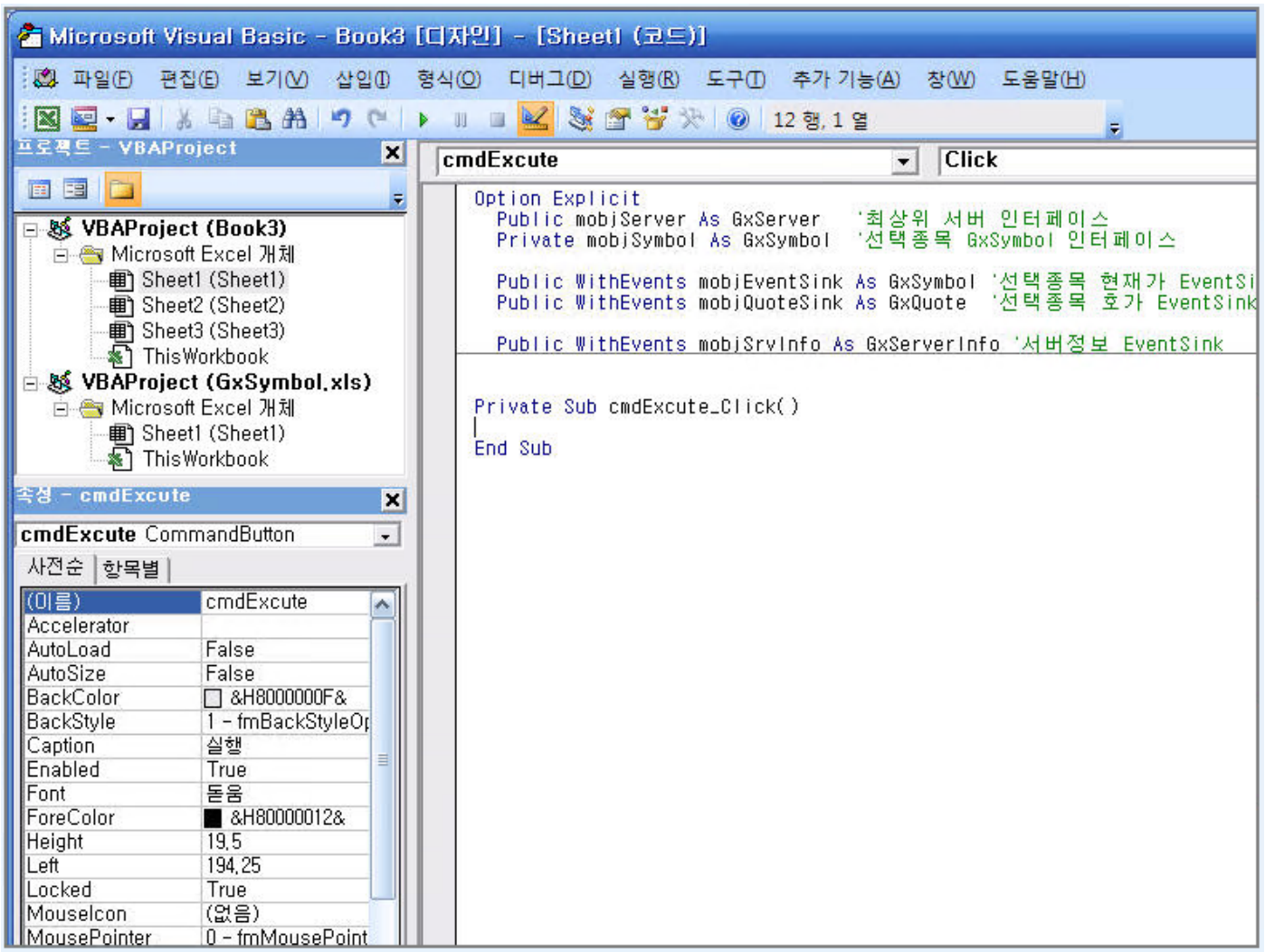
```
Public mobjServer As GxServer        '최상위 서버 인터페이스
Private mobjSymbol As GxSymbol       '선택종목 GxSymbol 인터페이스

Public WithEvents mobjEventSink As GxSymbol  '선택종목 현재가 EventSink
Public WithEvents mobjQuoteSink As GxQuote   '선택종목 호가 EventSink

Public WithEvents mobjSrvInfo As GxServerInfo  '서버정보 EventSink
```

첫번째로 예제에서 사용될 전역변수들을 선언해줍니다.

기본적으로 서버 인터페이스에 연결될 변수를 선언해줍니다.

이번예제에서 사용되는 종목 Object에 사용될 변수와, 종목이벤트에 연결될 변수, 마지막으로 서버정보를 받을 변수를 선언합니다.

버튼 클릭시 이벤트 연결디자인 작업에서 만들었던 Button 을 클릭했을 때 실행될 이벤트를 작성해줍니다.



엑셀 디자인 화면에서 '실행', '중지' Button 을 더블클릭하면 코드 창에 자동으로 버튼 클릭시 실행될 이벤트 함수가 선언됩니다.

실행 버튼 이벤트 함수에서 위와 같이 작성해줍니다.

실행버튼 클릭시 이벤트 작성

```
Private Sub cmdExcute_Click()
  Dim strSymbol As String

  If mobjServer Is Nothing Then
    On Error GoTo DoError      '서버 Object 연결 실패시 실행될 곳 지정
      Set mobjServer = GetObject(, "P2.GxServer")    '루트 인터페이스를 얻습니다.
      Set mobjSrvInfo = mobjServer.ServerInfo    '서버정보 이벤트 연결
      GoTo DoSuccess
  End If
  GoTo DoSuccess    '서버 Object 연결 성공시 실행될 곳 지정

'서버 Object 연결 실패시 실행
DoError:
  MsgBox ("서버와 연결할 수 없습니다")    '메세지 출력
  Exit Sub      '함수를 빠져나감

' 서버 Object 연결 성공시 실행
DoSuccess:
  strSymbol = txtSymbol.Text    'Edit Box 에 입력된 종목코드를 strSymbol 변수에 저장한다.

  '입력된 종목명으로 해당 종목을 찾는다.
  Set mobjSymbol = mobjServer.SymbolStore(strSymbol)

  '종목을 못찾았을 경우 오류 메세지를 출력하고 끝내준다.
  If mobjSymbol Is Nothing Then
    Set mobjEventSink = Nothing
    MsgBox "주어진 이름의 종목이 없습니다"
    Exit Sub
  End If

  RefreshQuote    '호가갱신함수

  '종목 Event 연결 초기화
  If (mobjSymbol Is Nothing) Or (mobjSymbol.Quote Is Nothing) Then
    Set mobjEventSink = Nothing
    Set mobjQuoteSink = Nothing
    Exit Sub
  End If

  '종목 Event 요청
  Set mobjEventSink = mobjSymbol
  Set mobjQuoteSink = mobjSymbol.Quote
End Sub
```

Visual Basic 의 'GetObject 함수'를 이용하여 GOM 의 Root Object 를 연결합니다. 이렇게 연결한 Root Object 를 통해서 GOM 의 모든 Object 에 접근 할 수가 있습니다.

'실행' Button 을 눌렀을 경우 서버에 연결됨과 동시에 Edit Box 에 입력한 종목코드로 해당종목에 대한 호가를 화면에 출력해주도록 코드를 작성합니다.

RefreshQuote 라는 함수를 작성하여 GOM 서버에 처음 연결했을 때와 종목 Event 가 발생했을 때 호가를 갱신해줍니다.

중지버튼 클릭시 이벤트 작성'중지' Button 을 눌렀을 때, 종목 Event 와 연결했던 GOM Object 를 연결 해제해 줍니다.

```
Private Sub cmdStop_Click()
  '이벤트 중지:종목 Event 연결해 주었던 것을 끊어준다.
  Set mobjEventSink = Nothing
  Set mobjQuoteSink = Nothing
  Set mobjTickSink = Nothing
End Sub
```

호가 갱신 함수 작성호가가 변할때 마다, GOM Object 에서 Event 를 수신받아서 RefreshQuote 함수를 호출하게 됩니다. 여기서는 Excel 의 화면에 Cell 단위로 종목의 호가 정보를 출력합니다.

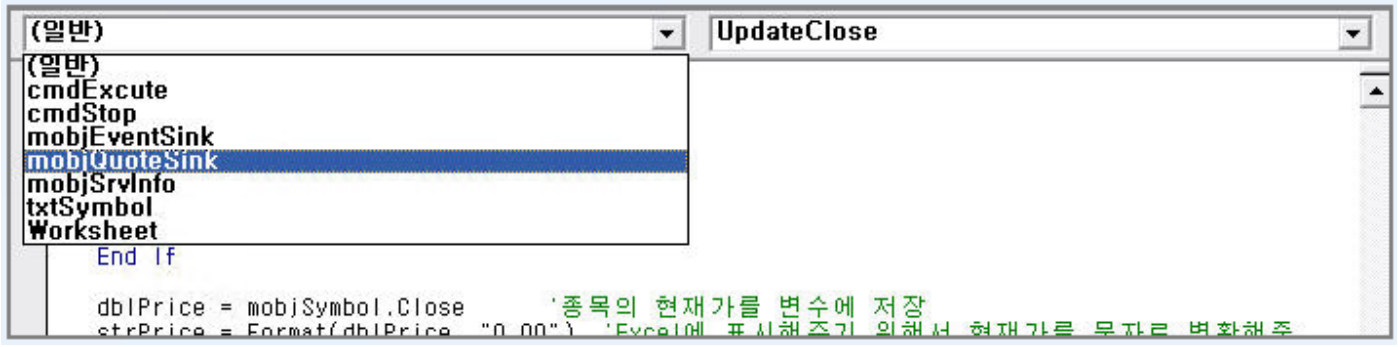
```
Private Sub RefreshQuote()
  Dim i As Integer       'roop 를 돌리기 위한 변수
  Dim vntPrice As Variant    '호가가격 저장을 위한 배열변수
  Dim vntCnt As Variant      '호가건수 저장을 위한 배열변수
  Dim vntQty As Variant      '호가수량 저장을 위한 배열변수

  If (mobjSymbol Is Nothing) Or (mobjSymbol.Quote Is Nothing) Then
    Exit Sub
  End If

  ' variant 로 변환
  vntPrice = mobjSymbol.Quote.VPrices
  vntCnt = mobjSymbol.Quote.Cnts
  vntQty = mobjSymbol.Quote.Qtys

  For i = 0 To 4      ' 매도 5 단계 호가를 Excel 화면에 출력
    Range("D" + CStr(9 - i)) = vntPrice(i) / 100
  Next i
  For i = 5 To 9      ' 매수 5 단계 호가를 Excel 화면에 출력
    Range("D" + CStr(i + 5)) = vntPrice(i) / 100
  Next i

  For i = 1 To 5      ' 매도 5 단계 건수를 Excel 화면에 출력
    Range("B" + CStr(10 - i)) = vntCnt(i)
  Next i
  Range("B15") = vntCnt(0)
  For i = 7 To 11      ' 매수 5 단계 건수를 Excel 화면에 출력
    Range("F" + CStr(i + 3)) = vntCnt(i)
  Next i
  Range("F15") = vntCnt(6)

  For i = 1 To 5      ' 매도 5 단계 잔량을 Excel 화면에 출력
    Range("C" + CStr(10 - i)) = vntQty(i)
  Next i
  Range("C15") = vntQty(0)
  For i = 7 To 11      ' 매수 5 단계 잔량을 Excel 화면에 출력
    Range("E" + CStr(i + 3)) = vntQty(i)
  Next i
  Range("E15") = vntQty(6)
```

```
    Range("D15") = vntQty(6) - vntQty(0)

    UpdateClose       '현재가를 화면에 표시해주는 함수
End Sub
```

현재가 갱신 함수 작성현재가가 변할때 마다, GOM Object 에서 Event 를 수신받아서 UpdateClose 함수를 호출하게 됩니다.

```
Private Sub UpdateClose()
    Dim i As Integer
    Dim dblPrice As Double
    Dim strPrice As String
    Dim objRange As Range

    '호가영역의 파란색 현재가 갱신
    If mobjSymbol Is Nothing Then
        Exit Sub
    End If

    dblPrice = mobjSymbol.Close      '종목의 현재가를 변수에 저장
    strPrice = Format(dblPrice, "0.00")    'Excel 에 표시해주기 위해서 현재가를 문자로 변환해줌

    For i = 5 To 14
        Set objRange = Range("D" + CStr(i))      '호가의 하나의 셀 설정

        '종목의 현재가와 해당셀의 가격이 같다면 현재가격이므로
        '현재가 셀인데 색은 현재가 색이 아니면 현재가색으로 바꾸어준다.
        If (strPrice = Trim(objRange.Text)) And _
            (objRange.Interior.ColorIndex <> 41) Then
            objRange.Interior.ColorIndex = 41

        '위치는 현재가 셀이 아닌데 색은 현재가 색이면 색을 복원한다.
        ElseIf (strPrice <> Trim(objRange.Text)) And _
                (objRange.Interior.ColorIndex = 41) Then

            If i <= 9 Then
                objRange.Interior.ColorIndex = 37      '매도호가
            Else
                objRange.Interior.ColorIndex = 38      '매수호가
            End If
        End If
    Next i
End Sub
```

종목 Event 를 받았을 때 처리하는 Event 함수 작성



Visual Basic Editor 코드 창 위에서, 프로그램 작성시 처음 선언해주었던 종목 Event 를 받을 Object 를 선택하면 각각 연결된 Event 들을 작성할 수 있습니다.

이와 같이 Event 연결 Object 인 mobjEventSink 의 OnPriceChanged 와 mobjQuoteSink 의 OnQuoteChanged 를 각각 선택하여 다음과 같이 Event 함수를 작성해줍니다.

```
'호가가 변할때마다 Event 수신을 받게 됩니다.
Private Sub mobjQuoteSink_OnQuoteChanged(ByVal aGxSymbol As Object, ByVal aGxQuote As Object)
On Error GoTo FAIL:
  If mobjSymbol Is aGxSymbol Then
    RefreshQuote
  End If

FAIL:
End Sub

'현재가가 변할때마다 Event 수신을 받게 됩니다.
Private Sub mobjEventSink_OnPriceChanged(ByVal aGxSymbol As Object)
On Error GoTo FAIL:
  If aGxSymbol Is mobjSymbol Then
    UpdateClose
  End If

FAIL:
End Sub
```

호가가 바뀔 때마다 OnQuoteChanged Event 가 현재가가 바뀔 때마다 , OnPriceChanged Event 가 발생하여 각각의 Event 함수를 호출해주게 됩니다.

현재가 갱신 함수 작성 GOM 서버와 연결이 끊어질 때, GOM 서버 오브젝트의 할당을 해제해줍니다.

```
Private Sub mobjSrvInfo_OnDisconnected(ByVal aGxServerInfo As Object)
On Error GoTo FAIL:
  Set mobjServer = Nothing

FAIL:
End Sub
```

GOM Symbol Object 활용

이런 방법으로 종목의 시세를 이용하여 자신만의 로직으로 계산되는 값들을 실시간으로 출력하여 볼수가 있습니다.

DDE 를 통해 Excel 에서 계산되는 것보다 조금더 빠르고 효율적으로 사용하실수 있습니다.

다. 계좌정보 불러오기

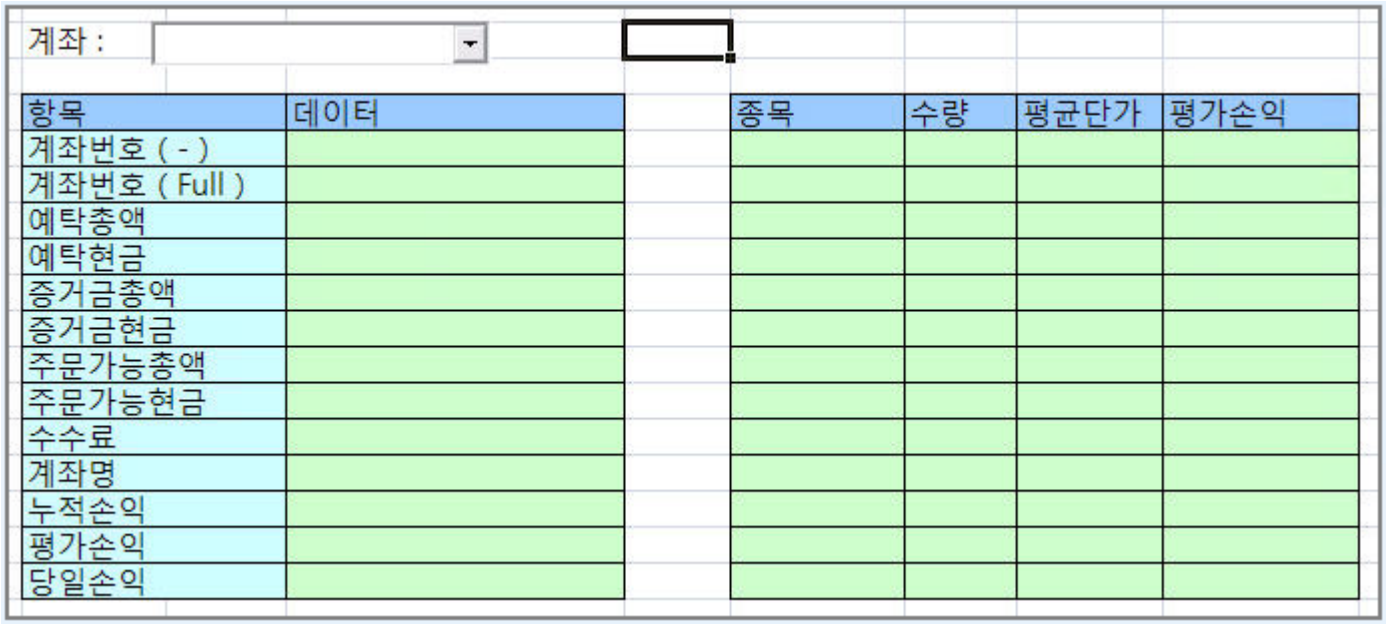
계좌정보 불러오기

개요

여기서는 GOM 의 계좌에 대한 Object 사용법을 간단하게 설명하겠습니다. 이번 예제에서는 GOM 의 종목 Object 를 사용하여 계좌 정보와 포지션 정보를 받아서 화면에 표시하는 방법을 설명하겠습니다.

엑셀에서 화면 디자인하기

삽입 > Visual Basic 컨트롤 양식 상자에 있는 컨트롤들을 이용해서 다음 화면과 같이 디자인해줍니다.



항목	데이터
계좌번호 (-)	
계좌번호 (Full)	
예탁총액	
예탁현금	
증거금총액	
증거금현금	
주문가능총액	
주문가능현금	
수수료	
계좌명	
누적손익	
평가손익	
당일손익	

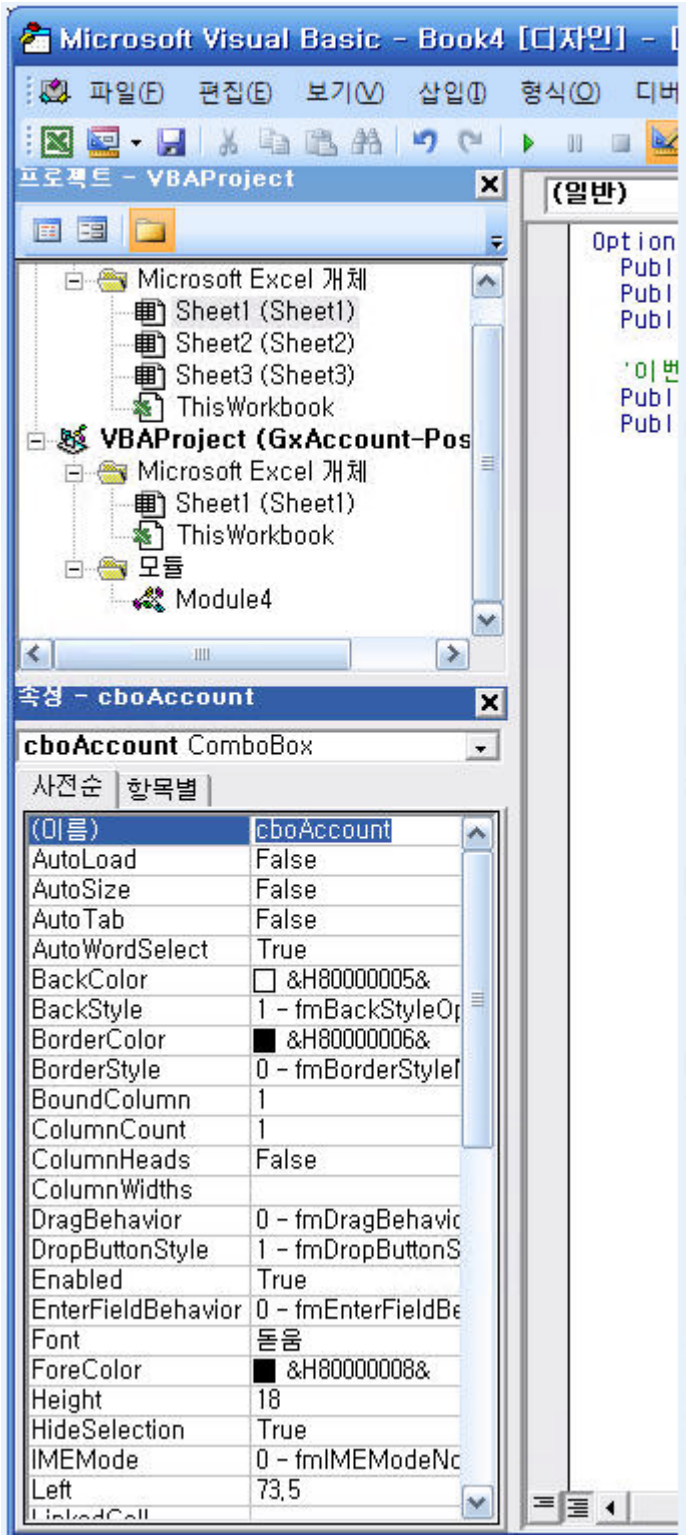
종목	수량	평균단가	평가손익

계좌를 선택할 수 있는 Combo Box 를 올리고, Excel 에 계좌정보와 포지션 정보를 출력할 수 있게 디자인해줍니다.

실행, 중지 버튼을 올려놓고, 이 버튼으로 GOM 서버와 연결하고 연결해제하는 작업을 하게 합니다.

Visual Basic Editor 속성창에서의 작업

디자인했을 때 올려놓은 Combo Box 의 이름을 지정해 줍니다. 여기서 지정한 이름은 프로그램안에서 통용되는 이름으로 다른 이름들과는 구별되어야 합니다.



속성 창을 이용하여 바로 전 단계에서 디자인했던 Combo Box 의 이름을 지정하여 줍니다.

Combo Box 의 이름은 cboAccount 로 이름을 정해줍니다.

'실행', '중지' 버튼은 각각 cmdExcute, cmdStop 으로 이름을 정해줍니다.

Visual Basic Editor 코드창에서의 작업

Visual Basic Editor 의 코드창에서 실제 프로그래밍해야되는 작업에 대해 설명하겠습니다.

```
Option Explicit
  Public mobjServer As GxServer        'GOM 서버의 Root Object 와 연결할 객체
  Public mobjTradeStore As GxTradeStore    'TradeStore 와 연결할 객체
  Public mobjAccount As GxAccount       '선택한 계좌를 담을 객체

  '이벤트 선언
  Public WithEvents mobjAccountSink As GxAccounts    '계좌 이벤트
  Public WithEvents mobjPositionSink As GxPositions    '포지션 이벤트
```

첫번째로 예제에서 사용될 전역변수들을 선언해줍니다.

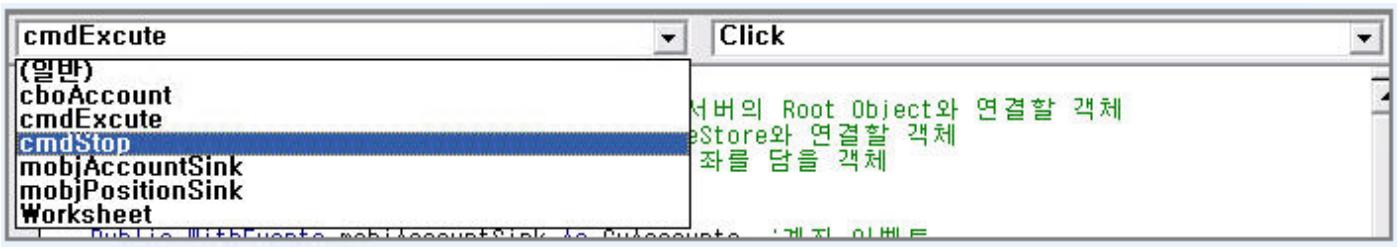
서버 인터페이스에 연결될 변수를 선언해줍니다.

이번 예제에서 사용될 계좌에 대한 정보와 포지션 정보를 담고 있는 TradeStore Object 를 연결할 변수를 선언합니다.

Combo Box 에서 선택한 계좌를 저장할 변수를 선언하고, 계좌와 포지션에 대해서 변경됐을 경우 수신되는 계좌이벤트와 포지션 이벤트를 받을 변수를 선언합니다.

Button 이벤트 작성

디자인 작업에서 만들었던 Button 을 클릭했을 때 실행될 이벤트를 작성해줍니다.



Visual Basic Editor 코드창 위에서 Object List 중에서 Button 의 이름인 cmdExcute 와 cmdStop 을 선택하여 Click Event 를 선택하면 자동으로 Event 함수의 원형이 만들어집니다.

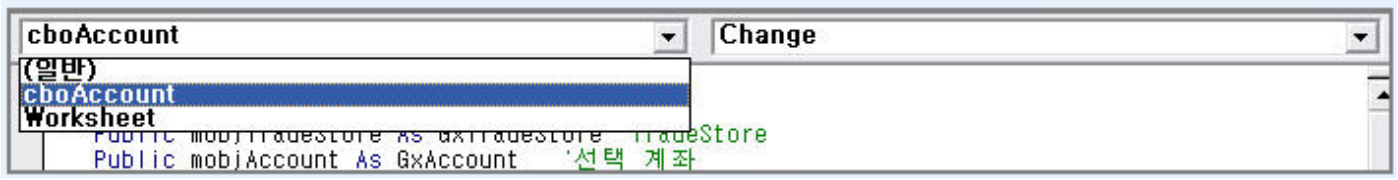
Button 의 Click 이벤트 함수에서 아래와 같이 작성해줍니다.

```
Private Sub cmdExcute_Click()
  Dim objAccount As GxAccount

  On Error GoTo DoFail
    Set mobjServer = GetObject(, "P2.GxServer")    'GOM 서버의 Root Object 연결
    Set mobjTradeStore = gobjServer.TradeStore    'TradeStore 를 연결시켜준다.

    GoTo DoSuccess
    Exit Sub

'서버 연결 실패시 메세지 출력후 끝냄
DoFail:
  MsgBox "서버와 연결하는데 실패 하였습니다"
  Exit Sub
```

```
DoSuccess:      'Combo Box 초기화
  cboAccount.Clear

  'TradeStore 의 계좌목록을 Combo Box 에 넣어준다.
  For Each objAccount In mobjServer.TradeStore.Accounts
    cboAccount.AddItem objAccount.Code
  Next

  '기본적으로 첫번째 계좌를 선택합니다.
  If cboAccount.ListCount > 0 Then
    cboAccount.ListIndex = 0
  End If
End Sub

Private Sub cmdStop_Click()
  mobjServer = Nothing
End Sub
```

Combo Box 이벤트 작성

디자인 작업에서 만들었던 Combo Box 에서 계좌를 선택했을 때 실행될 이벤트를 작성해줍니다.



Visual Basic Editor 코드창 위에서 Object List 중에서 Combo Box 의 이름인 cboAccount 를 선택하여 Change Event 를 선택하면 자동으로 Event 함수의 원형이 만들어집니다.

Combo Box 의 Change 이벤트 함수에서 아래와 같이 작성해줍니다.

```
Private Sub cboAccount_Change()
On Error GoTo FAIL
  If mobjTradeStore Is Nothing Then Exit Sub

  '이벤트 연결 (계좌, 포지션 이벤트 요청)
  Set mobjAccountSink = mobjTradeStore.Accounts       '계좌 이벤트 요청
  Set mobjPositionSink = mobjTradeStore.Positions     '포지션 이벤트 요청

  '선택 계좌 인터페이스를 얻습니다.
  Set mobjAccount = mobjTradeStore.Accounts(Trim(cboAccount.Text))

  ClearPosition       'Excel 의 포지션 정보란을 초기화해준다
  RefreshPositions    'Excel 의 포지션 정보란을 갱신해준다
  RefreshAccount      'Excel 의 계좌 정보란을 갱신해준다.

  Exit Sub

FAIL:
  MsgBox ("GOM 초기화에 실패하였습니다")
  Exit Sub
End Sub
```

포지션 정보 초기화 함수 작성

Excel 에 포지션 정보를 출력하는 부분을 초기화 해주는 함수를 다음과 같이 작성합니다.

```
Private Sub ClearPosition()
  Range("F5", "I17") = ""
End Sub
```

포지션 정보 갱신 함수 작성

Excel 에 포지션 정보를 처음 출력하거나 포지션이 추가될때 화면에 출력할 함수를 다음과 같이 작성합니다.

```
'선택 계좌에 해당하는 포지션을 갱신합니다.
Sub RefreshPositions()
  Dim objPosition As GxPosition    'GOM 의 포지션 Object 를 담을 변수
  Dim intIndex As Integer          'Excel 의 열을 지정할 변수

  If mobjAccount Is Nothing Then Exit Sub

  intIndex = 5       '포지션정보는 5 번째 열부터 출력되므로
  'loop 를 돌면서 Excel 에 포지션 정보를 출력합니다.
  For Each objPosition In mobjAccount.Positions
    Range("F" + CStr(intIndex)) = objPosition.Symbol.Code
    Range("G" + CStr(intIndex)) = objPosition.Qty
    Range("H" + CStr(intIndex)) = objPosition.AvgPrice
    Range("I" + CStr(intIndex)) = objPosition.EvalPL

    intIndex = intIndex + 1    'Excel 의 열을 증가시켜준다.
  Next
End Sub
```

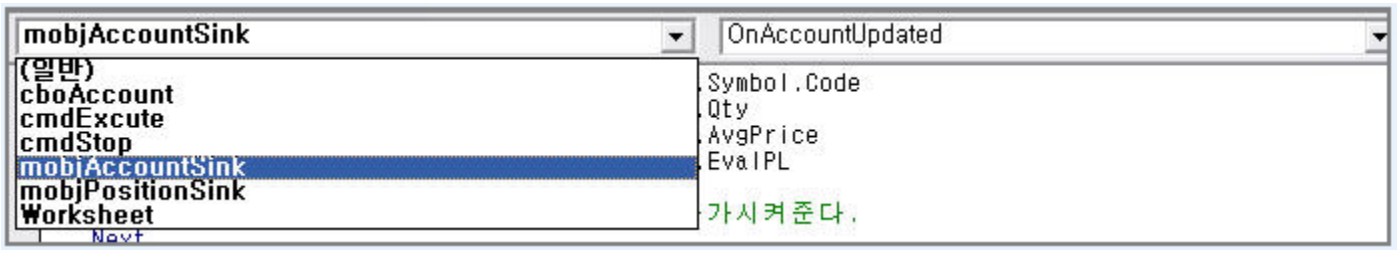
계좌 정보 갱신 함수 작성

Excel 에 계좌 정보를 처음 출력할 때 화면에 출력할 함수를 다음과 같이 작성합니다.

```
'계좌 전체 값을 갱신 합니다.
Public Sub RefreshAccount()
  If mobjAccount Is Nothing Then Exit Sub

  'Excel 의 지정된 Cell 에 계좌정보를 출력합니다.
  With mobjAccount
    Range("D5") = .HiphenedCode
    Range("D6") = .Code
    Range("D7") = FormatNumber(.DepositTotal, ["0"])
    Range("D8") = FormatNumber(.DepositCash, ["0"])
    Range("D9") = FormatNumber(.MarginTotal, ["0"])
    Range("D10") = FormatNumber(.MarginCash, ["0"])
    Range("D11") = FormatNumber(.LiquidTotal, ["0"])
    Range("D12") = FormatNumber(.LiquidCash, ["0"])
    Range("D13") = FormatNumber(.Commission, ["0"])
    Range("D14") = .Desc
```

```
    Range("D15") = FormatNumber(.AccPL, ["0"])
    Range("D16") = FormatNumber(.EvalPL, ["0"])
    Range("D17") = FormatNumber(.DailyPL, ["0"])
  End With
End Sub
```

계좌 정보 Event 를 받았을 때 처리하는 Event 함수 작성



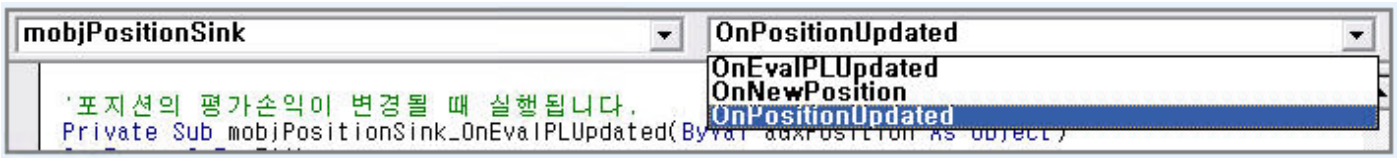
Visual Basic Editor 코드 창 위에서, 프로그램 작성시 처음 선언해주었던 계좌 Event 를 받을 Object 인 mobjAccountSink 를 선택하고 OnAccountUpdated Event 를 선택하면 자동으로 함수 원형이 만들어집니다.

OnAccountUpdated Event 함수를 다음과 같이 작성해줍니다.

```
'계좌 값이 변경될 때 실행됩니다.
Private Sub mobjAccountSink_OnAccountUpdated(ByVal aGxAccount As Object)
On Error GoTo FAIL
  Dim objAccount As GxAccount

  If aGxAccount Is Nothing Then Exit Sub

  Set objAccount = aGxAccount

  '현재 선택된 계좌의 이벤트 이면(선택된 계좌가 아닐경우는 실행안됨)
  If mobjAccount Is objAccount Then
    UpdateAccount      '실시간 계좌 값 갱신
  End If
  Exit Sub

FAIL:
  Exit Sub
End Sub
```

계좌정보가 변경될 때마다 OnAccountUpdated Event 가 발생하여 Event 함수를 호출해주게 됩니다.

포지션 정보 Event 를 받았을 때 처리하는 Event 함수 작성



Visual Basic Editor 코드 창 위에서, 프로그램 작성시 처음 선언해주었던 계좌 Event 를 받을 Object 인 mobjPositionSink 를 선택하고 OnEvalPLUpdated, OnNewPosition, OnPositionUpdated Event 를 선택하면 자동으로 함수 원형이 만들어집니다.

포지션 Event 함수를 다음과 같이 작성해줍니다.

```
'포지션의 평가손익이 변경될 때 실행됩니다.
Private Sub mobjPositionSink_OnEvalPLUpdated(ByVal aGxPosition As Object)
On Error GoTo FAIL
  Dim objEventPosition As GxPosition
  Dim objEventAccount As GxAccount

  Set objEventPosition = aGxPosition
  Set objEventAccount = objEventPosition.Account

  If (mobjAccount Is Nothing) Or (objEventAccount Is Nothing) Then Exit Sub

  '현재 선택된 계좌의 이벤트 이면(선택된 계좌가 아닐경우는 실행안됨)
  If mobjAccount Is objEventAccount Then
    UpdatePosition objEventPosition    '실시간 포지션 정보 갱신 함수 실행
  End If

FAIL:
  Exit Sub
End Sub

'포지션이 추가될 때 실행됩니다.
Private Sub mobjPositionSink_OnNewPosition(ByVal aGxPosition As Object)
On Error GoTo FAIL
  Dim objEventPosition As GxPosition
  Dim objEventAccount As GxAccount

  Set objEventPosition = aGxPosition
  Set objEventAccount = objEventPosition.Account

  If (mobjAccount Is Nothing) Or (objEventAccount Is Nothing) Then Exit Sub

  '현재 선택된 계좌의 이벤트 이면(선택된 계좌가 아닐경우는 실행안됨)
  If mobjAccount Is objEventAccount Then
    RefreshPositions     '포지션 정보를 새로 갱신함
  End If

FAIL:
  Exit Sub
End Sub

'포지션이 값이 변경될 때 실행됩니다.
Private Sub mobjPositionSink_OnPositionUpdated(ByVal aGxPosition As Object)
On Error GoTo FAIL
  Dim objEventPosition As GxPosition
  Dim objEventAccount As GxAccount

  Set objEventPosition = aGxPosition
  Set objEventAccount = objEventPosition.Account
  If (mobjAccount Is Nothing) Or (objEventAccount Is Nothing) Then Exit Sub
```

```
  '현재 선택된 계좌의 이벤트 이면(선택된 계좌가 아닐경우는 실행안됨)
  If mobjAccount Is objEventAccount Then
    UpdatePosition objEventPosition
  End If

FAIL:
  Exit Sub
End Sub
```

포지션 정보가 변경될 때, 평가손익이 변경될 때마다 OnEvalPLUpdated Event, 새로운 포지션이 발생하면 OnNewPosition Event, 포지션 정보가 갱신되면 OnPositionUpdated Event 가 각각 발생하여 Event 함수를 호출해주게 됩니다.

실시간으로 계좌 정보와 포지션 정보를 갱신하는 함수 작성

```
'실시간으로 변경될 수 있는 값을 갱신 합니다.
Public Sub UpdateAccount()
  If mobjAccount Is Nothing Then Exit Sub

  With mobjAccount
    Range("C7") = FormatNumber(.DepositTotal, ["0"])
    Range("C8") = FormatNumber(.DepositCash, ["0"])
    Range("C9") = FormatNumber(.MarginTotal, ["0"])
    Range("C10") = FormatNumber(.MarginCash, ["0"])
    Range("C11") = FormatNumber(.LiquidTotal, ["0"])
    Range("C12") = FormatNumber(.LiquidCash, ["0"])
    Range("C13") = FormatNumber(.Commission, ["0"])
    Range("C15") = FormatNumber(.AccPL, ["0"])
    Range("C16") = FormatNumber(.EvalPL, ["0"])
    Range("C17") = FormatNumber(.DailyPL, ["0"])
  End With
End Sub

Private Sub UpdatePosition(ByVal objPosition As IGxPosition)
  Dim i As Integer
  Dim objRange As Range

  For i = 5 To 18
    If Trim(Range("E" + CStr(i))) = objPosition.Symbol.Code Then
      Set objRange = Range("E" + CStr(i))

      objRange.Offset(0, 0) = objPosition.Symbol.Code
      objRange.Offset(0, 1) = objPosition.Qty
      objRange.Offset(0, 2) = objPosition.AvgPrice
      objRange.Offset(0, 3) = objPosition.EvalPL
    End If
  Next i
End Sub
```

Event 함수에서 Event 를 받을 때마다 계좌 정보와 포지션 정보를 새로 갱신하여 화면에 출력해주는 함수를 아래와 같이 작성합니다.

이미 위에서 작성했던 RefreshAccount, RefreshPosition 함수와 별도로 이런 함수를 작성하는 이유는 VBA 는 VB, MFC, Delphi 와 다르게 Excel 의 화면에 데이타를 출력하기 때문에 Excel 의 Cell 에 대한 접근을 최소한으로 줄여주는 것이 성능 향상에 좋습니다.

UpdateAccount, UpdatePosition 함수는 변경되는 항목에 해당하는 Cell 만을 갱신해줍니다.

라. 주문익히기

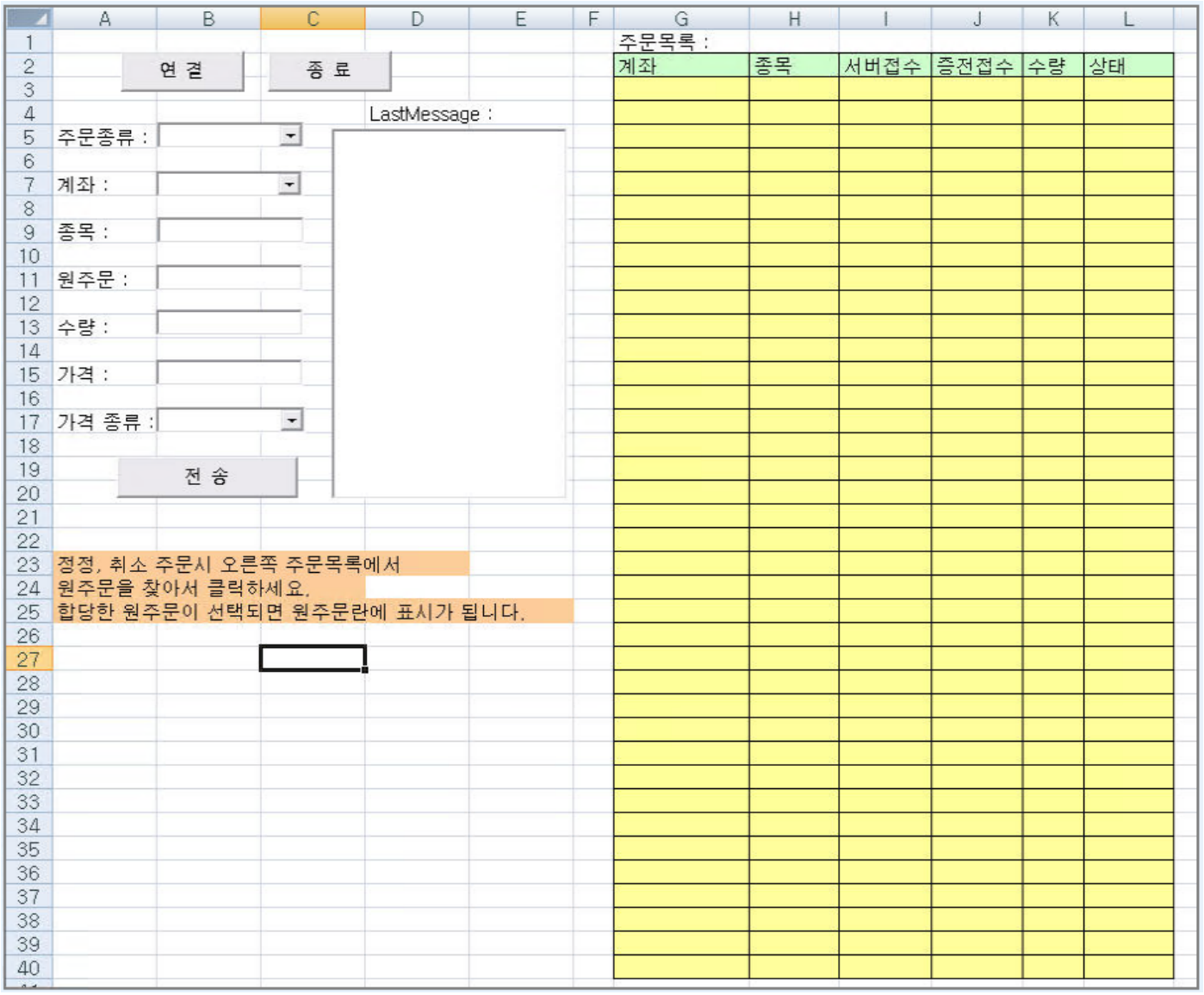
주문익히기

개요

여기서는 GOM 의 주문에 대한 Object 사용법을 간단하게 설명하겠습니다. 이번 예제에서는 GOM 의 주문 Object 를 사용하여 서버로 주문을 전송하고 처리 결과를 화면에 표시하는 프로그램을 작성해 보겠습니다.

엑셀에서 화면 디자인하기

삽입 > Visual Basic 컨트롤 양식 상자에 있는 컨트롤들을 이용해서 다음 화면과 같이 Excel 화면을 디자인 합니다.



GOM 서버와 연결하고 종료할 '연결', '종료' 버튼을 올려놓습니다.

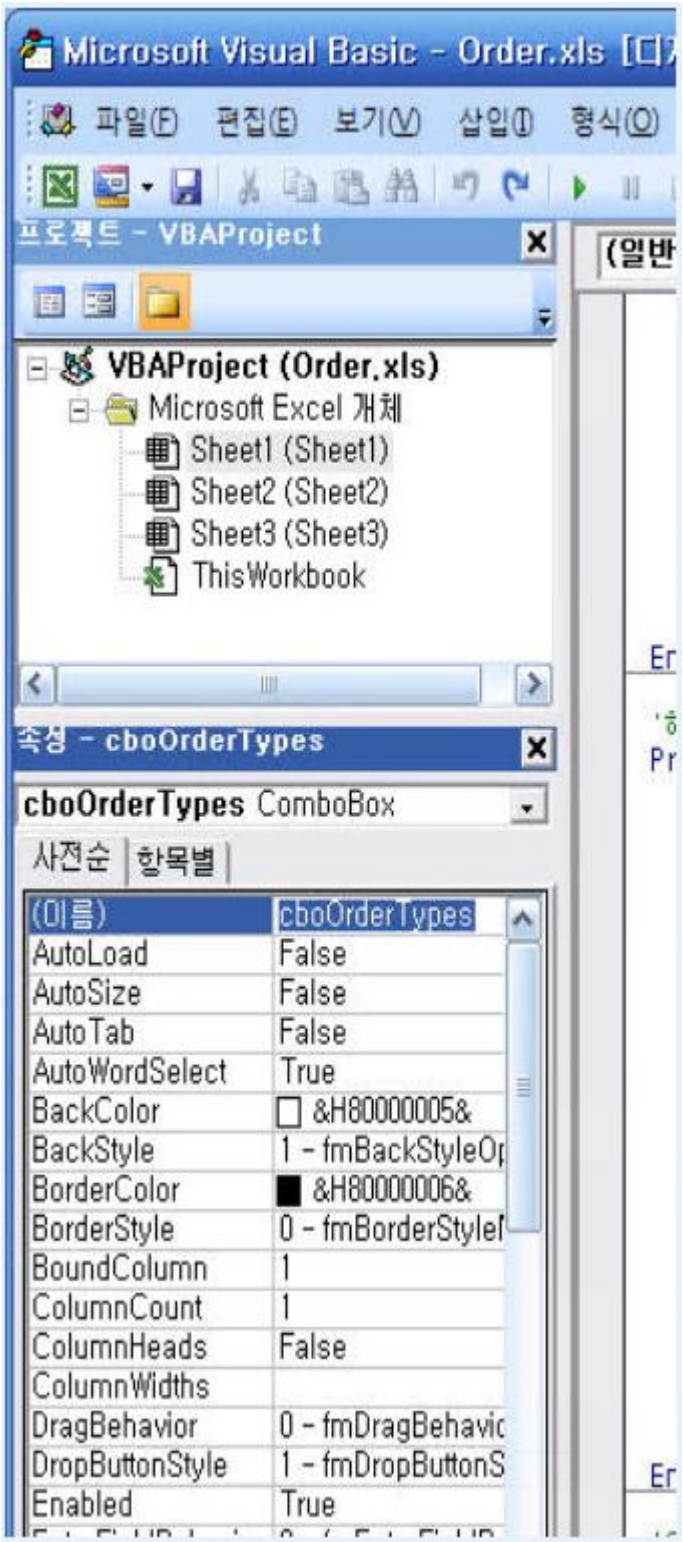
주문 전송시 사용할 버튼과 주문시 조건을 설정할 Combo Box, Text Box 등을 올려놓습니다.

주문 전송하고 서버에서 받는 메세지를 출력할 Text Box 를 올려놓습니다.

Excel 화면 오른쪽에 전송했던 주문목록을 출력할 화면을 디자인합니다. 여기서는 정정, 취소시에 지정할 원주문들도 출력됩니다.

Visual Basic Editor 속성창에서의 작업

디자인했을 때 올려놓은 Edit Box 와 Button 들의 이름을 지정해 줍니다. 여기서 지정한 이름은 프로그램안에서 통용되는 이름으로 다른 이름들과는 구별되어야 합니다.



위에 보시는 속성 창을 이용하여 바로 전 단계에서 디자인했던 컨트롤들의 이름을 지정합니다.

'실행', '종료', '주문전송' 버튼은 각각 cmdConnect, cmdDisconnect, cmdSend 로 이름을 지정합니다.

주문 전송 조건을 설정할 컨트롤들은 계좌, 주문종류, 종목, 원주문, 수량, 가격, 가격종류를 각각 cboAccount, cboOrderType, txtSymbol, txtTarget, txtQty, txtPrice, cboPriceType 으로 지정합니다.

주문 전송 메세지를 출력할 Text Box 는 txtLastMsg 로 지정합니다.

Visual Basic Editor 코드창에서의 작업

Visual Basic Editor 의 코드창에서 실제 프로그래밍해야되는 작업에 대해 설명하겠습니다.

```
Option Explicit

Const MAX_ORDER_ROW = 40

  Public mobjServer As GxServer       'Gom 서버와 연결할 루트 인터페이스를 할당할 Object
  Public mobjTradeStore As GxTradeStore     '주문관리를 위한 GxTradeStore 를 할당할 Object
  Public mobjOrderHandler As GxOrderHandler    'OrderHandler 를 할당할 Object
  Public mobjTarget As GxOrder      '원주문으로 선택되는 주문 할당을 위한 Object

  Public mobjOrderCols As Collection      'GxOrder 를 저장한 Collection

  '이벤트 선언
  Public WithEvents mobjOrderReqSink As GxOrderHandler     '주문 전송에 대한 응답 Event 연결할 Object
  Public WithEvents mobjOrderSink As GxOrders         '서버에서 주문에 대한 처리응답 Event 연결할 Object
```

첫번째로 예제에서 사용될 전역변수들을 선언해줍니다.

서버 Root 인터페이스에 연결될 변수를 선언해줍니다.

이번예제에서 사용되는 주문 관리 Object 인 GxTradeStore 에 사용될 변수와, 주문전송 Object 인 GxOrderHandler 를 할당할 변수를 선언합니다.

해당 계좌에서 나간 주문들을 따로 저장할 Collection Object 를 선언해줍니다. 여기에 주문 목록들을 저장하여 프로그램 내에서 필요할때 찾게 됩니다.

버튼 클릭시 이벤트 연결

디자인 작업에서 만들었던 Button 을 클릭했을 때 실행될 이벤트를 작성해줍니다.

```
'연결 버튼 Click
Private Sub cmdConnect_Click()
On Error GoTo FAIL:
  Set mobjServer = GetObject(, "P2.GxServer")      'Root 인터페이스 할당
  Set mobjTradeStore = mobjServer.TradeStore     'TradeStore 할당
  Set mobjOrderHandler = mobjServer.TradeStore.OrderHandler      'OrderHandler 인터페이스 할당

  'TradeStore 나 OrderHandler 연결에 실패할 경우 메세지 출력
  If (mobjOrderHandler Is Nothing) Or (mobjTradeStore Is Nothing) Then
    MsgBox "초기화에 실패하였습니다"
    Exit Sub
  End If

  '이벤트 재요청
  Set mobjOrderReqSink = Nothing
  Set mobjOrderReqSink = mobjOrderHandler
  Set mobjOrderSink = Nothing
  Set mobjOrderSink = mobjTradeStore.Orders

  'Collection 재생성
```

```
    Set mobjOrderCols = Nothing
    Set mobjOrderCols = New Collection

    InitCtrls       '화면 컨트롤들을 초기화하는 함수
    RefreshOrder    '현재 주문내역을 화면에 출력하는 함수
    Exit Sub

FAIL:
    MsgBox ("서버와 연결이 실패하였습니다")
    Exit Sub
End Sub

'종료 버튼 Click
Private Sub cmdDisconnect_Click()
    ClearCtrls      '컨트롤들을 초기화 하는 함수

    Set mobjOrderSink = Nothing     '이벤트 요청을 해제
    Set mobjServer = Nothing        '이벤트 요청을 해제
End Sub
```

엑셀 디자인 화면에서 '연결', '종료' Button 을 더블클릭하면 코드 창에 자동으로 버튼 클릭시 실행될 이벤트 함수가 선언됩니다.

실행 버튼 이벤트 함수에서 위와 같이 작성해줍니다.

Visual Basic 의 'GetObject 함수'를 이용하여 GOM 의 Root Object 를 연결합니다. 이렇게 연결한 Root Object 를 통해서 GOM 의 모든 Object 에 접근 할 수가 있습니다.

'연결' Button 을 눌렀을 경우 서버에 연결됨과 동시에 InitCtrls 함수에서 컨트롤들을 초기화 해주고 주문전송을 가능하게 준비합니다.

RefreshOrder 라는 함수를 작성하여 GOM 서버에 처음 연결했을 때 해당 계좌에

'종료' 버튼을 눌렀을 경우는 컨트롤들을 클리어하고 이벤트 요청을 해제합니다.

컨트롤 초기화 함수 작성

GOM 서버에 연결했을 때, Excel 화면에 컨트롤들을 초기화 하는 함수를 작성합니다.

```
'컨트롤 초기화
Private Sub InitCtrls()
    If (mobjServer Is Nothing) Or (mobjOrderHandler Is Nothing) Then Exit Sub

    Dim objAccount As GxAccount

    '계좌 콤보 박스를 초기화 하고, 계좌 목록을 만들어준다.
    cboAccounts.Clear
    For Each objAccount In mobjServer.TradeStore.Accounts
        cboAccounts.AddItem objAccount.Code
    Next objAccount

    '첫번째 계좌로 기본선택해준다.
    If cboAccounts.ListCount > 0 Then
        cboAccounts.ListIndex = 0
    End If
```

```
    '주문 종류 콤보 초기화
    cboOrderTypes.Clear
    With cboOrderTypes
        .AddItem "매수"
        .AddItem "매도"
        .AddItem "정정"
        .AddItem "취소"
    End With

    '주문 종류를 매수로 기본선택해준다.
    If cboOrderTypes.ListCount > 0 Then
        cboOrderTypes.ListIndex = 0
    End If

    '가격 종류 콤보 초기화
    cboPriceTypes.Clear
    With cboPriceTypes
        .AddItem "지정가"
        .AddItem "시장가"
        .AddItem "조건부지정가"
        .AddItem "최유리지정가"
    End With

    '가격 종류를 지정가로 기본선택해준다.
    If cboPriceTypes.ListCount > 0 Then
        cboPriceTypes.ListIndex = 0
    End If

    '종목란에 기본적으로 선물최근월물 코드 자동 입력
    Dim objSymbol As GxSymbol
    Set objSymbol = mobjServer.SymbolStore.NearestFuture
    txtSymbol.Text = objSymbol.Code

    '가격란에 선물 현재가 자동 입력
    txtPrice.Text = Format(objSymbol.Close, "0.00")
    txtQty.Text = 1
    txtLastMsg.Text = ""
End Sub
```

주문 전송 이벤트 작성

서버의 모든 주문을 클라이엔트에 출력하게 하는 함수를 작성해줍니다.

```
'현재 GOM 서버의 모든 주문을 클라이언트에 로딩합니다.
Private Sub RefreshOrder()
    Dim i, intRow As Integer
    Dim objOrder As GxOrder

    If mobjTradeStore Is Nothing Then Exit Sub

    intRow = 3
    For i = mobjTradeStore.Orders.Count To 1 Step -1
        Set objOrder = mobjTradeStore.Orders(i)
```

```
        AssignOrder objOrder, intRow      '셀 갱신
        AddOrder objOrder, -1      'Collection 갱신
        intRow = intRow + 1

        If intRow = MAX_ORDER_ROW Then Exit Sub
    Next i
End Sub
```

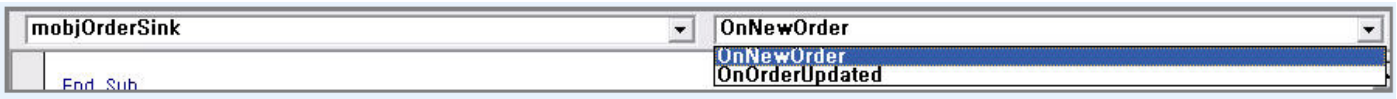
주문 전송 이벤트 작성

주문 전송 버튼을 눌렀을 때 이벤트로, 실제 서버로 주문을 전송하는 작업을 작성해줍니다.

```
'전송 버튼 Click
Private Sub cmdSend_Click()
    If (mobjServer Is Nothing) Or (mobjOrderHandler Is Nothing) Then Exit Sub

    Dim strSymbol, strAccount As String
    Dim iptPriceType As IPriceType
    Dim objOrderReq As GxOrderReq
    Dim intQty As Integer
    Dim dblPrice As Double
    Dim objTarget As GxOrder

On Error GoTo FAIL_ORDER
    '입력값 준비
    strSymbol = Trim(txtSymbol.Text)      '계좌코드
    strAccount = Trim(cboAccounts.Text)      '종목코드

    iptPriceType = iprtNA      '가격종류
    Select Case cboPriceTypes.ListIndex      '가격종류
        Case 0: iptPriceType = iptPrice
        Case 1: iptPriceType = iptMarket
        Case 2: iptPriceType = iptCondition
        Case 3: iptPriceType = iptBestPrice
    End Select

    intQty = txtQty.Text      '수량
    dblPrice = txtPrice.Text      '가격

    '실제 주문 전송부분
    Select Case cboOrderTypes.ListIndex
        Case 0:      '매수 주문
            Set objOrderReq = mobjOrderHandler.PutNewOrder(strAccount, strSymbol, iptLong, iptPriceType, intQty,
dblPrice)
        Case 1:      '매도 주문
            Set objOrderReq = mobjOrderHandler.PutNewOrder(strAccount, strSymbol, iptShort, iptPriceType, intQty,
dblPrice)
        Case 2:      '정정 주문
            Set objOrderReq = mobjOrderHandler.PutChangeOrder(mobjTarget, intQty, iptPriceType, dblPrice)
        Case 3:      '취소 주문
            Set objOrderReq = mobjOrderHandler.PutCancelOrder(mobjTarget, intQty)
    End Select

    '주문 전송이 실패하면 에러 메세지를 출력해준다.
```

```
    If objOrderReq Is Nothing Then
        txtLastMsg.Text = mobjOrderHandler.LastMessage
    Else
        txtLastMsg.Text = ""
        mobjOrderHandler.Send
    End If

    Exit Sub

FAIL_ORDER:
    txtLastMsg.Text = ""
    MsgBox "주문입력 값이 잘못되었습니다"
    Exit Sub

End Sub
```

종목 Event 를 받았을 때 처리하는 Event 함수 작성



Visual Basic Editor 코드 창 위에서, 프로그램 작성시 처음 선언해주었던 주문 Event 를 받을 Object 인 mobjOrderSink 선택하고 OnNewOrder Event 와 OnOrderUpdated Event 를 선택하면 자동으로 함수 원형이 만들어집니다.

OnNewOrder Event 함수를 다음과 같이 작성해줍니다.

```
'새로운 주문 이벤트
Private Sub mobjOrderSink_OnNewOrder(ByVal aGxOrder As Object)
On Error GoTo FAIL:
    Dim objOrder As GxOrder

    '새로운 주문이 전송되면 Excel 화면에 새로운 주문을 최상단에 출력해주기 위해서
    '기존의 주문목록들을 한칸씩 아래로 이동시킵니다.
    MoveDownOrder

    Set objOrder = aGxOrder

    AssignOrder objOrder, 3     '새로운 주문을 주문목록의 최상단에 출력해줍니다.
    AddOrder objOrder, 1        '주문내역을 저장할 Collectiond 의 제일 처음에 새로운 주문을 추가해줍니다.

FAIL:

End Sub
```

OnOrderUpdated Event 를 다음과 같이 작성해줍니다.

```
'주문 변경 이벤트
Private Sub mobjOrderSink_OnOrderUpdated(ByVal aGxOrder As Object)
On Error GoTo FAIL:
    Dim intIndex As Integer
```

```
    Dim objOrder As GxOrder
    Set objOrder = aGxOrder

    '변경된 주문을 주문을 저장한 Collection 에서 찾아서 몇번째 항목에 있는지를 찾습니다.
    intIndex = FindOrder(objOrder)

    If intIndex < 0 Then Exit Sub

    If (intIndex + 2 > 0) And (intIndex + 2 <= MAX_ORDER_ROW) Then
        AssignOrder objOrder, intIndex + 2      '변경된 주문목록에 대해서 Excel 화면을 갱신해줍니다.
    End If

FAIL:

End Sub
```

새로운 주문이 전송이 되면 OnNewOrder Event 가 발생하고 기존 주문에 대한 변경이 생기면 OnOrderUpdated Event 가 발생하여 작업이 이루어집니다.

Excel 에 주문 목록을 출력하기 위한 함수 작성

```
'주문 셀을 한 칸씩 아래로 이동시킵니다.
Private Sub MoveDownOrder()
    Dim i As Integer
    Dim objSource, objTarget As Range

    '주문내역에 있는 항목들을 한칸씩 아래로 이동합니다.
    For i = MAX_ORDER_ROW To 4 Step -1
        Set objTarget = Range(Cells(i, 7), Cells(i, 12))
        Set objSource = Range(Cells(i - 1, 7), Cells(i - 1, 12))
        objTarget.Value = objSource.Value
    Next i

End Sub
```

```
'해당 Row 에 mobjGxOrder 의 값을 할당
Private Sub AssignOrder(objGxOrder As IGxOrder, ByVal intRow As Integer)
    If objGxOrder Is Nothing Then Exit Sub
    If (intRow < 3) Or (intRow > MAX_ORDER_ROW) Then Exit Sub

    With objGxOrder
        Cells(intRow, 7) = objGxOrder.Account.Code
        Cells(intRow, 8) = objGxOrder.Symbol.Code
        Cells(intRow, 9) = objGxOrder.SrvAcptNo
        Cells(intRow, 10) = objGxOrder.KseAcptNo
        Cells(intRow, 11) = objGxOrder.Qty

        Select Case .OrderState
            Case iosConfirm: Cells(intRow, 12) = "확인됨"
            Case iosDead: Cells(intRow, 12) = "죽은주문"
            Case iosFullFill: Cells(intRow, 12) = "전량체결"
            Case iosKseAcpt: Cells(intRow, 12) = "증전접수"
```

```
        Case iosPartFill: Cells(intRow, 12) = "부분체결"
      End Select
    End With

End Sub
```

AssignOrder 함수는 Excel 의 지정된 행에 해당 주문을 출력해줍니다.

MoveDownOrder 함수는 새로운 주문을 최상단에 출력해주기 위해서, 기존의 주문목록들을 한행씩 아래로 이동시켜줍니다.

Collection 에 주문목록을 저장하기 위한 함수 작성

정정이나 취소주문에서 사용하기 위해서 프로그램 처음에 선언했던 Collection 에 새로운 주문을 저장합니다.

```
'Collection 에 추가 작업
Private Sub AddOrder(objGxOrder As IGxOrder, ByVal intIndex As Integer)
  If objGxOrder Is Nothing Then Exit Sub

  'Collection 에 저장된 주문이 없으면 바로 추가시켜주고
  '주문목록이 존재하면 지정된 위치에 추가해줍니다.
  If (intIndex < 0) Or (mobjOrderCols.Count = 0) Then
    mobjOrderCols.Add objGxOrder
  Else
    mobjOrderCols.Add Item:=objGxOrder, before:=intIndex
  End If

  If mobjOrderCols.Count > 18 Then
    mobjOrderCols.Remove (mobjOrderCols.Count)
  End If
End Sub
```

Collection 에 저장된 주문은 정정, 취소 주문시 원주문 번호를 기억하기 위해서 사용되며, Excel 의 화면에 주문목록을 출력할 때도 사용됩니다.

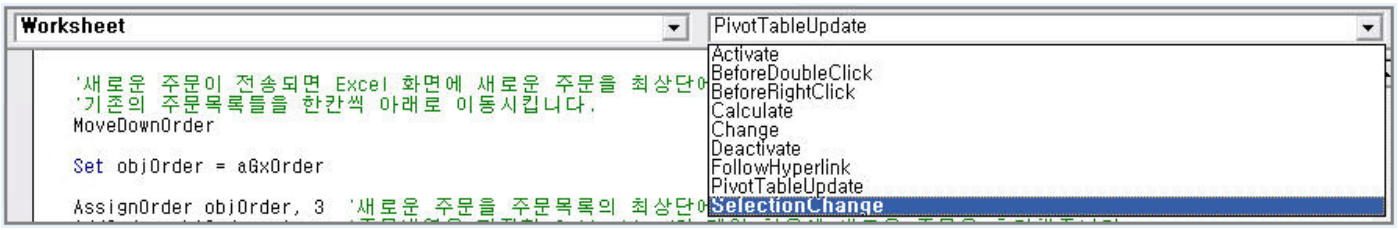
Collection 에 저장된 주문목록에서 원하는 주문의 위치를 찾기 위해서 FindOrder 함수를 작성해줍니다.

```
'Collection 에서 해당 아이템 찾기 (리턴값 : Collection 의 위치 인덱스)
Private Function FindOrder(mobjGxOrder As IGxOrder) As Integer
  Dim intIndex As Integer
  Dim objOrder As GxOrder
  FindOrder = -1
  intIndex = 1

  '주문목록이 저장된 Collection 에서 해당 주문을 찾습니다.
  For Each objOrder In mobjOrderCols
    If objOrder Is mobjGxOrder Then
      FindOrder = intIndex
      Exit Function
    End If

    '해당 주문이 몇번째 항목에 있는지 설정합니다.
```

```
    intIndex = intIndex + 1
  Next objOrder
End Function
```

정정, 취소 주문을 위해 주문을 선택했을 때의 이벤트 작성

Excel 의 주문목록에서 정정이나 취소할 주문을 선택했을 때, 주문설정 컨트롤에 해당 주문의 원주문번호를 설정해줍니다.



Excel 에서 해당 셀을 선택했을 때의 이벤트를 작성하기 위해서 Visual Basic Editor 코드 창 위에서 WorkSheet 객체를 선택하고, SelectionChange 이벤트를 선택하여 작성해줍니다.

```
'주문부 셀을 클릭 했을 때
Private Sub Worksheet_SelectionChange(ByVal Target As Range)
  Dim intRow, intCol As Integer
  Dim objOrder As GxOrder

  '선택한 주문 목록이 출력된 행과 열을 지정해줍니다.
  intRow = Target.Row
  intCol = Target.Column

  If mobjOrderCols Is Nothing Then Exit Sub

  If (intRow < 3) Or (intRow > mobjOrderCols.Count + 2) Then Exit Sub
  If (intCol < 7) Or (intCol > 12) Then Exit Sub

  Set objOrder = mobjOrderCols(intRow - 2)
  If objOrder Is Nothing Then Exit Sub

  '선택한 주문에 대해서 미체결 수량이 0 이면 함수를 빠져나갑니다.
  If objOrder.UnFillQty = 0 Then Exit Sub

  Set mobjTarget = Nothing
  txtTarget.Text = ""

  '정정이나 취소주문을 위해서 선택한 주문에 대한 주문번호를 원주문번호에 설정해줍니다.
  Set mobjTarget = objOrder
  txtTarget.Text = objOrder.Account.Desc + "/" _
                 + objOrder.Symbol.Code + "/" + CStr(objOrder.SrvAcptNo)
End Sub
```

마. 챠트익히기

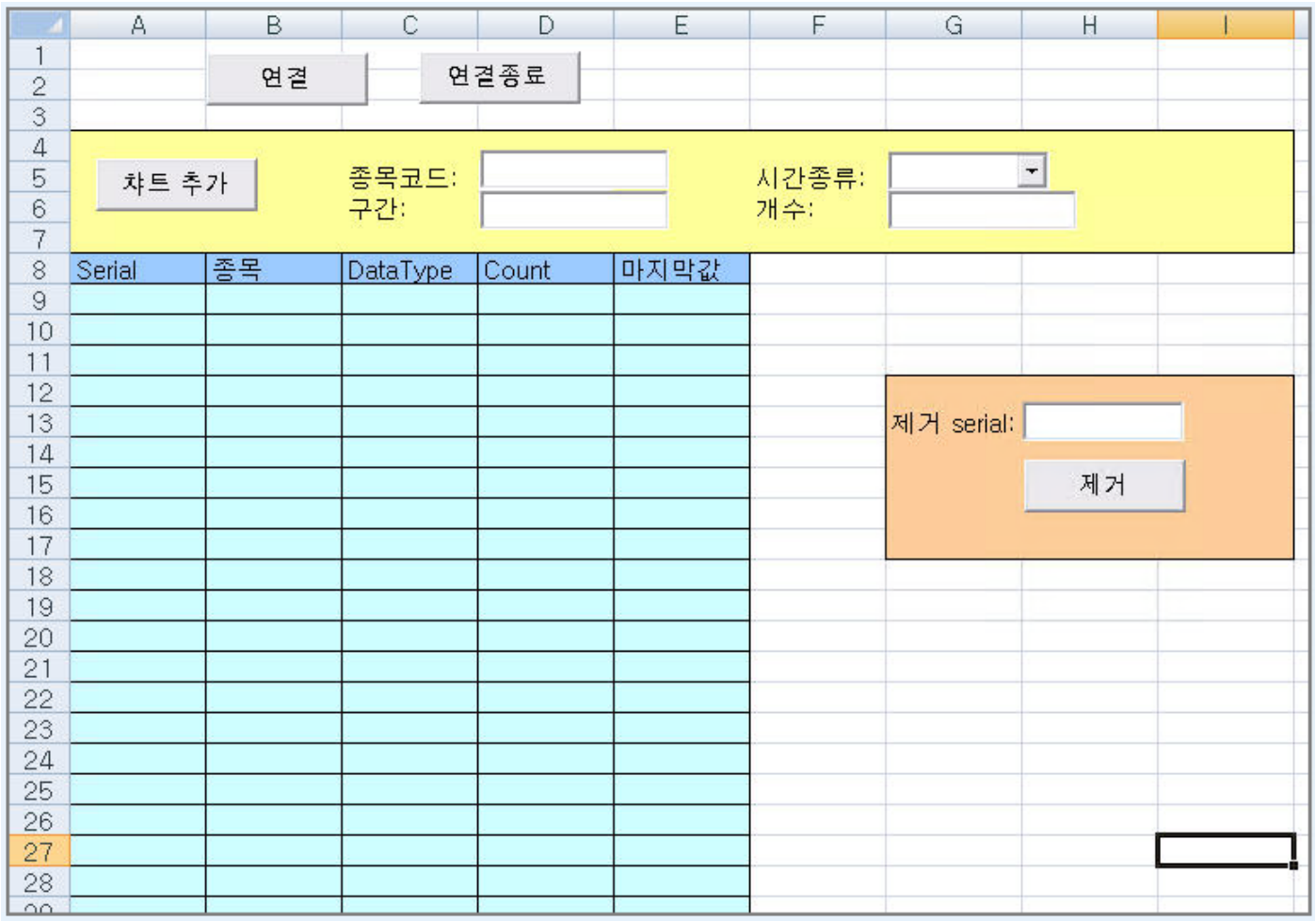
챠트익히기

개요

여기서는 GOM 의 챠트에 대한 Object 사용법을 간단하게 설명하겠습니다. GOM 의 챠트 Object 를 이용하여 종목의 챠트데이타를 Excel 화면에 표시할 수 있습니다. 이번 예제에서는 여러 종목의 챠트를 요청하여 마지막 봉의 현재가만을 출력하는 예제를 만들어보겠습니다.

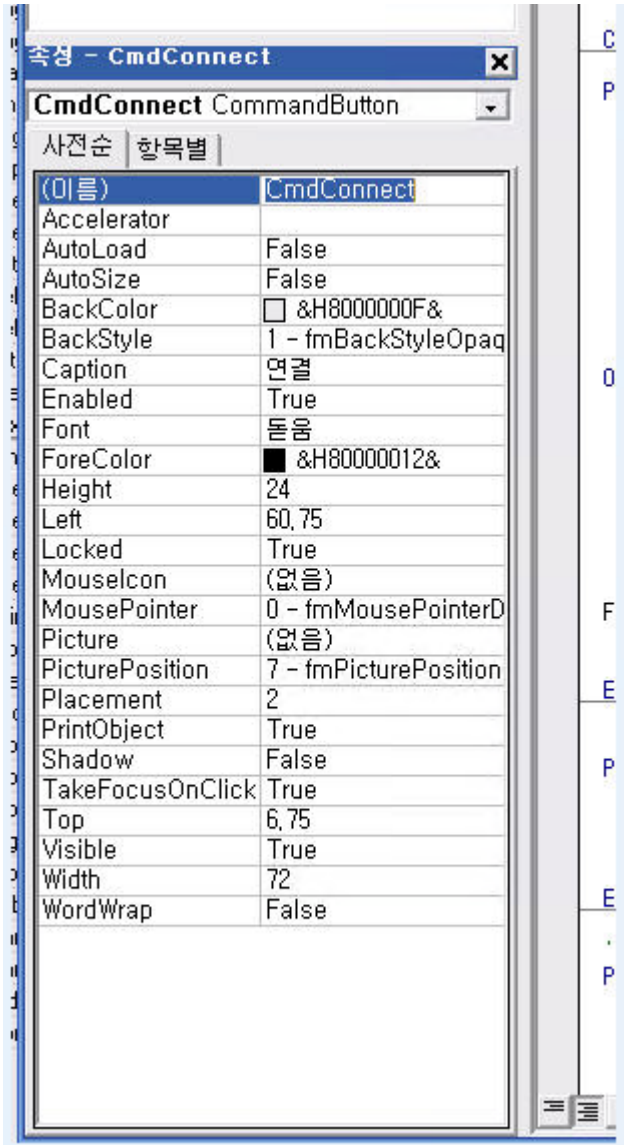
엑셀에서 화면 디자인하기

삽입 > Visual Basic 컨트롤 양식 상자에 있는 컨트롤들을 이용해서 다음 화면과 같이 종목코드와 구간, 챠트 데이타의 갯수, 추가한 챠트의 Serial 번호를 입력할 수 있는 Edit Box 와 챠트의 시간종류를 설정하는 Combo Box 를 만들고 GOM 서버에 연결, 연결종료, 챠트를 추가, 삭제하는 Button 을 올려놓습니다.



Visual Basic Editor 속성창에서의 작업

디자인했을 때 올려놓은 컨트롤들의 이름을 지정해 줍니다. 여기서 지정한 이름은 프로그램안에서 통용되는 이름으로 다른 이름들과는 구별되어야 합니다.



위에 보시는 속성 창을 이용하여 바로 전 단계에서 디자인했던 컨트롤들의 이름을 지정하여 줍니다.

종목코드, 구간, 챠트데이타의 갯수, 추가한 챠트의 Serial 번호를 입력하는 Edit Box 의 이름은 각각 txtSymbolCode, txtPeriod, txtCount, txtRemove 로 지정합니다.

서버 연결, 연결종료, 챠트추가, 챠트삭제 Button 의 이름은 각각 cmdConnect, cmdDisconnect, cmdAdd, cmdRemove 로 지정합니다.

챠트의 시간종류를 설정하는 Combo Box 는 cboType 로 지정합니다.

Visual Basic Editor 코드창에서의 작업

Visual Basic Editor 의 코드창에서 실제 프로그래밍해야되는 작업에 대해 설명하겠습니다.

```
Option Explicit
  'GOM Server 연결 Object
  Public mobjServer As GxServer

  'GxChartStore 연결 Object
  Public mobjChartStore As GxChartStore
```

```
  '여러종목의 챠트데이타와 각각에 발생하는 이벤트를 관리하기 위해 Collection 에 저장
  Public mobjChartDatas As New Collection
  Public mobjChartSinks As New Collection

'Excel 에 출력되는 시작 행을 상수형 변수로 선언
Const START_ROW = 9
```

첫번째로 예제에서 사용될 전역변수들을 선언해줍니다.

서버 인터페이스에 연결될 변수를 선언해줍니다.

이번예제에서 사용되는 챠트 Object 에 사용될 Object 변수와, 요청한 챠트데이타와 각각에서 발생하는 이벤트를 관라하는 Collection Object 를 선언합니다.

버튼 클릭시 이벤트 연결

디자인 작업에서 만들었던 Button 을 클릭했을 때 실행될 이벤트를 작성해줍니다.

엑셀 디자인 화면에서 '연결', '연결종료' Button 을 더블클릭하면 코드 창에 자동으로 버튼 클릭시 실행될 이벤트 함수가 선언됩니다.

연결 버튼과 연결종료 이벤트 함수에서 아래와 같이 작성해줍니다.

```
Private Sub CmdConnect_Click()
  '컨트롤 초기화
  cboTypes.Clear
  cboTypes.AddItem ("Tick")
  cboTypes.AddItem ("분")
  cboTypes.AddItem ("일")
  cboTypes.AddItem ("주")
  cboTypes.AddItem ("월")
  cboTypes.ListIndex = 0
  txtPeriod.Text = 1
  txtCount.Text = 10

On Error GoTo FAIL:
  'GOM Server 와 연결해준다.
  Set mobjServer = GetObject(, "P2.GxServer")
  'GxChartStore 와 연결
  Set mobjChartStore = mobjServer.ChartStore

  'Excel 화면의 종목 컨트롤에 종목명을 기본으로 설정해준다.
  txtSymbolCode.Text = mobjServer.SymbolStore.NearestFuture.Code

  '서버에 현재 생성된 챠트 데이터를 Collection 에 재수집함
  CollectChartList

  'ChartData 를 Excel 에 출력해준다.
  RefreshChartData
  Exit Sub

FAIL:
```

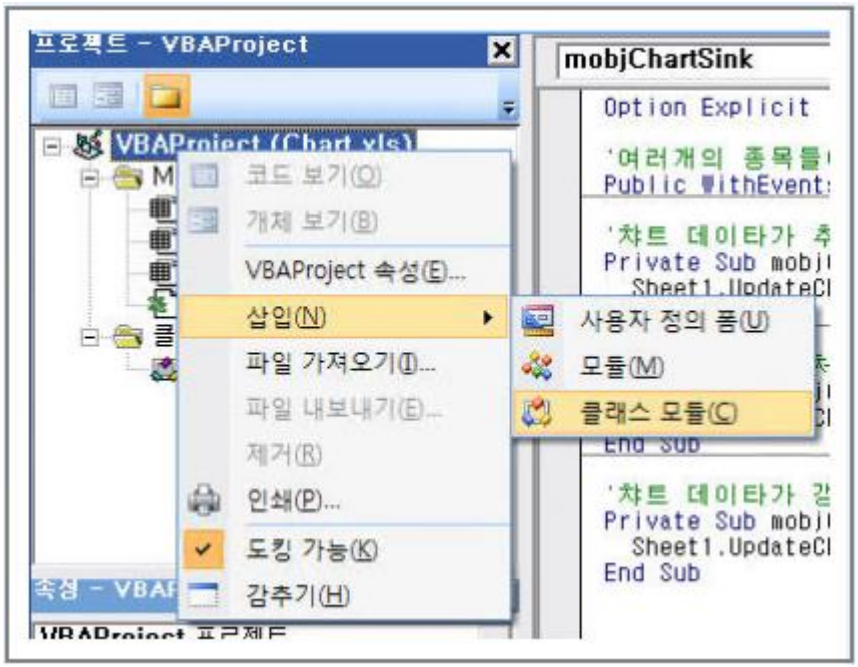
```
    MsgBox "서버와 연결하는데 실패하였습니다"
    Exit Sub
End Sub

Private Sub CmdDisconect_Click()
  '모든 종목의 Event 연결을 해제해준다.
  DisconnectAllData

  'ChartStore 와 GOMServer 연결을 해제해준다.
  Set mobjChartStore = Nothing
  Set mobjServer = Nothing
End Sub
```

연결 버튼 이벤트에서는 GOM 서버에 연결시키고 컨트롤들을 초기화하고 GOM 서버에 현재 생성된 챠트 데이타들을 불러와 Collection 에 저장하는 작업을 수행합니다.

종료버튼의 이벤트에서는 모든 종목의 Event 연결을 해제하고 GOM 서버와의 연결을 해제합니다.

챠트 데이타의 Class 화

여러개 종목에 대해서 Event 수신을 하기 위해서 GxChartData 를 클래스화하여 사용합니다.



VBA 프로젝트에서 팝업메뉴를 통해서 클래스 모듈을 삽입해줍니다.

삽입한 클래스 모듈은 clsEventSink 라는 이름으로 지정해주며, 이 클래스는 GxChartData Object 를 가지고서 ChartData 의 Event 를 받는 역할을 하게 됩니다.

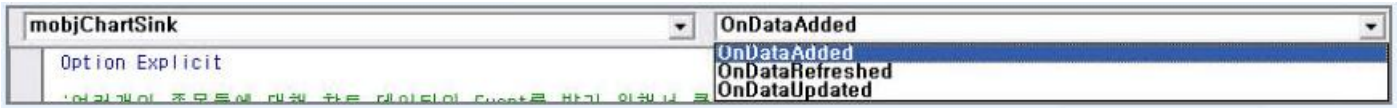
삽입한 클래스 모듈에 다음의 코드들을 작성합니다.

```
Option Explicit

'여러개의 종목들에 대해 챠트 데이타의 Event 를 받기 위해서 클래스로 만들어주었다.
Public WithEvents mobjChartSink As GxChartData
```

VBA 프로젝트에서 팝업메뉴를 통해서 클래스 모듈을 삽입해줍니다.

삽입한 클래스 모듈은 clsEventSink 라는 이름으로 지정해주며, 이 클래스는 GxChartData Object 를 하나 가지게 됩니다. 또한 이 GxChartData Object 의 Event 를 받는 역할을 하게 됩니다.

다음과 같이 clsEventSink 클래스에 GxChartData Object 를 선언해줍니다.



위에서 선언한 GxChartData Object 의 Event 함수를 작성합니다.

Visual Basic Editor 코드 창 위에서 위에서 선언한 mobjChartSink Object 를 선언하고 OnDataAdded, OnDataRefreshed, OnDataUpdated 이벤트를 다음과 같이 작성합니다.

이렇게 하여 여러종목에 대한 챠트데이타를 요청할 경우, 모든 종목에 대해서 Event 처리가 가능해집니다.

```
'챠트 데이타가 추가될때 발생
Private Sub mobjChartSink_OnDataAdded(ByVal aGxChartData As Object)
  Sheet1.UpdateChartData aGxChartData
End Sub

'챠트 데이타를 처음 요청했을 때 발생
Private Sub mobjChartSink_OnDataRefreshed(ByVal aGxChartData As Object)
  Sheet1.UpdateChartData aGxChartData
End Sub

'챠트 데이타가 갱신될 때 발생
Private Sub mobjChartSink_OnDataUpdated(ByVal aGxChartData As Object)
  Sheet1.UpdateChartData aGxChartData
End Sub
```

챠트 추가 버튼 클릭시 Event 작성

챠트 추가 버튼을 클릭할 경우, 설정한 종목에 대해서 챠트 데이타를 가지고 오게 됩니다.

```
'챠트 추가 버튼을 누를 때 -> 서버가 연결이 되 있어야 합니다.
Private Sub CmdAdd_Click()
  If mobjServer Is Nothing Then
    MsgBox "서버와 연결해야 합니다"
  Else
    Dim objChartData As GxChartData
    Dim DataType As IChartBaseType
    Dim iCount, iPeriod As Integer
    Dim objSink As clsEventSink

    On Error GoTo FAIL:

    '챠트 시간 종류 설정
    Select Case cboTypes.ListIndex
      Case 0: DataType = icbTick
      Case 1: DataType = icbMin
```

```
        Case 2: DataType = icbDaily
        Case 3: DataType = icbWeekly
        Case 4: DataType = icbMonthly
        Case Else: DataType = icbNA
    End Select

    iCount = txtCount.Text      '챠트 갯수
    iPeriod = txtPeriod.Text    '챠트 기간설정(챠트 시간 종류에 따라서 1 틱,2 틱,1 분,2 분,1 일,2 일등...)

    'ChartStore 에 챠트 데이터(GxChartData) 생성
    Set objChartData = mobjChartStore.Add

    'mobjChartDatas Collection 에 생성된 챠트데이타를 저장
    mobjChartDatas.Add objChartData

    '생성된 챠트 데이터 설정
    If objChartData.Define(Trim(txtSymbolCode.Text), icdOHLC, DataType, iPeriod, iCount) Then
      '이벤트 연결 -> 다수의 이벤트를 받기 위하여 clsEventSink 생성 후 Collection 에 저장
      Set objSink = New clsEventSink
      'clsEventSink 의 GxChartData 에 새로 생성한 챠트데이터(GxChartData)를 할당해준다.
      Set objSink.mobjChartSink = objChartData

      '새로 생성한 clsEvnetSink 를 mobjChartSinks Collection 에 저장
      mobjChartSinks.Add objSink
    Else
      MsgBox "데이터 요청 실패"
    End If

  End If

  Exit Sub

FAIL:
  MsgBox "입력값에 문제가 있습니다"
  Exit Sub

End Sub
```

GxChartData 의 Define 함수를 사용하여 챠트 데이타를 가지고 오게 되고, 이 GxChartData Object 는 mobjChartDatas Collection 에 저장하게 됩니다.

클래스 관리와 이벤트 관리를 위해서 mobjChartSinks 에 새로 생성한 clsEventSink Object 도 저장하게 됩니다.

Excel 화면에 챠트 데이타 출력 함수 작성

챠트를 처음 추가하거나 추가했던 챠트 데이타가 갱신되면, 해당 데이타를 Excel 화면에 출력하는 함수를 작성합니다.

```
'해당 챠트의 상태의 변화를 셀에 표시함
Public Sub UpdateChartData(objChartData As GxChartData)
  Dim i, iOffset As Integer
  Dim bFind As Boolean

  bFind = False
  iOffset = -1
```

```
    'mobjChartDatas 에 저장된 챠트중에서 찾는다.
    For i = 1 To mobjChartDatas.Count
      If mobjChartDatas(i) Is objChartData Then
        bFind = True
        iOffset = i
        Exit For
      End If
    Next i

    'Excel 화면에 저장된 순서대로 출력해준다.
    If bFind Then
      Cells(START_ROW + iOffset - 1, "A") = objChartData.SerialNo

      If Not objChartData.Symbol Is Nothing Then
        Cells(START_ROW + iOffset - 1, "B") = objChartData.Symbol.Code
      End If

      Cells(START_ROW + iOffset - 1, "C") = objChartData.Base
      Cells(START_ROW + iOffset - 1, "D") = objChartData.DataCount

      '챠트데이타의 마지막데이타만 출력해준다.
      'objChartData.Terms(index) 의 형식으로 원하는 index 의 챠트 데이타에 접근할 수 있다.
      If Not objChartData.DataCount = 0 Then
        Cells(START_ROW + iOffset - 1, "E") = objChartData.Terms(objChartData.DataCount).Close
      End If

    End If
  End Sub

  '전체 챠트의 상태의 변화를 셀에 표시함
  Private Sub RefreshChartData()
    Dim i As Integer

    Sheet1.Range("A9", "E100") = ""

    'mobjChartDatas Collection 에 있는 ChartData 들을 Excel 에 출력한다.
    For i = 1 To mobjChartDatas.Count
      UpdateChartData mobjChartDatas(i)
    Next i

  End Sub
```

GOM 서버에 저장되어 있는 챠트 데이타를 불러오는 함수 작성

GOM 서버에 처음 연결하거나, 추가했던 챠트 데이타를 삭제할 경우, GOM 서버에서 챠트데이타를 다시 갱신받는 함수를 작성합니다.

```
  '서버에 현재 생성된 챠트 데이터를 재수집함
  Private Sub CollectChartList()
    If mobjChartStore Is Nothing Then Exit Sub

    Dim objChartData As GxChartData
    Dim objEventSink As clsEventSink
```

```
    'ChartStore 에 저장된 챠트 데이타를 mobjChartDatas, objEventSink Collection 에 넣어준다.
    For Each objChartData In mobjChartStore
        'mobjChartDatas 에 챠트 데이타 저장
        mobjChartDatas.Add objChartData

    'objEventSink 에 clsEventSink 를 생성하여 저장
    Set objEventSink = New clsEventSink
    Set objEventSink.mobjChartSink = objChartData

    mobjChartSinks.Add objEventSink

    Next objChartData
End Sub
```

GOM 서버에 저장되어 있는 모든 챠트 데이타를 mobjChartDatas 에 저장하고 이에 따라서 새로 생성된 clsEventSink Object 들도 mobChartSinks 에 저장합니다.

챠트 삭제 버튼 클릭시 Event 작성

추가했던 챠트의 Serial 번호를 통해서 해당 챠트 데이타를 삭제해줍니다.

```
'해당 serial 에 대한 챠트 데이터 삭제
Private Sub cmdRemove_Click()
    If Not mobjChartStore Is Nothing Then
        'ChartStore 에서 해당 Serial 을 가진 데이타를 삭제해준다.
        mobjChartStore.RemoveBySerialNo (txtRemove.Text)

    End If

    'mobjChartDatas, mobjChartSinks Collection 에 저장된 데이타를 초기화해준다.
    DisconnectAllData

    '서버에 현재 생성된 챠트 데이터를 Collection 에 재수집함
    CollectChartList

    'ChartData 를 Excel 에 출력해준다.
    RefreshChartData
End Sub

'모든 챠트 데이터와 이벤트 연결 해제
Private Sub DisconnectAllData()
    Dim i As Integer
    Dim objEventSink As clsEventSink

    'objEventSink 에 할당된 챠트 데이타를 해제해준다.
    For i = 1 To mobjChartSinks.Count
        Set objEventSink = mobjChartSinks(i)
        Set objEventSink.mobjChartSink = Nothing
    Next i

    Set mobjChartSinks = Nothing
    Set mobjChartDatas = Nothing
End Sub
```

챠트 삭제시 입력한 해당 Serial 번호의 GxChartData 를 mobjChartStore 에서 찾아서 삭제해줍니다.

MoveDownOrder 함수는 새로운 주문을 최상단에 출력해주기 위해서, 기존의 주문목록들을 한행씩 아래로 이동시켜줍니다.

3. GOM 구조도

가. GOM 구조도

개요

GOM 에 정의된 열거형 타입을 설명 합니다. 이에 정의된 타입을 참고하여 메소드를 호출하는데 사용하거나 속성을 참조하는데 사용합니다.

Constant 설명

IPositionType

포지션 타입 입니다.

iptLong (0) : 매수 타입입니다.

iptShort (1) : 매도 타입입니다.

iptNA (100) : N.A

IOrderType

주문 종류 입니다.

iotNew (0) : 신규 주문 타입 입니다.

iotChange (1) : 정정 주문 타입 입니다.

iotCancel (2) : 취소 주문 타입 입니다.

iotNA (100) : N.A

IPriceType

가격 유형 입니다.

iptPrice (0) : 지정가 입니다.

iptMarket (1) : 시장가 입니다.

iptCondition (2) : 조건부 지정가 입니다.

iptBestPrice (3) : 최유리 지정가 입니다.

iprtNA (100) : N.A

IUnderlyingType

기초자산 타입 입니다.

iutKospi200 (0) : KOSPI200 입니다.

<유안타> iiutKosdaq50 (1): KOSDAQ50 입니다.

iutStock (2) : 주식 입니다.

iutBond (3) : 국채 입니다.

iutCurrency (4) : 통화 입니다.

iutProduct (5) : 상품 입니다.

iutNA (100) : N.A

ISymbolType

종목 타입 입니다.

istIndex (0) : 지수 타입 입니다.

istStock (1) : 주식 타입 입니다.

istFuture (2) : 선물 타입 입니다.

istOption (3) : 옵션 타입 입니다.

istCombi (4) : 스프레드 타입 입니다.

istBond (5) : 채권 타입 입니다.

istNA (100) : N.A

IOrderState

체결 조건 입니다.

iosKseAcpt (0) : 미체결 / 미확인 상태 입니다.

iosDead (1) : 죽은 주문 상태 입니다.

iosFullFill (2) : 전량 체결 상태 입니다.

iosPartFill (3) : 부분 체결 상태 입니다.

iosConfirm (4) : 확인된 상태 입니다.

iosNA (100) : N.A

IChartMarketFlag

차트 장구분 입니다.

icmDay (0) : 주간장

icmNight (1) : 야간장

icmDayNight (2) : 주간장 + 야간장

IChartBaseType

챠트 종류 입니다.

icbTick (0) : Tick 챠트 입니다.

icbMin (1) : 분 챠트 입니다.

icbDaily (2) : 일 챠트 입니다.

icbWeekly (3) : 주 챠트 입니다.

icbMonthly (4) : 월 챠트 입니다.

icbNA (100) : N.A

IOrderReqState

주문 요청 상태 입니다.

irsNew (0) : 신규 입니다.

irsSendRjt (1) : 전송 거부 입니다.

irsSendWait (2) : 전송 대기 입니다.

irsSrvWait (3) : 서버 대기 입니다.

irsSrvRjt (4) : 서버거부 입니다.

irsKseWait (5) : 증전대기 입니다.

IEmOrderType

비상주문 타입 입니다.

ieoLong (0) : 매수 주문 입니다.

ieoShort (1) : 매도 주문 입니다.

ieoChange (2) : 정정 주문 입니다.

ieoCancel (3) : 취소 주문 입니다.

ieoNA (100) : N.A

IChartDataType

챠트 데이터 타입입니다.

icdOHLC (0) : 시/고/저/종 챠트 입니다.

icdOpenInterest (1) : 미결제 약정 챠트 입니다.

icdNA (100) : N.A

IOptionType

옵션 Call/Put 여부 입니다.

iotCall (0) : Call 종목 입니다.

iotPut (1) : Put 종목 입니다.

ioptNA (100) : N.A

IGreeksCalcType

옵션 그릭스 잔존일수 계산 방식 입니다.

igcCalDate(0) : 달력일수 날짜기준 입니다.

igcCalTime(1) : 달력일수 시간기준 입니다.

igcTrdDate(2) : 거래일수 날짜기준 입니다.

igcTrdTime(3) : 거래일수 시간기준 입니다.

igcNA(100) : N.A

IFillType

체결 유형 입니다.

<유안타증권>

iftFOK(0) : FOK 주문입니다.

iftIOC(1) : IOC 주문입니다.

iftFAS(2) : 일반주문입니다.

igcNA(100) : N.A

IMarketType

장구분 조회입니다.

imtDay(0) : 주간장입니다.

imtNight(1) : 야간장입니다.

개요

GOM 의 모든 Object 는 LastMessage(BSTR) 속성을 제공합니다. LastMessage 에는 GOM Object 의 최근 에러 사항을 기록합니다. LastMessage 는 소스위치함수위치[ErrorCode]:ErrorMsg 형식을 가짐니다. 소스위치나 함수 위치는 관리자의 식별용이므로 신경쓰지 않으셔도 되고 사용자 들에게 중요한 것은 [ErrorCode]:ErrorMsg 입니다.
아래에 열거한 ErrorCode 는 사용자들이 신경써야 되는 최소한의 에러코드이고 아래에 나오지 않는 에러코드는 시스템 내부 문제 이므로 궁금증을 가지지 마시고 관리자에게 알리십시요.

공통 Error Log

[GX-ERROR200]:CHASING_EVENTSINK
GOM 서버가 클라이언트에게 이벤트를 보낼 때 실패를 하여 추적을 하고 있는 중입니다.
5 초동안 5 번의 실패를 할 경우 클라이언트의 이벤트 요청을 제거합니다.
이 에러 발생 시 클라이언트는 클라이언트의 이벤트 수신부를 확인 해야 합니다.

[GX-ERROR210]:FIX_EVENTSINK
ERROR200 이 발생하여 추적 중 5 초동안 5 번의 실패를 한 경우 클라이언트의 이벤트 요청을 제거했음을 의미합니다. 이 후로 클라이언트는 이벤트를 받지 못하고 다시 이벤트 요청을 해야 합니다.

GxOrderHandler, GxEmhandler Error Log (1000 ~ 1100)

[GX-ERROR1001]:ACCOUNT_NOT_AUTHENTICATED
주문 계좌가 계좌 비밀번호가 확인 되지 않았습니다.

[GX-ERROR1003]:CANNOT_ORDERED_SYMBOL
주문 종목이 고수의 종목설정에서 종목이 주문이 나가지 않도록 설정 되었습니다.

[GX-ERROR1004]:SYMBOL_UNDERLYING_CANNOT_ORDERED
주문설정에 주문 종목의 기초자산이 주문이 나가지 않도록 설정 되었습니다.

[GX-ERROR1005]:OVER_MAXCOUNT/1
주문설정의 주문 종목의 신규수량을 초과 하였습니다.

[GX-ERROR1006]:OVER_MAXAMTS/1
주문설정의 주문 금액의 신규주문금액을 초과 하였습니다.

[GX-ERROR1007]:NOTFOUND_TARGETORDER
원주문을 찾지 못했습니다.

[GX-ERROR1008]:NOTFOUND_ACCOUNT
계좌코드로 계좌를 찾지 못했습니다.

[GX-ERROR1009]:NOTFOUND_SYMBOL
종목코드로 종목을 찾지 못했습니다.

[GX-ERROR1010]:FAIL_PUTORDER
GxOrderHandler 의 PutxxxOrder 가 실패하였습니다.

[GX-ERROR1030]:NOTFOUND_ACCOUNT
비상주문시 계좌코드로 계좌를 찾지 못했습니다.

[GX-ERROR1031]:NOTFOUND_SYMBOL
비상주문시 종목코드로 계좌를 찾지 못했습니다.

[GX-ERROR1050]:HIGHLIMIT_UPPERPRICE
주문가격이 종목 상한가를 초과하였습니다.

[GX-ERROR1060]:LOWLIMIT_LOWERPRICE
주문가격이 종목 하한가를 초과하였습니다.

[GX-ERROR1069]:TARGETORDER_NULL
PutChangeOrder, PutCancelOrder 시 TargetOrder 에 NULL 값을 입력하였습니다.

[GX-ERROR1070]:ZERO_ORDERQTY
주문수량을 0 을 입력하였습니다.

[GX-ERROR1071]:INVALID_POSITIONTYPE
잘못된 포지션 타입입니다.

[GX-ERROR1072]:INVALID_PRICETYPE
잘못된 가격 타입입니다.

[GX-ERROR1072]:INVALID_TARGETORDER_TYPE
PutChangeOrder, PutCancelOrder 시 TargetOrder 에 IGxOrder 타입이 아닌 타입을 입력하였습니다.

[GX-ERROR1074]:INVALID_TARGETORDER_NOTFOUND_ACCOUNT
잘못된 원주문 Dispatch 를 입력하였습니다.

[GX-ERROR1075]:INVALID_TARGETORDER_NOTFOUND_SYMBOL
잘못된 원주문 Dispatch 를 입력하였습니다.

[GX-ERROR1076]:INVALID_TARGETORDER_NOTFOUND_ORDER
잘못된 원주문 Dispatch 를 입력하였습니다.

[GX-ERROR1077]:CANNOT_CANCELORDER_INCANCELORDER
취소 주문에 대해 취소를 할 수 없습니다.

[GX-ERROR1078]:CANNOT_CHANGEORDER_INCANCELORDER
취소 주문에 대해 정정을 할 수 없습니다.

[GX-ERROR1080]:INVALID_TARGETNUMBER
잘못된 원주문 번호 입니다.

[GX-ERROR1081]:QTY_OVER_UNFILLQTY
정정, 취소시 미체결수량을 초과했습니다.

[GX-ERROR1082]:EQUAL_CHANGEPRICE
정정가격이 현재 주문가격과 동일합니다.

[GX-ERROR1083]:CANNOT_SEND_THE_CHANGED_ORDER_THAT_EXCEED_THE_300_TIMES_LIMIT
같은계좌, 같은종목에 대해서 정정주문 횟수가 300 번을 초과합니다.

[GX-ERROR1084]:INVALID_FILLTYPE
잘못된 체결 타입입니다.

[GX-ERROR1100]:FAIL_RPC_CALL
주문결과 통보를 위한 클라이언트 RPC 호출이 실패했습니다. 클라이언트 이벤트 수신부를 확인하십시요.

GxSymbolStore Error Log (2000 ~ 2999)

[GX-ERROR2030]:FAIL_RPCCALL_SYMBOL
종목 현재가 이벤트를 위한 클라이언트 RPC 호출이 실패했습니다.

[GX-ERROR2105]:OUTOFRANGE_CALLVOL
SetGreeksParam 호출 시 입력 CallVol 이 범위를 벗어났습니다.

[GX-ERROR2110]:OUTOFRANGE_PUTVOL
SetGreeksParam 호출 시 입력 PutVol 이 범위를 벗어났습니다.

[GX-ERROR2115]:OUTORRANGE_MIXRATIO
SetGreeksParam 호출 시 입력 MixRatio 가 범위를 벗어났습니다.

[GX-ERROR2120]:NOTSUPPORT_GREEKS
그릭스를 지원하지 않는 종목 입니다.

[GX-ERROR2125]:NOTREADY_GREEKSPARAM
그릭스를 계산하는데 실패 하였습니다.

[GX-ERROR2150]:INVALID_TIMERINTERVAL
입력 QuoteEventInterval, PriceEventInterval 이 상하한치를 벗어남니다.

GxChartStore Error Log (5000 ~ 5999)

[GX-ERROR5010]:CHART_REQUESTING
현재 누적 챠트 데이터 요청중 이므로 새로운 요청이 불가능합니다.

[GX-ERROR5030]:DEFINE_ZEROPERIOD
요청 시간간격이 1 미만이나 100 초과입니다.

[GX-ERROR5040]:NOTFOUND_SYMBOL
요청 종목코드에 해당하는 종목이 존재하지 않습니다.

[GX-ERROR5050]:WRONG_DATATYPE
잘못된 요청 챠트 데이터 타입입니다.

[GX-ERROR5100]:NOTSUPPORT_SYMBOL
요청 데이터 타입이 지원하는 종목이 아닙니다.

[GX-ERROR5110]:NOTSUPPORT_BASE
요청 데이터 타입이 지원하는 ChartBase 가 아닙니다.

[GX-ERROR5200]:INDEXOUTOFERROR
ChartStore 에서 Remove 시 입력 index 에 해당하는 GxChartData 가 없습니다.

[GX-ERROR5210]:NOTEXIST_KEY
ChartStore 에서 RemoveByKey 시 입력 key 에 해당하는 GxChartData 가 없습니다.

[GX-ERROR5230]:CHARTKEY_HIGHLIMIT
관리자에게 문의하십시요.

[GX-ERROR5235]:CHARTDATA_HIGHLIMIT
ChartStore 에서 Add 시 ChartData 개수가 상한치를 초과 하였습니다.

[GX-ERROR5240]:FAIL_RPC_OPEN_CHARTREFRESH
미결제약정 챠트에서 refresh RPC Call 시 문제 발생

[GX-ERROR5250]:FAIL_RPC_OPEN_CHARTUPDATE
미결제약정 챠트에서 update RPC Call 시 문제 발생

[GX-ERROR5260]:FAIL_RPC_OPEN_CHARTADD
미결제약정 챠트에서 ADD RPC Call 시 문제 발생

[GX-ERROR5310]:FAIL_RPC_CHARTREFRESH
OHLC 챠트에서 Refresh RPC 시 문제 발생

[GX-ERROR5320]:FAIL_RPC_CHARTUPDATE
OHLC 챠트에서 UPDATE RPC 시 문제 발생

[GX-ERROR5330]:FAIL_RPC_CHARTADD
OHLC 챠트에서 ADD RPC 시 문제 발생

[GX-ERROR5400]:FIX_CHARTSINKLIST
ChartData 에서 RPC 호출 실패가 5 초동안 지속되어 클라이언트 요청을 제거합니다.

History

GOM Histroy

GOM 변경내역

변경일	버전	변경내용
2007.10.25	1.24	**GxOrderhandler :** PutNewOrder2, PutChangeOrder2 메소드 추가 **GxEmhandler :** SendEmNewOrder2, SendEmChangeOrder2 메소드 추가 (IOC/FOK 주문가능)
2004.12.09	1.23	**GxTradeStore:** Synchronize 메소드 추가
2004.11.23	1.22	**GxChartStore:** UpateEventFiltered, UpdateEventInterval 속성 추가
2004.09.23	1.21	**GxSymbolSymbolStore:** TickEventFiltered, TickEventInterval 속성 추가
2004.08.12	1.2	**GxSymbolSymbolStore:** QuoteEventFiltered, QuoteEventInterval, PriceEventFiltered, PriceEventInterval 추가 **GxSymbol, GxQuote:** EventOmitCount 추가

2004.06.24	1.23	GxSymbol: Greeks 추가, OnQuoteChanged, OnChartDataAdded 버그 수정

GGOM 지원 파일

GOM 지원파일

Type Libraray (version 1.24)
IDL (version 1.24) GOM 인터페이스 정의 파일
P2_i.c (version 1.24) C++ 사용자들을 위한 source file (GOM 의 GUID 들이 담겨 있습니다.)

나. 계좌 및 주문

계좌 및 주문관리

GxTradeStore

개요

주문, 계좌 조회, 포지션 조회 등 거래 전반 데이터 Object들의 인터페이스와 주문 도구 Object들의 인터페이스를 제공합니다.

IGxTradeStore - GUID {7F7A69DF-C4DF-4A2C-9D7C-DFD9F750CB2E}

목록

Fields

ID: 1	IDispatch	Accounts 계좌(GxAccount) 목록을 관리하는 GxAccounts 타입의 하위 Collection Object를 반환합니다.
ID: 2	IDispatch	Positions 포지션(GxPosition) 목록을 관리하는 GxPositions 타입의 하위 Collection Object를 반환합니다.
ID: 3	IDispatch	Orders 주문(GxOrder) 목록을 관리하는 GxOrders 타입의 하위 Collection Object를 반환합니다.

ID: 4	IDispatch	**Fills** 체결(GxFill) 목록을 관리하는 GxFills 타입의 하위 Collection Object를 반환합니다.
ID: 5	IDispatch	**Confirms** Q정정 확인 및 취소 확인 목록(GxConfirm)을 관리 하는 GxConfirms 타입의 하위 Collection Object를 반환합니다.
ID: 6	IDispatch	**OrderHandler** 주문을 전송하는데 사용하는 GxOrderHandler 오브젝트입니다.
ID: 7	IDispatch	**EmHandler** *<유안타증권>* 서버 장애로 인하여 주문접수 및 체결 결과가 원할하게 전달이 되지 않을때 긴급 주문 및 강제로 서버로 조회를 수행하는 GxEmHandler 오브젝트입니다.
ID: 8	BSTR	**LastMessage** 최신 에러 메시지를 조회합니다.

Methods

ID: 9	n.a.	**Synchronize**:Gosu의 주문 / 체결 내역 수동 조회 기능을 요청합니다. **이 메소드를 사용하려면 아래의 상세설명을 필히 읽어 보셔야 합니다.**

메소드 상세설명

n.a. Synchronize

Gosu의 주문 / 체결 내역 수동 조회 기능을 요청합니다. 주문 접수 및 체결이 예상보다 늦어질 때는 이 기능을 사용하여 강제적으로 최신 정보를 갱신합니다. 이 메소드는 최소한 사용 빈도를 줄여야 하며 고수에서도 1초에 최대 한번 밖에 기능을 하지 않습니다. 그래서 만약 이 메소드를 5번 호출한다면 실제 기능은 한 번만 사용되며 고수에서 주문이 나간 후 5초 후에는 자동으로 고수에서 이 기능을 호출하므로 이 후에 강제 조회를 하실 필요가 없습니다. 하루종일 1초에 한 번씩 이 메소드를 호출하는 등의 행위는 피하시고**가급적으로 꼭 필요한 시점에서만 호출하십시요. 이 메소드의 무리한 호출은 주문에 안좋은 영향을 줄 수 있습니다.**

GxAccounts

개요

계좌 목록을 관리하고, 계좌 관련된 이벤트를 발생시킵니다.

IGxAccounts - GUID {D07D4C9A-987C-4C1F-B113-4A0783614778}
IGxAccountEvents - GUID {2BE6D91E-19FD-47D2-88F0-64BAFB27DFDF}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant를 구현하는 IUnknown 객체
ID: 3	long	Count 총 계좌 수
ID: 0	IDispatch	Item [Variant index] 개별 계좌를 순서 또는 코드로 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 내용

Methods

ID: 1	IDispatch	FindAccount(BSTR stAccountName) 계좌 이름으로 계좌 객체를 찾습니다.

Events

ID: 1	IGxAccountEvents	OnAccountUpdated(IDispatch aGxAccount) 계좌의 속성값(예탁금, 증거금, 손익, 수수료 등)이 변경될때 발생합니다.

필드 상세설명

IDispatch **Item** [Variant index]

개별 계좌(IGxAccount 객체)를 순서 또는 코드로 조회합니다.

Parameters

Index : 순서에 따른 계좌 조회를 할 경우 long 형을 사용하고, 계좌 코드('-' 포함)로 조회할 경우는 계좌코드를 사용합니다.

Return : 계좌 (IGxAccount) 객체

메소드 상세설명

IDispatch **FindAccount** (BSTR stAccountName)

계좌명으로 계좌를 찾습니다.

Parameters

Index **stAccountName** : 계좌명

Return

IGxAccount 객체

이벤트 상세설명

<IGxAccountEvents> **OnAccountUpdated** (IDispatch aGxAccount)

계좌의 속성값(예탁금, 증거금, 손익, 수수료 등)이 변경될때 발생합니다.

Parameters

aGxAccount 속성값이 변경된 계좌(IGxAccount 객체)

계좌목록

계좌정보

GxAccount

개요

계좌 정보를 담고 있는 객체입니다.

IGxAccount - GUID {38AC09C3-A997-4FC7-ABE1-EFB4CC04C121}

목록

Fields

ID: 1	BSTR	**Code**: '-' 이 제외된 계좌번호 입니다. GOM 에서 계좌번호는 모두 이 Code 를 사용합니다.
ID: 2	BSTR	**Desc** 계좌명
ID: 3	BSTR	**HiphenedCode** Code 사이에 '-' 이 들어간 형태로서 화면에 표시할때 인식을 편하게 하기 위해 제공합니다
ID: 4	VARIANT_BOOL	**Authenticated** 비밀번호 확인 여부입니다. 주문을 내기 위해서는 이 필드의 값이 True 가 되어야 합니다.
ID: 5	VARIANT_BOOL	**ApplyBiasMargin** 해당 계좌가 편향증거금 대상인가를 조회합니다.
ID: 6	double	**DepositTotal** 예탁금 총액
ID: 7	double	**DepositCash** 예탁금 현금
ID: 8	double	**MarginTotal** 증거금 총액
ID: 9	double	**MarginCash** 증거금 현금
ID: 10	double	**LiquidTotal** 주문 가능 총액
ID: 11	double	**LiquidCash** 주문 가능 현금
ID: 12	double	**Commission** 잠정 수수료

ID: 13	double	**DailyPL** 당일 총 손익을 조회합니다. 당일 손익 = 실현손익(AccPL) + 평가손익(EvalPL)
ID: 14	double	**AccPL** 당일 누적 실현(청산) 손익
ID: 15	double	**EvalPL** 미결제 약정의 평가 손익
ID: 16	IDispatch	**Orders** 해당 계좌의 주문 목록(IGxOrders)입니다. 단, 이 주문 목록은 신규 주문이나 주문 상태 변경시 이벤트를 발생시키지는 않습니다.
ID: 17	IDispatch	**Positions** 해당 계좌의 포지션 목록(IGxPositions)입니다. 단, 이 포지션 목록은 포지션 추가나 변경시 이벤트를 발생시키지는 않습니다.
ID: 18	BSTR	**LastMessage** 최신 에러 내용

Methods

ID: 19	void	ForceUpdate ( )예탁금, 증거금, 수수료 정보를 서버로부터 새로 수신을 합니다.
ID: 20	long	GetMaxOrderQty (BSTR stSymbolCode, IPositionType PositionType, double dPrice) 최대주문가능수량을 조회합니다.
ID: 21	double	GetShortOptMargin (BSTR stSymbolCode)KOSPI200 옵션의 매도증거금을 조회합니다. 단, 매도증거금이 0일 경우 유안타증권에서 업무 정의된 값으로 설정됨.

메소드 상세설명

void **ForceUpdate**( )

고수에서는 예탁금, 증거금, 수수료를 자동으로 최신정보로 갱신을 합니다. 하지만, 서버에서의 처리 지연 등으로 실제와 달라질 수 있으므로, 이때 서버에서 최신정보를 받아 오도록 할 때 이 메소드를 사용합니다.

long **GetMaxOrderQty**(BSTR stSymbolCode, IPositionType PositionType, double dPrice)

최대주문가능수량은 계좌의 주문가능금액에 따라 변경됩니다. 주문가능금액은 주문을 낼때마다 변경되는 금액으로

서버에 요청해서 가져오는 금액입니다. GOM에서 주문을 빠른 속도로 낼 경우 주문가능금액이 바로 업데이트되지 않을 수도 있기때문에 빠른 속도의 연속주문에서는 이 최대가능수량이 모두 적용되지 않을 수 있습니다.
이점 주의하시기 바랍니다. 만약 0보다 작은 수가 return된다면 dPrice가 0이거나 오류입니다.

double **GetShortOptMargin**(BSTR stSymbolCode)

KOSPI200 옵션종목의 순위험증거금중 가격변동증거금에 해당됩니다. 가격변동증거금이 0일 경우에는 유안타증권의 업무정의에 따라 10만원으로 적용됩니다. 가격변동증거금은 거래소의 제도변경에 따라 변경되거나 없어질 수 있는 항목이고 이 금액이 0일 경우에 금액은 유안타증권의 업무정의에 따라 변경될 수 있습니다. 현재 거래소의 2011년 5월 개정된 내용입니다.

포지션 목록

GxPositions

개요

포지션 목록을 관리하며, 포지션의 변동시 통보하는 기능을 합니다.

IGxPositions - GUID {9E3300C2-38C6-460E-9CF1-FDC35A5399C4}
IGxPositionEvents - GUID {2BE6D91E-19FD-47D2-88F0-64BAFB27DFDF}

목록

Fields

ID: -4	IUnknown	**_NewEnum** IEnumVariant 를 구현하는 객체
ID:3	long	**Count** 포지션 수
ID: 0	IDispatch	**Item**[Variant index] 개별 포지션을 순서에 따라 조회합니다.
ID: 4	BSTR	**LastMessage** 최신 에러 메시지

Methods

ID: 1	IDispatch	FindPosition ( )(BSTR stAccountCode, BSTR stSymbolCode) 계좌 코드와 종목 코드를 포지션 객체를 찾습니다.

Events

ID: 1	IGxPositionEvents	OnNewPosition ( )(BSTR stAccountCode, BSTR stSymbolCode) 계좌 코드와 종목 코드를 포지션 객체를 찾습니다.
ID: 2	IGxPositionEvents	OnPositionUpdated ( )(IDispatch aGxPosition) GxPosition의 속성이 변경될때 발생합니다.
ID: 3	IGxPositionEvents	OnEvalPLUpdated ( )(BSTR stAccountCode, BSTR stSymbolCode) 계좌 코드와 종목 코드를 포지션 객체를 찾습니다.

필드 상세설명

IDispatch **Item**[Variant index]

포지션 목록에서 순서에 따라 포지션 객체를 조회합니다.

Parameters

Index Index가 long 타입일 경우만 가능하며, 목록에서 Index 번째의 포지션 객체를 조회합니다

Return

포지션(IGxPosition) 객체

메소드 상세설명

IDispatch **FindPosition**(BSTR stAccountCode, BSTR stSymbolCode)

계좌 코드와 종목 코드로 해당 포지션 객체를 찾습니다.

Parameters

stAccountCode 계좌 Code

stSymbolCode 종목 Code

Return

찾는 포지션이 있을 경우 포지션(IGxPosition) 객체, 포지션이 없으면 NULL

이벤트 상세설명

<IGxPositionEvents> **OnNewPosition** (IDispatch aGxPosition)

새로운 포지션이 추가될 때 발생합니다.

Parameters

aGxPosition 새로 추가된 포지션(IGxPosition) 객체

Return

찾는 포지션이 있을 경우 포지션(IGxPosition) 객체, 포지션이 없으면 NULL

<IGxPositionEvents> **OnPositionUpdated** (IDispatch aGxPosition)

포지션의 수량이 변경되면 발생합니다.

Parameters

aGxPosition변경된 포지션(IGxPosition) 객체

<IGxPositionEvents> **OnEvalPLUpdated** (IDispatch aGxPosition)

포지션의 평가 손익이 변경되었을때(포지션 종목의 현재가 변경) 발생합니다.

Parameters

aGxPosition평가손익이 변경된 포지션(IGxPosition) 객체

포지션 정보

GxPosition

개요

개별 포지션에 대한 모든 정보를 담고 있는 객체입니다.

GxPosition – GUID {8965A261-20DD-44E2-9180-ED76FB221E73}

목록

Fields

ID: 1	IDispatch	**Account** 포지션의 계좌(IGxAccount)
ID: 2	IDispatch	**Symbol** 포지션의 종목(IGxSymbol)
ID: 3	long	**Qty** :최종 포지션 수량(매수: 양수, 매도: 음수)
ID: 4	long	**PrevQty** 전일 최종 포지션 수량
ID: 5	long	**TodayQty** 당일(변동) 포지션 수량
ID: 6	double	**AvgPrice** 평균 단가
ID: 7	double	**EvalPL** 평가 손익
ID: 8	long	**ShortOrderQty** 매도 미체결 주문 수량
ID: 9	long	**LongOrderQty** 전일 최종 포지션 수량
ID:10	BSTR	**LastMessage** 최신 에러 정보

주문목록

GxOrders

개요

주문 목록을 관리하며, 주문 관련 변경시 통보하는 역할을 합니다.

IGxOrders – GUID {AE3371A0-CF37-4EBF-ACB7-28DDF83C0D3A}
IGxOrderEvents – GUID {1538C50A-2BFD-46C0-BD14-78A592B71674}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 주문 목록 수
ID: 0	IDispatch	Item[Variant index] 순서에 따라 주문 객체를 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 메시지

Methods

ID: 1	IDispatch	FindOrder (BSTR stAccountCode, BSTR stSymbolCode, long iKseAcptNo) 계좌 코드, 종목코드, 주문접수번호로 주문을 찾습니다.

Events

ID: 1	IGxOrderEvents	OnNewOrder (IDispatch aGxOrder) 새로운 주문이 추가되면(증전접수) 발생합니다.
ID: 2	IGxOrderEvents	OnOrderUpdated (IDispatch aGxOrder) 체결, 정정, 취소 등으로 주문의 상태가 변경되면 발생합니다.

필드 상세설명

IDispatch **Item**[Variant index]

순서에 따라 주문 객체를 조회합니다.

Parameters

Index 목록에서 index 번째 주문 객체를 조회합니다

Return

포지션(IGxOrder) 객체

메소드 상세설명

IDispatch **FindOrder**(BSTR stAccountCode, BSTR stSymbolCode, long iKseAcptNo)

계좌 코드, 종목코드, 주문 접수번호로 주문을 찾습니다.

Parameters

stAccountCode 계좌 Code

stSymbolCode 종목 Code

iKseAcptNo 주문 접수번호

Return

해당되는 주문이 있으면 해당 주문(IGxOrder) 객체, 없으면 Null (IGxOrder) 객체, 포지션이 없으면 NULL

이벤트 상세설명

<IGxOrderEvents>**OnNewOrder** (IDispatch aGxOrder)

새로운 주문이 추가되면(증전접수) 발생합니다.

Parameters

aGxOrder 새로 추가된 주문(IGxOrder) 객체

<IGxOrderEvents>**OnOrderUpdated** (IDispatch aGxOrder)

체결, 정정, 취소 등으로 인하여 주문의 상태가 변경되면 발생합니다.

Parameters

aGxOrder변경된 주문(IGxOrder) 객체

주문정보

GxOrder

개요

개별 주문에 대한 정보 및 상태를 담고 있는 객체입니다. 주문접수 부터, 정정, 취소, 체결에 대한 정보를 모두 이 개체를 통해 확인할 수 있습니다. 주문이 증전에 접수되었을 때 생성됩니다.

IGxOrder - GUID {0B985283-DCAF-4768-AB87-B0EFF1FDE174}

목록

Fields

ID: 1	IDispatch	**Account** 주문 대상 계좌 (IGxAccount)
ID: 2	IDispatch	**Symbol** 주문대상 종목 (IGxSymbol)
ID: 3	IDispatch	**Position** 주문에 대한 포지션 객체 (IGxPosition)
ID: 4	IOrderType	**OrderType** 주문 종류(신규/정정/취소)
ID: 5	IPositionType	**PositionType** 매수/ 매도 구분
ID: 6	IPriceType	**PriceType** 주문 가격 유형(지정가/시장가/조건부지정가/최유리지정가)
ID: 7	long	**Qty** 주문 수량(정정 및 취소에 의해 감소됨)
ID: 8	long	**InitQty** 최초 주문수량(정정 및 취소에 의해서 변하지 않음)
ID: 9	double	**Price** 주문 가격
ID: 10	IOrderState	**OrderState** 주문 상태(접수,미체결/부분체결/전량체결/죽은주문/확인됨). 해당 주문이 정정 되면 기존 주문은 '죽은주문'이 되며, 새로운 주문 객체가 생성이 됩니다. 주문이 전량 취소가 되면 기존 주문은 '죽은주문'이 되며, 최소주문은 '확인됨' 상태가 됩니다.
ID: 11	long	**SrvAcptNo** 서버 접수 번호. 정정 및 취소시 이 주문번호가 사용됩니다.
ID: 12	long	**KseAcptNo** 거래소 접수번호. 종목별 고유접수번호입니다.
ID: 13	double	**KseAcptTime** 거래소 주문 접수시간
ID: 14	long	**FillQty** 체결 수량 (OrderState가 iosConfirm일 경우 확인수량을 의미)
ID: 15	double	**LastFillPrice** 가장 최근 체결가
ID: 16	long	**UnFillQty** 미체결 수량 (= Qty - FillQty)
ID: 17	IDispatch	**TargetOrder** 정정 및 취소의 대상 주문 (IGxOrder). 신규주문은 이 값이 Null 입니다.

ID: 18	long	TargetSrvAcptNo 정정 및 취소 대상 주문의 서버 접수번호. (신규주문은 해당 없음) TargetOrder 객체를 통해서 확인할 수도 있습니다.
ID: 19	long	TargetKseAcptNo 정정 및 취소 대상 주문의 거래소 접수번호. (신규주문은 해당 없음) TargetOrder 객체를 통해서 확인할 수도 있습니다.
ID: 20	IDispatch	Fills 본 주문에 대해 발생한 체결 목록(IGxFills) 본 체결 목록은 이벤트를 발생시키지 않습니다.
ID: 21	IDispatch	Confirms 본 주문이 정정 및 취소 주문일 경우 발생한 확인 목록(IGxConfirms) 본 확인 목록은 이벤트를 발생시키지 않습니다.
ID: 22	BSTR	LastMessage 최신 에러 정보

체결목록

GxFills

개요

체결 목록을 관리하고, 체결이 발생하거나 체결 내용이 변경되면 이를 알립니다.

IGxFills - GUID {7B2DA71A-0421-4CFC-8B96-79E237025276}
IGxFillEvents - GUID {9C7F9B73-6A1D-4DE9-994D-29AC73E60C31}{}

목록

Fields

ID: -4	IUnknown	_NewEnum:IEnumVariant 를 구현하는 객체
ID:3	long	Count 포지션 수
ID: 0	IDispatch	Item 목록에서 체결 객체를 순서에 따라 참조합니다.

ID: 4	BSTR	LastMessage 최신 에러 메시지

Events

ID: 1	IGxFillEvents	OnNewFill(IDispatch aGxFill) : 신규 체결시 통보합니다.
ID: 2	IGxFillEvents	OnFillUpdated(IDispatch aGxFill) 체결정보 수정시(체결 수량) 통보합니다.

필드 상세설명

IDispatch **Item**[Variant index]

목록에서 체결 객체(IGxFill)를 순서에 따라 참조합니다.

Parameters

Index index 번째 체결 객체(IGxFill)를 참조합니다.

Return

체결 객체(IGxFill)

이벤트 상세설명

<IGxFillEvents> **OnNewFill** (IDispatch aGxFill)

신규 체결 발생시 통보 됩니다.

Parameters

aGxFill 신규 체결 객체(IGxFill)

<IGxFillEvents> **OnFillUpdated** (IDispatch aGxFill)

체결 수량이 변경될 때 이를 통보합니다. 같은 주문에 대한 체결이 짧은 시간 안에 연달아 발생할 때는 같은 체결로 간주하고 체결 수량을 합산하게 됩니다.

Parameters

aGxFill 체결수량이 변경된 체결 객체(IGxFill)

체결정보

GxFill

개요

체결에 대한 정보를 담고 있는 객체입니다. 체결이 될 때마다 생성이 됩니다. 단, 동일 주문에 대한 체결이 10 초 이내에 두개 이상 들어올 때는 한 개의 체결로 처리합니다.

IGxFill – GUID {BEDADF9E-5879-4883-AE1C-2F3D9175838C}

목록

Fields

ID: 1	IDispatch	**Order** 체결 대상 주문 (IGxOrder)
ID: 2	long	**FillNo** 거래소 체결번호
ID: 3	long	**FillQty** 체결수량
ID: 4	double	**FillPrice** 체결가
ID: 5	double	**FillTime** 체결시간
ID: 6	double	**NearPrice** 근월물 체결가(스프레드 종목의 체결인 경우)
ID: 7	double	**FarPrice** 원월물 체결가(스프레드 종목의 체결인 경우)
ID: 8	BSTR	**LastMessage** 최신 에러 내용

확인목록

GxConfirms

개요

정정 확인 및 취소 확인을 관리하며, 새로운 확인 발생시 이를 통보합니다.

IGxConfirms - GUID {5045FA98-5584-478A-B088-8D5549934B21}

IGxConfirmEvents - GUID {F24BE35A-9DEA-4E99-804A-38EAD1D2FE71}

목록

Fields

ID: -4	IUnknown	**_NewEnum** IEnumVariant 를 구현하는 객체
ID: 3	long	**Count** 확인 객체 수
ID: 0	IDispatch	**Item** 목록에서 확인 객체를 순수에 따라 참조합니다.
ID: 4	BSTR	**LastMessage** 최신 에러 정보

Event

ID: 1	IGxConfirmEvents	OnNewConform(IDispatch aGxConfirm) 새로운 정정확인 및 취소확인 발생했을때 이를 통보합니다.

필드 상세설명

IDispatch **Item**[Variant index]

목록에서 확인 객체를 순서에 따라 참조합니다.

Parameters

Index index 번째 체결 객체(IGxConfirm)를 참조합니다.

Return

확인 객체(IGxConfirm)

이벤트 상세설명

<IGxConfirmEvents> **OnNewConfirm** (IDispatch aGxConfirm)

새로운 정정확인 및 취소확인이 발생하면 이를 통보합니다.

Parameters

aGxConfirm 신규 확인 객체(IGxConfirm)

확인정보

GxConfirm

개요

정정 및 취소 주문의 확인에 대한 정보를 담고 있는 객체입니다.

IGxConfirm – GUID {A06C536F-ED78-4D58-8C2B-94184BF3F0E2}

목록

Fields

ID: 1	IDispatch	**Order** 정정 및 취소 확인 대상 주문(IGxOrder).
ID: 2	double	**ArrivedTime** 주문 확인 (PC)도착 시간
ID: 3	long	**ReqQty** 주문 요청 수량
ID: 4	long	**ConfirmQty** 확인 수량. 확인수량이 0 이면 확인이 거부된 것입니다.
ID: 5	double	**ConfirmPrice** 확인 가격(정정 주문에 대한 확인만 해당, 정정주문의 실제 정정가임)
ID: 6	IPriceType	**NearPrice** 정정가격 유형(정정 주문에 대한 확인만 해당)
ID: 7	BSTR	**RjtReason** 확인 거부 사유(주문이 정정되거나 취소되기전 체결 등으로 사라질 경우 등) 확인 거부 시만 값을 갖습니다.

ID: 8	BSTR	LastMessage 최신 에러 내용

주문송수신

GxOrderHandler

개요

주문을 내고, 주문의 거래소 접수까지 상태를 알려 주는 객체입니다. 주문을 내는 순서는 다음과 같습니다.

주문 요청 목록을 만듭니다. 이때 개별 주문 요청 객체를 보관하고 있어야만, 해당 주문 요청의 처리 상황을 알 수가 있습니다.

Send()를 호출하여 주문을 서버로 전송합니다.

이벤트를 통해서 개별 주문 요청들의 결과를 확인합니다.

IGxOrderHandler – GUID {018A70D4-C01E-4D90-B446-1198800588A0}
IGXOrderHandlerEvents – GUID 016402D5-714B-47CF-B292-03B22C659736}

목록

Fields

ID: 1	IDispatch	Order: 주문 요청 목록 (IGxOrderReqs)
ID: 2	long	WaitCount 서버로 보내기 전 상태에 있는 주문요청 건수 입니다.
ID: 3	BSTR	LastMessage 최종 에러 메시지

Methods

ID: 4	IDispatch	PutNewOrder (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, double dPrice) 신규 주문을 주문요청 목록에 추가합니다

ID: 5	IDispatch	PutChangeOrder IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice) 정정 주문을 주문요청 목록에 추가합니다.
ID: 6	IDispatch	PutCancelOrder (IDispatch aTargetOrder, long iCancelQty) 취소 주문을 주문요청 목록에 추가합니다.
ID: 7	void	Send () 주문 요청 목록에 새로 추가된 주문들을 실제 서버로 전송합니다
ID: 8	IDispatch	PutNewOrder2 (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, double dPrice, FillType: IFillType) 신규 주문을 주문요청 목록에 추가합니다. (IOC, FOK 주문가능)
ID: 9	IDispatch	PutChangeOrder2 (IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice, FillType: IFillType) 정정 주문을 주문요청 목록에 추가합니다. (IOC, FOK 주문가능)

Events

ID: 4	IGxOrderHandlerEvents	OnStateChanged: ((IDispatch aGxOrderReq) send() 를 통해 나간 주문들의 상태가 변할 시 발생한다. 입력 aGxOrderReq 는 상태가 변한 대상 GxOrderReq이다.

메소드 상세설명

IDispatch **PutNewOrder** (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, double dPrice)

신규 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

stAccountCode 계좌코드

stSymbolCode 종목코드

PostionType 매수/매도 구분

PriceType 가격 유형

iQty 주문수량

dPrice 주문가격

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutChangeOrder** (IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

단, 같은계좌, 같은 종목에 대해 정정주문 시도건수가 **300건** 초과일 경우 정정주문이 제한 됩니다. 정정주문이 제한되더라도 신규, 취소 주문은 가능합니다.

정정이 필요한 경우에는 고수로 주문을 내시거나 고수를 재로그인 하시면 정정횟수가 초기화됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iChangeQty 정정수량

PriceType 가격 유형

dPrice 정정 가격

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutCancelOrder** (IDispatch aTargetOrder, long iCancelQty)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iCancelQty 취소 수량

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

void Send()

주문 요청에 목록에 있는 주문들을 실제로 서버로 전달을 합니다.

IDispatch PutNewOrder2 (BSTR stAccountCode, BSTR stSymbolCode,IPositionType PositionType,IPriceType PriceType, long iQty, double dPrice, FillType: IFillType)

신규 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

stAccountCode 계좌코드

stSymbolCode 종목코드

PostionType 매수/매도 구분

PriceType 가격 유형

iQty 주문수량

dPrice 주문가격

FillType 체결유형(일반/IOC/FOK)

Return

주문요청 객체(Return : 주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutChangeOrder2** (IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice, FillType: IFillType)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iChangeQty 정정수량

PriceType 가격 유형

dPrice 정정 가격

FillType 체결유형(일반/IOC/FOK)

dPrice 주문가격

FillType 체결유형(일반/IOC/FOK)

'**일반**'으로 주문시에는 체결유형이 **DIFM**으로 변경됩니다.

Return

주문요청 객체(Return : 주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

이벤트 상세설명

<IGxOrderHandlerEvents> **OnStateChanged** (IDispatch aGxOrderReq)

주문의 상태 변화(증전접수까지)를 통보합니다.

Parameters

aGxOrderReq 상태가 바뀐 주문 요청 객체(IGxOrderReq)

주문요청목록

GxOrderReqs

개요

주문 요청 목록을 관리하는 개체입니다

IGxOrderReqs - GUID {2AE9A6E6-AE76-4008-8DFF-652F976B6D6B}

목록

ID: -4	IUnknown	**_NewEnum** IEnumVariant 를 구현하는 객체
ID:3	long	**Count** 총 주문 요청 건수. 이미 접수가 끝난 것과 아직 서버에 보내진 않은 것들이 함께 포함된 것입니다.
ID: 0	IDispatch	**Item** [Variant index] 주문 요청 내용을 목록에서 순서에 따라 참조합니다.
ID: 4	BSTR	**LastMessage**최신 에러 정보

Fields

필드 상세설명

IDispatch **Item** Variant index

목록에서 개별 주문 요청을 순서에 따라 참조를 합니다.

Parameters

index index 번째의 주문요청 객체를 참조합니다.

Return

주문요청 객체(IGxOrderReq)

주문요청

GxOrderReq

개요

건별 주문 요청 내용을 담고 있는 객체입니다.

IGxOrderReq - GUID {546BF980-F456-4288-A50A-B93440EB4D7F}

목록

Fields

ID: 1	IDispatch	**Account** 주문 대상 계좌(IGxAccount)
ID: 2	IDispatch	**Symbol** 주문 대상 종목(IGxSymbol)
ID: 3	IPositionType	**PositionType** 매수/매도 구분
ID: 4	long	**Qty** 주문 수량
ID: 5	IPriceType	**PriceType** 주문 가격 유형
ID: 6	double	**Price** 주문 가격
ID: 7	IDispatch	**TargetOrder** 정정 및 취소 주문 요청의 경우 정정 및 취소 대상 주문 객체 (IGxOrder)
ID: 8	IOrderType	**OrderType** 주문 유형
ID: 9	IOrderReqState	**State** 주문 요청 상태 신규 : Send() 호출 전 전송대기 : Send() 호출 후 상태 전송거부 : 비밀번호 미확인등의 사유로 서버로의 전송이 거부됨 서버접수대기 : 서버에 주문 전송후 응답 대기 서버거부 : 서버에서 주문 접수가 거부됨 증전접수대기 : 서버접수가 된 후 거래소에서의 접수를 기다림 증전거부 : 거래소에서 주문이 거부된 상태
ID: 10	long	**SrvAcptNo** 서버 접수 번호. 서버 접수시 값을 갖습니다.
ID: 11	BSTR	**RjtReason** 주문 거부 사유(전송거부,서버거부,증전거부 시)
ID: 12	VARIANT_BOOL	**IsDone** 처리가 종료여부(전송거부,서버거부,증전거부,증전접수)
ID: 13	VARIANT_BOOL	**IsDoing** 처리중 여부(신규,전송대기,서버접수대기,증전접수대기)

ID: 14	BSTR	LastMessage 최신 에러 정보

비상주문송수신

GxEmHandler

개요

서버 체결 통보 지연 등 서버 장애 발생시 고수의 엔진을 거치지 않고 서버에 직접 주문을 전송하는데 사용합니다.

유안타증권 고수에만 제공됩니다.

IGxEmHandler – GUID {50B9FF14-5792-4050-991B-519448949AB0}
IGXEmHandlerEvents – GUID {9B904CA0-84A1-449D-ABBA-CAFB8E1B5BD1}

목록

Fields

ID: 1	BSTR	LastMessage 최신 에러 메시지

Methods

ID: 2	VARIANT_BOOL	SendNewOrder (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, , double dPrice) 신규 주문을 냅니다.
ID: 3	VARIANT_BOOL	SendChangeOrder (BSTR stAccountCode, BSTR stSymbolCode, long iSrvAcptNo, long iQty, IPriceType PriceType, double dPrice) 정정 주문을 냅니다.
ID: 4	VARIANT_BOOL	SendCancelOrder (BSTR stAccountCode, BSTR stSymbolCode, long iSrvAcptNo, long iQty) 취소 주문을 냅니다.

ID: 5	VARIANT_BOOL	SendNewOrder2 (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, , double dPrice, FillType: IFillType) 신규 주문을 냅니다.
ID: 6	VARIANT_BOOL	SendChangeOrder2 (BSTR stAccountCode, BSTR stSymbolCode, long iSrvAcptNo, long iQty, IPriceType PriceType, double dPrice, FillType: IFillType) 정정 주문을 냅니다.

Events

ID: 1	IGxOrderHandlerEvents	OnEmOrderArrived (IEmOrderType OrderType, VARIANT_BOOL bSuccess, long iSrvAcptNo, BSTR stMsg) 서버 주문 접수 결과를 통보합니다. GxEmHandler 에서는 주문 전송 및 접수가 고수 엔진을 거치지 않으므로 주문 접수결과 통보시 계좌, 종목등 상세 주문 정보가 제공이 되지 않습니다. 사용하실 때 유의하시기 바랍니다.

메소드 상세설명

VARIANT_BOOL **SendNewOrder** (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType,

IPriceType PriceType, long iQty, double dPrice)

신규 주문을 냅니다.

Parameters

stAccountCode 계좌코드

stSymbolCode 종목코드

PostionType 매수/매도 구분

PriceType 가격 유형

iQty 주문수량

dPrice 주문가격

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutChangeOrder** (IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iChangeQty 정정수량

PriceType 가격 유형

dPrice 정정 가격

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutCancelOrder** (IDispatch aTargetOrder, long iCancelQty)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iCancelQty 취소 수량

Return

주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

void **Send**()

주문 요청에 목록에 있는 주문들을 실제로 서버로 전달을 합니다.

IDispatch **PutNewOrder2** (BSTR stAccountCode, BSTR stSymbolCode, IPositionType PositionType, IPriceType PriceType, long iQty, double dPrice, FillType: IFillType)

신규 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

stAccountCode 계좌코드

stSymbolCode 종목코드

PostionType 매수/매도 구분

PriceType 가격 유형

iQty 주문수량

dPrice 주문가격

FillType 체결유형(일반/IOC/FOK)

Return

주문요청 객체(Return : 주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

IDispatch **PutChangeOrder2** (IDispatch aTargetOrder, long iChangeQty, IPriceType PriceType, double dPrice, FillType: IFillType)

정정 주문 요청 내용을 목록에 추가합니다. 이때 주문은 서버에 전송되는 것이 아닙니다. 주문 요청 목록을 만드신 후 Send() 를 호출할 때, 실제 서버로 주문이 전달됩니다.

Parameters

aTargetOrder 정정 대상 주문(IGxOrder)

iChangeQty 정정수량

PriceType 가격 유형

dPrice 정정 가격

FillType 체결유형(일반/IOC/FOK)

dPrice 주문가격

FillType 체결유형(일반/IOC/FOK)

Return

주문요청 객체(Return : 주문요청 객체(IGxOrderReq). 차후 주문의 접수 여부를 확인할 때 사용합니다. 주문 요청에 문제가 있을 경우에는 Null 값이 됩니다. 주문 요청의 추가가 실패하면 LastMessage 에서 원인을 확인하시기 바랍니다.

이벤트 상세설명

<IGxOrderHandlerEvents> OnStateChanged (IDispatch aGxOrderReq)

주문의 상태 변화(증전접수까지)를 통보합니다.

Parameters

aGxOrderReq 상태가 바뀐 주문 요청 객체(IGxOrderReq)

다. 시세

종목관리

GxSymbolStore

개요

종목을 종합적으로 관리합니다. 종목을 참조할 수 있는 다양한 방법을 제공하므로 편리하게 종목 정보를 조회하실 수 있습니다.

IGxSymbolStore - GUID {31E86383-A860-48B2-869E-90F3142B4D78}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 전체 종목 수(분류되지 않은 전체 종목 목록입니다)
ID: 0	IDispatch	Item[Variant index] 종목을 전체 목록에서 순서 및 코드에 따라 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 정보
ID: 5	IDispatch	Stocks 시가 총액 상위 5개 주식 목록만 따로 분류해 놓은 것입니다. (IGxSymbols)
ID: 6	IDispatch	Spreads Kospi200 지수 선물 스프레드 종목만 따로 분류해 놓은 것입니다. (IGxSymbols)
ID: 7	IDispatch	Futures Kospi200 지수 선물만 따로 분류해 놓은 것입니다. (IGxSymbols)
ID: 8	IDispatch	OptionMonths Kospi200 지수 옵션을 월물별로 분류해 놓은 것입니다. (IGxOptionMonths)
ID: 9	IDispatch	K200Index KOSPI200 지수 객체 (IGxSymbol)
ID: 10	IDispatch	NearestFuture 선물 최근월물 (IGxSymbol)
ID: 11	IDispatch	NearestOptMonth 옵션 최근월 참조 객체(IGxOptionMonth)
ID: 13	VARIANT_BOOL	QuoteEventFiltered 일정 시간 간격으로 호가 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.
ID: 14	long	QuoteEventInterval QuoteEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.
ID: 15	VARIANT_BOOL	PriceEventFiltered 일정 시간 간격으로 현재가 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.

ID: 16	long	**PriceEventInterval** PriceEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.
ID: 17	VARIANT_BOOL	**TickEventFiltered** 체결 거르기(이전 전달된 체결과 동일 호가 이면서 같은 분봉 이내 이면 거래량을 누적하고 체결을 구독 객체로 보내지 않음)을 사용할 것 인지를 결정하거나 설정되어 있는 값을 읽습니다.
ID: 18	long	**TickEventInterval** TickEventFiltered를 사용할 경우 필요한 '최대 대기 시간' 값을 설정하거나 그 값을 읽습니다.
ID: 19	IDispatch	**AllFutures** KOSPI200지수선물, 국내주식선물, 국내선물 객체 (IGxSymbol)
ID: 20	IDispatch	**AllSpreads** KOSPI200지수선물, 국내주식선물, 국내선물 스프레드 객체 (IGxSymbol)
ID: 21	IDispatch	**BondFutures** 국내선물중 국채선물 객체 (IGxSymbol)
ID: 22	IDispatch	**CurrencyFutures** 국내선물중 통화선물 객체 (IGxSymbol)
ID: 23	IDispatch	**ProductFutures** 국내선물중 상품선물 객체 (IGxSymbol)
ID: 24	VARIANT_BOOL	**EstFillIncluded** EstFillIncluded 사용할 경우 예상체결가 이벤트(GxLastTick)를 발생시키거나 현재 설정 되어 있는 값을 읽습니다.
ID: 25	IDispatch	**MiniSpreads** Kospi200 Mini 지수 선물 스프레드 종목만 따로 분류해 놓은 것입니다. (IGxSymbols)
ID: 26	IDispatch	**MiniFutures** Kospi200 Mini 지수 선물만 따로 분류해 놓은 것입니다. (IGxSymbols)
ID: 27	IDispatch	**MiniOptionMonths** Kospi200 Mini 지수 옵션을 월물별로 분류해 놓은 것입니다. (IGxOptionMonths)
ID: 28	IDispatch	**K200MiniIndex** KOSPI200 Mini 지수 객체 (IGxSymbol)
ID: 29	IDispatch	**NearestMiniFuture** Kospi200 Mini 선물 최근월물 (IGxSymbol)
ID: 30	IDispatch	**NearestMiniOptMonth** Kospi200 Mini 옵션 최근월 참조 객체(IGxOptionMonth)
ID: 31	IDispatch	**IndexFutures** 지수선물 객체(K200,K200미니제외)(IGxSymbols)

ID: 32	IDispatch	OptionWeeks K200 위클리옵션을 월물별로 분류해 놓은 것입니다.(IGxOptionMonths)

Methods

ID: 12	VARIANT_BOOL	SetGreeksParam (IGreeksCalcType GreeksCalcType, double dUnderMixRatio, double dCallVol, double dPutVol) GxSymbol의 Greeks 계산에 필요한 값 (잔존일수 계산방식, Call, Put 입력 변동성값, 기초자산 선물비율 등을 설정합니다.

메소드 상세설명

IDispatch **Item** [Variant index]

종목을 전체 목록에서 순서 및 코드에 따라 조회합니다.

Parameters

index index 가 숫자인 경우 index 번째의 종목을 참조하고, index 가 문자인 경우 종목 코드가 index인 종목을 참조합니다.

Return

종목 객체(IGxSymbol)

VARIANT_BOOL **QuoteEventFiltered**

일정 시간 간격으로 호가 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시 :

True 일정 시간 간격으로 호가 이벤트를 발생 시킵니다.
False 호가 변동 시 마다 호가 이벤트를 발생 시킵니다.

Return

VARIANT_BOOL

True 현재 일정 시간 간격으로 호가 이벤트를 발생시키고 있습니다.

False 현재 호가 변동 시 마다 호가 이벤트를 발생 시킵니다.

long QuoteEventInterval

QuoteEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시 :

시간 간격을 설정 합니다. 단위는 mili second 입니다. 기본값은 1000(1초), 상한값은 5000(5초), 하한값은 50(0.05초) 입니다.

Return

현재 설정된 시간 간격을 반환합니다.

VARIANT_BOOL PriceEventFiltered

일정 시간 간격으로 현재가 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시 :

True 일정 시간 간격으로 호가 이벤트를 발생 시킵니다.
False 호가 변동 시 마다 호가 이벤트를 발생 시킵니다.

Return

VARIANT_BOOL

True 현재 일정 시간 간격으로 호가 이벤트를 발생시키고 있습니다.
False 현재 호가 변동 시 마다 호가 이벤트를 발생 시킵니다.

long PriceEventInterval

PriceEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시 :

시간 간격을 설정 합니다. 단위는 mili second 입니다. 기본값은 1000(1초), 상한값은 5000(5초), 하한값은 50(0.05초) 입니다.

Return

현재 설정된 시간 간격을 반환합니다.

VARIANT_BOOL **TickEventFiltered**

체결 거르기(이전 전달된 체결과 동일 호가 이면서 같은 분봉 이내 이면 거래량을 누적하고 체결을 구독 객체로 보내지 않음)을 사용할 것 인지를 결정하거나 설정되어 있는 값을 읽습니다. 상세한 기준은 다음과 같습니다.

다음과 같은 경우 해당 체결을 바로 구독 객체에 이벤트를 전달합니다.

봉(1분)의 경계선을 넘은 경우(시가) 바로 구독 객체에 이벤트를 전달합니다.

같은 봉내에서 체결가가 변경되면 바로 구독 객체에 이벤트를 전달합니다.

구독 객체에 이벤트를 전달 후 다음 첫 체결 시점부터 '최대 대기 시간(TickEventInterval)'이 지나면 '최대 대기 시간' 동안 보내지 않은(누적된) 체결을 구독 객체에 이벤트를 전달합니다. **단 대기시간 동안 위의 두 가지 경우 중 하나라도 만족한다면 대기 시간은 무시되고 그 동안 누적된 체결을 한건으로 처리하여 구독 객체에 이벤트를 전달합니다.**

<기타규칙>

매수/매도 기준 : 누적 체결 중 최종 체결의 '매도/매수 기준'을 사용합니다.

시간 : 누적 체결 중 최종 체결의 시간을 사용합니다.

체결 거르기를 선택하지 않을 경우 모든 체결 발생 시 마다 구독 객체에 이벤트를 발생합니다.

속성 : Read / Write

설정 시

True 체결 거르기를 선택합니다
False 체결 거르기를 해제 합니다.

Return

VARIANT_BOOL

True 현재 체결 거르기를 진행 중입니다.
False 현재 체결 거르기 해제 상태입니다.

long **TickEventInterval**

TickEventFiltered를 사용할 경우 필요한 '최대 대기 시간' 값을 설정하거나 그 값을 읽습니다.

속성 : Read / Write

설정 시 :

시간 간격을 설정 합니다. 단위는 mili second 입니다.
기본값은 1000(1초), 상한값은 10000(10초), 하한값은 500(0.5초) 입니다.

Return

현재 설정된 시간 간격을 반환합니다.

VARIANT_BOOL **EstFillIncluded**

EstFillIncluded 사용할 경우 예상체결가 이벤트(GxLastTick)를 발생시키거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시 :

True GxLastTick을 통하여 예상체결가 이벤트를 발생킵니다.
False GxLastTick을 통하여 예상체결가 이벤트 발생을 취소합니다.

기본값은 False로 설정되어 있습니다.

Return

VARIANT_BOOL

True 현재 예상체결가 이벤트를 발생시키고 있습니다.
False 현재 예상체결가 이벤트를 발생시키고 있지 않습니다.

메소드 상세설명

VARIANT_BOOL **SetGreeksParam** (IGreeksCalcType GreeksCalcType, double dUnderMixRatio, double dCallVol, double dPutVol)

GxSymbol의 Greeks 계산에 필요한 값(잔존일수 계산방식, Call, Put 입력 변동성값, 기초자산 선물비율 등을 설정합니다. 이 설정을 하지 않는 경우 Greeks 값들은 기본값인 전일 대표 내재 변동성, 달력날짜, 선물비율=0 인 값으로 계산됩니다.

Parameters

GreeksCalcType
잔존일수 계산방식
igcNA 사용시 해당 설정 무시

dUnderMixRatio
기초자산 선물 비율 (0 ~ 1 사이의 실수, 0.0이면 Kospi200만 반영 1.0이면 선물만 반영)
"-1" 입력시 선물 자동 비율 사용
"-100" 입력시 해당 설정 무시

dCallVol
그릭스 계산에 필요한 Call 입력 변동성 값 (1 ~ 300)
"-100" 입력시 해당 설정 무시
"-1" 입력시 종목 내재 변동성 사용
"-2" 입력시 현재 ATM 변동성 사용
"-3" 입력시 현재 대표 변동성 사용
"-4" 입력시 전일 대표 변동성 사용
"-15" 입력시 역사적 변동성 15일 사용
"-30" 입력시 역사적 변동성 30일 사용
"-40" 입력시 역사적 변동성 40일 사용
"-60" 입력시 역사적 변동성 60일 사용
"-90" 입력시 역사적 변동성 90일 사용

dPutVol
그릭스 계산에 필요한 Put 입력 변동성 값 (1 ~ 300)
특정한 변동성 사용은 dCallVol 설정과 동일합니다.

종목목록

GxSymbols

개요

종목 목록을 관리합니다.

IGxSymbols - GUID {615DCD08-B2FF-4A96-ADDA-8DECE0C56841}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 종목 수
ID: 0	IDispatch	Item [Variant index] 종목을 목록에서 순서 및 코드에 따라 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 메시지

필드 상세설명

IDispatch **Item** [Variant index]

종목을 전체 목록에서 순서 및 코드에 따라 조회합니다.

Parameters

index index 가 숫자인 경우 index 번째의 종목을 참조하고, index 가 문자인 경우 종목 코드가 index인 종목을 참조합니다.

Return

종목 객체(IGxSymbol)

종목정보

GxSymbol

개요

종목 목록을 관리합니다.

IGxSymbol - GUID {A0CEADD8-BA76-43A1-8F7F-930652737CA9}
IGXSymbolEvents - GUID {2343D08D-ADD6-4C10-9F27-641D493331E0}

목록

Fields

ID: 1	BSTR	**Code** 종목 코드
ID: 2	BSTR	**Desc** 종목명(축약)
ID: 3	BSTR	**KorDesc** 한글 종목명
ID: 4	BSTR	**EngDesc** 영문 종목명
ID: 5	IUnderlyingType	**UnderlyingType** 기초자산 종류
ID: 6	ISymbolType	**SymbolType** 상품 종류
ID: 7	IOptionType	**OptionType** 옵션 종류(옵션만 해당)
ID: 8	double	**StrikePrice** 행사가(옵션만 해당)
ID: 9	BSTR	**ClosingMonth** 만기월일(형식: 'YYYYMMDD')
ID: 10	double	**MatureDate** 만기일
ID: 11	long	**TrailingDays** 잔존일수
ID: 12	long	**PointValue** 1.0 포인트당 가격(원)
ID: 13	double	**CDRate** CD 금리

ID: 14	double	HighLimit 상한가
ID: 15	double	LowLimit 하한가
ID: 16	double	CBHigh CB 상한가
ID: 17	double	CBLow CB 하한가
ID: 18	double	BasePrice 기준가
ID: 19	SAFEARRAY	LongThValues ** 지원하지 않습니다. 구간이론가는 ID:50번 ThValues 참고하세요
ID: 20	SAFEARRAY	ShortThValues ** 지원하지 않습니다. 구간이론가는 ID:50번 ThValues 참고하세요
ID: 21	VARIANT	VLongThValues ** 지원하지 않습니다. 구간이론가는 ID:51번 VThValues 참고하세요
ID: 22	VARIANT	VShortThValues ** 지원하지 않습니다. 구간이론가는 ID:51번 VThValues 참고하세요
ID: 23	long	Precision 가격 표시 단위(소수점 자리수)
ID: 24	VARIANT_BOOL	IsNearest 근월물 여부
ID: 25	VARIANT_BOOL	IsATM ATM 여부(전영업일 기준)
ID: 26	double	IV 종목 내재변동성(옵션만 해당) - 이 값은 증권거래소에서 장시작전 일회 제공하는 값입니다.
ID: 27	long	PrevOpenInterest 전일 미결제 약정 수량
ID: 28	double	Open 시가
ID: 29	double	High 고가
ID: 30	double	Low 저가
ID: 31	double	Close 현재가

ID: 32	double	**Change** 현재가 전일비
ID: 33	long	**OpenInterest** 종목 미결제약정
ID: 34	long	**AccVolume** 누적 거래량
ID: 35	CURRENCY	**AccAmount** 누적 거래 대금
ID: 36	IDispatch	**Quote** 5단계 호가 (IGxQuote)
ID: 37	IDispatch	**LastTick** 최종 체결 정보 (IGxLastTick)
ID: 38	VARIANT_BOOL	**CanOrder** 주문가능여부(고수의 '주문설정'에서 정함)
ID: 39	BSTR	**LastMessage** 최신 에러 메시지
ID: 40	double	**ThPrice** 이론가(선물, 옵션만 해당)
ID: 41	double	**IV2** 종목 내재변동성(옵션만 해당) 입니다. IV2는 IV와는 달리 고수에서 제공하는 내재 변동성 계산식에 의한 값입니다.
ID: 42	double	**Delta** Delta(옵션만 해당)
ID: 43	double	**Gamma** Gamma(옵션만 해당)
ID: 44	double	**Theta** Theta(옵션만 해당)
ID: 45	double	**Vega** Vega(옵션만 해당)
ID: 46	double	**Lamda** Lamda(옵션만 해당)
ID: 47	double	**IntrinsicValue** 옵션 내재 가치 (옵션만 해당)
ID: 48	double	**TimeValue** 시간 가치(옵션만 해당)
ID: 49	long	**EventOmitCount** 최근 PriceEventFiltered를 사용함으로써 생략된 이벤트 수.

		PriceEventFiltered를 사용한 효율을 알 수 있습니다.
ID: 50	SAFEARRAY	**ThValues** 구간 이론가 목록
ID: 51	VARIANT	**VThValues** 구간 이론가(자료형식: Variant array)

Event

ID: 1	IGxSymbolEvents	**OnPriceChanged** (IDispatch aGxSymbol) 종목이 현재가가 변할 때 발생합니다.

이벤트 상세설명

OnPriceChanged (IDispatch aGxSymbol)

종목의 현재가가 변할 때 발생 합니다.

Parameters

aGxSymbol 변경 대상 종목의 인터페이스

호가정보

GxQuote

개요

GxSymbol의 하위 Object로써 GxSymbol의 종목 정보 중 호가 정보를 이 Object에 위임하여 관리합니다.

GxQuote Object는 종목의 5단계 호가 정보를 클라이언트에 제공하며 클라이언트에 호가 이벤트를 발생합니다.

IGxQuote – GUID {988F61D7-31D7-439C-B766-E0D674E2851B}
IGxQuoteEvents – GUID {7E881DFD-8A0C-4DFB-81F0-F58B361A8259}

목록

Fields

ID: 1	double	**UpdateTime** 호가 최종 갱신 시간	
ID: 2	SAFEARRAY	**Prices** 5단계 호가, 단 원래 수치에 100이 곱해진 값이므로 실제 사용 할 때는 100으로 나누어야 함 0 ~ 4 매도 호가 5 ~ 9 매수 호가	
ID: 3	SAFEARRAY	**Qtys** 5단계 호가 잔량 0 : 매도 총 잔량 1 ~ 5 : 매도 잔량 6 : 매수 총잔량 7 ~ 11 : 매수잔량	VBA
ID: 4	SAFEARRAY	**Cnts** 5단계 호가 건수 0 : 매도 총 건수 1 ~ 5 : 매도 건수 6 : 매수 총건수 7 ~ 11 : 매수건수	VBA
ID: 5	SAFEARRAY	**Quotes** 5단계 호가/잔량/건수를 한 번에 조회 0 ~ 4 : 매도호가 5 ~ 9 : 매수호가 10 : 매도총잔량 11 ~ 15 매도잔량 16 : 매수총잔량 17 ~ 21 : 매수잔량 22 : 매도총건수 23 ~ 27 : 매도건수 28 : 매수 총건수 29 ~ 33 : 매수건수	
ID: 6	Variant	**VPrices** 5단계 호가(Variant Array로 조회) 인덱스는 safearray와 동일, 원래 수치에 100이 곱해짐	VBA
ID: 7	Variant	**VQtys** 5단계 호가 잔량(Variant Array로 조회) 인덱스는 safearray와 동일	
ID: 8	Variant	**VCnts** 5단계 호가 건수(Variant Array로 조회) 인덱스는 safearray와 동일	
ID: 9	Variant	**VQuotes** 5단계 호가/잔량/건수를 한 번에 조회(Variant Array로 조회) 인덱스는 safearray와 동일	VC++
ID: 13	BSTR	**LastMessage** 최신 에러 정보	
ID: 14	long	**EventOmitCount** 최근 QuoteEventFiltered를 사용함으로써 생략된 이벤트 수. QuoteEventFiltered를 사용한 효율을 알 수 있습니다.	

Methods

ID: 10	double	Price(IPositionType PositionType, long iIndex) 5단 호가중 개별 호가를 조회합니다.
ID: 11	long	Qty(IPositionType PositionType, long iIndex) 5단 호가중 개별 호가 잔량을 조회합니다.
ID: 12	long	Cnt(IPositionType PositionType, long iIndex) 5단 호가중 개별 호가 건수를 조회합니다.

Events

ID: 1	IGxQuoteEvents	OnQuoteChanged(IDispatch aGxSymbol, IDispatch aGxQuote) 종목이 호가가 변할 때 발생합니다.

이벤트 상세설명

doublePrice (IPositionType PositionType, long iIndex)

5단 호가중 개별 호가를 조회합니다.

Parameters

PositionType 매수/매도 구분
iIndex 호가 단계 (1..5)

Return

해당 호가

longQty (IPositionType PositionType, long iIndex)

5단 호가중 개별 호가를 조회합니다.

Parameters

PositionType 매수/매도 구분
iIndex 호가 단계 (0:총잔량, 1..5)

Return

해당 호가

longCnt (IPositionType PositionType, long iIndex)

5단 호가중 개별 호가 건수를 조회합니다.

Parameters

PositionType 매수/매도 구분
iIndex 호가 단계

Return

해당 호가

이벤트 상세설명

< IGxQuoteEvents >OnQuoteChanged (IDispatch aGxSymbol, IDispatch aGxQuote)

종목의 호가가 변할 때 발생 합니다.

Parameters

aGxSymbol 대상 종목 (IGxSymbol)aGxQuote
aGxQuote

최종체결정보

GxLastTick

개요

종목의 최종 체결에 대한 정보를 담고 있습니다.
이 Object는 System Trading을 위해 사용 시 체결이 순간적으로 발생하여 발생하는 부하로 인한 시스템이 불안정을 해결하기 위해 OnNewTick 이벤트를 두 가지 방법으로 발생 시키고 있습니다.

첫번째는 단순하게 체결 발생 시점 마다 OnNewTick을 발생시키는 방법입니다. 이 방법은 순간적으로 다수의 체결이 발생할 때 사용하는 용도(ex.Trade Station)에 따라 많은 부하가 발생할 수 있습니다.

두번째는 발생한 체결이 이전 전달된 체결과 동일 호가 이면서 같은 분봉 이내(0-59초)이면 거래량을 누적하고 구독 객체로 OnNewTick을 발생 시키지 않는 방법 입니다.

자세한 설명은'GxSymbolStore'의'TickEventFiltered'를 참고 하십시요.

IGxLastTick - GUID {532A38A2-2923-444D-9C8D-E7A7182100C4}
IGxLastTickEvents - GUID {312BEF52-4E4E-449A-BA40-7E4DAAAD8DA1}

목록

Fields

ID: 1	double	**Time** 최종 체결 시각
ID: 2	IPositionType	**LSType** 매수체결/매도체결 구분
ID: 3	double	**Price** 체결가
ID: 4	double	**Volume** 체결량
ID: 5	double	**AccVolume** 누적체결량
ID: 6	long	**OpenInterest** 종목 미결제약정
ID: 7	BSTR	**LastMessage** 최신 에러 메시지

Events

ID: 1	IGxLastTickEvents	OnNewTick(IDispatch aGxSymbol, IDispatch aGxLastTick) 종목의 체결이 들어올 때 발생합니다.

이벤트 상세설명

< IGxLastTickEvents > **OnNewTick** (IDispatch aGxSymbol, IDispatch aGxLastTick)

종목 체결이 들어올 때 발생 합니다.

Parameters

aGxSymbol 대상 종목(IGxSymbol)
aGxLastTick

옵션월물목록

GxOptionMonths

개요

옵션의 월물 목록을 관리합니다.

IGxOptionMonths - GUID {A19DAC39-E7BC-454F-9D1F-A6CF5CE44428}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 옵션월물 수
ID: 0	IDispatch	Item[Variant index] 옵션월물을 목록에서 순서에 따라 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 정보

필드 상세설명

IDispatch Item [Variant index]

옵션 월물을 목록에서 순서에 따라 조회합니다.

Parameters :

index index 번째 월물을 조회합니다.
index 가 '1' 이면 최근월물입니다.

Return

월물 객체(IGxOptionMonth)

옵션월물

GxOptionMonth

개요

옵션 월물에 대한 정보를 담고 있습니다.

IGxOptionMonth - GUID {054113C8-A0A8-42B6-B3E9-C31A743536DE}

목록

Fields

ID: 1	BSTR	**MonthCode** 옵션 월물의 코드입니다. (형식: 'YYYYMMDD')
ID: 2	BSTR	**MonthDesc** 옵션 월물의 설명입니다. (형식 월물: 'MM월' 위클리: 'W주')
ID: 3	IDispatch	**StrikePrices** 옵션월물의 행사가를 조회합니다. 옵션월물의 종목들을 조회하고자 할 때는 행사가를 통해서 조회를 합니다. (IGxStrikePrices)
ID: 4	BSTR	**LastMessage** 최신 에러 정보

행사가목록

GxStrikePrices

개요

옵션월물의 행사가 목록을 관리합니다.

IGxStrikePrices - GUID {4B66F544-2E23-430C-A237-CFA5092C2EB5}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 행사가 수
ID: 0	IDispatch	Item[Variant index] 행사가를 목록에서 순서에 따라 조회합니다.
ID: 4	BSTR	LastMessage 최신 에러 정보

필드 상세설명

IDispatch **Item** [Variant index]

행사가를 목록에서 순서에 따라 조회합니다.

Parameters

index index번째 행사가를 조회합니다.

Return

행사가 객체(IGxStrikePrices)

행사가

GxStrikePrice

개요

행사가에 대한 정보를 담고 있습니다.

IGxStrikePrice - GUID {6B8CDCB5-52A4-46E4-A4ED-F35E7B70FBAB}

목록

Fields

ID: 1	double	**StrikePrice** 행사가
ID: 2	BSTR	**StrikeDesc** 행사가 설명(형식: '127.5', '127.0')
ID: 3	IDispatch	**Call** 해당 행사가의 콜 종목(IGxSymbol)
ID: 4	IDispatch	**Put** 해당 행사가의 풋 종목(IGxSymbol)
ID: 5	BSTR	**LastMessage** 최신 에러 정보

라. 챠트 및 기타

차트자료관리

GxChartStore

개요

챠트자료를 얻고자 할 때 사용합니다. 챠트 자료를 사용하고자 할 때는 다음과 같은 절차를 따릅니다.

Add() 를 통해서 챠트 자료 객체를 추가합니다. (IGxChartData)

Add() 를 통해서 얻어진 객체를 이용해 챠트 자료를 요청합니다.

IGxChartStore - GUID {C2F8CC6C-E32F-412A-9A24-C9B8A883CC41}

목록

Fields

ID: -4	IUnknown	**_NewEnum** IEnumVariant 를 구현하는 객체
ID: 3	long	**Count** 챠트 객체 수
ID: 0	IDispatch	**Item**[Variant index] 챠트 객체를 목록에서 순서에 따라 참조합니다.
ID: 4	BSTR	**LastMessage** 최신 에러 정보

Methods

ID: 5	IDispatch	**Add**( ) 새로운 챠트 객체를 생성합니다.
ID: 6	VARIANT_BOOL	**Remove**(long index) index 번째 챠트 객체를 삭제합니다.
ID: 7	VARIANT_BOOL	**RemoveBySerialNo**(long iSerialNo) 고유번호를 통해 챠트 객체를 삭제합니다.
ID: 8	VARIANT_BOOL	**UpdateEventFiltered** 일정 시간 간격으로 OnDataUpdated 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.
ID: 9	long	**UpdateEventInterval** UpdateEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.

필드 상세설명

IDispatch **Item** [Variant index]

챠트 객체를 목록에서 순서에 따라 참조합니다.

Parameters

index index 번째의 챠트 객체(IGxChartData)

Return

index 번째의 챠트 객체(IGxChartData)

VARIANT_BOOL **UpdateEventFiltered**

일정 시간 간격으로 OnDataUpdated 이벤트를 발생시킬 것인지를 결정하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시

True 일정시간 간격으로 OnDataUpdated 이벤트를 발생시킵니다.
False Term의 Close 변경 시 마다 OnDataUpdated 이벤트를 발생 시킵니다.

Return : VARIANT_BOOL

True 현재 일정 시간 간격으로 OnDataUpdated 이벤트를 발생시키고 있습니다.
False 현재 Term의 Close 변경 시 마다 OnDataUpdated 이벤트를 발생 시킵니다.

long**UpdateEventInterval**

UpdateEventFiltered를 사용할 경우 그 시간 간격을 설정 하거나 현재 설정 되어 있는 값을 읽습니다.

속성 : Read / Write

설정 시

시간 간격을 설정합니다. 단위는 mili second 입니다.
기본값은 1000(1초), 상한값은 5000(5초) 하한값은 50(0.05초) 입니다.

Return

현재 설정된 시간 간격을 반환합니다.

메소드 상세설명

IDispatch **Add**( )

새로운 챠트 객체를 생성합니다. 원하는 챠트 자료를 얻기 위해서는 이 메소드로 얻어진 객체로 작업을 합니다.

Return

새로운 챠트 객체(IGxChartData)

VARIANT_BOOL**Remove**(long Index)

index 번째 챠트 객체를 제거 합니다.

Parameters

long Index 제거할 위치의 Index 입니다. (1 ~ Count)

Return

정상적으로 제거되면 True, 실패하면 False

VARIANT_BOOL **RemoveBySerialNo**(long iSerialNo)

고유번호가 iSerialNo 인 챠트 객체를 제거합니다

Parameters

iSerialNo

차트자료정보

GxChartData

개요

챠트 자료를 관리합니다. 챠트 종류를 정하게 되면 서버에서 자료를 가져 오며, 실시간으로 자료를 만들어 갑니다. 챠트를 요청하는 순서는 다음과 같습니다

챠트 정의 : 종목, 자료종류, 자료기준, 자료간격, 자료수 등, (Define() 이용)

챠트 준비 이벤트 확인 : 챠트자료가 준비가 되면 이벤트가 발생하며, 이때부터 정상적으로 챠트자료를 이용하시면 됩니다. (OnDataRefreshed 이벤트)

자료 추가 및 갱신 확인 : 실시간 시세가 들어 옴에 따른 이벤트(OnDataAdded, OnDataUpdated)가 발생하면 그에 따른 작업을 합니다.

IGxChartData – GUID {16DE1E36-4D9B-4B91-924D-1B44831335CA}
IGxChartDataEvents – GUID {0799439F-5608-40BA-96FF-40FA0E3E9D29}

목록

Fields

ID: 1	IDispatch	**Symbol** 종목(IGxSymbol)
ID: 2	IChartDataType	**DataType** 챠트 종류(가격챠트/미결제약정챠트)
ID: 3	IChartBaseType	**Base** 자료기준(가격챠트: 틱/분/일봉/주/월봉, 미결제약정:분/일봉)
ID: 4	long	**Period** 챠트 데이터 간격(예, 3분봉인 경우 '3'이 됨)
ID: 5	long	**DataCount** 챠트 자료 수
ID: 6	IDispatch	**Terms** 챠트 자료(IGxTerms)
ID: 7	VARIANT_BOOL	**Ready** 챠트 자료 준비 상태. 이 값이 True 가 되어야만 챠트 자료를 정상적으로 사용하실 수 있습니다.
ID: 8	VARIANT_BOOL	**Requesting** 서버에서 챠트 누적 자료 수신 중 여부를 나타냅니다. 서버에서 챠트 누적 자료를 수신 중에는 챠트를 재정의를 할 수 없습니다.
ID: 9	long	**SerialNo** 챠트의 고유번호. 삭제시 사용합니다.
ID: 10	BSTR	**LastMessage** 최신 에러 메시지 입니다.

Methods

ID: 11	VARIANT_BOOL	Define(BSTR stSymbolCode, IChartDataType DataType, IChartBase Base, long iPeriod, long iCount) 챠트 자료를 정의합니다.
ID: 12	void	Undefine( ) 실시간 시세를 받아 누적하는 것을 중단합니다.
ID: 13	VARIANT_BOOL	Define2(BSTR stSymbolCode, IChartDataType DataType, IChartMarketFlag MarketFlag, IChartBase Base, long iPeriod, long iCount) 챠트 자료를 정의합니다.(장구분별)

Events

ID: 1	IGxChartDataEvents	OnDataRefreshed(IDispatch aGxChartData) 서버에서 누적자료 수신이 완료되면 통보합니다. 이때, Ready 속성이 True 가 됩니다.
ID: 2	IGxChartDataEvents	OnDataAdded(IDispatch aGxChartData) 실시간 시세로 인하여 자료가 추가될때 알려 줍니다.
ID: 3	IGxChartDataEvents	OnDataUpdated(IDispatch aGxChartData) 실시간 시세로 인하여 마지막 자료가 갱신이 될 때 통보됩니다.

메소드 상세설명

VARIANT_BOOL**Define** (BSTR stSymbolCode, IChartDataType DataType,IChartBase Base, long iPeriod, long iCount)

챠트 자료를 정의합니다.

Parameters :
stSymbolCode 종목코드
DataType 자료 유형(가격챠트/미결제약정 챠트)
Base 자료 기준(틱/분/일/주/월봉 등)
iPeriod 챠트 간격
iCount 자료 수.

Return

성공하면 True, 실패하면 False

void**UnDefine**()

실시간 시세를 이용 챠트 자료를 늘려 나가는 작업을 중단합니다.

VARIANT_BOOL**Define2** (BSTR stSymbolCode, IChartDataType DataType,

IChartMarketFlag MarketFlag, IChartBase Base, long iPeriod, long iCount)

장구분별 챠트 자료를 정의합니다.

Parameters :
stSymbolCode 종목코드
DataType 자료 유형(가격챠트/미결제약정 챠트)
MarketFlag 장구분(주간/야간/주간+야간)
Base 자료 기준(틱/분/일/주/월봉 등)
iPeriod 챠트 간격
iCount 자료 수.

Return

성공하면 True, 실패하면 False

이벤트 상세설명

OnDataRefreshed (IDispatch aGxChartData)

서버에서 누적자료 수신이 완료되면 통보합니다. 이때, Ready 속성이 True 가 됩니다.

Parameters

aGxChartData 이벤트가 발생한 챠트 자료 객체(IGxChartData)

< IGxChartDataEvents >**OnDataAdded** (IDispatch aGxChartData)

실시간 시세로 인하여 자료가 추가될때 알려 줍니다.

Parameters

aGxChartData 자료가 추가된 챠트 자료 객체(IGxChartData)

< IGxChartDataEvents >**OnDataUpdated** (IDispatch aGxChartData)

실시간 시세로 인하여 마지막 자료가 갱신이 될 때 통보됩니다.

Parameters

aGxChartData

Terms

GxTerms

개요

챠트 자료를 관리합니다.

IGxTerms - GUID {10137DAF-376A-4734-AD23-060A2E807D3B}

목록

Fields

ID: -4	IUnknown	_NewEnum IEnumVariant 를 구현하는 객체
ID: 3	long	Count 챠트 자료 수
ID: 0	IDispatch	Item[Variant index] 챠트 자료를 목록에서 순서에 따라 참조합니다.
ID: 4	BSTR	LastMessage 최신 에러 정보

필드 상세설명

IDispatch Item [Variant index]

챠트 자료를 목록에서 순서에 따라 참조합니다.

Parameters

index index 번째의 챠트 자료(IGxTerm) 참조합니다.

Return

챠트자료 객체(IGxTerm)

Term

GxTerm

개요

시간대별 시/고/저/종, 미결제 약정 정보를 담고 있습니다.

IGxTerm - GUID {B90F5661-86F3-4F09-AD47-AC0EA96FC3EB}

목록

Fields

ID: 1	double	StartTime 시작 시각
ID: 2	double	LastTime 끝 시각
ID: 3	double	Open 시가
ID: 4	double	High 고가
ID: 5	double	Low 저가
ID: 6	double	Close 종가
ID: 7	double	Volume 거래량
ID: 8	double	AccVolume 누적 거래량
ID: 9	double	OpenInterest 미결제 약정
ID: 10	BSTR	LastMessage 최신 에러 정보

서버연결정보

GxServerInfo

개요

증권사 서버 연결 정보를 조회하고, 서버와 연결이 끊어졌을 경우 이를 알려주는 역할을 합니다.

IGxServer - GUID {86356355-B0F7-4922-9794-09E53775E2C7}

목록

Fields

ID: 1	VARIANT_BOOL	**Connected** 고수와 증권사 서버와의 연결 여부를 알려줍니다. 연결되어 있으면 값이 True 가 됩니다.
ID: 2	double	**ServerDate** 서버 날짜를 조회합니다.
ID: 3	BSTR	**LastMessage** 최신 에러 메시지
ID: 4	double	**TradeDate** 영업일자를 조회합니다.
ID: 5	IMarketType	**MarketType** 주간장/야간장을 조회합니다.

Event

ID: 1	IGxServerInfoEvents	OnDisconnected(IGxServerInfo aInfo) 고수가 종료됐을 때 이를 통보합니다.

이벤트 상세설명

< IGxServerInfoEvents >**OnDisconnected** (IGxServerInfo aInfo)

고수가 종료됐을 때 이를 통보합니다.

Parameters
aInfo IGxServerInfo 객체

4. 기타

가. 시작하기전 유의사항

Early Binding 과 Late Binding

Early Binding 은 vtable 에 서버 객체의 프로퍼티나 메소드의 메모리 주소를 두고 Client 가 접근할 때 이를 이용하는 빠른 접근 방법을 제공합니다. Late Binding 은 속성이나 메소드 이름을 통하여 얻은 dispatch ID 를

통해 접근하는 방법으로 접근 속도가 느립니다. Microsoft 는 이를 보완하기 위해 이 두 가지 방법을 같이 쓰는 dual interface 를 권고하며 GOM 에서는 dual interface 를 지원하기 때문에 두 가지 방식의 사용이 모두 가능합니다. 그러나 HTS 는 속성상 빠른 속도가 필요로 하고 많은 접근 횟수를 필요로 하므로 사용자들에게 Early Binding 을 권하는 바 입니다.

객체의 접근 횟수

무엇보다도 클라이언트 제작시 유의점은 최소한의 CPU 로드를 줄이는 것입니다. 이는 서버, 클라이언트의 원활한 동작을 보장하므로 클라이언트를 작성하고 동작을 시키면서 자주 프로세서(CPU) 가동율을 확인하는 것이 좋습니다. 프로세서 가동율이 100%가 계속 유지가 되면 클라이언트가 할 일들이 수 초간 지연 되거나 프로그램이 멈추어 있는 현상, 이벤트 수신 실패등 문제가 발생하므로 프로세서(CPU) 가동률을 자주 점검해야 합니다.

COM 객체의 메소드, 속성의 접근은 동일 프로그램 내의 객체 접근보다 부하가 훨씬 드는 작업입니다. 이는 다른 프로세스 간의 통신을 위해 우리 눈에는 보이지 않는 많은 작업이 존재함을 의미합니다.

GOM 도 COM 객체 이므로 이와 같은 부하의 문제에 자유로울 수 는 없습니다. 그래서 클라이언트는 GOM 객체에 대한 접근을 최대한 줄여야 합니다. 이를 테면 클라이언트의 모듈에서 GxSymbol 의 현재가가 여러 번 필요하다고 가정합니다. 이런 상황에 클라이언트가 현재가가 변경되지 않은 상태에서는 서버의 GxSymbol 객체의 현재가 속성을 여러번 호출하는 것보다 한 번 접근하여 그 값을 클라이언트 내에서 저장하고 그 값을 사용하는 것이 부하를 경감하는 방법 일 수 있습니다.

이벤트 핸들러

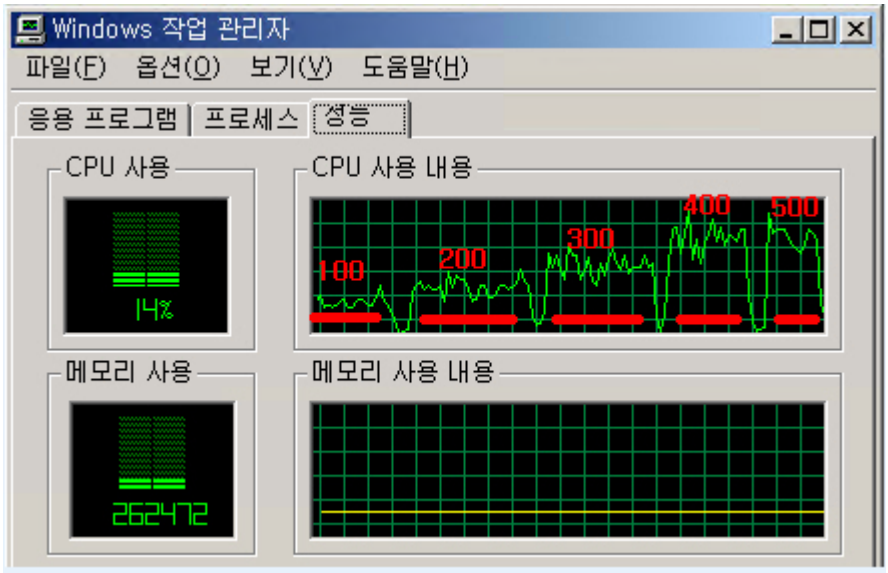
COM 이벤트 수신 작업을 약간 살펴 보면 서버측에서 이벤트를 발생 시키면 서버측은 멈추게 되어 클라이언트는 제어권이 순간적으로 넘어 갑니다. 이 때 클라이언트의 이벤트 핸들러를 실행하고 핸들러가 종료되면 다시 서버로 제어권을 넘기게 됩니다.

이는 클라이언트의 이벤트 핸들러가 마칠 때까지 서버가 멈추게 되는 의미이므로 클라이언트는 이벤트 처리를 빠른 시간 내에 마쳐야 합니다. 윈도우 메시지가 사용 가능한 Language 는 윈도우 메시지를 날리고 이벤트 핸들러를 바로 마쳐서 제어권을 서버에 넘겨주고 윈도우 메시지를 받아 실제 이벤트 핸들링을 하는 방법이 좋습니다.

그리고 이벤트 핸들러 상에서 부하가 많은 작업이나 모달 다이얼로그를 띄우는 것은 절대 피해야 합니다. 모달 메시지 다이얼로그는 제어권을 멈추어 버리게 하여 서버를 멈추게 합니다. 그리고 이벤트 핸들러상에서 꼭 Exception 핸들링을 해야 합니다. 이벤트 핸들링 과정에서 에러가 발생하여 에러 창등이 나타나는 현상도 서버를 멈추게 합니다.

EventFiltered 사용

EventFilter 사용하지 않는 경우의 CPU (숫자는 초당 입력 시세 건수)



EventFilter 사용하는 경우의 CPU (숫자는 초당 입력 시세 건수)

위의 두 그림을 비교하면 EventFilter 를 사용하지 않는 경우가 훨씬 많은 시세 입력을 받아 들이는 것을 볼 수 있습니다.

종목 이벤트는 어떠한 이벤트 보다 자주 발생하는 이벤트 입니다. 특히 호가, 체결 이벤트의 경우 때에 따라 빈도수가 무척 많아짐니다. 그러나 GxSymbolStore 의 QuoteEventFiltered, PriceEventFiltered 속성을 이용하면 호가나 현재가가 변할 때 마다 이벤트를 발생시키는 것에서 일정시간 마다 이벤트를 발생시키는 것으로 변경할 수 있습니다. 이와 같은 Filtering 은 호가, 현재가의 변화를 민감도를 떨어 뜨리나 클라이언트의 부하는 많이 줄일 수 있습니다. 이 Filtering 의 사용 여부는 자신의 시스템 환경이나 자신이 필요로 하는 민감도를 감안 하여 결정하셔야 합니다.

GxChartStore 관리

GxChartStore 는 'Add'를 하여 GxChartData 를 생성시킵니다. 만일 많은 수의 'Add'를 하고 이에 대해 이벤트 요청을 하면 프로세서 부하나 메모리 문제등이 발생할 수 있습니다. 그래서 GOM 에서는 GxChartData 의 갯수 상한치를 100 개로 지정하고 이 이상의 'Add'를 할 경우 NULL 을 반환하지만 사용하지 않은 GxChartData 는 GxChartStore 의 Remove 나 RemoveByKey 를 사용하여 제거하거나 GxChartData 의 UnDefine 을 사용, 이벤트 중지등을 사용하는 것이 좋습니다.

UI 작업과 I/O 작업

이곳에서는 프로그래밍 전반에 대한 요령을 얘기 하겠습니다. 프로그래밍에서 부하를 많이 잡는 작업으로는 I/O(예:파일작업), UI 작업등이 있습니다. UI 작업은 화면에 나타나게 하는 작업인데 Optimize 작업을 소홀히 하면 이 작업에서 많은 부하를 잡게 됩니다. 예를 들어 MSFlexGrid 컨트롤같은 경우 빠른 시간내에 현재 포커스되어 있는 셀을 계속 바꾸어가면서 셀의 색깔을 바꾸는 작업등은 부하를 굉장히 많이 잡습니다. 그리고 시세를 수신함과 동시에 파일을 저장하는 작업등도 피해야 합니다. 일반적인 그리고 프로그래밍에서 부하 처리는 굉장히 중요한 문제 이므로 GOM 사용자 분들은 이에 신경을 많이 써 주시기 바랍니다.

고수시스템의 변경시 작업

서버나 고수의 배포가 발생할 경우는 GOM 에 대한 테스트가 필요합니다. GOM 사용자분들은 고수의 배포내역과 공지내역을 꼭 확인부탁드리고 이에 대한 적절한 조치를 꼭 하시기 바랍니다.

나. VB, VBA 예제

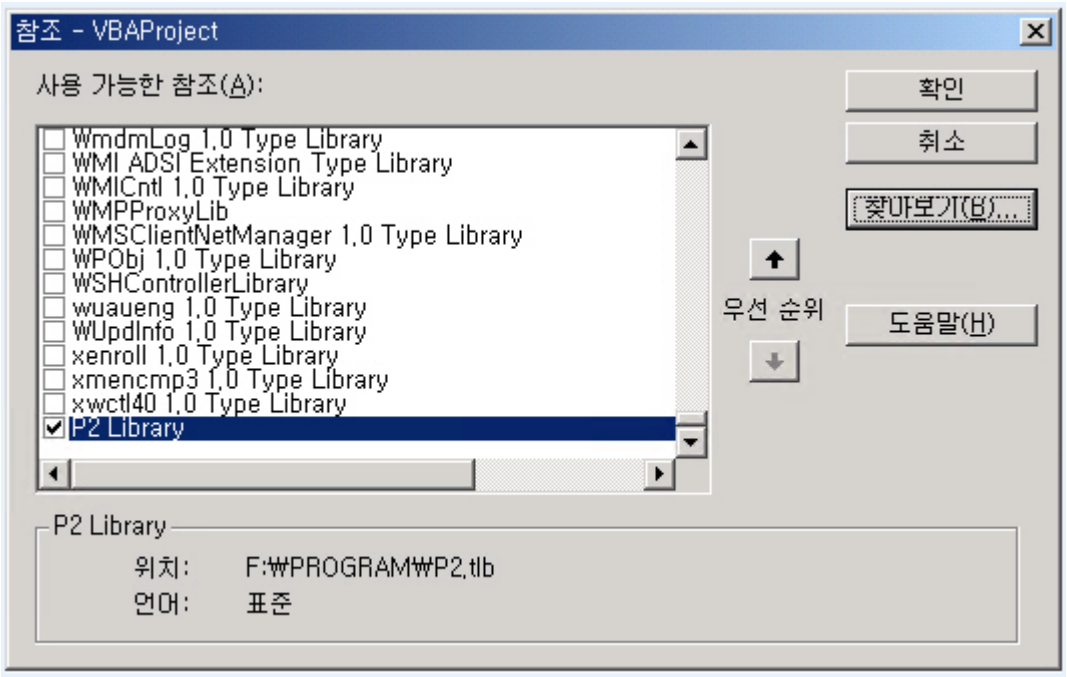
개요

이곳에서는 Visual Basic과 Microsoft Excel에서 사용가능한 VBA(Visual Basic Application)를 사용하는 방법에 대해 설명하겠습니다.

MS-Excel VBA와 GOM 시작

Microsoft Excel에서는 VB 엔진을 탑제하였기 때문에 사용자의 간단한 프로그래밍을 통하여 Excel Document에 대한 컨트롤을 할 수 있습니다. 이를 위해 사용자는 도구가 되는 Visual Basic의 문법과 Excel Document에 대하여 어느 정도 이해가 있어야 합니다.

Late Binding 을 사용하는 경우 타입 라이브러리를 사용할 필요가 없지만 저희가 Early Binding을 강력 권고하므로 Early Binding 이용을 가정하고 설명을 진행 하겠습니다.

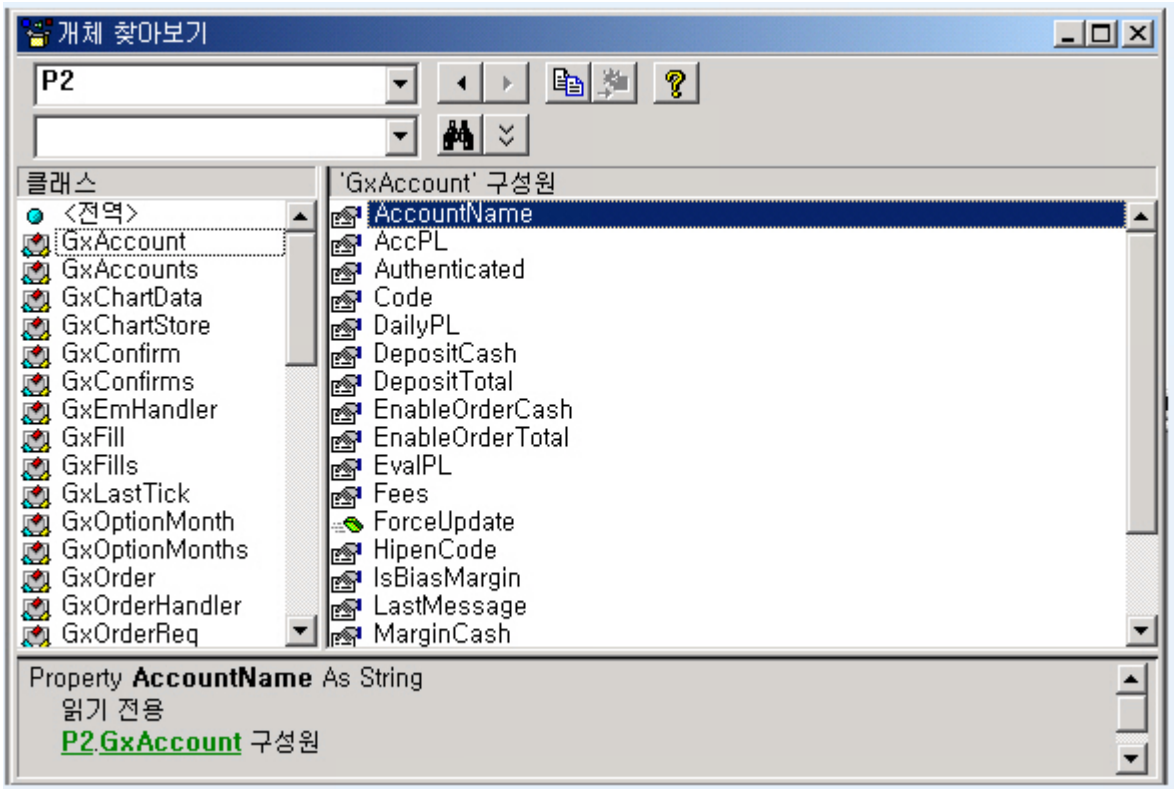


타입 라이브러리를 import 하는 화면

VBA 프로젝트에서 GOM 을 시작위한 첫 단계는 GOM 타입 라이브러리 import 입니다. 작업은 다음과 같은 순서로 진행합니다.

- Excel을 연 후 Visual Basic Editor를 엽니다.
- Visual Basic Editor 메뉴의 도구/참조를 실행합니다.
- '사용 가능한 참조'에서 P2 Library가 있는지 확인 합니다.
- P2 Library가 있는 경우 체크하고 확인을 누릅니다.
- P2 Library가 없는 경우 자료실에서 P2.tlb를 다운받아 찾아보기를 통하여 선택하고 확인을 누릅니다.
- 이로서 GOM 타입 라이브러리가 프로젝트가 import 되었습니다.

타입 라이브러리를 추가한 후 타입 라이브러리의 내용을 보실 수 있습니다. View menu 의 Object Brower 를 사용하여 P2 라이브러리를 선택 하시면 다음과 같은 화면을 보실 수 있습니다.



GOM 타입 라이브러리의 내용

VB, VBA에서의 Early Binding 과 Late Binding

Binding은 클라이언트가 메소드나 속성을 내부적으로 요청을 하는 방법이고 앞서 'COM, Automation 개요'에서 말한 바와 같이 Early Binding, Late Binding 두 가지 방식이 있습니다. VB, VBA에서 Binding 방식은 사용자의 변수를 선언하는 방법에 따라 결정 됩니다.

Early Binding 방식은 클라이언트가 GOM 타입 라이브러리를 import하고 타입 라이브러리에 선언된 타입대로 변수를 선언하는 방식이고 다음과 같습니다.

```
Dim objSymbol As GxSymbol
```

타입 라이브러리의 정보는 컴파일 시 이용 가능 하기 때문에 VB, VBA가 메소드나 속성을 실행하는데 빠른 접근 시간을 제공합니다.

VB, VBA에서 Late Binding 은 타입 라이브러리를 import하지 않고 Object 타입으로 변수를 선언 합니다. Object 타입으로 변수를 선언 한다면 VB, VBA는 컴파일시에는 속성이나 메소드의 주소를 결정하지 않습니다. 프로그램 실행시 속성이나 메소드의 실행이 필요할 때 'COM, Automation 개요' 에서 언급한 바와 같이 내부적으로 GetIDsOfNames와 Invoke 를 사용하여 작업을 수행하므로 느린 접근 시간을 제공합니다. 다음은 Late Binding의 한 예 입니다.

```
Dim objSymbol As Object
```

GOM 에서는 HTS의 특성상 빠른 접근을 위하여 Early Binding을 사용하는 것을 강력히 권고합니다.

VB, VBA GOM 프로그래밍

GOM 클라이언트 제작 다음 단계는 최상위 Object의 인터 페이스를 얻는 작업 입니다. GetObject, CreateObject 두 함수에 PROGID를 사용하여 최상위 Object의 인터페이스를 얻습니다. GetObject, CreateObject 둘 다 클라이언트 측에서 보면 별 차이가 없으므로 둘 중 하나를 사용합니다만 메모리 절약을 위해 GetObject를 사용하시기를 추천 합니다. 가져온 인터 페이스는 프로그램 종료 시 까지 유지하기 위해 로컬 변수가 아닌 클래스의 멤버나 글로벌에 변수에 저장해야 합니다. GOM은 Tree 모양의 Object 구조이기 때문에 이렇게 얻은 최상위 인터페이스를 통해 GOM의 모든 Object에 대한 접근이 가능합니다.

VBA 에서 Object에 대한 이벤트를 받는 방법은 여타 어느 Lauguage 보다 간단 합니다. 우선 WithEvents 지시어를 사용하여 Event Sink(Event를 받는 Object)를 선언 합니다. 그리고 코드 에디터에의 상단 왼쪽에 Object가 리스트가 되는 콤보박스에서 방금 전 선언한 Event Sink Object를 선택 합니다. 만약 Event Sink 타입이 Connectable Object(이벤트를 지원하는 GOM Object) 타입으로 선언 되었다면 콤보박스 오른쪽의 프로시저 선택 콤보박스에 선택 가능한 이벤트 핸들러가 나열 됩니다. 이벤트 핸들러를 선택하면 자동으로 이벤트 핸들러가 코드 에디터에 작성이 되고 이 안에 자신의 코드를 작성합니다. 그리고 사용자가 생성한 Event Sink Object에 이벤트를 발생 시키는 GOM Object를 할당합니다.

VB, VBA GOM 프로그래밍시 유의점

앞서 '클라이언트 유의사항'에서 언급 했듯이 클라이언트에서의 UI 처리는 중요 합니다. Excel은 화면 처리는 어떤 Language로 구현 했을 때보다 부하가 많이 듭니다. (GOM 에 상관 없이 빠른 시간내에 셀을 바꿔가면서 색상을 바꿔가는 것을 프로그래밍을 하시면 느끼실 수 있는 점 입니다.) 특히 시세 쪽 구현 시 다른 이벤트 보다 빈도 수가 많으므로 화면 처리를 최소화해야 합니다.

예제 파일

경고 : 이 예제는 GOM 사용법을 쉽게 익히기 위해 제공 하는 것으로, 이 예제를 거래에 이용 하는 중 에러나 다른 이유로 발생하는 피해는 당사에서 책임을 지지 않습니다.
2004. 05. 18 (주)델타 익스체인지

VBA 예제

GxChartStore 예제 <제작 버전:1.21> : 다수의 챠트 데이터 수신
GxChartStore 예제 <제작 버전:1.22> : 다수의 챠트 데이터 수신 예제에 간단한 이벤트 필터 기능 추가
GxSymbol 예제 <제작 버전:1.1> : 입력 종목코드의 5단계 호가, 현재가, 체결 표시
Multi GxSymbol 예제 <제작 버전:1.0> : 선택 월물 옵션종목의 현재가, 미결제 표시
Greeks 예제 <제작 버전:1.1> : 선택 월물 옵션종목의 그릭스 표시
Account, Position 예제 <제작 버전:1.0>
GxOrders, GxFills, GxConfirms 예제 <제작 버전:1.0>
GxOrderHandler 예제 <제작 버전:1.0>
EventFiltering을 이용한 GxSymbol 예제 <제작 버전:1.2> : VBA GxSymbol 예제에 EventFiltering 기능 추가
평가예탁금 예제 <제작 버전:1.23>

VB 예제

GxOrderHandler, GxEmHandler 예제 <제작 버전:1.0>

GxSymbol을 이용한 ATM 예제 <제작 버전:1.1>
GxChartData 예제 <제작 버전:1.1>
옵션 다종목 예제 <제작 버전:1.23>
포지션, 체결, 미체결 예제 <제작 버전:1.23>
자동 정정 주문 <제작 버전:1.23>
스탑, 리미트 주문 <제작 버전:1.23>
간단한 곰 연결 예제 <제작 버전:1.23>
간단한 종목 찾기 예제 <제작 버전:1.23>
간단한 최근선물조회 예제<제작 버전:1.23>
간단한 주문목록 조회 예제 <제작 버전:1.23>
간단한 계좌목록 조회 예제 <제작 버전:1.23>

다. DELPHI 예제

DELPHI 클라이언트

개요

이곳에서는 Delphi 를 이용한 GOM을 사용하는 방법을 알아 봅니다. Delphi Client는 내부 DAX 아키텍처를 구현하여 GOM 을 사용하는데 있어 VB와 같이 손쉽고 C++과 같은 저레벨 접근이 가능합니다.

Delphi GOM 프로그래밍

Delphi로 GOM 을 사용하는 방법은 크게 두 가지 입니다. tlb 파일을 Import하여 tlb Unit을 생성하여 tlb Unit만을 통한 방법과 tlb파일을 Import하여 Object를 컴포넌트화 하여 컴포넌트 팔레트에 올려 놓고 사용하는 방법 두 가지 입니다. 컴포넌트화 방법은 TButton 컴포넌트 사용하듯이 간단한 방법입니다. 그러나 tlb Unit만을 사용하는 방법은 Object의 프로퍼티와 메소드 접근하는데 있어 컴포넌트 방식과 유사합니다만 GOM 이벤트 수신 시 Event Sink Object를 직접 작성해야 하는 어려움이 있습니다. 이 점 때문에 사용자 여러분께는 컴포넌트화 방법을 추천합니다.

Project>Import TypeLibrary 메뉴를 통하여 tlb 파일을 import 하고 Install을하여 사용자 패키지에 추가 합니다. 그러면 컴포넌트 팔레트에 각 Object들의 컴포넌트가 생성됩니다. 사용하고자 하는 하는 컴포넌트를 폼에 올려 보통의 컴포넌트와 같이 사용 합니다. Object Inspector 에는 Object의 속성은 나타나지 않습니다. 그러나 속성과 메소드는 코드상에서 사용하시면 됩니다. 그러나 이벤트는 Object Inspector 에서 사용하실 수 있습니다.

tlb 내용을 보기 위해서는 Delphi 상에서 'open' 메뉴를 이용하여 P2.tlb 를 엽니다. 화면은 다음과 같습니다.

Delphi 를 이용한 GOM 프로그래밍

GOM의 GxChartData를 사용하는 Delphi 클라이언트를 작성 합니다.

소스 : Delphi GxChartData 샘플

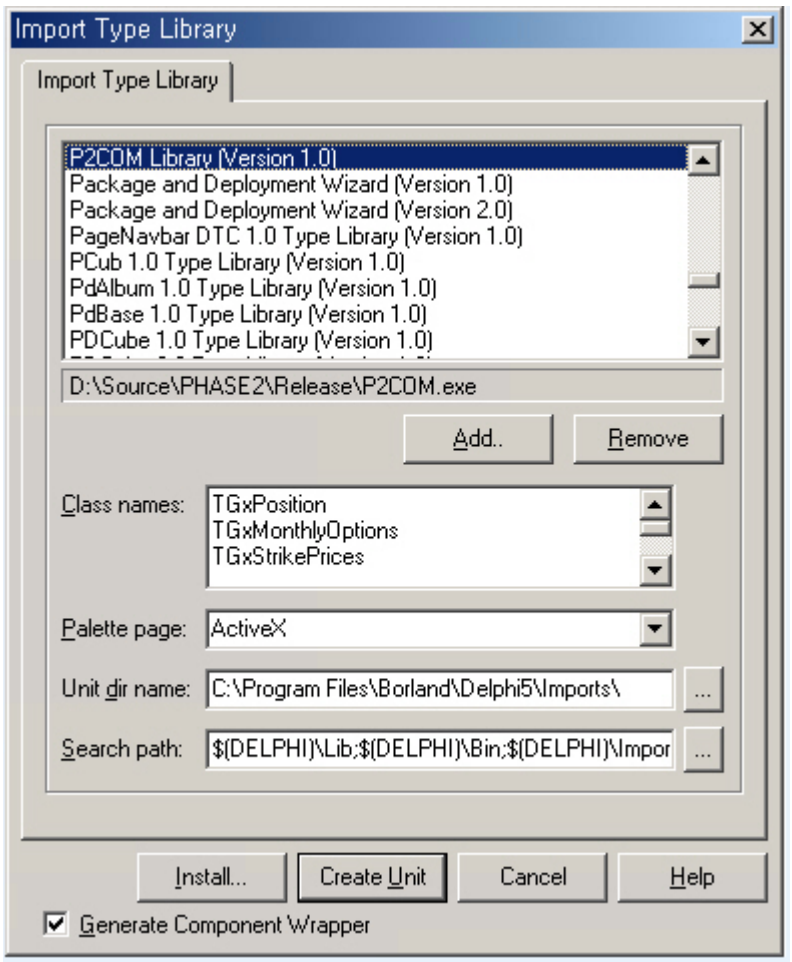
프로젝트 시작

이 예제는 GOM을 이용하여 사용자가 요청한 챠트 데이터를 수신받아 표시하는 Delphi 클라이언트 입니다. Client에는 종목, 단위, 표시단위, 자료수를 설정하여 요청하는 버튼이 있습니다. 그리고 챠트 데이터를 표시하는 리스트와 GOM의 서버 Object의 현황을 보여 주는 리스트로 구성이 됩니다.

요청 버튼은 두 종류 입니다. 하나는 새로운 챠트 데이터를 요청하는 버튼이고 또 하나는 기존의 챠트 데이터에 기존 데이터를 삭제하고 새 챠트 데이터를 요청하는 버튼입니다. 전자는 '새요청'버튼이고 후자는 '재요청'버튼입니다. '삭제' 버튼은 챠트 데이터를 삭제하고 삭제종료는 챠트 데이터를 삭제와 함께 클라이언트를 종료 합니다.

GOM을 사용하기 위한 Delphi 사용법은 크게 두 가지 입니다. 하나는 tlb 파일을 Unit화 하여 Unit을 참조하여 사용하는 방법이고 또 하나는 컴포넌트화 하여 사용하는 방법입니다. 이벤트 구현 시 컴포넌트화 하여 사용하는 방법이 훨씬 사용하기 용이 하므로 이 예제 에서는 컴포넌트화 방법을 사용 합니다.

우선 Delphi에 GOM 컴포넌트를 만드는 방법을 설명 하겠습니다. 우선 Project메뉴의 'Import Type Library' 메뉴를 선택 합니다. 그러면 아래와 같은 화면이 나타납니다. 'Add' 버튼을 눌러 P2.tlb 파일을 찾아 선택합니다. 그리고 'Install' 버튼을 눌러 컴포넌트를 만듭니다. 컴포넌트 만드는 자세한 방법은 모든 Delphi 서적에 나와 있으므로 이를 참조 합니다. 설치 후 컴포넌트 팔레트에 GOM 컴포넌트가 나타납니다.



프로젝트 작성

ListView, EditBox등을 화면에 배치 합니다.

GxChartData 컴포넌트를 화면에 올려 놓습니다.

폼 Class에 IGxServer, 즉 대표 인터페이스를 선언 합니다.

화면이 시작할 때 대표 인터 페이스를 가져 와야 하므로 FormCreate 이벤트 핸들러에 GetActiveOleObject나

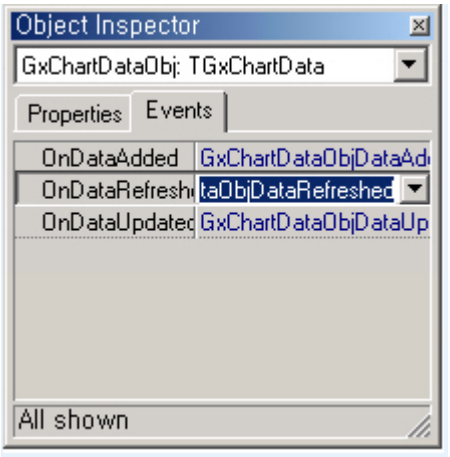
CreateOleObject를 사용하여 대표 인터페이스를 얻습니다. 두 API 를 사용하기 위해서는 'ComObj' Unit을 uses 에 추가 합니다.

'새요청'버튼의 Click 이벤트 핸들러를 작성 합니다. 이 곳에 대표 인터페이스의 하위 IGxChartStore 에 Add 메소드를 호출합니다. 그러면 새로 만들어진 IGxChartData가 리턴이 되는데 이를 IGxChartData의 이벤트를 수신하기 위해 앞서 올려 놓은 GxChartData 컴포넌트에 연결 합니다.

Add 후 리턴된 IGxChartData에 Define 메소드를 호출함으로써 실제 챠트 데이터 요청을 합니다.

그 다음 연결한 GxChartData 컴포넌트의 이벤트를 작성해야 합니다.

GxChartData 컴포넌트를 선택하고 Object Inspector의 Events 탭을 보면 OnDataAdded, OnDataRefresh, OnDataUpdate 가 표시 되어 있는 것을 보실 수 있습니다. 이를 더블클릭하여 이벤트 핸들러 작성에 들어 갑니다. 이제 이 세개의 이벤트를 작성 하겠습니다.



OnDataRefrehed 는 Define시 요청한 챠트 데이터가 GxChartData에 모두 준비가 됐을 때 발생합니다. 챠트 데이터는 IGxChartData 인터페이스 하위의 IGxTerms Collection을 통하여 읽을 수 있습니다. 예제의 OnDataRefreshed 이벤트 핸들러는 IGxChartData의 IGxTerms Collection을 사용하여 데이터를 ListView에 기록하는 작업을 합니다.

OnDataAdded 는 OnDataRefreshed 이벤트 발생 후 실시간 체결을 통해 새로운 챠트 데이터가 추가 되었을 경우 발생 합니다. 예제에서는 OnDataAdded 발생 시 IGxTerms의 마지막 IGxTerm이 새롭게 추가된 인터페이스이므로 제일 마지막 IGxTerm의 내용은 ListView에 추가하는 작업을 합니다.

OnDataUpdated 는 OnDataAdded 발생 이후 마지막 IGxTerm의 값이 변경되었을 때 발생 합니다. 예제에서는 OnDataUpdated 발생 시 마지막 IGxTerm의 값을 ListView에 마지막 값에 변경시키는 작업을 합니다.

이 세개의 이벤트 타입을 고수의 종합챠트에 비유하면 최초 종합챠트에서 챠트 요청 시 약간의 시간 이후에 과거 시점에서부터 현재까지 그려집니다. 이 사건이 OnDataRefreshed 이벤트이고 시간이 진행되면서 봉이 추가 될 때가 OnDataAdded 이벤트이고 현재가의 변경되면서 봉의 모양이 변하는 사건은 OnDataUpdated 이벤트 입니다.

예제 파일

경고 : 이 예제는 GOM 사용법을 쉽게 익히기 위해 제공 하는 것으로, 이 예제를 거래에 이용 하는 중 에러나 다른 이유로 발생하는 피해는 당사에서 책임을 지지 않습니다.
2004. 05. 18 (주)델타 익스체인지

Delphi 예제

GxSymbol 예제 <제작 버전:1.0>
GxSymbol 예제(델파이버전6이상) <제작 버전:1.0>
GxSymbol LastTick 예제 <제작 버전:1.21>
GxSymbol LastTick 예제(델파이버전6이상) <제작 버전:1.21>
GxTrade(GxPosition, GxAccount, GxOrder, GxConfirm, GxFill) 예제 <제작 버전:1.0>
GxTrade(GxPosition, GxAccount, GxOrder, GxConfirm, GxFill) 예제(델파이버전6이상) <제작 버전:1.0>
GxChartData 예제 <제작 버전:1.0>
GxChartData 예제(델파이버전6이상) <제작 버전:1.0>
3개 종목 수신 예제 <제작 버전:1.23>
동적 종목 추가 예제(델파이버전6이상) <제작 버전:1.23>
그릭스 및 그릭스 설정 예제 <제작 버전:1.23>
그릭스 및 그릭스 설정 예제(델파이버전6이상) <제작 버전:1.23>

라. MFC 예제

MFC 클라이언트

개요

이곳에서는 Visual C++의 MFC를 이용한 GOM 프로그래밍에 대해 알아보겠습니다.

OLE/COM Object Viewer

서버 객체 타입 정보가 있는 tlb 파일을 보기 위해 Visual Studio에 있는 OLE/COM Object Viewer Utility를 사용하면 아래 그림과 같이 모든 class의 정보를 보실 수 있습니다. Visual C++에서 Tool 메뉴에서 OLE/COM Object Viewer를 선택합니다.

IUnknown과 IDispatch

인터페이스는 관련 함수의 주소의 테이블 입니다. 사용자가 인터페이스의 주소를 얻으면 사용자는 인터페이스의 함수를 접근 할 수 있습니다. IUnknown, IDispatch는 Automation 의 핵심 인터페이스 입니다. 모든 COM 객체는 IUnknown 인터페이스를 상속 받습니다. IUnknown 인터페이스는 COM 객체의 수명 관리와 사용자에게 객체가 지원하는 다른 인터페이스를 접근케 합니다. IUnknown 인터페이스를 COM 객체 가 지원을 해야 하는 세 개의 함수를 가지고 있습니다. 그 함수들은 다음과 같습니다.

IUnknown 인터페이스

```
IUnknown :: QueryInterface()
```

객체가 지원하는 인터페이스를 식별하고 접근하는데 호출 됩니다.

IUnknown :: AddRef() 클라이언트가 인터페이스에 요청을 하였을 때 호출 됩니다.

IUnknown :: Release() 클라이언트가 인터페이스를 제거 할 때 호출 됩니다.

COM 객체는 AddRef와 Release를 사용하여 클라이언트에 의해 다뤄지는 인터페이스 수를 추적해야 합니다. 이를 'reference count' 라고 합니다. reference count가 0이 되면 객체는 메모리에서 제거 됩니다. automation 객체는 IDispatch 인터페이스도 구현을 해야 합니다. IDispatch 인터페이스는 4 개의 함수를 제공합니다. 이는 다음과 같습니다.

IDispatch 인터페이스

IDispatch :: GetTypeInfoCount() 타입 정보가 이용 가능하는지 결정할 때 호출됩니다.

IDispatch :: GetTypeInfo() 타입 정보를 얻고자 할 때 호출 됩니다.

IDispatch :: GetIDsOfNames() 속성 이름이나 메소드 이름으로 부터 DISPID를 얻고자 할 때 호출 됩니다.

IDispatch :: Invoke() 메소드나 속성을 요청할 때 호출 됩니다.

GOM을 사용하기 위해선 실행되고 있는 최상위 객체를 얻거나 최상위 객체를 생성해야 합니다. 객체을 생성 하기 위해서 ::CoCreateInstance를 사용해서 IUnknown 인터페이스를 얻어야 합니다. IUnknown의 QueryInterface를 사용 IDispatch 인터페이스를 얻어야 합니다. 이 IDispatch를 사용하여 객체가 공개하는 메소드와 속성을 Invoke를 이용하여 사용할 수 있습니다.

MFC 를 이용한 GOM 프로그래밍 1

GOM의 GxAccount를 사용하는 MFC 클라이언트를 작성합니다.

소스 : MFC GxAccount 샘플

프로젝트 시작

프로젝트를 시작합니다. 프로젝트명은 GxAccounts 이고 Dialog Base 로 하였습니다. 주의할 점은 MFC AppWizard 중 아래 그림과 같이 Automation 체크 박스를 선택 해야 합니다. 선택 시 Automation 클라이언트가 해야 되는 AfxOleInit코드를 작성 합니다.

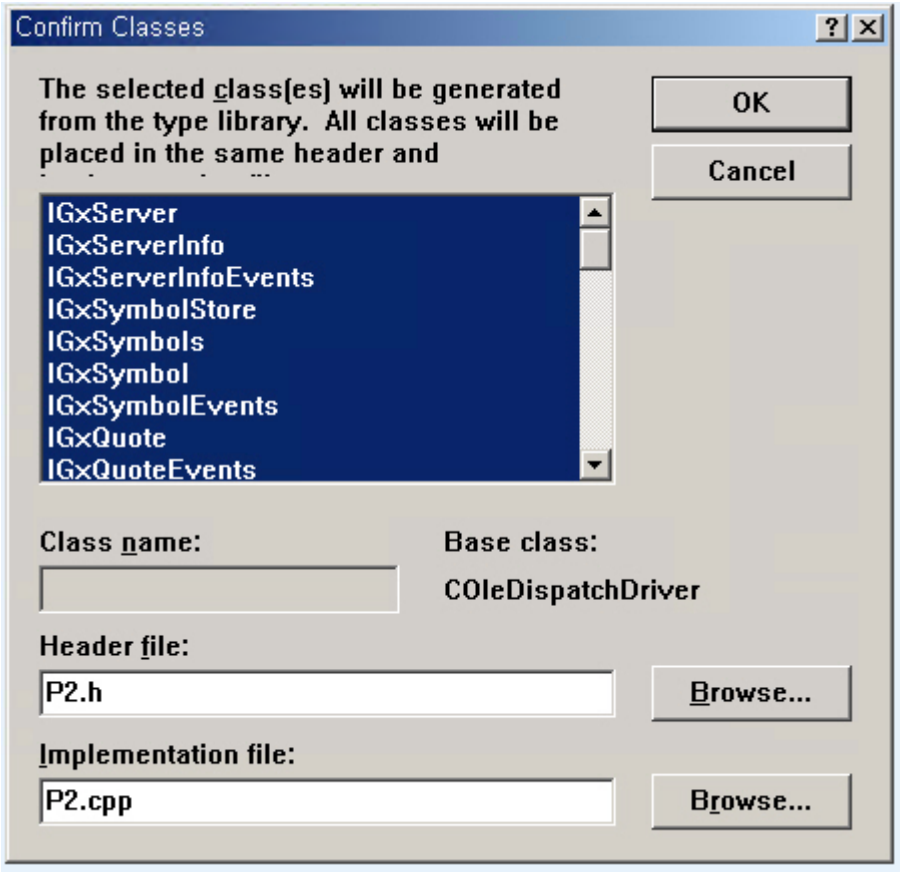
Import Type Libray와 초기 작업

ClassWizard를 실행 합니다.

Add Class 버튼을 누르고 아래로 나타나는 From a Type Libarary 를 선택 합니다.

GOM tlb 파일을 찾아 선택 합니다.

그림과 같이 GOM 인터페이스들이 나타납니다.



리스트에서 선택한 인터페이스만 프로젝트에 포함 되므로 모든 인터페이스를 선택한 후 OK버튼을 누릅니다. 대화창 아래에 'Header file:', 'Implementation file:' 이라고 나오는데 OK 버튼을 누르면 GOM 인터페이스의 wrapper Class가 'Header file:', 'Implementation file:'에 지정한 파일에 자동으로 작성이 됩니다.

WorkSpace 창을 보면 여러 GOM 인터페이스들이 추가 되어 있는 것을 볼 수 있습니다.

CGxAccountDlg 에서 이들 인터페이스 wrapper 들을 사용하기 위해 P2.h를 include 합니다.

우선 GOM 최상위 객체의 인터페이스 IGxServer를 얻어야 합니다. CGxAccountDlg 클래스에 IGxServ 타입의 멤버 변수 m_Serv를 선언 합니다.

다음은 IGxServ 의 Dispatch 를 생성 해야 하는데 창이 완전히 로딩되기 전에 콤보박스에 계좌번호가 로딩이 되어야 하므로 아래 그림과 같이 InitDialog()에서 m_Serv의 Dispatch를 생성 합니다.

```
// -- 서버 Dispatch 생성
```

```
if (!m_lpserver.Createdispatch("P2.GxServer"))
{
AfxMessageBox("서버 실행을 실패 하였습니다. 종료 합니다.");
EndDialog(-1);
}

// -- IGxAccounts 를 이용 계좌 콤보박스 로딩
IGxTradeStore lpTradeStore;
IGxAccount lpAccount;

lpTradeStore = m_lpServer.GetTradeStore();
m_lpAccounts = lpTradeStore.GetAccounts();

// -- GxAccounts Collction Object에 접근
for (int i=1;i<=m_lpAccounts.GetCount();i++)
{
lpAccount = m_lpAccounts.GetItem(COleVareant((long)i));
m_cboAccounts.AddString(lpAccount.GetCode());
}
```

계좌 콤보박스에서 계좌 선택시 계좌의 내용을 보여 주여야 하므로 이에 대한 이벤트 핸들러를 작성 합니다.

이벤트 핸들러에서는 선택 계좌 코드를 이용 IGxAccounts 인터페이스에서 해당 IGxAccount를 찾아 멤버 변수에 저장하고 각 프로퍼티를 호출하여 화면을 갱신합니다.

MFC GOM Event 수신

계좌 이벤트를 수신받기 위해서는 Event Sink Class를 작성해서 서버 객체에 Advise() 해야 합니다.

그리고 서버의 객체나 인터페이스의 GUID가 필요합니다. 이는 직접 GUID를 GOM 도움말을 찾아서 GUID를 직접 설정하는 방법도 있지만 저희나 사용자분들이 idl 파일을 컴파일하여 만들어진 p2_i.c 파일을 include하여 사용하면 됩니다. p2_i.c 파일을 보면 서버의 각종 GUID 들이 선언되어 있는 것을 볼 수 있습니다.

```
const IID LIBID_P2 = { 0x746AB4B8, 0x598E, 0x4B9B, { 0xAB,0x69,0x1F,0x2F,0xE0,0x86,0xO1,0x50 } };
const IID IID_IGxServer = { 0x3FD18384, 0x29E7, 0x4D5A, { 0xAF,0x7D,0x24,0x4A,0x10,0xCA,0x29,0xC4 } };
const CLSID CLSID_GxServer = { 0x9D317706, 0x9BED, 0x4699,
{ 0x8F,0x30,0x78,0x1D,0xBC,0xBD,0x27,0xA8 } };
const IID IID_IGxServerInfo = { 0x86356355, 0xB0F7, 0x4922, { 0x97,0x94,0x09,0xE5,0x37,0x75,0xE2,0xC7 } };
const IID DIID_IGxServerInfoEvents = { 0xA0DB5BA1, 0x7F32, 0x464B,
{ 0x9F,0x0F,0x0B,0x16,0x0E,0xEF,0xAC,0xB3 } };
```

New Class를 통해 새로운 Class 를 작성합니다. 클래스 이름은 CEventSink 로 하겠습니다. 선언과 구현은 EventSink.h 에서 하므로 EventSink.cpp는 프로젝트에서 삭제 합니다.

CEventSink에 IDispatch를 상속케 합니다.

CEventSink에 Constructor, Destructor, IUnknown의 AddRef, Release, QueryInterface IDispatch의 GetTypeInfoCount, GetTypeInfo, GetIDsOfNames, Invoke를 구현해야 합니다. 각 Event Sink마다 거의 구현이 비슷하기 때문에 예제의 구현을 복사하면 됩니다. 그리고 약간의 수정이 필요 합니다.

우선 QueryInterface의 IsEqualGUID를 사용하여 GUID 비교하는 부분이 나타납니다. IDispatch, IUnknown 은 그대로 뒤고 세번째에 자신이 이벤트를 받고자 하는 객체의 GUID 비교부분을 추가 합니다.

그 다음 수정을 하는 부분은 Invoke 함수인데 이것은 이벤트가 발생할 때 호출되는 함수 입니다. 그러므로 이 부분에서 이벤트 발생시의 클라이언트 내 작업을 작성하여야 합니다.

예제에서는 이 부분에서 Dispatch ID가 50 일 경우(OnAccountUpdated) 메인윈도우에 특정 윈도우 메시지를 전달하는 부분을 추가 하였습니다.

다음은 작성한 EventSink 클래스를 인스턴스화하여 서버 객체에 Advise()하는 단계 입니다.

이벤트 요청 시점은 폼이 초기화 할 때 이벤트 종료 요청 시점은 폼이 종료 될 때로 하겠습니다.

우선 CEventSink를 CGxAccountsDlg 의 멤버 변수로 추가하여 인스턴스화 합니다.

OnInitDialog 에 앞 단계에서 서버 Dispatch 를 생성한 부분을 찾습니다. 이 부분뒤로 다음과 같은 코드를 작성 합니다. 우선 우리가 이벤트를 요청할 객체에 QueryInterface()를 사용하여 IConnectionPointContainer 인터페이스를 찾습니다. 찾은 IConnectionPointContainer에 FindConnectionPoint()를 사용하여 IGxTradeStoreEvents에 해당하는 IConnectionPoint 을 찾고 멤버 변수로 저장합니다. 찾은 IConnectionPoint에 Advise()를 호출합니다. Advise()호출시 우리가 작성한 Event Sink 객체의 포인터를 파라미터로 추가 해야 합니다. 그리고 Cookie값을 Event Sink 객체와 같이 입력 해야 하는데 이 때 얻은 쿠키는 멤버 변수에 꼭 저장하여 후에 UnAdvise()할 때 사용해야 합니다.

```
// -- Event 요청 작업
IConnectionPointContainer *lpCpc = NULL;
HRESULT hr;

// -- 서버 Object의 IConnectionPointContainer 인터페이스를 찾음.
hr = m_lpAccounts.m_lpDispatch->QueryInterface(IID_IConnectionPointContainer,(void **)&lpCpc);

if FAILED(hr)
{
AfxMessageBox("IConnectionPointContainer를 찾지 못했습니다.");
EndDialog(-1);
}

// -- IConnectionPoint 인터페이스를 찾음.
hr = lpCpc->FindConnectionPoint(DIID_IGxAccountEvents, &m_lpCp);

if FAILED(hr)
{
AfxMessageBox("FindConnectionPoint 에 실패 하였습니다.");
EndDialog(-1);
}

// -- 이벤트 요청
hr = m_lpCp->Advise(&m_Sink, &m_dwCookie);
```

프로그램 종료 때의 이벤트 종료 요청은 전 단계에서 저장한 IConnectionPoint에 쿠키를 사용하여 UnAdvise()를 호출합니다.

C++, MFC 등에서는 안정적인 서버, 클라이언트를 위해 UnAdvise()를 하는 것이 좋습니다.

Event Sink를 이용하여 Advise()를 하면 서버에서 이벤트 발생시 CEventSink 객체의 Invoke() 메소드가 호출 됩니다. 앞서 우리는 Invoke() 함수 호출 시 윈도우 메시지를 메인윈도우에 발생 시켰습니다. 예제에서는 이 메시지를 수신하는 WindowProc에서 계좌의 값들을 갱신 시키는 작업을 합니다.

MFC 를 이용한 GOM 프로그래밍 2

VC++을 이용하여 GOM의 종목 객체를 이용 지정 종목의 호가, 현재가 등을 보여주는 Dialog Base의 샘플을 작성합니다.

소스 : MFC GxSymbol 샘플
샘플의 소스코드와 아래 설명에서 예시하는 코드는 약간 틀릴 수가 있습니다.

프로젝트 시작

<그림>완성된 예제 화면

1. Visual Studio 를 실행합니다.(이 예제에서는 Visual Studio 6.0 를 사용하였습니다.)

2. MFC AppWizard(exe)을 이용하여 프로젝트를 새로 시작합니다.

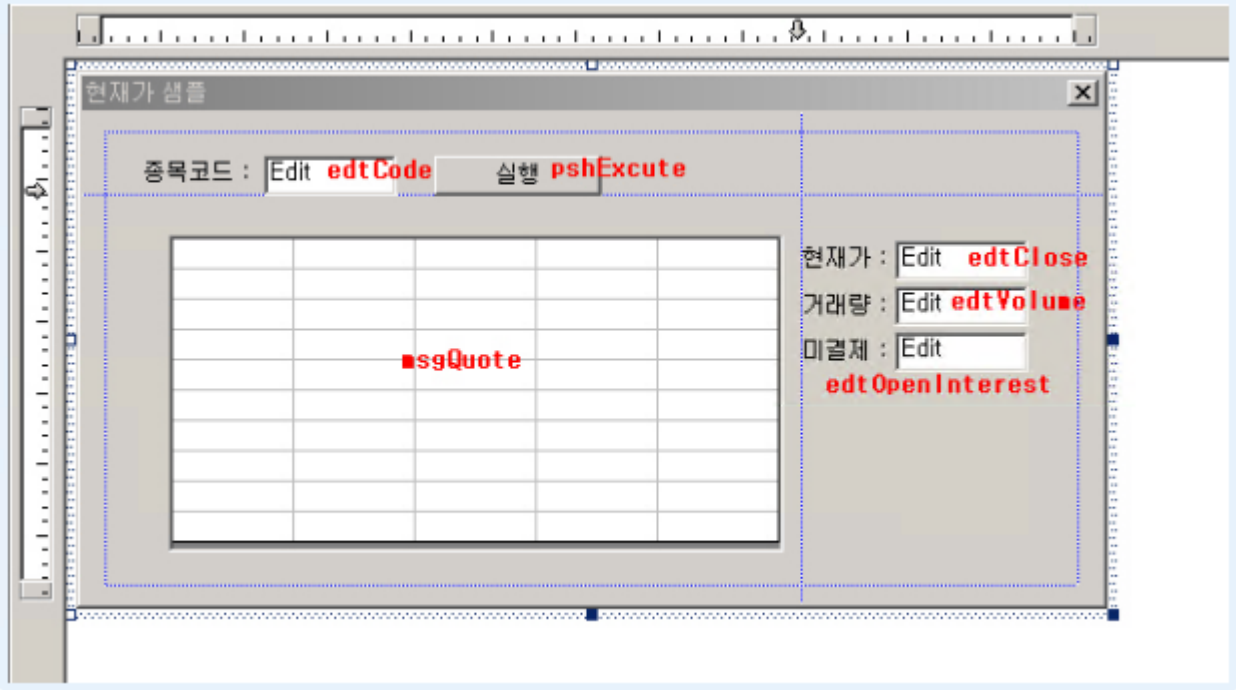
MFC AppWizard 중 'Dialog based'를 선택하고 반드시 'Automation'을 체크합니다.
프로젝트명은 'PriceSample'로 하였습니다.

기본 작업

3. 화면을 설계합니다.

종목의 호가를 보여주기 위해 MSFlexGrid를 이용합니다.
MSFlexGrid는 기본 컨트롤이 아니므로 Project>Add to Project>Components and Controls를 통하여 MSFlexGrid를 import해야 합니다.

화면 상단에 종목 코드를 입력할 수 있는 EditBox와 Push Button을 배치합니다. 그리고 종목의 호가를 보여주는 MSFlexGrid를 배치하고 화면 오른쪽에 현재가, 거래량, 미결제를 보여주는 EditBox를 배치 합니다.



<그림>화면 디자인

4.GOM 타입 라이브러리를 import 합니다.

ClassWizard 를 연 후 'Add Class'를 눌러 'From a TypeLibrary'를 선택합니다.

선택 후 타입 라이브러리 파일 선택 창이 나타나는데 GOM 자료실에서 다운받은 p2.tlb 파일을 선택합니다.

선택 후 'Confirm Classes' 창이 GOM 의 인터페이스들과 함께 나타나는데 이 들을 모두 선택한 후 'OK' 버튼을 누릅니다.

다시 나타나는 ClassWizard 창에서 'OK' 버튼을 눌러 GOM 타입 라이브러리를 import 를 완료 합니다.

import 된 GOM 인터페이스들은 p2.h 파일에 자동으로 추가가 됩니다.

5.GOM 초기화/종료 작업을 합니다.

화면이 초기화 할 때 GOM 을 초기화 하고 화면이 종료 될 때 GOM 마무리 작업을 합니다.

GOM 초기화 작업은 최상위 인터페이스인 IGxServer 를 얻어 클라이언트에 저장하는 작업이고 GOM 종료 작업은 초기화 때 얻은 IGxServer 인터페이스를 다시 돌려주는 작업 입니다.

우선 IGxServer Dialog 클래스인 CPriceSampleDlg 의 멤버로 IGxServer 타입으로 m_Server 를 선언합니다.

이 멤버에 클라이언트가 실행되는 동안 최상위 인터페이스를 보관합니다. 선언 전 IGxServer 타입이 있는 p2.h 를 import 해야 합니다.

다음은 최상위 인터페이스를 얻는 작업 입니다.

CPriceSampleDlg::OnInitDialog() 에서 m_Server.CreateDispatch 를 호출합니다.

입력 파라미터는 GOM 의 PROGID 인 P2.GxServer 를 입력하면 됩니다.

다음은 GOM 종료 작업입니다.
화면이 닫힐 때 실행되는 CPriceSampleDlg::OnCancel()에 m_Server.ReleaseDispatch 를 호출합니다.
추가로 현재 선택된 종목의 인터페이스를 저장하는 IGxServer 타입의 m_Symbol 을 m_Server 와 함께 선언합니다.

6. pshExcute 이벤트 핸들러를 작성합니다.

pshExcute 버튼을 클릭할 때 지정 종목코드에 대한 종목 인터페이스를 찾습니다.

우선 edtCode Edit Box와 대응되는 Value 형 변수를 m_strCode 라고 선언합니다.

그리고 pshExcute에 대한 Click 이벤트 핸들러를 작성합니다.

이벤트 핸들러는 다음과 같이 작성 합니다.

UpdateData를 호출하여 사용자가 입력한 종목코드를 변수에 저장하고 SymbolStore의 'Item' 속성을 이용하여 종목을 찾아 이전에 CPriceSampleDlg에 멤버로 선언한 m_Server에 저장합니다.

MFC가 자동으로 작성한 GOM 랩퍼 클래스의 속성을 호출시 실제 속성 명 앞에 읽기속성을 이용하면 'Get'이 붙고 쓰기 속성을 이용하면 'Set'이 붙습니다.
코드는 다음과 같습니다.

```
void CPriceSampleDlg::OnpshExcute()
{

…

UpdateData(TRUE);

…

if (m_Server != NULL)
{
SymbolStore = m_Server.GetSymbolStore();

if (SymbolStore != NULL)
{
m_Symbol = SymbolStore.GetItem(COleVariant(m_strCode));

if (m_Symbol == NULL)
{
AfxMessageBox("종목을 찾지 못했습니다.");
…
}
else
{
RefreshQuote();
RefreshPrice();
}

}
}
}
```

7.다음은 m_Symbol에 해당하는 종목의 호가를 화면에 표시하는 RefreshQuote, 현재가/거래량을 표시하는 RefreshPrice등의 CPriceSampleDlg의 멤버 함수를 작성합니다.

우선 상대적으로 쉬운 RefreshPrice를 작성합니다.

우선 edtClose, editVolume, editOpenInterest Edit Box 컨트롤에 해당하는 Value형 CString 타입의 변수 m_strClose, m_strVolume, m_strOpenInterest를 추가 합니다.
RefreshPrice에서는 m_Symbol의 현재가, 미결제, 거래량 속성을 통해 값을 얻고 이 들을 Value형 멤버 변수에 저장하고 화면에 동기화 합니다.

```
void CPriceSampleDlg::RefreshPrice()
{
if (m_Symbol == NULL) return;

m_strClose.Format("%5.2f", m_Symbol.GetClose());
m_strOpenInterest.Format("%d", m_Symbol.GetOpenInterest());
m_strVolume.Format("%d", m_Symbol.GetAccVolume());

UpdateData(FALSE);
}
```

다음은 RefreshQuote를 작성합니다.
GOM에서는 5단계 호가를 얻기 위해 array로 접근하는 방법과 array를 사용하지 않는 접근방법이 있습니다.
그러나 array가 빠른 접근 방법을 제공하므로 이 예제에서는 array를 사용하여 진행합니다.
GxQuote 객체에는 VQuotes, VQtys, VPrices, VCnts 등을 제공하는데 VQtys, VPrices, VCnts 등은 각각 수량, 호가, 건수 만 각각 제공하지만 VQuotes 는 앞의 3가지를 통합하여 제공합니다. 우리는 호가, 수량, 건수 모두 필요 하므로 VQuotes를 사용합니다.
호가 array에서 값을 뽑아 내는 과정은 다음과 같습니다.

```
void CPriceSampleDlg::RefreshQuote()
{

VARIANT vntQuotes;
SAFEARRAY *sa;
SAFEARRAYBOUND saDims[1]; //-- 1차원 array 지정
long lIndex, lElement;
CString strElement;

IGxQuote Quote;

if (m_Symbol == NULL) return;

Quote = m_Symbol.GetQuote();

if (Quote == NULL) return;

//-- safearray의 처음과 최종 인덱스 지정
saDims[0].lLbound = 0;
saDims[0].cElements = 34;

vntQuotes = Quote.GetVQuotes();
sa = vntQuotes.parray;
//-- 매도 호가 (0 ~ 4)
for(int i=0; i<5; i++)
{
lIndex = i;
SafeArrayGetElement(sa, &lIndex, &lElement);
strElement.Format("%5.2f", static_cast(lElement) / 100);
```

```
m_msgQuote.SetTextMatrix(5-i, 2, LPSTR(LPCTSTR(strElement)));
}
//-- 매수 호가 (5 ~ 9)
for(i=5; i<10; i++)
{
lIndex = i;
SafeArrayGetElement(sa, &lIndex, &lElement);
strElement.Format("%5.2f", static_cast(lElement) / 100);
m_msgQuote.SetTextMatrix(i+1, 2, LPSTR(LPCTSTR(strElement)));
}
....
//-- 매수 건수 (29 ~ 33)
for(i=5; i<10; i++)
{
lIndex = i + 24;
SafeArrayGetElement(sa, &lIndex, &lElement);
strElement.Format("%d", lElement);
m_msgQuote.SetTextMatrix(i+1, 4, LPSTR(LPCTSTR(strElement)));
}
}
```

RefreshPrice, RefreshQuote 가 완성되면 앞서의 pshExcute 버튼의 이벤트 핸들러에 이 두 함수를 추가합니다.

EventSink 작성

8. 다음 단계는 이벤트를 수신하기 위한 EventSink 객체를 만드는 작업입니다.

'File>New...' 메뉴를 통하여 'C/C++ Header File' 를 통하여 파일명이 EventSink.h인 새로운 Header 파일을 만듭니다.
EventSink 객체는 클라이언트 측 COM 객체이므로 IUnknown 인터페이스 의 AddRef, Release, QueryInterface, IDispatch 인터페이스의 GetTypeInfoCount, GetTypeInfo, GetIDsOfNames, Invoke 등을 구현해야 합니다. 이는 다음과 같은 예제 코드를 참조하여 작성하도록 합니다.

```
class CEventSink : public IDispatch
{
public:
ULONG refCount;

CEventSink::CEventSink() {
refCount = 1;
}
CEventSink::~CEventSink() {
}

// IUnknown methods.
virtual HRESULT __stdcall QueryInterface(
REFIID riid, void **ppvObject) {
if( IsEqualGUID(riid, IID_IDispatch) ||
IsEqualGUID(riid, IID_IUnknown) ||
IsEqualGUID(riid, DIID_IGxSymbolEvents) ||
IsEqualGUID(riid, DIID_IGxQuoteEvents)
) {
this->AddRef();
```

```
*ppvObject = this;
return S_OK;
}
*ppvObject = NULL;
return E_NOINTERFACE;
}

virtual ULONG _stdcall AddRef(void) {
return ++refCount;
}

virtual ULONG _stdcall Release(void) {
if(--refCount <= 0) {
//Delete this;
return 0;
}
return refCount;
}

// IDispatch methods.
virtual HRESULT _stdcall GetTypeInfoCount(UINT *pctinfo) {
if(pctinfo) *pctinfo = 0;
return E_NOTIMPL;
}

virtual HRESULT _stdcall GetTypeInfo(
UINT iTInfo, LCID lcid, ITypeInfo **ppTInfo) {
return E_NOTIMPL;
}

virtual HRESULT _stdcall GetIDsOfNames(
REFIID riid, LPOLESTR *rgszNames, UINT cNames, LCID lcid,
DISPID *rgDispId) {
return E_NOTIMPL;
}

virtual HRESULT _stdcall Invoke(
DISPID dispIdMember, REFIID riid, LCID lcid, WORD wFlags,
DISPPARAMS *pDispParams, VARIANT *pVarResult,
EXCEPINFO *pExcepInfo, UINT *puArgErr)
{
//--
}
```

위의 EventSink 객체 코드는 다른 GOM 객체들을 사용할 때 Copy를 해서 사용해도 됩니다.
단 QueryInterface의 경우는 서버에서 클라이언트가 작성한 이 EventSink 객체가 어떤 GOM Event를 취하는지를 알기 위해 QueryInterface를 호출하므로 EventSink 객체의 용도에 따라 IsEqualGUID(...) 부분을 추가하거나 삭제하시면 됩니다. 그리고 Invoke(...)의 경우 서버가 이벤트를 발생시키면 클라이언트의 EventSink 객체의 Invoke(...)가 호출 되므로 이벤트 시 클라이언트 동작은 Invoke(...)에 추가 하면 됩니다.

이 예제에서는 종목 현재가, 종목 호가 이벤트를 수신하기 위한 EventSink 이므로 IsEqualGUID(...)에 IGxSymbolEvents, IGxSymbolQuoteEvents를 추가 하였고 이를 위해 IGxSymbolEvent등을 정의해 놓은 자료실의 P2_i.c를 다운로드 하여 이를 include 하였습니다.
그리고 EventSink 가 호가 이벤트를 받는 것인지 현재가 이벤트를 받는 것인지를 표시하기 위해 CEventSink에 bool형의 m_bPriceReceived 멤버를 추가 하였습니다.

그래서 Invoke(...)에서 m_bPriceReceived 가 True이면 사용자 정의 메시지인 WM_PRICE_CHANGED 이벤트를

CPriceSampleDlg에 전달하고 False 이면 마찬가지로 사용자 메시지인 WM_QUOTE_CHANGED 이벤트를 CPriceSampleDlg에 전달합니다.

추가로 CPriceSampleDlg의 윈도우 메시지를 받는 WindowProc에 WM_PRICE_CHANGED 이벤트를 수신하면 RefreshPrice를 WM_QUOTE_CHANGED 이벤트를 수신하려면 RefreshQuote를 실행하도록 작성합니다. 이 때 CPriceSampleDlg 에 EventSink.h 를 include 하도록 합니다.

```
LRESULT CPriceSampleDlg::WindowProc(UINT message, WPARAM wParam, LPARAM lParam)
{
// TODO: Add your specialized code here and/or call the base class
switch (message)
{
case WM_PRICE_CHANGED :
RefreshPrice();
break;

case WM_QUOTE_CHANGED :
RefreshQuote();
break;
}

return CDialog::WindowProc(message, wParam, lParam);
}
```

9. 다음 작업은 Event 요청 및 해지 작업입니다. 앞 단계에서 작성한 EventSink 객체는 작성만 상태이지 아직 사용하지 않는 상태 입니다. 이제 EventSink를 통하여 이벤트를 구독, 취소 하는 작업을 진행합니다. CPriceSampleDlg 에 멤버로 CEventSink 타입으로 m_PriceSink, m_QuoteSink 를 선언합니다. 그리고 IConnectionPoint 포인터 타입으로 m_lpQuoteCP, m_lpPriceCP를 선언합니다. 마지막으로 DWORD 타입으로 m_dwQuoteCookie, m_dwPriceCookie 를 선언합니다. m_PriceSink, m_QuoteSinks는 이벤트를 수신하는 EventSink이고 m_lpQuoteCP등은 서버 객체에 이벤트 구독, 해지를 위한 인터페이스인 IConnectionPoint를 저장하는 변수 입니다. 마지막으로 이벤트 해지를 위해 사용될 값을 저장하는 변수가 m_dwQuoteCookie등 입니다. CPriceSampleDlg::OnInitDialog()로 돌아가서 m_PriceSink, mQuoteSink 등에 m_bPriceReceived 멤버를 용도에 맞게 설정합니다. 멤버 변수 선언 후 종목 선택 시 마다 이벤트 구독, 해지를 해야 하므로 pshExcute 버튼 이벤트 핸들러로 돌아가서 RefreshPrice 이 후에 이벤트 구독을 하는 다음과 같은 코드를 작성 합니다.

```
//-- 현재가 이벤트 구독
hr = m_Symbol.m_lpDispatch->QueryInterface(IID_IConnectionPointContainer, (void **)&lpCPC);

if FAILED(hr)
{
AfxMessageBox("IConnectionPointContainer를 찾지 못했습니다.");
return;
}

hr = lpCPC->FindConnectionPoint(DIID_IGxSymbolEvents, &m_lpPriceCP);

if FAILED(hr)
{
AfxMessageBox("IConnectionPoint를 찾지 못했습니다");
return;
}

hr = m_lpPriceCP->Advise(&m_PriceSink, &m_dwPriceCookie);

if FAILED(hr)
{
```

```
AfxMessageBox("Advise에 실패 하였습니다");
return;
}

Quote = m_Symbol.GetQuote();

if (Quote == NULL)
{
AfxMessageBox("서버에 문제가 발생하였습니다");
return;
}

//-- 호가 이벤트 구독
hr = Quote.m_lpDispatch->QueryInterface(IID_IConnectionPointContainer, (void **)&lpCPC);

if FAILED(hr)
{
AfxMessageBox("IConnectionPointContainer를 찾지 못했습니다.");
return;
}

hr = lpCPC->FindConnectionPoint(DIID_IGxQuoteEvents, &m_lpQuoteCP);

if FAILED(hr)
{
AfxMessageBox("IConnectionPoint를 찾지 못했습니다");
return;
}

hr = m_lpQuoteCP->Advise(&m_QuoteSink, &m_dwQuoteCookie);

if FAILED(hr)
{
AfxMessageBox("Advise에 실패 하였습니다");
return;
}
```

이벤트 구독 작업을 완료 했으면 다음은 이벤트 구독 작업 해지를 진행합니다. 이벤트 해지 작업 부분의 코드는 다음과 같습니다.

```
try
{
if (m_lpPriceCP != NULL)
{
m_lpPriceCP->Unadvise(m_dwPriceCookie);
m_lpPriceCP->Release();
}

if (m_lpQuoteCP != NULL)
{
m_lpQuoteCP->Unadvise(m_dwQuoteCookie);
m_lpQuoteCP->Release();
}
}
catch(CException e)
```

```
{
}
```

이 코드는 pshExcute 이벤트 핸들러의 이벤트 구독 이전에 추가 되어야 하고 프로그램의 종료시에도 이 작업이 필요하므로 CPriceSampleDlg::OnCancel()에도 이 코드를 추가합니다.

MFC GOM 프로그래밍

우선 프로젝트 시작 시 Automation 옵션을 꼭 체크 합니다. 체크 할 시 Automation 초기화 작업을 자동으로 해 주기 때문입니다. 그리고 ClassWizard 를 통해 Add Class > From a TypeLibray 에 통해 tlb 파일을 import 합니다. 그러면 자동으로 VC는 자동으로 GOM wrapper 를 만들어 줍니다. 사용자는 이 wrapper에 선언된 메소드를 호출하여 사용하시면 됩니다.

MFC GOM 객체 사용

ClassWizard를 통하여 wrapper를 생성 후 최상위 인터페이스를 멤버 변수로 선언하고 이에 대한 Dispatch를 생성해야 합니다. CreateDispatch("P2.GxServer")를 통해 Dispatch 를 생성 후 하위 객체들을 wrapper를 통해 사용합니다.

MFC GOM Event 수신

사용자는 Event Sink 객체를 직접 생성해야 합니다. Event Sink Class 에는 IUnknown, IDispatch를 구현합니다. 사용 목적에 따라 IDispatch의 Invoke 메소드를 작성 합니다. 서버에서 Event를 발생시키면 클라이언트 쪽에서 이 Invoke 함수가 호출되기 때문 입니다. Event Sink 작성 후 Event Sink을 생성 시키고 IConnectionPointContainer, IConnectionPoint를 이용 서버 객체의 ConnectionPoint를 찾아 Advise을 합니다. Event Sink는 Advise의 파라미터로 입력되어야 합니다. 그리고 Event를 수신을 중지 하려면 앞서 Advise한 ConnectionPoint에 UnAdvise를 합니다. 원활한 프로그램밍을 위해 Advise한 후 종료 시 꼭 UnAdvise 하는 습관을 갖도록 해야 합니다.

예제 파일

경고 : 이 예제는 GOM 사용법을 쉽게 익히기 위해 제공 하는 것으로, 이 예제를 거래에 이용 하는 중 에러나 다른 이유로 발생하는 피해는 당사에서 책임을 지지 않습니다.
2004. 05. 18 (주)델타 익스체인지

C++, MFC 예제

GxSymbol 예제<제작 버전:1.2>
GxAccount 예제 <제작 버전:1.0>
Greeks 예제 <제작 버전:1.1>
GxChartStore 예제 <제작 버전:1.23>
MFC를 이용하지 않고 C++ 라이브러리만으로 KOSPI200 현재가를 조회하는 간단한 콘솔 프로그램입니다.<제작 버전:1.1>