TRADING TECHNOLOGIES®

X_TRADER API Class Reference

Version 7.17

Document Version 7.17 DV1 3/9/16

Trading Technologies International, Inc.

Legal Notices

This document and all related computer programs, example programs, and all TT source code are the exclusive property of Trading Technologies International, Inc. (“TT”), and are protected by licensing agreements, copyright law and international treaties. Unauthorized possession, reproduction, duplication, or dissemination of this document, or any portion of it, is illegal and may result in severe civil and criminal penalties.

Unauthorized reproduction of any TT software or proprietary information may result in severe civil and criminal penalties, and will be prosecuted to the maximum extent possible under the law.

Information in this document is subject to change without notice. Companies, names, and data used in examples herein are fictitious unless otherwise noted. No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of TT.

All trademarks displayed in this document are subject to the trademark rights of TT, or are used under agreement by TT. These trademarks include, but are not limited to, service brand names, slogans and logos and emblems including but not limited to: Trading Technologies®, the Trading Technologies Logo, TT™, X_TRADER®, X_RISK®, MD Trader®, Autospreader®, TT_TRADER®, Autotrader™, ADL™ (Algo Design Lab), TT Trainer™, TTNET™, X_STUDY™, TT CVD®. All other referenced companies, individuals and trademarks retain their rights. All trademarks are the property of their respective owners. The unauthorized use of any trademark displayed in this document is strictly prohibited.

Copyright © 2004-2012 **Trading Technologies International, Inc.**
All trademarks are the property of their respective owners.

About this Manual 13

Chapter 1: Introduction 15

X_TRADER API overview 16
Namespaces overview 18
X_TRADER API samples 19

Chapter 2: XTAPI Namespace 21

X_TRADER API data flow for XTAPI environments 22
XTAPI classes 25

Chapter 3: InstrCollection Class 29

InstrCollection class overview 30
Property: Count 31
Property: Item 32

Chapter 4: ITTCustomerProfile Class 33

ITTCustomerProfile class overview 34
Property: Acct 35
Property: AcctType 36
Property: Customer 37
Property: FFT2 38
Property: FFT3 39
Property: GiveUp 40
Property: Market 41
Property: MaxOrderQty 42
Property: OpenClose 43
Property: OrderRestr 44
Property: OrderType 45
Property: ProdType 46
Property: Product 47
Property: TIF 48
Property: UseMaxQty 49

Chapter 5: ITTPriceEntry Class 51

ITTPriceEntry class overview 52
Property: Name 53
Property: PriceID 54
Property: toString 55
Property: toDouble 56
Property: toTicks 57

Chapter 6: ITTPriceList Class 59

ITTPriceList class overview 61
Property: Count 63
Property: Item 64

Chapter 8: ITTPriceUpdate Class 65

ITTPriceUpdate class overview 66
Property: Count 67
Property: Instrument 68

Property: Item 70
Property: PriceTimeStamp 71
Property: TimeStamp 72

Chapter 9: OrderProfileCol Class 73

OrderProfileCol class overview 74
Property: Count 75
Property: Item 76

Chapter 10: OrderSetCol Class 77

OrderSetCol class overview 78
Property: Count 79
Property: Item 80

Chapter 11: TTDropHandler Class 81

TTDropHandler class overview 82
Property: Count 83
Property: DropEffect 84
Property: DropKeyState 86
Property: DropX 88
Property: DropY 89
Property: Item 90
Method: RegisterDropWindow 91
Method: Reset 92
Event: OnNotifyDrop 93

Chapter 12: TTFillObj Class 95

TTFillObj class overview 96
Property: Get 97
Property: Instrument 106
Property: IsValidGet 107
Property: ReadProperties 109
Method: Confirm 110

Chapter 13: TTGate Class 111

TTGate Class class overview 113
Property: CustomerDefaults 117
Property: Customers 118
Property: DebugAppInfo 119
Property: DebugLogLevel 120
Property: EnableOrderSend 123
Property: Get 124
Property: InstrData 127
Property: Instruments 128
Property: IsValidGet 129
Property: OrderProfiles 131
Property: OrderSetData 132
Property: OrderSets 133
Property: RapidFillDelivery 134
Property: ReadProperties 136
Method: AuditXTServices 137

Method: DestroyDownloadedFills . . . 138
Method: DisableExtendedPrices . . . 139
Method: EnableAutoSessionRollover . . . 140
Method: EnableServerMode . . . 141
Method: GetExchangeRate . . . 143
Method: GetPrimaryCurrency . . . 144
Method: OpenExchangeFills . . . 145
Method: OpenExchangeOrders . . . 146
Method: OpenExchangePrices . . . 147
Method: SetExchangeLogon . . . 148
Method: SetExchangeLogonEx . . . 150
Method: SetUniversalLogin . . . 152
Method: StartNewSession . . . 153
Method: StartXTrader . . . 154
Method: XTAPITerminate . . . 155
Event: OnExchangeMessage . . . 156
Event: OnExchangeStateUpdate . . . 157
Event: OnLoginFailed . . . 159
Event: OnSessionRollMessage . . . 160
Event: OnStatusUpdate . . . 161

Chapter 14: TTHotKeyNotify Class . . . 163

TTHotKeyNotify class overview . . . 164
Method: AddHotKey . . . 165
Method: DelHotKey . . . 167
Event: OnHotKey . . . 168

Chapter 15: TTInstrNotify Class . . . 169

TTInstrNotify class overview . . . 170
Property: DebugLogLevel . . . 173
Property: DeliverAllPriceUpdates . . . 176
Property: EnableDepthUpdates . . . 178
Property: EnableOrderSetUpdates . . . 181
Property: EnablePriceUpdates . . . 182
Property: InstrEnableUpdates . . . 185
Property: InstrFilter . . . 187
Property: Instrument . . . 190
Property: IsValidUpdateFilter . . . 193
Property: PriceList . . . 195
Property: UpdateFilter . . . 197
Method: AttachInstrument . . . 200
Method: AttachOrderSet . . . 202
Method: DetachInstrument . . . 204
Method: DetachOrderSet . . . 206
Event: OnInstrumentExpiryChanged . . . 207
Event: OnNotifyDepthData . . . 208
Event: OnNotifyFound . . . 210
Event: OnNotifyNotFound . . . 211
Event: OnNotifyUpdate . . . 212
Event: OnOrderSetUpdate . . . 214
Event: OnPriceListUpdate . . . 215

Chapter 16: TTInstrObj Class 217

TTInstrObj class overview 218
Property: Alias 220
Property: CalculateTTImplieds 221
Property: Contract 222
Property: ContractFormatted 225
Property: CreateNotifyObject 226
Property: CreateOrderSelector 227
Property: DebugLogLevel 229
Property: Exchange 232
Property: GatewayType 233
Property: Get 234
Property: IsTradablePrice 254
Property: IsValidGet 256
Property: IsValidUpdateFilter 258
Property: IsVolatile 260
Property: MergeImpliedsIntoDirect 262
Property: OrderSet 264
Property: ProdType 265
Property: Product 266
Property: ReadProperties 267
Property: RoundPrice 268
Property: SeriesKey 270
Property: TickPrice 271
Property: TickPriceEx 273
Method: Open 275
Method: Retrieve 277

Chapter 17: TTOrderObj Class 279

TTOrderObj class overview 280
Property: CreateOrderProfile 281
Property: GetChanged 283
Property: Get 285
Property: GetIsChanged 296
Property: HasNextOrder 297
Property: Instrument 299
Property: IsNull 300
Property: IsValidGet 301
Property: NextOrder 303
Property: ReadProperties 305
Method: DeleteOrder 306
Method: HoldOrder 307

Chapter 18: TTOrderProfile Class 309

TTOrderProfile class overview 310
Property: Alias 312
Property: Customer 313
Property: CustomerDefaults 315
Property: Customers 316
Property: Get 317
Property: GetLast 328

Property: GetLastOrder 330
Property: Instrument 331
Property: IsValidGet 332
Property: ReadProperties 334
Property: RoundOption 335
Property: WriteProperties 336
Method: Set 337

Chapter 19: TTOrderSelector Class 343

TTOrderSelector class overview 345
Property: AllMatchesRequired 347
Property: AllowAnyMatches 348
Property: TestMatch 349
Method: AddSelector 350
Method: AddTest 351
Method: AddTestEx 353
Method: InvertTest 354
Method: Reset 356

Chapter 20: TTOrderSet Class 357

TTOrderSet class overview 359
Property: Alias 362
Property: AvgOpenPriceMode 363
Property: Count 365
Property: CreateOrderSelector 366
Property: DebugLogLevel 368
Property: DeleteContracts 371
Property: DeleteOrders 373
Property: EnableFillCaching 376
Property: EnableOrderAutoDelete 377
Property: EnableOrderFillData 378
Property: EnableOrderRejectData 379
Property: EnableOrderSend 380
Property: EnableOrderSetUpdates 381
Property: EnableOrderUpdateData 382
Property: IsValidGet 383
Property: Get 385
Property: IsVolatile 389
Property: Item 390
Property: NextOrderTracker 392
Property: OrderSelector 394
Property: OrderStatusNotifyMode 397
Property: ReadProperties 398
Property: SendOrder 400
Property: SiteKeyLookup 402
Property: SiteKeyLookupEx 403
Property: WriteProperties 404
Method: Close 405
Method: HoldOrder 406
Method: HoldOrders 408
Method: Open 410
Method: Set 411

Method: SetValidExchanges 414
Method: UpdateOrder 416
Method: UpdateXTOrder 418
Event: OnOrderActionRejected 421
Event: OnOrderDeleted 423
Event: OnOrderFillBlockEnd 425
Event: OnOrderFillBlockStart 426
Event: OnOrderFillData 427
Event: OnOrderFilled 429
Event: OnOrderHeld 431
Event: OnOrderInstrumentChanged 432
Event: OnOrderRejected 433
Event: OnOrderSetUpdate 434
Event: OnOrderSubmitted 435
Event: OnOrderUpdated 437

Chapter 21: TTOrderTrackerObj Class 439

TTOrderTrackerObj class overview 440
Property: GetChanged 442
Property: GetIsChanged 443
Property: HasNewOrder 445
Property: HasOldOrder 446
Property: NewOrder 447
Property: OldOrder 449

Chapter 22: MEAPI Namespace 451

About the Market Explorer API 452
MEAPI classes 454

Chapter 23: ITTContractCollection Class 457

ITTContractCollection class overview 458
Property: Count 459
Property: Item 460

Chapter 24: ITTContractInfo Class 461

ITTContractInfo class overview 462
Property: CallPut 463
Property: ComboCode 464
Property: Currency 465
Property: Decimals 466
Property: ExpirationDate 467
Property: LegInfo 468
Property: Name 469
Property: Precision 470
Property: PriceDisplayFormat 471
Property: PriceDisplayType 472
Property: Product 473
Property: SeriesKey 474
Property: SeriesName 475
Property: Strike 476
Property: TickPriceDenominator 477
Property: TickPriceNumerator 478

Property: TickSize 479
Method: CopyToClipboard 480
Method: ToXML 481

Chapter 25: ITTGateway Class 483

ITTGateway class overview 484
Property: AvailableServices 485
Property: ID 486
Property: IsAvailable 487
Property: IsLoggedIn 489
Property: MarketName 490
Property: Name 491
Property: Products 492
Method: RegisterEventHandler 493
Method: RequestProducts 495
Method: RequestProductTypes 497
Method: UnregisterEventHandler 499

Chapter 26: ITTGatewayEvents Class 501

ITTGatewayEvents class overview 502
Event: OnGatewayAttributes 503
Event: OnNewContract 505
Event: OnNewProduct 506
Event: OnProducts 507
Event: OnProductTypes 509
Event: OnServerStatus 510

Chapter 27: ITTMarketExplorerEvents Class 511

ITTMarketExplorerEvents class overview 512
Event: OnGateway 513

Chapter 28: ITTProduct Class 515

ITTProduct class overview 516
Property: Alias 517
Property: Description 518
Property: Gateway 519
Property: Name 520
Property: Type 521
Property: TypeAsString 522
Method: RegisterEventHandler 523
Method: RequestContracts 524
Method: UnregisterEventHandler 525

Chapter 29: ITTProductCollection Class 527

ITTProductCollection class overview 528
Property: Count 529
Property: Item 530

Chapter 30: ITTProductEvents Class 531

ITTProductEvents class overview 532
Event: OnContracts 533

Event: OnNewContract 535
Event: OnRequestContractsTimeout 537

Chapter 31: TTMarketExplorer Class 539

TTMarketExplorer class overview 540
Method: MEAPITerminate 541
Method: RegisterEventHandler 542
Method: RequestGateways 544
Method: UnregisterEventHandler 545

Appendix A: XTAPI Enumerations and Codes. 547

Constants and enumerations 548
Codes, types, and restrictions 552

Appendix B: MEAPI Types and Enumerations 559

Constants and Enumerations 560

Appendix C: Order Rejections Not Exposed to Clients 567

About this Manual

Overview

This manual describes the XTAPI (X_TRADER API) classes and functionality that programmers can use to create custom trading applications in a TT trading environment. It explains the purpose of each class and describes all of the properties, methods, and events associated with each class. Also, it provides code samples that demonstrate how to call them.

Assumptions

This guide assumes the following:

- The target audience is an experienced API developer.
- The reader is familiar with Trading Technologies software.
- The reader understands X_TRADER®.

Conventions

This guide uses a set of terms, symbols, and typographic conventions to categorize specific information. Familiarity with these conventions helps you use this documentation more effectively.

Convention	Use
Courier New	Sample code, arguments, properties, arguments, methods, events, or anything that you must type exactly as it appears. When found in the body of the manual (i.e., paragraphs), this text is also in bold. Text in Courier New is also 1 point size smaller than the surrounding text.
Italic	Variables, including variable text. Variable text is used when dialog boxes or their components are not labeled. Variable text is also used for labels that change dynamically based on their current context.
[argument]	Optional argument.
pXXXX	Pointer to the object data.

Related documentation

This manual is part of a comprehensive Trading Technologies document set. For additional manuals and documentation on other TT™ topics, contact your TT™ sales representative.

1 Introduction

Chapter overview

This chapter provides a brief overview of the X_TRADER API and its namespaces.

In this chapter

Section	
X_TRADER API overview	16
Namespaces overview	18
X_TRADER API samples	19

X_TRADER API overview

About the XTAPI

The X_TRADER® API (XTAPI) is an application programming interface that allows you to create custom applications to supplement an X_TRADER application, which can subscribe for prices (including depth) and manipulate orders. As the order book is shared by X_TRADER® and XTAPI, both can access and manipulate any orders in it. Consequently, the potential uses of the XTAPI include:

- Automated trading based on proprietary strategies
- Custom order entry screens
- Synthetic order types
- Order and fill monitoring
- Filtered position and P&L reporting
- Data extraction for back office
- Status monitoring of TT Gateways and exchanges
- Custom charts and graphs

COM compliance

The XTAPI is fully COM (Component Object Model) compliance and uses COM for its interface and data access processes. As a result, the XTAPI is compatible with any development language that supports COM including (but not limited to) Visual Basic, Visual C++, Visual C#, and Microsoft Excel. The XTAPI is also compatible with any program written using the .NET environment.

Backward compatibility

The XTAPI COM interface is fully backwards compatible. Therefore, applications compiled with one version of the XTAPI do not need to be recompiled when a subsequent version is installed.

Note: You cannot run 32-bit XTAPI applications with the 64-bit version of XTAPI. If you want to use existing 32-bit XTAPI application, you must either install the 32-bit XTAPI or recompile the existing applications with the 64-bit XTAPI.

Because the X_TRADER plug-ins, Back Office Bridge, and Trade Tracker applications are 32-bit XTAPI applications, you cannot use these applications if you install the 64-bit version of the XTAPI.

Prerequisites

To develop custom applications with the XTAPI, you must have the following prerequisites:

- The XTAPI must be installed.
- You must have an XTAPI Pro license.
- X_TRADER® Pro must be running on the same machine as the XTAPI application with "Pro" mode enabled.
- You must have application development environment that supports the Microsoft COM interface.
- You must register with TT Developer Support through the Development Support Web site (http://developer.tradingtechnologies.com).

Installed files

The XTAPI setup package installs the XTAPI files and libraries in <xtrader_root_dir>\xtapi\bin.

Note: During the installation process, XTAPI places two Microsoft Excel add-in files (**XTAPI.xla** and **XTAPI2K2-RTD.xla**) that include various helper functions. If you want to use these add-ins, you must register them with Excel.

For more information about registering these add-ins, refer to http://office.microsoft.com/en-us/excel-help/add-or-remove-add-ins-HP010342658.aspx.

Namespaces overview

About namespaces

The X_TRADER API comprises the following namespaces that focus on different areas of the trading process:

- XTAPI namespace
- MEAPI namespace

You can create applications that focus on either or that combine both.

About the XTAPI namespace

The XTAPI namespace provide classes that help you manage prices, orders, and fills. Using these objects, you can place, modify, or cancel orders; keep track of fills and monitor P&L; and inject SOD records or manual fills to keep positions updated.

About the MEAPI namespace

The MEAPI namespace provides classes that help you monitor TT Gateways and to retrieve the available products and contracts in real time. Using these objects, you can connect to a TT Gateway and request all products or create search criteria to return any subset of products and contracts.

X_TRADER API samples

About the sample applications

The TT Development Technical Support team provides several sample applications that illustrate how to use XTAPI objects, events, methods, and properties. You can find information specific to each sample in the readme file for the sample program.

TT recommends that you use these samples as foundations for your own customized programs. Although they are designed to run "right of the box," TT also recommends that you compile and run the programs to ensure they compile correctly.

Where the samples are located

You can access the sample applications through the TT Developer Support Website at:

http://developer.tradingtechnologies.com

Note: You must have an approved login to download the sample code. If you do not yet have one, you can sign up for a full-access login account through the Web site.

Supported languages

The Web site provides sample application for the following programming languages:

- C#.NET
- C++.NET
- C++ 6.0
- VB.Net

Development Support Website Sample Code page

Figure 1 shows the Sample Code section of the TT Development Support Website. Because TT continues to expand the sample applications, the available applications might differ from those shown.

Figure 1. TT Development Support Website Samples

How to access the sample applications

To access the sample applications:

- In a web browser, open the **TT Development Support Website**.
- Click the appropriate **Sample Code** link.
- Save the zip file to the desired location.
- Decompress the zip file.

2 XTAPI Namespace

Chapter overview

This chapter provides a brief overview of the XTAPI namespace and its classes.

In this chapter

Section	Page
X_TRADER API data flow for XTAPI environments	22
XTAPI classes	25

X_TRADER API data flow for XTAPI environments

Data flow diagram

Figure 2 illustrates how data flows between an XTAPI client application and the other components in an TT network.

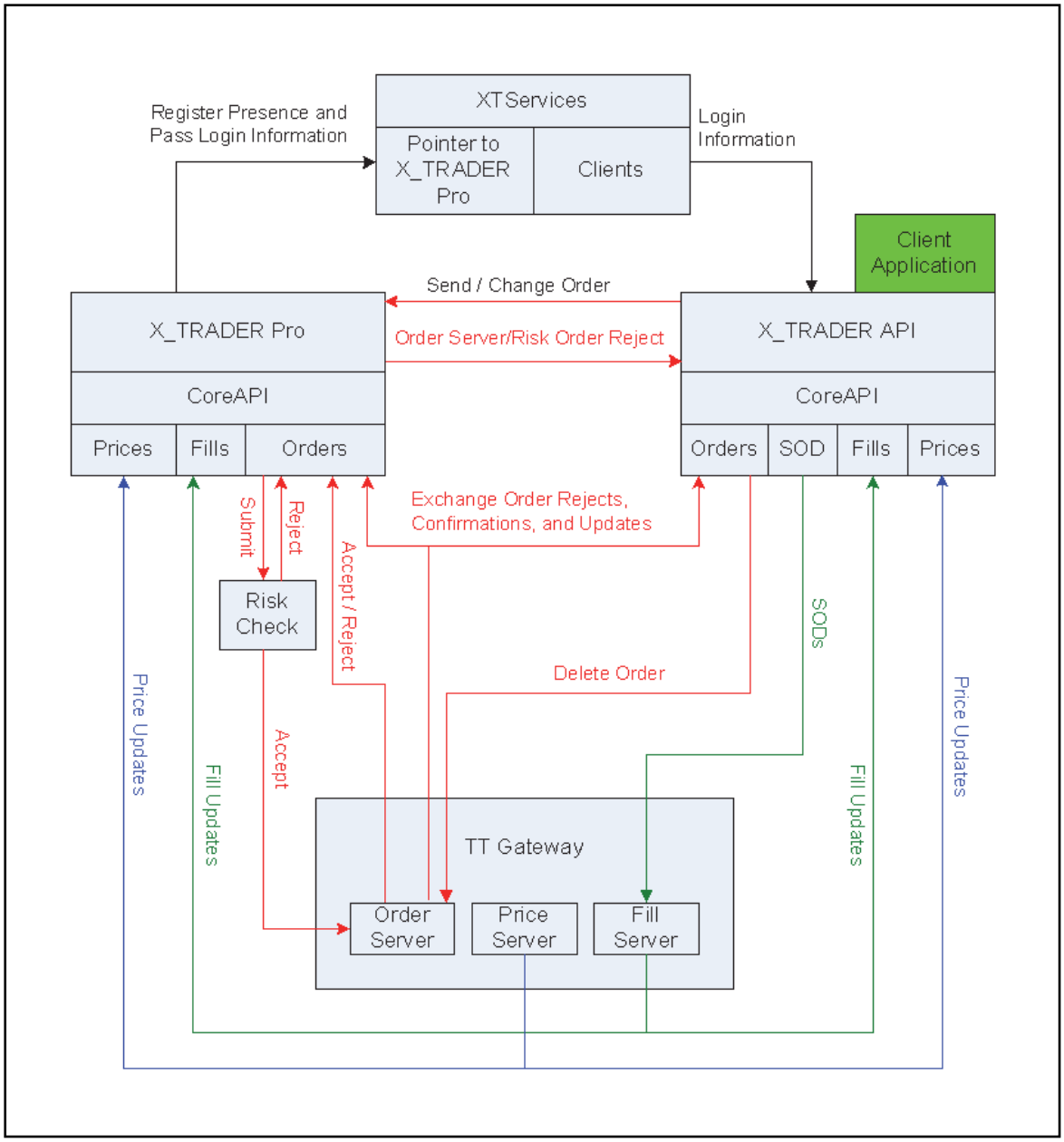


Figure 2. Data flow for XTAPI applications in an X_TRADER environment

Connection data flow

The data flow of a connection occurs as follows:

- When X_TRADER Pro starts, it registers its presence and passes its login information to XTServices.
- When an XTAPI client application starts, the XTAPI retrieves the login information from XTServices and logs into the TT Gateway(s) and exchange(s).

Note: If an X_TRADER® client application starts before X_TRADER® Pro, the X_TRADER® waits until X_TRADER® Pro starts before it proceeds with the login process.

Price subscription data flow

The data flow of a price subscription occurs as follows:

- When an XTAPI client application subscribes for prices, the XTAPI submits the request to the Price Server.
- The Price Server begins to multicast price updates to all client applications which have subscribed to the instrument.

Note: You cannot programmatically unsubscribe for price updates. Stopping the X_TRADER® client application ends a price subscription.

Order submission data flow

The data flow of a send/change order occurs as follows:

- When an XTAPI client application submits a new or change order request, the XTAPI submits the request to X_TRADER Pro
- X_TRADER Pro checks risk limits. If the order does not pass risk checking:
  - Guardian sends a reject message back to X_TRADER Pro.
  - X_TRADER sends the reject message back to the XTAPI.
  - The XTAPI sends the reject message to the XTAPI client application.
- If the order passes risk checking, XTAPI sends it to the TT Gateway. If the TT Gateway rejects the order:
  - The TT Gateway sends a reject message back to Guardian.
  - Guardian sends a reject message back to X_TRADER Pro.
  - X_TRADER sends the reject message back to the XTAPI.
  - The XTAPI sends the reject message to the XTAPI client application.
- If the TT Gateway accepts the order, it submits it to the exchange.
- All subsequent order state changes (reject from exchange, partial fill, full fill, etc.) are sent by the Order Server to both X_TRADER Pro and the XTAPI.
- The XTAPI forwards all messages to the XTAPI client application.

Note: An X_TRADER® client application is unable to perform a change or delete of an order until the order status confirmation is received from the exchange after submission.

Order deletion data flow

The data flow of a delete order occurs as follows:

- When an XTAPI client application submits a request to delete an order, the XTAPI submits the request to the Order Server.
- The Order Server submits the request directly to the exchange.
- The Order Server sends the TT Gateway and exchange acknowledgements to both X_TRADER Pro and the XTAPI.
- The XTAPI forwards all messages to the XTAPI client application.

Note: Because deleting an order does not increase your risk, the X_TRADER® submits all order delete requests directly to the Order Server. Acknowledgement of the order deletion from the exchange is sent to Guardian so your current number of working orders is properly updated.

Fill data flow

The data flow of a fill occurs as follows:

- When an XTAPI client application subscribes for fills, the XTAPI submits the request to the Fill Server.
- The Fill Server then begins to send fill updates to all client applications which have subscribed for them.

XTAPI classes

Available classes

Table 3 lists the classes available in the XTAPI.

Class	Description
InstrCollection	Stores a collection of instrument objects
ITTPriceEntry	Stores market data for a single price update
ITTPriceList	Queues uncoalesced prices
ITTPriceUpdate	Represents an instrument at its updated price information
OrderProfileCol	Stores a collection of order profile objects
OrderSetCol	Stores a collection of order set objects
TTDropHandler	Supports drag-and-drop functionality between XTAPI and an XTAPI application
TTFillObj	Stores information about fills
TTGate	Stores global XTAPI application objects and settings
TTHotKeyNotify	Supports custom hot key definitions
TTInstrNotify	Monitors price and order set updates
TTInstrObj	Stores information about a tradable instrument
TTManualFill	Manages manually-entered fills
TTOrderObj	Represents a state of an order in the market
TTOrderProfile	Submits and modifies orders
TTOrderSelector	Filters orders to include in an order set
TTOrderSet	Represents a collection of individual orders
TTOrderTracker	Stores the set of changes to an order over its lifetime
TTRiskManager	Manages access to the risk limit database for SOD publishing
TTSODObj	Represents a single start-of-day (SOD) record for a trader's instrument
TTSODSet	Stores the collection of SOD records for managed traders

Table 3. XTAPI classes

Class relationships

Figure 4 illustrates the functional representation between the different classes to help you understand how the corresponding objects interact with each other. Please note that the illustration does not imply any hierarchical structure or inheritance between the objects. The solid lines represent direct relationships, while the dashed lines represent indirect relationships. For example, a **`TTOrderObj`** object has a direct relationship with a **`TTOrderSet`** object and an indirect relationship with a **`TTInstrObj`** object

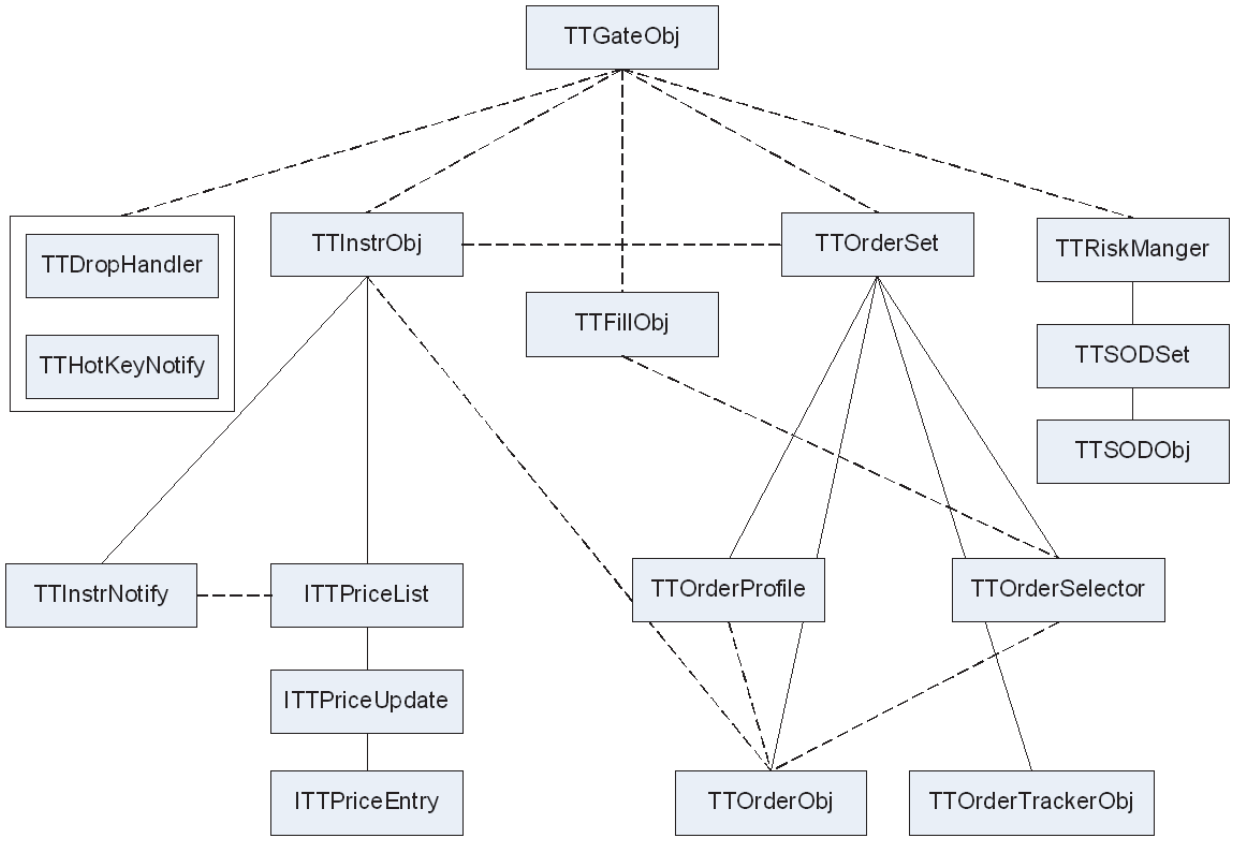


Figure 4. Class relationships

Class function types

Each class defines a set of functions you can use to create, query, or update instantiated objects and to execute code when certain events occur for an object. To help identify the basic role of the functions, the XTAPI categorizes the functions into three areas: properties, methods, and events. Table 5 describes these function types.

Function type	Description
Property	Functions that return information about an object
Method	Functions that create or modify an object and its attributes
Event	Functions that are executed when object-specific events occur

Table 5. Class function types

About Boolean values

The XTAPI uses properties that function logically as Boolean values. These properties convert all incoming parameters into expected data types. The XTAPI conforms to the following data type conversion for non-Boolean incoming values.

Incoming value	Converted value
0	False
<>0	True
Non-integer	Undefined

Thus, if an XTAPI application expects a Boolean value for one of its parameters, it interprets any non-Boolean incoming values in Boolean terms. When XTAPI

returns values for attributes marked as VARIANT_BOOL; it uses 0 for FALSE and a non-zero integer for TRUE. Your application can convert these values to the Boolean representations supported by the application programming language.

The XTAPI converts data types according to the VariantChangeType Windows API.

About array indexing

Arrays returned from XTAPI properties, methods, and arrays use a 1-based index.

About optional parameters

Some XTAPI functions, such as the **`TTOrderSet`** `DeleteOrders` property, support optional parameters. If you omit a value, XTAPI uses the default value for the parameter.

If you develop in a .NET language, however, you must supply a special value, System.Type.Missing, for optional parameters for which you do not want to supply an actual value.

Generic data type conventions used in this guide

Because different programming languages use different data types to represent similar data, this guide uses generic terms to represent the type of data represented in parameters. For example, an integer value might declared as an int, a long, etc., depending on which programming language you use. Rather than list all the different data types for common types of information, this guide uses the generic terms described in Table 6.

Term used	Type of data
Integer	32-bit integer value (i.e. int, long)
Decimal	Decimal value (i.e. float, double, real)
String	Characters (i.e. string, char *)
Array	List of values of similar or different data types (i.e. array, linked list)

Table 6. Generic data type conventions

Error handling

For .NET applications, the XTAPI throws exceptions when unexpected errors occur. This document describes the exceptions thrown for each class property or method. For non-.NET applications, the XTAPI returns an error code if an error occurs.

3 InstrCollection Class

Chapter overview

The chapter describes the **`InstrCollection`** class. It provides an overview of the class and includes detailed information about its properties.

In this chapter

Section	Page
InstrCollection class overview	30
Property:Count	31
Property:Item	32

InstrCollection class overview

Overview

The **`InstrCollection`** class contains a collection of instrument (**`TTInstrObj`**) objects. This class allows you to access instruments directly from a **`TTGate`** object, which returns the collection of aliased instruments from its `Instruments` property. You can use this class to manipulate all of the aliased instruments known to the **`TTGate`** object.

This class primarily supports the .NET foreach construct, so that you can easily loop through the available instruments. If your development environment does not support the construct, you can use this class to iterate over the instruments.

For more information about instrument aliases, refer to the **`TTInstrObj`** object's Property:Alias section on page 220.

Properties

Table 7 lists the **`InstrCollection`** properties.

Property	Purpose
Count	Returns the number of instruments in the collection
Item	Returns the instrument at the specified index or alias in a collection

Table 7. InstrCollection properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of instruments in the collection
Syntax	*instrCount* = myInstrCol.Count
Parameters	None
Return value	*instrCount* — Number of instruments in the collection Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`TTInstrObj`** objects contained in the collection.
Example	The following code fragment uses the `Count` property to return each **`TTInstrObj`** object in the collection.

```
instrCol = myGate.Instruments
'Loop through each TTInstrObj
For i = 1 To instrCol.Count
   Try
      Dim instrument As TTInstrObj
      instrument = instrCol.Item(i)

      'Get the property values
      Dim oProp() As Object = instrument.Get(propList)
      ' ... omitted code
```

Related information

Property:Item, on page 32

`OrderProfileCol` Property:Count, on page 75

`OrderSetCol` Property:Count, on page 79

Property: Item

Purpose

Returns the instrument at the specified index or alias in a collection

Syntax

`instrObj = myInstrCol.Item(index_or_alias)`

Parameters

index_or_alias	Position in the collection of the desired instrument or an alias for the instrument Data type: Integer or String

Return value

instrObj	Order profile at the specified location or matching the specified alias Data type: `TTInstrObj`

Exceptions

Exception	Meaning
The parameter is incorrect	The specified index is either out of range or not a valid instrument alias.

Description

The `Item` property returns an instrument from the **`InstrCollection`** collection. You can request an instrument either by specifying the index position or by providing an alias associated with an instrument.

Note: The **InstrCollection** collection uses a 1-based index.

Example

The following code fragment uses the `Item` property to return each **`TTInstrObj`** object in the collection.

```
instrCol = myGate.Instruments
'Loop through each TTInstrObj
For i = 1 To instrCol.Count
   Try
      Dim instrument As TTInstrObj
      instrument = instrCol.Item(i)

      'Get the property values
      Dim oProp() As Object = instrument.Get(propList)
      ' ... omitted code
End For
```

Related information

Property:Count, on page 31

`OrderProfileCol` Property:Item, on page 76

`OrderSetCol` Property:Item, on page 80

4 ITTCustomerProfile Class

Chapter overview

The chapter describes the **`ITTCustomerProfile`** class. It provides an overview of the class and includes detailed information about its properties.

In this chapter

Section	Page
ITTCustomerProfile class overview	34
Property:Acct	35
Property:AcctType	36
Property:Customer	37
Property:FFT2	38
Property:FFT3	39
Property:GiveUp	40
Property:Market	41
Property:MaxOrderQty	42
Property:OpenClose	43
Property:OrderRestr	44
Property:OrderType	45
Property:ProdType	46
Property:Product	47
Property:TIF	48
Property:UseMaxQty	49

ITTCustomerProfile class overview

Overview

The **`ITTCustomerProfile`** class contains properties that allow you to extract information from an X_TRADER customer profile.

Properties

Table 8 lists the **`ITTCustomerProfile`** properties.

Property	Description
Acct	Returns the account number for the customer profile
AcctType	Returns the account type for the customer profile
Customer	Returns the name of the customer profile
FFT2	Returns the FFT2 value for the customer profile
FFT3	Returns the FFT3 value for the customer profile
GiveUp	Returns the clearing member code for the customer profile
Market	Returns the name of the market for the customer profile
MaxOrderQty	Returns the maximum order quantity allowed for this customer profile
OpenClose	Returns the Open/Close value for this customer profile
OrderRestr	Returns the order restriction for this customer profile
OrderType	Returns the default order type for this customer profile
ProdType	Returns the default product type for this customer profile
Product	Returns the name of product for this customer profile
TIF	Returns the time-in-force for this profile
UseMaxQty	Returns whether this profile uses the maximum quantity restriction

Table 8. ITTCustomerProfile properties

Methods

None

Events

None

Property: Acct

Purpose	Returns the account number for the customer profile
Syntax	*acctNum* = myITTCustomerProfile.Acct
Parameters	None
Return value	*acctNum* — Account ID for the customer Data type: String
Exceptions	None

Description

The `Acct` property returns the account ID for the Customer Defaults entry represented by this customer profile object. The value corresponds to the value in the Account # column in the X_TRADER **Customer Defaults** window for the selected customer.

Example

The following example uses the `Acct` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Market = CME and Acct = 12345
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ACCT = "12345"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:AcctType, on page 36

TTOrderProfile Property:CustomerDefaults, on page 315

Property: AcctType

Purpose

Returns the account type for the customer profile

Syntax

`acctType = myITTCustomerProfile.AcctType`

Parameters

None

Return value

acctType	Trading account type for the customer profile. Possible values include: • A1, A2, A3 (Allocation accounts) • G1, G2, G3 (Give-up accounts) • M1, M2, M3 (Market maker accounts) • P1, P2, P3 (Principal accounts) • U1, U2, U3 (Unallocated accounts) Data type: String

Exceptions

None

Description

The `AcctType` property returns the type of trading account associate with the customer profile. The value corresponds to the value in the **Acct** column in the X_TRADER Customer Defaults window for the selected customer.

Example

The following example uses the `AcctType` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Acct = 12345 and AcctType = G1
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Acct = "12345" And
           oCustProfile.AcctType = "G1"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:Acct, on page 35

TTOrderProfile Property:CustomerDefaults, on page 315

Property: Customer

Purpose

Returns the name of the customer profile

Syntax

`custName = myITTCustomerProfile.Customer`

Parameters

None

Return value

custName	Profile name Data type: String

Exceptions

None

Description

The `Customer` property returns the name of the customer profile. The value corresponds to the value in the **Customer** column in the X_TRADER Customer Defaults window for the selected customer.

Example

The following example uses the `Customer` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Customer = jdoe
    For Each cdObj In oCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Customer = "jdoe"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

TTOrderProfile Property:CustomerDefaults, on page 315

Property: FFT2

Purpose	Returns the FFT2 value for the customer profile
Syntax	*fft2* = myITTCustomerProfile.FFT2
Parameters	None
Return value	*fft2* — Freeform text typically used for annotation purposes by the trader or back office. Data type: String
Exceptions	None
Description	The FFT2 property returns the value of FFT2 field. The value corresponds to the value in the **FFT2** column in the X_TRADER Customer Defaults window for the selected customer.

Example

The following example uses the FFT2 property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has market = CME;ProdType = FUTURE and Product =
GE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Customer = "jdoe" And
           oCustProfile.FFT2 = "Trial Period" And
           oCustProfile.FFT3 = "Jun13"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

TTOrderProfile Property:CustomerDefaults, on page 315

Property: FFT3

Purpose	Returns the FFT3 value for the customer profile
Syntax	*fft3* = myITTCustomerProfile.FFT3
Parameters	None
Return value	*fft3* — Freeform text typically used for annotation purposes by the trader or back office. Data type: String
Exceptions	None
Description	The FFT3 property returns the value of FFT3 field. The value corresponds to the value in the **FFT3** column in the X_TRADER **Customer Defaults** window for the selected customer.

Example

The following example uses the FFT3 property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has market = CME;ProdType = FUTURE and Product =
GE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Customer = "jdoe" And
           oCustProfile.FFT2 = "Trial Period" And
           oCustProfile.FFT3 = "Jun13"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information	**TTOrderProfile** Property:CustomerDefaults, on page 315

Property: GiveUp

Purpose	Returns the clearing member code for the customer profile
Syntax	`clearingCode = myITTCustomerProfile.GiveUp`
Parameters	None
Return value	*clearingCode* — Clearing member code Data type: String
Exceptions	None
Description	The `GiveUp` property returns clearing member code associated with a Give-up trade accounts. The value corresponds to the value in the **Give Up** column in the X_TRADER **Customer Defaults** window for the selected customer.
Example	None
Related information	**TTOrderProfile** Property:CustomerDefaults, on page 315

Property: Market

Purpose	Returns the name of the market for the customer profile
Syntax	*mktName* = myITTCustomerProfile.Market
Parameters	None
Return value	*mktName* — Name of the market associated with the TT Gateway Data type: String
Exceptions	None
Description	The `Market` property returns the market for which the trader is authorized to trade. The value corresponds to the value in the **Market** column in the X_TRADER **Customer Defaults** window for the selected customer.
Example	The following example uses the `Market` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Market = CME;ProdType = FUTURE and Product =
GE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ProdType = "FUTURE" And
           oCustProfile.Product = "GE"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information	**TTOrderProfile** Property:CustomerDefaults, on page 315

Property: MaxOrderQty

Purpose	Returns the maximum order quantity allowed for this customer profile
Syntax	`qty = myITTCustomerProfile.MaxOrderQty`
Parameters	None
Return value	*qty* — Maximum order size Data type: Integer
Exceptions	None
Description	The `MaxOrderQty` property returns the maximum quantity allowed per trade for this customer profile. The value corresponds to the value in the **Max Order Qty** column in the X_TRADER **Customer Defaults** window for the selected customer.
Example	None
Related information	Property:UseMaxQty, on page 49 **TTOrderProfile** Property:CustomerDefaults, on page 315

Property: OpenClose

Purpose	Returns the Open/Close value for this customer profile
Syntax	*position* = myITTCustomerProfile.OpenClose
Parameters	None
Return value	*position* — State of the position Data type: String
Exceptions	None
Description	The OpenClose property returns whether the position is being opened or closed. The value corresponds to the value in the **O/C** column in the X_TRADER **Customer Defaults** window for the selected customer.
Example	The following example uses the OpenClose property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has market = CME and OpenClose = Open
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.OpenClose = "Open"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information	**TTOrderProfile** Property:CustomerDefaults, on page 315

Property: OrderRestr

Purpose	Returns the order restriction for this customer profile
Syntax	*typesRestrs* = myITTCustomerProfile.OrderRestr
Parameters	None
Return value	*typesRestrs* — Order restriction defined for the profile Data type: String
Exceptions	None

Description

The `OrderRestr` property returns customer profile's order restriction. The value corresponds to the value in the **Restrict** column in the X_TRADER **Customer Defaults** window for the selected customer.

For a list of possible values, refer to the X_TRADER documentation.

Example

The following example uses the `OrderRestr` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Market = CME and OrderRestr = STOP
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.OrderRestr = "STOP"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:OrderType, on page 45

TTOrderProfile Property:CustomerDefaults, on page 315

Property: OrderType

Purpose	Returns the default order type for this customer profile
Syntax	*orderType* = myITTCustomerProfile.OrderType
Parameters	None
Return value	*orderType* — Selected order type Data type: String
Exceptions	None
Description	The OrderType property returns the value in the **Type** column in the X_TRADER **Customer Defaults** window for the selected customer. For a list of possible values, refer to the X_TRADER documentation.
Example	The following example uses the OrderType property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile with Market = CME and ProdType = FUTURE and OrderType
= Mkt
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ProdType = "FUTURE" And
           oCustProfile.OrderType = "Mkt"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information	Property:OrderRestr, on page 44 **TTOrderProfile** Property:CustomerDefaults, on page 315

Property: ProdType

Purpose	Returns the default product type for this customer profile
Syntax	*prodType* = myITTCustomerProfile.ProdType
Parameters	None
Return value	*prodType* — Selected product type Data type: String
Exceptions	None

Description

The `ProdType` property returns the default product type defined for this customer. The value corresponds to the value in the **Product Type** column in the X_TRADER **Customer Defaults** window for the selected customer.

For a list of possible values, refer to the X_TRADER documentation.

Example

The following example uses the `ProdType` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Market = CME;ProdType = FUTURE and Product =
GE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ProdType = "FUTURE" And
           oCustProfile.Product = "GE"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:Product, on page 47

TTOrderProfile Property:CustomerDefaults, on page 315

Property: Product

Purpose	Returns the name of product for this customer profile
Syntax	`product = myITTCustomerProfile.Product`
Parameters	None
Return value	*product* — Specific product or * for all Data type: String
Exceptions	None
Description	The `Product` property returns the default product for the customer. The value corresponds to the value in the **Product** column in the X_TRADER **Customer Defaults** window for the selected customer.
Example	The following example uses the `Product` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has Market = CME;ProdType = FUTURE and Product =
GE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ProdType = "FUTURE" And
           oCustProfile.Product = "GE"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information	Property:ProdType, on page 46 **TTOrderProfile** Property:CustomerDefaults, on page 315

Property: TIF

Purpose	Returns the time-in-force for this profile
Syntax	`entryName = myITTCustomerProfile.TIF`
Parameters	None
Return value	*entryName* — Default time an order is in force Data type: String
Exceptions	None

Description

The `TIF` property returns default time-in-force for orders submitted through this customer profile. The value corresponds to the value in the **TIF** column in the X_TRADER **Customer Defaults** window for the selected customer.

For a list of possible values, refer to the X_TRADER documentation.

Example

The following example uses the `TIF` property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has market = CME;ProdType = FUTURE and TIF = GTD
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.ProdType = "FUTURE" And
           oCustProfile.TIF = "GTD"
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

TTOrderProfile Property:CustomerDefaults, on page 315

Property: UseMaxQty

Purpose

Returns whether this profile uses the maximum quantity restriction

Syntax

useMax = myITTCustomerProfile.UseMaxQty

Parameters

None

Return value

useMax	Whether to use the maximum order quantity Data type: Boolean

Exceptions

None

Description

The UseMaxQty property returns whether the trader's orders are subject to the maximum order quantity. The value corresponds to the value in the **Use Max** column in the X_TRADER **Customer Defaults** window for the selected customer.

Example

The following example uses the UseMaxQty property to help select a Customer Defaults entry.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object

    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults

    'Select the profile which has market = CME and UseMaxQty = TRUE
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
        oCustProfile = cdObj
        If oCustProfile.Market = "CME" And
           oCustProfile.UseMaxQty = TRUE
        Then
          oOrderProfile.CustomerDefaults = oCustProfile
        End If
    Next
  Catch ex As Exception
      MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:MaxOrderQty, on page 42

TTOrderProfile Property:CustomerDefaults, on page 315

5 ITTPriceEntry Class

Chapter overview

The chapter describes the **`ITTPriceEntry`** class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
ITTPriceEntry class overview	52
Property:Name	53
Property:PriceID	54
Property:toDouble	56
Property:toString	55
Property:toTicks	57

ITTPriceEntry class overview

Overview

The **`ITTPriceEntry`** class contains properties that allow you to return instrument prices in different formats.

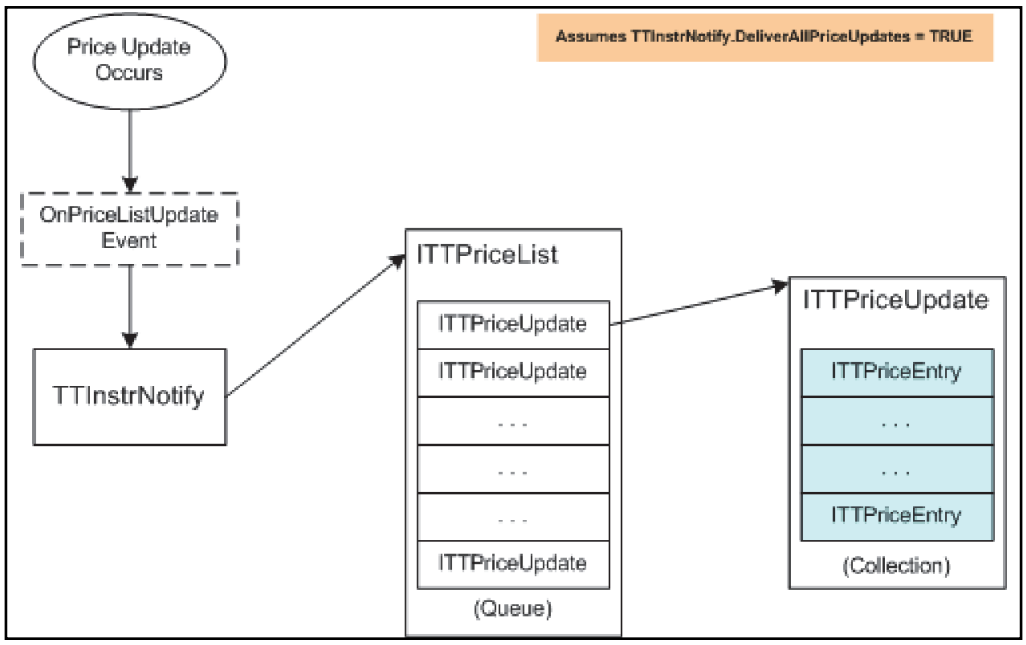


Figure 9. ITTPriceEntry class overview

This class primarily supports the .NET `foreach` construct, so that you can easily loop through the available instruments. If your development environment does not support the construct, you can use this class to iterate over the instruments.

Properties

Table 10 lists the **`ITTPriceEntry`** properties.

Property	Description
Name	Returns the name of the instrument attribute associated with price entry
PriceID	Returns the price field that was updated
toDouble	Returns the price as a decimal number
toString	Returns the price in the X_TRADER display format
toTicks	Returns the price as an integer value representing the price in ticks

Table 10. ITTPriceEntry properties

Methods

None

Events

None

Property: Name

Purpose	Returns the name of the instrument attribute associated with price entry
Syntax	*entryName* = myITTPriceEntry.Name
Parameters	None
Return value	*entryName* — Name of the object attribute corresponding to the updated price field Data type: String
Exceptions	None
Description	The Name property returns the name of the corresponding instrument's attribute that changed on the exchange.
Example	The following example uses the Name property to display the name of the updated price attribute.

```
Dim oPriceUpdate As ITTPriceUpdate
Dim oPriceEntry As ITTPriceEntry

'Iterate through the PriceList queue to get PriceUpdate
For Each oPriceUpdate In pNotify.PriceList
  'Iterate throgh the PriceUpdate collection to get Name and Price Information
  For Each oPriceEntry In oPriceUpdate
    Debug.Print("Name=" + oPriceEntry.Name + "Price=" + oPriceEntry.toString)
  Next
Next
```

Related information

Property:PriceID, on page 54

TTInstrObj Class, on page 217

Property: PriceID

Purpose

Returns the price field that was updated

Syntax

```
priceField = myITTPriceEntry.PriceID
```

Parameters

None

Return value

priceField	Price field updated for the instrument, such as **ttHighPrice** or **ttLowPrice** quantity Data type: enumPriceID

Exceptions

None

Description

The `PriceID` property returns the price field contained in the price entry. The value is an enumeration that corresponds to the string value contained in the `Name` property. This property allows you to perform quick comparisons to see which price attribute changed without having to compare attribute name strings.

This property returns one of the following values:

- ttUnknown
- ttBestAskCntrPty
- ttBestBidCntrPty
- ttClosePrice
- ttHighPrice
- ttHitTake
- ttLastTradedPrice
- ttLastTradedQty
- ttLowPrice
- ttLowerLimitPrice
- ttOTCTradePrice
- ttOTCTradeQty
- ttOpenPrice
- ttSeriesStatus
- ttSettlePrice
- ttTotalTradedQty
- ttTradeState
- ttUpperLimitPrice

Example

The following example uses the `PriceID` property to display the changed price attribute.

```
Dim oPriceUpdate As ITTPriceUpdate
Dim oPriceEntry As ITTPriceEntry

'Iterate through the PriceList queue to get PriceUpdate
For Each oPriceUpdate In pNotify.PriceList
  'Iterate throgh the PriceUpdate collection to get Name and Price Information
  For Each oPriceEntry In oPriceUpdate
    Debug.Print("Field=" + oPriceEntry.PriceId + "Price=" +
oPriceEntry.toString)
  Next
Next
```

Related information

Property:Name, on page 53

Property:toString, on page 55

TTInstrObj Class, on page 217

Property: toString

Purpose

Returns the price in the X_TRADER display format

Syntax

```
price = myITTPriceEntry.toString
```

Parameters

None

Return value

price	Returns the value of the entry formatted as a string Data type: String

Exceptions

None

Description

The `toString` property returns the price entry value as a string. The contents of the string value varies based on the type of value contained in the price entry, as shown in the following table.

Value type	Resulting string
Price data (i.e. **Ask**)	Price in X_TRADER display format
Integer data (i.e. **BidQty**)	Integer converted to a string (i.e. "10")
Enumeration (i.e. **SeriesStatus**)	Enumeration name converted to a string (i.e. "Trading")
Invalid price	Empty string

Example

The following example uses the `toString` method to convert the updated price to a string value.

```
Dim oPriceUpdate As ITTPriceUpdate
Dim oPriceEntry As ITTPriceEntry

'Iterate through the PriceList queue to get PriceUpdate
For Each oPriceUpdate In pNotify.PriceList
  'Iterate throgh the PriceUpdate collection to get Name and Price Information
  For Each oPriceEntry In oPriceUpdate
    Debug.Print("Name=" + oPriceEntry.Name + "Price=" + oPriceEntry.toString)
  Next
```

Related information

Property:PriceID, on page 54

Property:toDouble, on page 56

Property:toTicks, on page 57

Property: toDouble

Purpose	Returns the price as a decimal number
Syntax	*price* = myITTPriceEntry.toDouble
Parameters	None
Return value	*price* — Returns the value of the entry as a double value. Data type: Decimal
Exceptions	None
Description	The `toDouble` property returns the price entry value as a decimal number. For values that are always integers, such as a quantity, the method returns the decimal equivalent. For example, if the price entry represents a **BidQty** of 10, the method returns 10.0. For invalid prices, the method returns `TT_INVALID_PRICE`.
Example	The following example uses the `toDouble` method to convert the updated price to a real number.

```
Dim oPriceUpdate As ITTPriceUpdate
Dim oPriceEntry As ITTPriceEntry

'Iterate through the PriceList queue to get PriceUpdate
For Each oPriceUpdate In pNotify.PriceList
  'Iterate throgh the PriceUpdate collection to get Name and Price Information
  For Each oPriceEntry In oPriceUpdate
    Debug.Print("Name=" + oPriceEntry.Name + "Price=" + oPriceEntry.toDouble)
  Next
```

Related information	Property:PriceID, on page 54 Property:toString, on page 55 Property:toTicks, on page 57

Property: toTicks

Purpose	Returns the price as an integer value representing the price in ticks
Syntax	*price* = myITTPriceEntry.toTicks
Parameters	None
Return value	*price* — Returns the value of the entry as an integer value. Data type: Integer
Exceptions	None

Description

The `toTicks` property returns the price entry value as an integer value. For entries containing price data, such as **Bid**, the value represents the tick price of the instrument.

For invalid prices, the method returns `TT_INVALID_PRICE`.

Example

The following example uses the `toTicks` method to convert the updated price to a tick value.

```
Dim oPriceUpdate As ITTPriceUpdate
Dim oPriceEntry As ITTPriceEntry

'Iterate through the PriceList queue to get PriceUpdate
For Each oPriceUpdate In pNotify.PriceList
  'Iterate throgh the PriceUpdate collection to get Name and Price Information
  For Each oPriceEntry In oPriceUpdate
    Debug.Print("Name=" + oPriceEntry.Name + "Price=" + oPriceEntry.toTicks)
  Next
```

Related information

Property:PriceID, on page 54

Property:toDouble, on page 56

Property:toString, on page 55

6 ITTPriceList Class

Chapter overview

The chapter describes the **`ITTPriceList`** class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
ITTPriceList class overview	61
Property:Count	63
Property:Item	64

ITTPriceList class overview

Overview

The **ITTPriceList** class represents a queue of **ITTPriceUpdate** objects. An **ITTPriceList** is created and maintained when price update notifications are registered with the **TTInstrNotify** object (`DeliverAllPriceUpdates` property). Each time an exchange sends updated price information for a **TTInstrNotify** member instrument, a new **ITTPriceUpdate** object is added to the price list.

If your application wants to monitor the changes in prices, you can use this class to access all price changes. Figure 11 shows how the **ITTPriceList** object provides access to non-coalesced prices. When a price update is received from an exchange, the **TTInstrNotify** `OnPriceListUpdate` event is triggered to indicate that updated prices are available. The **TTInstrNotify** `PriceList` property returns a pointer to the **ITTPriceList** object. From this object, you can access the individual price updates.

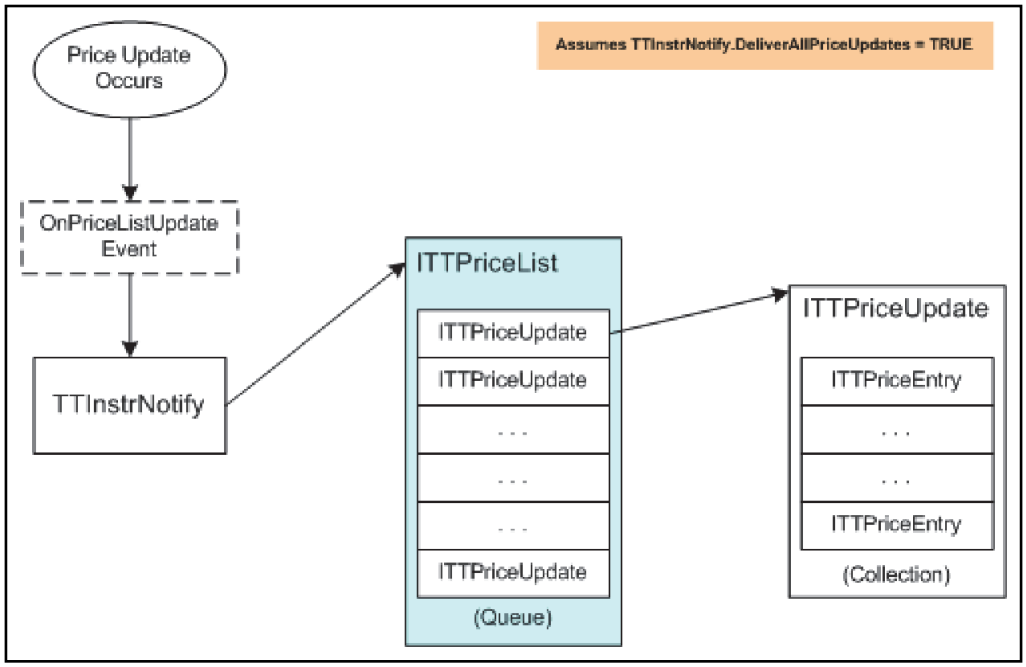


Figure 11. ITTPriceList class overview

When you retrieve a price list, the XTAPI transfers the queue to the event handler and creates a new **ITTPriceList** object for subsequent price updates. Any **ITTPriceUpdate** objects that are not processed by the event handler are lost when the next **ITTPriceList** object is retrieved.

Properties

Table 12 lists the **ITTPriceList** properties.

Property	Description
Property:Count	Returns the number of price updates in the list
Property:Item	Returns the price update at the specified position

Table 12. ITTPriceList Properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of price updates in the list
Syntax	*priceUpdateCount* = myIttPriceList.Count
Parameters	None
Return value	*priceUpdateCount* — Number of price updates in the list Data type: Integer
Exceptions	None

Description

The `Count` property returns the number of **`ITTPriceUpdate`** objects contained in the list.

Note: Because the XTAPI clears its queue of updates after it creates an **`ITTPriceList`** object, you must store the list in a temporary variable before accessing either the `Count` or `Item` property.

Example

The following code fragment uses the `Count` property to return each **`ITTPriceUpdate`** object in the list.

```
Dim cnt As Integer
        Dim p As XTAPI.ITTPriceList
        p = pNotify.PriceList

        Try
            cnt = p.Count
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try

        For i = 1 To cnt
            PriceUpdate1 = p.Item(i)

            For j = 1 To PriceUpdate1.Count
                PriceEntry = PriceUpdate1.Item(j)
            Next
        Next
```

Related information

Property:Item, on page 64

ITTPriceUpdate Class, on page 65

Property: Item

Purpose

Returns the price update at the specified position

Syntax

priceUpdate = myIttPriceList.`Item`(*index*)

Parameters

index	Position in the collection of the desired price update Data type: Integer

Return value

priceUpdate	Price update at the specified position Data type: `ITTPriceUpdate`

Exceptions

Exception	Meaning
The parameter is incorrect	The specified index is out of range.

Description

The `Item` property returns a price update from the **`ITTPriceList`** collection.

Note: The **InstrCollection** collection uses a 1-based index.

Note: Because the XTAPI clears its queue of updates after it creates an **`ITTPriceList`** object, you must store the list in a temporary variable before accessing either the `Count` or `Item` property.

Example

The following code fragment uses the `Item` property to return each **`ITTPriceUpdate`** object in the collection.

```
Dim cnt As Integer
        Dim p As XTAPI.ITTPriceList
        p = pNotify.PriceList

        Try
            cnt = p.Count
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try

        For i = 1 To cnt
            PriceUpdate1 = p.Item(i)

            For j = 1 To PriceUpdate1.Count
                PriceEntry = PriceUpdate1.Item(j)
            Next
        Next
```

Related information

Property:Count, on page 63

ITTPriceUpdate Class, on page 65

8 ITTPriceUpdate Class

Chapter overview

The chapter describes the **`ITTPriceUpdate`** class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	
ITTPriceUpdate class overview	66
Property:Count	67
Property:Instrument	68
Property:Item	70
Property:PriceTimeStamp	71
Property:TimeStamp	72

ITTPriceUpdate class overview

Overview

The **ITTPriceUpdate** class represents a snapshot of price updates for an instrument. A price update object is a collection of **ITTPriceEntry** objects with the price updates that have occurred for the instrument since the last update was received. The class supports "`foreach`" iteration over the **ITTPriceEntry** objects and allows access to a **TTInstrObj**.

As shown in Figure 13, the **ITTPriceList** object contains a queue of **ITTPriceUpdate** objects. Each price update can contain one or more price entries.

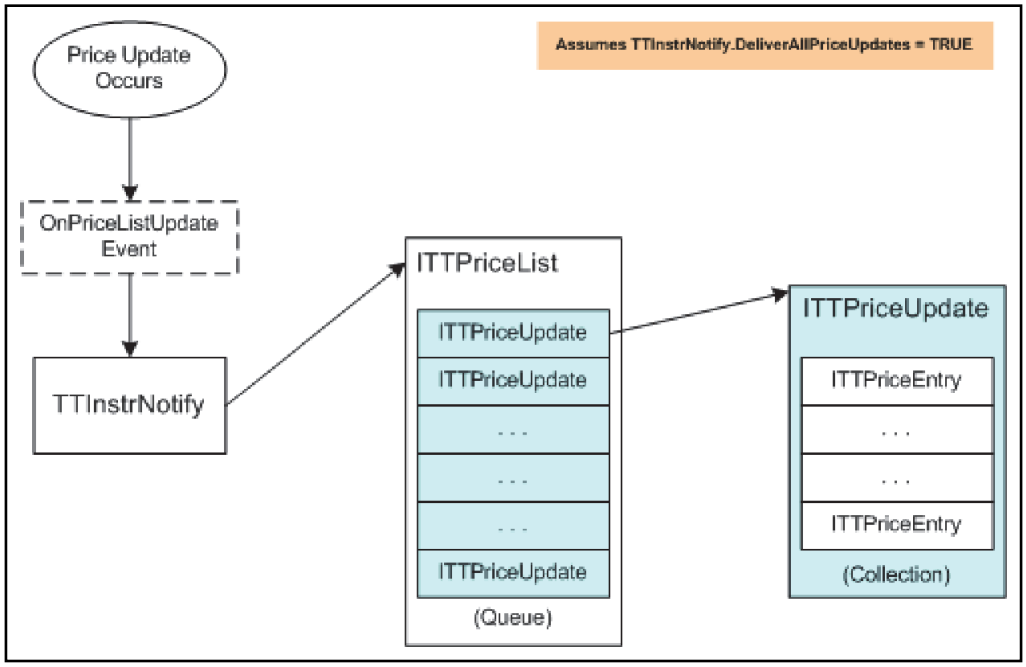


Figure 13. ITTPriceUpdate class overview

Properties

Table 14 lists the **ITTPriceUpdate** properties.

Property	Description
Count	Returns the number of price entries in the list
Instrument	Returns the instrument that received an update
Item	Returns the price entry at the specified position
PriceTimeStamp	Returns the time the XTAPI received the price update as an integer
TimeStamp	Returns the time the XTAPI received the price update

Table 14. ITTPriceUpdate properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of price entries in the list
Syntax	*priceEntryCount* = myIttPriceUpdate.Count
Parameters	None
Return value	*priceEntryCount* — Number of price entries in the list Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`ITTPriceEntry`** objects contained in the list.
Example	The following code fragment uses the `Count` property to return each **`ITTPriceEntry`** object in the list.

```
Dim cnt As Integer
        Dim p As XTAPI.ITTPriceList
        p = pNotify.PriceList

        Try
            cnt = p.Count
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try

        For i = 1 To cnt
            PriceUpdate1 = p.Item(i)

            For j = 1 To PriceUpdate1.Count
                PriceEntry = PriceUpdate1.Item(j)
            Next
        Next
```

Related information	Property:Item, on page 64 ITTPriceEntry Class, on page 51

Property: Instrument

Purpose

Returns the instrument that received an update

Syntax

```
instrObj = myIttPriceUpdate.Instrument
```

Parameters

None

Return value

instrObj	Instrument for which the price update occurred Data type: `TTInstrObj`

Exceptions

Exception	Meaning
get_Instrment, Error: text	Some COM error occurred, represented by text.

Description

The `Instrument` property returns the **`TTInstrObj`** object for the instrument that received the price update. Because a single instrument notifier can monitor prices for multiple instruments, you use this property to sort through the updates contained in the price list.

Example

The following example uses the `Instrument` property to access the instrument associated with each price update. Then it uses a shorthand technique to return the instrument's `Contract` property.

```
Private Sub m_TTInstrNotify_OnPriceListUpdate(ByVal pNotify As
XTAPI.TTInstrNotify) Handles m_TTInstrNotify.OnPriceListUpdate

  Try

    Dim oPriceUpdate As XTAPI.ITTPriceUpdate
    Dim oPriceEntry As XTAPI.ITTPriceEntry

    'Iterate through the PriceList queue to get PriceUpdate
    For Each oPriceUpdate In pNotify.PriceList

      'Iterate through the PriceUpdate collection for Name and Price Info
      'Note that this is the actual instrument returned, not a
      'string representing the name of the instrument
      Console.WriteLine("Instrument=" + oPriceUpdate.Instrument.Contract)

      For Each oPriceEntry In oPriceUpdate
        Console.WriteLine("Name=" + oPriceEntry.Name + " Price=" +
            oPriceEntry.toString)
      Next
    Next

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Related information

Property:TimeStamp, on page 72

Property: Item

Purpose

Returns the price entry at the specified position

Syntax

`priceUpdate = myIttPriceUpdate.Item(index)`

Parameters

index	Position in the collection of the desired price entry Data type: Integer

Return value

priceEntry	Price entry at the specified position Data type: `ITTPriceEntry`

Exceptions

Exception	Meaning
The parameter is incorrect	The specified index is out of range.

Description

The `Item` property returns a price update from the **`ITTPriceUpdate`** collection.

Note: The **InstrCollection** collection uses a 1-based index.

Example

The following code fragment uses the `Item` property to return each **`ITTPriceEntry`** object in the collection.

```
Dim cnt As Integer
        Dim p As XTAPI.ITTPriceList
        p = pNotify.PriceList

        Try
            cnt = p.Count
        Catch ex As Exception
            MsgBox(ex.Message)
        End Try

        For i = 1 To cnt
            PriceUpdate1 = p.Item(i)

            For j = 1 To PriceUpdate1.Count
                PriceEntry = PriceUpdate1.Item(j)
            Next
        Next
```

Related information

Property:Count, on page 67

ITTPriceEntry Class, on page 51

Property: PriceTimeStamp

Purpose	Returns the time the XTAPI received the price update as an integer
Syntax	`time = myIttPriceUpdate.PriceTimeStamp`
Parameters	None
Return value	*time* — Time, in UTC, the update was received, as an integer representing the time in 100-nanosecond units Data type: 64-bit Integer
Exceptions	None

Description

The `PriceTimeStamp` property returns the time, as an integer, when the XTAPI received the instrument's price update from the exchange. Because TT Gateways do not transmit timestamps for updates, this time does not represent when the update actually occurred at the exchange. To compensate, the XTAPI stores the time when it received the price update.

You can pass this value to the `System.DateTime()` function to convert the integer to a `DateTime` object, which allows you to retrieve date information, such as a month or day, more efficiently than parsing the expiration date string value returned by the `TimeStamp` property.

Note: The XTAPI receives this timestamp from the Exchange in UTC format, not as local time. To convert the value to local time, you can call the `System.ToLocalTime()` method.

Example

The following example uses the `PriceTimeStamp` property and converts it to local time.

```
Dim ticks as Long
ticks = oPriceUpdate.PriceTimeStamp
Dim dt As System.DateTime
dt = New DateTime(ticks).ToLocalTime()
'
' code to parse the dt variable omitted
'
```

Related information

Property:Instrument, on page 68

Property:TimeStamp, on page 72

Property: TimeStamp

Purpose	Returns the time the XTAPI received the price update
Syntax	*time* = myIttPriceUpdate.TimeStamp
Parameters	None
Return value	*time* — Time the update was received, in the format: *HH*:*MM*:*SS*:*mmm* Data type: String
Exceptions	None
Description	The TimeStamp property shows when the XTAPI received the instrument's price update from the exchange. Because the exchanges do not transmit timestamps for updates, this time does not represent when the update actually occurred at the exchange. To compensate, the XTAPI stores the time when it received the price update.
Example	The following example uses the TimeStamp property to display the time a price updated.

```
Private Sub m_TTInstrNotify_OnPriceListUpdate(ByVal pNotify As
XTAPI.TTInstrNotify) Handles m_TTInstrNotify.OnPriceListUpdate

  Try

    Dim oPriceUpdate As XTAPI.ITTPriceUpdate
    Dim oPriceEntry As XTAPI.ITTPriceEntry

    'Iterate through the PriceList queue to get PriceUpdate
    For Each oPriceUpdate In pNotify.PriceList

      'Iterate through the PriceUpdate collection to get Name and Price Info
      Console.WriteLine("TimeStamp=" + oPriceUpdate.TimeStamp)

      For Each oPriceEntry In oPriceUpdate
        Console.WriteLine("Name=" + oPriceEntry.Name + " Price=" +
            oPriceEntry.toString)
      Next
    Next

    Catch ex As Exception
      Console.WriteLine(ex.Message)
  End Try
End Sub
```

Related information	Property:Instrument, on page 68 Property:PriceTimeStamp, on page 71

9 OrderProfileCol Class

Chapter overview

The chapter describes the `OrderProfileCol` class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
OrderProfileCol class overview	74
Property:Count	75
Property:Item	76

OrderProfileCol class overview

Overview

The **`OrderProfileCol`** class contains a collection of instrument (**`TTOrderProfile`**) objects. This class allows you to access instruments directly from a **`TTGate`** object, which returns the collection of aliased order profile from its `OrderProfiles` property. You can use this class to manipulate all of the aliased instruments known to the **`TTGate`** object.

This class primarily supports the .NET `foreach` construct, so that you can easily loop through the available order profiles. If your development environment does not support the construct, you can use this class to iterate over the order profiles.

For more information about instrument aliases, refer to the **`TTInstrObj`** object's Property:Alias section on page 312.

Properties

Table 15 lists the **`OrderProfileCol`** properties.

Property	Description
Count	Returns the number of order profile objects contained in the collection
Item	Returns the order profile at the specified index or alias in a collection

Table 15. OrderProfileCol properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of order profile objects contained in the collection
Syntax	*orderProfCount* = myOrderProfileCol.Count
Parameters	None
Return value	*orderProfCount* — Number of order profiles in the collection Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`OrderProfileCol`** objects contained in the collection.

Example

The following code fragment displays the number of order profile objects.

```
Dim orderprofcol As XTAPI.OrderProfileCol
orderprofcol = Me.m_TTGate.OrderProfiles()

Dim intOrderProfCount As Integer
intOrderProfCount = orderprofcol.Count

Console.WriteLine(intOrderProfileCount)
```

Related information

InstrCollection Property:Count, on page 31

OrderSetCol Property:Count, on page 79

TTOrderProfile Class, on page 309

Property: Item

Purpose

Returns the order profile at the specified index or alias in a collection

Syntax

```
orderProfileObj = myOrderProfileCol.Item(index_or_alias)
```

Parameters

index_or_alias	Position in the collection of the desired profile or an alias for the order profile Data type: Integer or String

Return value

orderProfileObj	Order profile at the specified location or matching the specified alias Data type: `TTOrderProfile`

Exceptions

Exception	Meaning
The parameter is incorrect	The specified index is either out of range or not a valid order profile alias.

Description

The `Item` property returns an order profile from the **`OrderSetCol`** collection. You can request an order profile either by specifying the index position or by providing an alias associated with an order profile.

Note: The **InstrCollection** collection uses a 1-based index.

Example

None

Related information

InstrCollection Property:Item, on page 32

OrderSetCol Property:Item, on page 80

TTOrderProfile Class, on page 309

10 OrderSetCol Class

Chapter overview

The chapter describes the `OrderSetCol` class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
OrderSetCol class overview	78
Property:Count	79
Property:Item	80

OrderSetCol class overview

Overview

The **`OrderSetCol`** class contains a collection of instrument (**`TTOrderSet`**) objects. This class allows you to access order sets directly from a **`TTGate`** object, which returns the collection of aliased order sets from its `OrderSets` property. You can use this class to manipulate all of the aliased order sets known to the **`TTGate`** object.

This class primarily supports the .NET `foreach` construct, so that you can easily loop through the available order sets. If your development environment does not support the construct, you can use this class to iterate over the order sets.

For more information about order set aliases, refer to the **`TTOrderSet`** object's Property:Alias section on page 362.

Properties

Table 16 lists the **`OrderSetCol`** properties.

Property	Description
Count	Returns the number of order set objects contained in the collection
Item	Returns the instrument at the specified index or alias in a collection

Table 16. OrderSetCol properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of order set objects contained in the collection
Syntax	*orderSetCount* = myOrderProfileCol.Count
Parameters	None
Return value	*orderSetCount* — Number of order sets in the collection Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`TTOrderSet`** objects contained in the collection.
Example	The following code fragment displays the number of order set objects in the **`OrderSetCol`** object.

```
Dim ordersetcol As XTAPI.OrderSetCol
ordersetcol = Me.m_TTGate.OrderSets()

Dim intOrderSetCount As Integer
intOrderSetCount = ordersetcol.Count

Console.WriteLine(intOrderSetCount)
```

Related information

InstrCollection Property:Count, on page 31

OrderProfileCol Property:Count, on page 75

TTOrderSet Class, on page 357

Property: Item

Purpose

Returns the instrument at the specified index or alias in a collection

Syntax

```
instrObj = myOrderProfileCol.Item(index_or_alias)
```

Parameters

index_or_alias	Position in the collection of the desired instrument or an alias for the instrument Data type: Integer or String

Return value

instrObj	Order profile at the specified location or matching the specified alias Data type: `TTOrderSet`

Exceptions

Exception	Meaning
The parameter is incorrect	The specified index is either out of range or not a valid instrument alias.

Description

The `Item` property returns an instrument from the **`OrderSetCol`** collection. You can request an order set either by specifying the index position or by providing an alias associated with an order set.

Note: The **InstrCollection** collection uses a 1-based index.

Example

The following code fragment uses the `Item` property to return each **`TTOrderSet`** object in the collection.

```
osCol = myGate.OrderSets
'Loop through each TTOrderSet
For i = 1 To osCol.Count
   Try
      Dim os As TTOrderSet
      os = osCol.Item(i)

      'Get the property values
      Dim oProp() As Object = os.Get(propList)
      ' ... omitted code
End For
```

Related information

InstrCollection Property:Item, on page 32

OrderProfileCol Property:Item, on page 76

TTOrderSet Class, on page 357

11 TTDropHandler Class

Chapter overview

The chapter describes the **`TTDropHandler`** class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
TTDropHandler class overview	82
Property:Count	83
Property:DropEffect	84
Property:DropKeyState	86
Property:DropX	88
Property:DropY	89
Property:Item	90
Method:RegisterDropWindow	91
Method:Reset	92
Event:OnNotifyDrop	93

TTDropHandler class overview

Overview

The **TTDropHandler** class provides support for using Microsoft Windows drag-and-drop functionality in an XTAPI application. Instantiating a drop handler allows users to drag instruments from X_TRADER to an XTAPI application window instead of manually entering instrument names.

When using a **TTDropHandler** object, you must call the `RegisterDropWindow` method to register the application with the Windows operating system.

Properties

Table 17 lists the **TTDropHandler** properties.

Property	Description
Count	Returns the number of instruments dropped in the window
DropEffect	Detects whether a user pressed the `CTRL` key when dropping instruments onto the window
DropKeyState	Returns the keys pressed when dropping instruments onto a window
DropX	Returns the X-axis coordinate of the mouse pointer when dropping instruments
DropY	Returns the Y-axis coordinate of the mouse pointer when dropping instruments
Item	Returns an instrument that was dropped on the application window

Table 17. TTDropHandler properties

Methods

Table 18 lists the **TTDropHandler** methods.

Method	Description
Method:RegisterDropWindow	Enables an application window to support dropped instruments
Method:Reset	Clears dropped instruments from the application window

Table 18. TTDropHandler methods

Events

Table 19 lists the **TTDropHandler** events.

Event	Description
Event:OnNotifyDrop	Users drop instruments on an application window

Table 19. TTDropHandler events

Property: Count

Purpose	Returns the number of instruments dropped in the window
Syntax	`numInstr = myDropHandler.Count`
Parameters	None
Return value	*numInstr* — Number of instruments dropped Data type: Integer
Exceptions	None

Description

The `Count` property indicates how many instruments the user dropped from X_TRADER on your application window. You can use this property as the loop control when iterating over the array of dropped instruments returned in the Item property.

The XTAPI maintains a cumulative count of dropped instruments until the count is cleared by the `Reset` method. If you do not reset the count after you process the dropped instruments, the dropped instruments are added to the existing count.

Example

The following code fragment uses the `Count` property to determine how many instruments a user dropped on the application window.

```
Dim WithEvents dropInstr As TTDropHandler
Dim instrObj As TTInstrObj
Dim WithEvents myInstrNotify As TTInstrNotify

Set dropInstr = New TTDropHandler
dropInstr.RegisterDropWindow Me.hWnd

Private Sub dropInstr_OnNotfyDrop()
   Dim i, numDropped As Long
   numDropped = dropInstr.Count

   For i = 1 to numDropped
      ' Code to process instrment here
   Next i
End Sub
```

Related information

Property:Item, on page 90

Property: DropEffect

Purpose

Detects whether a user pressed the CTRL key when dropping instruments onto the window

Syntax

isCtrlPressed = myDropHandler.DropEffect

Parameters

None

Return value

isCtrlPressed	Whether the CTRL key was pressed. This property returns one of the following values: **1**: CTRL key was pressed **2**: All other situations Data type: Integer

Exceptions

None

Description

The `DropEffect` property indicates whether the CTRL key was pressed when the user dropped instruments from X_TRADER® onto the XTAPI application window. You can use this property to perform special logic if the user drops the instruments while pressing the CTRL key.

Example

The following example shows how to use the `DropEffect` property to determine whether the user pressed the CTRL key when dropping an instrument.

```
Private Sub m_DropInstr_OnNotifyDrop() Handles m_DropInstr.OnNotifyDrop

  Try
    'Clears the instrument object
    If Not m_TTInstrObj Is Nothing Then
      ' Detach previously attached instrument.
      m_TTInstrNotify.DetachInstrument(m_TTInstrObj)

      ' Set the TTInstrObj to NULL.
          m_TTInstrObj = Nothing
    End If

    'copy the instrument to the instrument object
    m_TTInstrObj = m_DropInstr(1)

    'Attach the instrument object to the notify obj
    m_TTInstrNotify.AttachInstrument(m_TTInstrObj)

    'Create an integer to hold the value 1 if the control button was held by
    'the user and 2 if it was not
    Dim intIsControlPressed As Integer
    intIsControlPressed = Me.m_DropInstr.DropEffect

    If (intIsControlPressed = 1) Then
      Console.WriteLine("User held the CTRL key when droppping instrument")
    ElseIf (intIsControlPressed = 2) Then
      Console.WriteLine("User did not hold Control when dropping instrument")
    End If

    'clear the value
    intIsControlPressed = 0
```

```
    'Open the instrument object
    m_TTInstrObj.Open()

    'resets the drop instrument
    m_DropInstr.Reset()

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try

End Sub
```

Related information

Property:DropKeyState, on page 86

Event:OnNotifyDrop, on page 93

Property: DropKeyState

Purpose

Returns the keys pressed when dropping instruments onto a window

Syntax

```
keyMask = myDropHandler.DropKeyState
```

Parameters

None

Return value

keyMask	Bitmask representing the keys pressed when dragging and dropping instruments to the application window. The value is a bitmask based on the following values: **4**: **SHIFT** key **8**: **CTRL** key **32**: **ALT** key Data type: Integer

Exceptions

None

Description

The `DropKeyState` property indicates which keys were pressed when dragging instruments from X_TRADER to your XTAPI application window. When the XTAPI triggers the `OnNotifyDrop` event, it also populates the `DropKeyState` property. The property dynamically updates while the user drags instruments to the window using the left mouse button (**LMB**), so it reflects changes in the keys or keys pressed during the drag-and-drop operation. You can also specify one or more key-modifiers for such as **CTRL-SHIFT-LMB** or **ALT-LMB**.

You can use this property to perform different functionality when different key and button combinations are used.

Note: The property is valid only for windows that have registered for drag-and-drop events.

Example

The following example uses the `DropKeyState` property to determine which key a user pressed when dropping an instrument onto the XTAPI application.

```
'Example illustrates the use of the key mask property
'This property notifies the application if the user
'held down a key while dropping the instrument
Private Sub m_DropInstr_OnNotifyDrop() Handles m_DropInstr.OnNotifyDrop

  Try
       'Clears the instrument object
       If Not m_TTInstrObj Is Nothing Then
            ' Detach previously attached instrument.
            m_TTInstrNotify.DetachInstrument(m_TTInstrObj)

            ' Set the TTInstrObj to NULL.
            m_TTInstrObj = Nothing
       End If

       'copy the instrument to the instrument object
       m_TTInstrObj = m_DropInstr(1)

       'Attach the instrument object to the notify obj
       m_TTInstrNotify.AttachInstrument(m_TTInstrObj)
```

```
        'Get the keyMask value
        Dim intKeyMask As Integer
        intKeyMask = Me.m_DropInstr.DropKeyState

        Console.WriteLine(intKeyMask)

        'Open the instrument object
        m_TTInstrObj.Open()

        'resets the drop instrument
        m_DropInstr.Reset()

    Catch ex As Exception
        Console.WriteLine(ex.Message)
    End Try

End Sub
```

Related information

Property:DropEffect, on page 84

Event:OnNotifyDrop, on page 93

Property: DropX

Purpose	Returns the X-axis coordinate of the mouse pointer when dropping instruments
Syntax	`xCoord = myDropHandler.DropX`
Parameters	None
Return value	*xCoord* — Location along the X-axis of the window Data type: Integer
Exceptions	None
Description	The `DropX` property returns the X-coordinate in the application window of the mouse pointer when the user drops instruments from X_TRADER. The window origin (0,0) is located at the top left corner of the window.
Example	The following code fragment uses the `DropX` property to display the X-coordinate of the mouse when it dropped an instrument onto the application window.

```
'copy the instrument to the instrument object
m_TTInstrObj = m_DropInstr(1)

'Attach the instrument object to the notify obj
m_TTInstrNotify.AttachInstrument(m_TTInstrObj)

'Get the X and Y coordinates where
'the instrument was dropped by the user
Dim intXCord As Integer
Dim intYCord As Integer
intXCord = Me.m_DropInstr.DropX
intYCord = Me.m_DropInstr.DropY

Console.WriteLine("Drop X value {0}", intXCord)
Console.WriteLine("Drop Y value {0}", intYCord)

'Open the instrument object
m_TTInstrObj.Open()
```

Related information	Property:DropY, on page 89

Property: DropY

Purpose	Returns the Y-axis coordinate of the mouse pointer when dropping instruments
Syntax	*yCoord* = myDropHandler.DropY
Parameters	None
Return value	*yCoord* — Location along the Y-axis of the window Data type: Integer
Exceptions	None
Description	The DropY property returns the Y-coordinate in the application window of the mouse pointer when the user drops instruments from X_TRADER. The window origin (0,0) is located at the top left corner of the window.
Example	The following example uses the DropY property to display the Y-coordinate of the mouse when it dropped an instrument onto the application window.

```
'copy the instrument to the instrument object
m_TTInstrObj = m_DropInstr(1)

'Attach the instrument object to the notify obj
m_TTInstrNotify.AttachInstrument(m_TTInstrObj)

'Get the X and Y coordinates where
'the instrument was dropped by the user
Dim intXCord As Integer
Dim intYCord As Integer
intXCord = Me.m_DropInstr.DropX
intYCord = Me.m_DropInstr.DropY

Console.WriteLine("Drop X value {0}", intXCord)
Console.WriteLine("Drop Y value {0}", intYCord)

'Open the instrument object
m_TTInstrObj.Open()
```

Related information	Property:DropX, on page 88

Property: Item

Purpose	Returns an instrument that was dropped on the application window
Syntax	`instr = myDropHandler.Item(index)`
Parameters	*index* — Index in the collection (1-based) of dropped instruments Data type: Integer
Return value	*instr* — TTInstrObj for the specified index in the instrument collection Data type: TTInstrObj
Exceptions	None
Description	The `Item` property returns one of the instruments dropped on your application window, based on the specified collection index. When the XTAPI triggers the `OnNotifyDrop` event, it makes the dropped instruments available to your application through the `Item` property.
Example	The following code fragment uses the `Item` property to return each instrument dropped on the application window.

```
Dim i, iDropped As Integer
Dim LiveInstr as Variant
Dim DropInstr As TTInstrobj
For i = 0 To iDropped - 1     ' setup the instruments object
  If Not (LiveInstr(i) Is Nothing) Then
    InstrNoti.DetachInstrument LiveInstr(i)
  Set LiveInstr(i) = DropInstr.Item(i + 1) ' set the instrument information
  InstrNoti.AttachInstrument LiveInstr(i)  ' ask for notifications
  LiveInstr(i).Open                        ' Open/access the instrument
Next i
```

Related information

Property:Count, on page 83

Event:OnNotifyDrop, on page 93

TTInstrObj Class, on page 217

Method: RegisterDropWindow

Purpose

Enables an application window to support dropped instruments

Syntax

`myDropHandler.RegisterDropWindow(hWnd)`

Parameters

hWnd	Windows operating system handle for the desired application window Data type: Integer

Return value

None

Exceptions

Exception	Meaning
Invalid Window Handle	The *hWnd* parameter does not identify a valid Windows window.

Description

The `RegisterDropWindow` property registers your application window with the Windows operating system so that it can receive drag-and-drop events. After you register a window, you can drag and drop instruments from an X_TRADER window onto your application window.

If you want to support drag-and-drop for multiple windows in your application, you must define a **`TTDropHandler`** object for each window.

Example

The following example shows how to register the default application window (`Me`) to for drag-and-drop events.

```
Dim WithEvents DropInstr As TTDropHandler
Dim myInstr As TTInstrObj
Dim WithEvents myInstrNotify As TTInstrNotify

Set DropInstr = New TTDropHandler          ' Create the drop object
DropInstr.RegisterDropWindow Me.hWnd       ' Enable drag and drop

Private Sub DropInstr_OnNotifyDrop()
  Set myInstr = DropInstr.Item(1)          ' Set the instrument info
  DropInstr.Reset                          ' Clear the list for next time
  Set myInstrNotify = myInstr.CreateNotifyObject
  myInstr.Open                             ' Open/access the instrument
End Sub
```

Related information

Method:Reset, on page 92

Event:OnNotifyDrop, on page 93

Method: Reset

Purpose	Clears dropped instruments from the application window
Syntax	`myDropHandler.Reset`
Parameters	None
Return value	None
Exceptions	None
Description	The `Reset` property removes the dropped instruments from your application window's instrument array. Each time the XTAPI triggers an `OnNotifyDrop` event, it appends the dropped instruments to the window's array of instruments. All of the instruments can be accessed through the Item property until cleared by the `Reset` method. **Note:** If you do not call the `Reset` method after processing the dropped instruments, the XTAPI adds subsequently dropped instruments to the existing collection.
Example	The following code fragment clears the dropped instrument array when the number of dropped instruments meets or exceeds the max number permitted by the client application.

```
If iDropped > maxDropped Then iDropped = maxDropped   ' Max instruments
For i = 0 To iDropped - 1     ' setup the instruments object

   ' ... omitted code to process instruments

Next i
Prices.Caption = "Dropped: " & DropInstr.Count & " Instruments"
If iDropped >= maxDropped Then DropInstr.Reset  ' If full, Clear the list
```

Related information	Property:Count, on page 83 Event:OnNotifyDrop, on page 93

Event: OnNotifyDrop

Triggered when	Users drop instruments on an application window
Syntax	`Sub myDropHandler_OnNotifyDrop`
Parameters	None
Description	The `OnNotifyDrop` event is triggered when a user drags and drops one or more instruments from an X_TRADER window, such as **Market Explorer**, onto an application window that has registered for the events. **Note:** The XTAPI triggers this event only for windows that have registered for drag-and-drop events.
Example	The following callback routine subscribes to price updates for the instruments dropped onto the application window.

```
Private Sub DropInstr_OnNotifyDrop()
   Dim i As Integer
   Dim iDropped As Integer

   lblDragDrop.Visible = False   ' Hide the instruction to the user
   iDropped = DropInstr.Count    ' Get number of dropped instruments
   For i = 0 To iDropped - 1     ' setup the instruments object
      If Not (LiveInstr(i) Is Nothing) Then
       InstrNoti.DetachInstrument LiveInstr(i)
      Set LiveInstr(i) = DropInstr.Item(i + 1) ' set the instrument information
       InstrNoti.AttachInstrument LiveInstr(i)  ' ask for notifications on
updates
      LiveInstr(i).Open                        ' Open/access the instrument
   Next i
   Prices.Caption = "Dropped: " & DropInstr.Count & " Instruments"
End Sub
```

Related information

Property:Count, on page 83

Property:Item, on page 90

Method:RegisterDropWindow, on page 91

TTInstrObj Class, on page 217

12 TTFillObj Class

Chapter overview

The chapter describes the `TTFillObj` class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
TTFillObj class overview	96
Property:Get	97
Property:Instrument	106
Property:IsValidGet	107
Property:ReadProperties	109
Method:Confirm	110

TTFillObj class overview

Overview

The **TTFillObj** class represents a market fill. As a trader's orders fill, the exchange gateways pass the fills to the XTAPI. The **TTFillObj** object stores information about each fill.

The XTAPI generates a **TTFillObj** each time an order receives a fill. For example, a trader enters a ten-lot order. The trader receives a 3-lot fill, followed by a 5-lot fill, and then a 2-lot fill. The XTAPI generates a total of three **TTFillObj** objects for the order.

If the XTAPI cannot find the order that generated a fill, it creates the fill as a pseudo-order. Once the fill matches a corresponding order (or pseudo-order), the XTAPI adds it to all **TTOrderSet** objects whose **TTOrderSelector** objects include it in the order filter. All fills that match this filter modify the net position and update the summary information within the **TTOrderSet**.

The **OrderNo** and **SiteOrderKey** object attributes provide the only ties between the fill and the original order. To track an order throughout its lifetime, your application must use the **SiteOrderKey** as the unique ID for the order.

Note: A **TTOrderSet** does not store fills. Instead, the XTAPI passes fills directly as a fill feed to your application (see `OnOrderFillData` in **TTOrderSet** for more information). To re-request all fills, you must re-initialize the **TTOrderSet**.

Properties

Table 20 lists the **TTFillObj** properties.

Property	Description
Get	Returns a TTFillObj object's attributes
Instrument	Returns the instrument associated with the fill
IsValidGet	Returns a TTFillObj object's attributes
ReadProperties	Returns information about TTFillObj attributes you can access through the Get property

Table 20. TTFillObj properties

Methods

Table 21 lists the **TTFillObj** methods.

Method	Description
Confirm	Reassigns a fill to another trader's account

Table 21. TTFillObj methods

Events

None

Property: Get

Purpose

Returns a **TTFillObj** object's attributes

Syntax

attrValues = myFillObj.Get(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **TTFillObj** object attribute

Description

The Get property returns the values for an arbitrary number of **TTFillObj** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single ChangeMe property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 22 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 22. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 22. Price and quantity value formats

Table 23 lists the **`TTFillObj`** object attributes you can supply to the `Get` property.

Attribute	Description	DataType
AccountType	Code indicating the desired account type	String
Acct	Customer's or trader's account number	String
AcctType	Code indicating the desired account type	String
Broker	Broker's name (multi-broker only)	String
Broker#	Broker's ID (multi-broker only)	String
Buy	Number of contracts bought (same as Long)	Integer
BuySell BuySell$	Type of order. Possible values include: **B**: Buy **S**: Sell For a buy order, BuySell returns B, while BuySell$ returns Buy.	String
ClearingDate	Clearing date for the fill, as specified by an Exchange. Some Exchanges, such as the TT SGX Gateway, use this attribute to support different dates for when a trade executes (**DateExec** attribute) and when the clearing firm clears the trade (**ClearingDate** attribute).	COleDateTime
ClrMember	Clearing Member ID	String
CntrClearing	Counter party trader's Firm ID	String
CntrMember	Counter party trader's Member ID	String
ColorPri ColorSec	Color to display in the MD TRADER® working orders grid, where: • **ColorPri** represents the order's main color (right-side of the grid) • **ColorSec** represents the order's secondary color (left-side of the grid) The value is an integer representing the RGB value of the color. The value corresponds to the Microsoft `ColorRef` data type.	Integer
ComboCode	Combination code for the fill. Possible values include: "" (empty string): Non-spread fill **C**: Calendar spread **F**: All other spread fills	String
Company	Broker's company name (multi-broker only)	String
Company#	Broker's company ID (multi-broker only)	String
ConnectionName	Broker's connection name (multi-broker only)	String
Contract Contract$	Instrument's name with expiration (e.g. ES Sep13)	String
Contract.objAttribute	Object attribute for the attached instrument (**`TTInstrObj`**)	Various

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
ContractAlias	Instrument's alias name with expiration (eg. E-Mini Sep13)	String
Date	Date of the order in *MM*/*DD*/*YY* (e.g. 03/18/13) The value contains valid data only in the date portion of the `COleDateTime` object.	COleDateTime
DateExec	Date the fill occurred in *MM*/*DD*/*YY* (e.g. 03/18/13) The value contains valid data only in the date portion of the COleDateTime object.	COleDateTime
Ex:OrderNo	Order number for the specified exchange (eg. CME-S: 107578464)	String
Exchange	Name of the TT Gateway that sent the fill	String
ExchCred	Exchange credentials (multi-broker only)	String
ExGroup	Exchange Group ID for the fill	String
ExMember	Exchange Member ID for the fill	String
ExOrderID	Exchange Order ID assigned to the fill	String
	Note: If the exchange does not provide a value, the X_TRADER API populates it with the OrderNo attribute value.	
ExRecNo	Transaction ID provided by the exchange	String
ExTrader	Exchange Trader ID for the fill	String
ExTransNo	Transaction ID provided by the exchange	String
FFT1 FFT2 FFT3	Free-form text fields. FFT1 typically contains the account.	String
Fee	Fee code for the fill. The fee codes are exchange-dependent.	String
FillKey	Fill ID supplied by the exchange	String
FillTicks	Total value of the fill in ticks	Integer
FillType	Type of fill. Possible values include: **P**: Partial **F**: Full	String
GiveUp	Give-up member's clearing ID, when the fill is assigned to another member	String
	Note: The TT ICE Gateway does not support the **GiveUp** attribute; instead, it uses the **ClrMember** attribute for Give Up orders.	
Group	Group name of the trader from the Member/Group/Trader ID	String
Instr Instr$	Default instrument name, as displayed in the XTAPI **Market Grid Contract** field (e.g., ES Sep13)	String
InstrAlias	Instrument's alias name with expiration (e.g. E-Mini Sep13)	String
Instr.objAttribute	Value of the specified object attribute for the related `TTInstrObj` object	Various
	Note: You can access only static, contract-related attributes, such as **Contract**, **Expiry**, and **Exchange**.	

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
IsAON	Whether the order is an All-Or-None order	Boolean
IsAuction	Whether the order is only valid for auction	Boolean
IsAuto-Agress IsAuto-Aggress	Whether the order is auto-aggressive (Currently supported only for BrokerTec)	Boolean
IsAutomated	Whether the matching order orginated from an automated program.	Boolean
IsAwayMarket	Whether an away market maker should match BBO of an away market (Currently supported only for BrokerTec)	Boolean
IsBasis	Whether the fill represents a basis order	Boolean
IsBestLimit	Whether the fill represents is a best limit order	Boolean
IsBlock	Whether the fill represents a block order.	Boolean
IsBuy	Whether the fill represents a buy order	Boolean
IsConfirmed	Whether the fill is a confirmed fill	Boolean
IsDetailedSOD	Whether the fill represents a detailed SOD record	Boolean
IsFAS	Whether the fill represents a fill and save order	Boolean
IsFlex	Whether the fill represents a flex option order	Boolean
IsFOK	Whether the fill represents a fill or kill order	Boolean
IsIceberg	Whether the fill represents an iceberg order	Boolean
IsIfTouched	Whether the order requires additional action when a price is touched	Boolean
IsInverted	Whether the fill is canceled and that the trader's position has been adjusted to reflect the cancelled fill	Boolean
IsIOC	Whether the fill represents an immediate-or-cancel order	Boolean
IsLOC	Whether the fill represents a market to limit order to be filled or killed on market close	Boolean
IsLOO	Whether the fill represents a market to limit order to be filled or killed on market open	Boolean
IsLSM	Whether the fill represents a limit-stop-market order	Boolean
IsLTM	Whether the fill represents a limit order to be converted to a market order on close	Boolean
IsManualFill	Whether the fill was entered as a manual fill	Boolean
IsMarketToLimit	Whether the fill represents a market-to-limit order	Boolean
IsMLM	Whether the fill represents a market-limit-market order	Boolean
IsMOC	Whether the fill represents a market-on-close order	Boolean
IsMOO	Whether the fill represents a market-on-open order	Boolean
IsMTL	Whether the fill represents a market-to-limit order	Boolean
IsMV	Whether the fill represents a minimum volume order	Boolean

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
IsNon-leave	Whether the fill represents a non-leave order (Currently supported only for BrokerTec)	Boolean
IsPassive	Whether the fill represents a passive (Currently supported only for BrokerTec)	Boolean
IsSell	Whether the fill represents a sell order	Boolean
IsShortExempt	Whether the fill represents a sell short order exempt from notification	Boolean
IsSOD	Whether the fill is a start-of-day fill	Boolean
IsStop	Whether the fill represents a stop order	Boolean
IsTimed	Whether the fill represents a timed order	Boolean
IsTriggered	Whether the order has been triggered	Boolean
IsTriggeredTime	Whether the order was triggered after a time submission	Boolean
IsVolatility	Whether the fill represents a volatility order	Boolean
Key	Internal static key that represents the order throughout its lifetime	SiteOrderKey
LinkType	Type of link identifying the relationship between orders. Values include: • Algo • AlgoC • AlgoL • AS • ASQ • ASH • ASR • OCO • OMA • SSE • SSEC • SSER • Stage-BKR • Stage-INT • Stage-UNK • UNM **Note**: In some instances an order can have more than one link type, separated by a comma (e.g. SSE, OCO).	String
Location	Order sender country and region code.	String
Long	Number of contracts bought (same as Buy)	Integer
Member	Member ID of the trader from the Member/Group/Trader ID	String
NetQty	Total quantity of contracts or shares executed for this fill	Integer
NetTicks	Total value of the fill in ticks	Integer

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
OpenClose OpenClose$	For market fills, whether the fill opened or closed a position; for injected fills, the source of an injected fill. Possible values include: **A**: XRisk Admin **C**: Close **D**: Detailed SOD **F**: FIFO **M**: Manual Fill **O**: Open **R**: Rollover **S**: SOD **Note:** The TOCOM Exchange changes the **OpenClose** value of a fill corresponding to a FIFO order as either **C** (Close) or **O** (Open).	String
OrderNo	Order number assigned by the exchange. For CME products, this property contains a different representation of the order number that matches the format used by other exchanges. To see the actual order number received from the CME, check the **ExOrderID** attribute.	String
OrderRestr	Order restriction on the order. For a list of possible values, see Order restrictions, on page 636.	String
OrderSource OrderSource$	Where the order originated: **0**: X_TRADER **1**: Autospreader Desktop **2**: Autotrader **3**: XTAPI **4**: Ext Source (Exchange) **5**: FA Licensed (Licensed FIX Adapter) **6**: FA Unlicensed (Unlicensed FIX Adapter) **7**: XT Lite (X_TRADER Lite) **8**: TT_TRADER **9**: Autospreader Engine **10**: Synthetic SE Engine **11**: TT API in X_TRADER Mode **12**: Algo SE **21**: TT API in Universal Login Mode **22**: ADL	Integer or String
OrderTag	User-definable text (15 char max) containing user-defined information included in the corresponding order profile.	String
OrderTraits	Traits associated with the order as a concatenated string containing the order status history, order flag, and order restriction.	String

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
OrderType	Type of order corresponding to the fill. Possible values include: **B**: OCO order **C**: CMO order **L**: Limit order **M**: Market order **O**: Batch order **Q**: Quote order **S**: Best Limit order (TOCOM only) **T**: Market To Limit order **X**: Cross order	String
Partial	Type of fill executed. Possible values include: **F**: Full fill **P**: Partial fill **Note:** A TT Eurex Gateway that enables Fast Fills does not set this attribute to "F" for partial Fast Fills. When your application later reconciles its fills with the Eurex Fill Server, one of the partial fills marked as "P" will be changed to "F" to indicate which fill completely filled the order.	String
Price Price$ Price# Price&	Fill price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
Price@	Fill price converted to the primary currency (as set in the Guardian currency tables). If you access this attribute before the currency table finishes loading, the attribute returns TT_INVALID_PRICE.	Decimal
Price[Currency]	Current market value of open positions in the currency whose code you specify (Currency). Valid only when the instrument has an attached **`TTOrderSet`**.	Decimal
Price^	Fill price converted to the native currency (as set in the Guardian currency tables)	String
Qty	Number of contracts contained in the fill	Integer
RiskAccount	ID for the risk account	String
Sell	Number of contracts sold (same as Short)	Integer
SenderSubID	Unique trader (or operator) ID used to distinguish between different traders when the current MGT routes orders for multiple traders. **Note:** Some exchanges, such as CME, require clients to provide this information when routing orders on behalf of multiple traders.	String (32 character limit)
Series Series$	Unique ID for the instrument, provided by the exchange	String
Series.objAttribute	Value of the specified object attribute for the instrument (**`TTInstrObj`**)	Varied

Table 23. TTDropHandler Get attributes

Attribute	Description	DataType
SeriesKey	Internal ID that uniquely identifies the instrument contained in the order	String
SessionId	Session ID for the fill	Integer
Short	Number of contracts sold (same as Sell)	Integer
SiteKey SiteOrderKey	Internal static key that represents the order throughout its lifetime	String
SOD	Whether the fill represents a start-of-day record	Boolean
SODType	Type of start-of-day message. Possible values include: • "Start of Day" • "Current" • "Correction"	String
Source	Source of the fill. For normal fills, the value contains a unique ID for the Fill server. For manual fills generated by an XTAPI application, the value contains an ID based on the system running the application.	String
Tag(x)	User-defined tag, where x is any integer.	String
Time Time$	Time the X_TRADER API sent the order to the exchange, in the form HH:MM:SS:mmm.	Time String
TimeExec TimeExec$	Time the exchanged matched the trade, in the form *HH*:*MM*:*SS*:*mmm*.	Time String
TimeLocalUpd TimeLocalUpd$	Time the X_TRADER API received the fill, in the form HH:MM:SS:mmm.	String
Trader	Trader ID of the trader from the Member/Group/Trader ID.	String
TTFillCode	Fill code associated with the fill. Possible values include: **1000**: Normal fill **1001**: Fast Fill **1002**: Reverse Trade **1003**: Adjustment **1004**: Clearing Error **1005**: Rollover	Integer
UserData(*x*)	User-defined tag, where *x* is any integer. Valid values: 0-1023	Integer
UserTag	User-definable text (15 char max) containing user-defined information included in the corresponding order profile.	String
Username	Trader login. The value differs based on the type of credentials used: • For MGT-based logins, the value contains the concatenation of the Member, Group, and Trader attributes. • For universal logins, the value contains the trader's login ID	String

Table 23. TTDropHandler Get attributes

Example

The following example uses the `Get` property to display the values for the specified **`TTFillObj`** attributes.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)
```

```
  Dim isValidNameArr As Array
  Dim attrValuesArr As Array
  Dim isValidList As Boolean
  Dim attrName As String
  Dim i As Integer
  i = 1

  'good practice to validate list before calling Get
  isValidNameArr = Me.m_TTFillObj.IsValidGet(objAttributes)

  ' code to verify attribute list omitted

  If isValidList = True Then
    attrValuesArr = Me.m_TTFillObj.Get(objAttributes)

    'Display each attribute and value
    For Each attrValue in attrValuesArr
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Attribute: {0} = {1}", attrName, attrValue)
      i = i + 1
      Next
    End For

  End If
End Sub
```

Related information

Property:IsValidGet, on page 107

Property:ReadProperties, on page 109

Property: Instrument

Purpose

Returns the instrument associated with the fill

Syntax

```
instrFilled = myFillObj.Instrument
```

Parameters

None

Return value

instrFilled	Instrument object associated with the fill Data type: `TTInstrObj`

Exceptions

Exception	Meaning
COM QI Error	Some external COM error occurred.
Invalid Fill Object	The specified pointer does not reference a valid **`TTFillObj`** object.
Invalid Instrument	The instrument has not yet been retrieved from the Price server. This error typically occurs when initially downloading fills during your application startup.
Unable to Create Instrument	Some internal error prevented the XTAPI from creating the TTInstrObj object.

Description

The `Instrument` property returns the instrument object (**`TTInstrObj`**) for the order that was filled. You can then access the object attributes of the returned instrument object to extract specific information about the instrument.

Example

The following subroutine uses the `Instrument` property to retrieve the instrument associated with the fill object selected in the application window.

```
Private Sub btnGetSet_Click(ByVal sender As Object,
                    ByVal e As System.EventArgs) Handles btnGetSet.Click
   If cmbPropMeth.SelectedItem = "Instrument" Then
       oInstr = oFill.Instrument
       oInstr.Open()
       oInstr.OrderSet = oOrderSetFill
       PrintInstr()
   End If

   If cmbPropMeth.SelectedItem = "ReadProperties" Then
       PrintFill(oFill.ReadProperties, PrintArg.PropsOnly)
   End If

   If cmbPropMeth.SelectedItem = "Confirm" Then
       oFill.Confirm(txtAcct.Text)
   End If
End Sub
```

Related information

TTInstrObj Class, on page 217

Property: IsValidGet

Purpose

Returns a **`TTFillObj`** object's attributes

Syntax

attrValues = myFillObj.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information

Property:Get, on page 97

Property:ReadProperties, on page 109

Property: ReadProperties

Purpose

Returns information about **TTFillObj** attributes you can access through the `Get` property

Syntax

attributes = myFillObj.ReadProperties

attrDescription = myFillObj.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **TTFillObj** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 97

Property:IsValidGet, on page 107

Method: Confirm

Purpose

Reassigns a fill to another trader's account

Syntax

`myFillObj.Confirm(traderAccount)`

Parameters

traderAccount	Trader account ID Data type: String

Return value

None

Exceptions

Exception	Meaning
Could not confirm fill	An internal system error occurred that prevented the XTAPI from confirming the fill.

Description

The `Confirm` method changes the accounting information for the fill by assigning the fill to another trader.

Example

The following subroutine uses the `Confirm` method to confirm the fills and change the account number to another account.

```
'This code also assumes that an order set enabled fill data and that the
'OnOrderFillData callback populated arrayOfFills with fill data
Private Sub ConfirmFills(arrayOfFills As Variant)

   Dim i As Integer

   'Iterate through the list of fills
   For i = 0 To arrayOfFills.Count - 1

      'create a fill object
      Dim fillobj As XTAPI.TTFillObj

      'get the fillobj from the array
      fillobj = arrayOfFills.Item(i)

      'Change the account to TTSim001001
      fillobj.Confirm("TTSim001001")
   Next

End Sub
```

Related information

None

13 TTGate Class

Chapter overview

The chapter describes the **`TTGate`** class. It provides an overview of the class and includes detailed information about its properties and methods.

In this chapter

Section	Page
TTGate Class class overview	113
Property:CustomerDefaults	117
Property:CustomerProfile	119
Property:Customers	118
Property:DebugAppInfo	119
Property:DebugLogLevel	120
Property:EnableOrderSend	123
Property:Get	124
Property:InstrData	127
Property:Instruments	128
Property:IsValidGet	129
Property:OrderProfiles	131
Property:OrderSetData	132
Property:OrderSets	133
Property:RapidFillDelivery	134
Property:ReadProperties	136
Method:AuditXTServices	137
Method:DestroyDownloadedFills	138
Method:DisableExtendedPrices	139
Method:EnableServerMode	141
Method:GetExchangeRate	143
Method:GetPrimaryCurrency	144
Method:OpenExchangeFills	145
Method:OpenExchangeOrders	146
Method:OpenExchangePrices	147
Method:SetExchangeLogon	148
Method:SetExchangeLogonEx	150
Method:StartNewSession	153
Method:StartXTrader	154
Method:XTAPITerminate	155
Event:OnExchangeMessage	156
Event:OnExchangeStateUpdate	157
Event:OnLoginFailed	159
Event:OnSessionRollMessage	160

Section	Page
Event:OnStatusUpdate	161

- Property:CustomerProfile

TTGate Class class overview

Overview

The **TTGate** class serves as a central repository for all XTAPI objects. Your application can iterate through the object collections or can access a specific object through the user-defined alias that was assigned when the object was created.

It also stores global variables that control XTAPI behavior. Additionally, **TTGate** enables applications to register to receive general exchange, TT Gateway, and X_TRADER® status messages.

When your application starts, the XTAPI automatically instantiates a **TTGate** object to handle basic object management. Instantiating additional **TTGate** objects returns pointers to the same **TTGate** object, as it is a singleton. You need not instantiate another **TTGate** for typical XTAPI application functionality.

Application Modes

An XTAPI application can operate in one of two modes: standard and server. By default, applications run in standard mode, but the **TTGate** class provides a mechanism to change modes, as described in Table 24.

Application mode	Description
Standard	When an application is run in standard mode, it shares the license of the X_TRADER that is running on the machine. Therefore, one or more XTAPI applications can be run in standard mode and no additional licenses are required.
Server	When an application is run in server mode, it consumes its own X_TRADER license and, therefore, does not require that X_TRADER be running. All XTAPI functionality is available to a server mode application except the ability to submit, change, and delete orders. You typically use server mode to monitor applications such as back office recorders.

Table 24. Application modes

When to create a TTGate object

While not required, you should consider instantiating a **TTGate** object for most XTAPI applications. Automated applications, such as those that automatically submit orders based on some logic, benefit from instantiating a **TTGate** object because a **TTGate** object is needed to register and receive updates from the exchanges. If you do not instantiate the object, your application cannot receive notifications when order and fill servers go down, which can then cause errors when the application tries to submit orders.

You must, however, instantiate a **TTGate** object in the following circumstances:

- When creating a server-mode application
- If you want to use rapid fill delivery
- If you want to access the customer defaults in X_TRADER

TTGate and .NET

If your application uses the .NET environment, you should always call the `XTAPITerminate` method before exiting.

Properties

Table 25 lists the **TTGate** properties.

Property	Description
CustomerDefaults	Returns customer defaults selected in X_TRADER

Table 25. TTGate Class properties

Property	Description
Property:CustomerProfile	Returns a customer profile matching specified customer defaults criteria
Customers	Returns the customer defaults created in X_TRADER
DebugAppInfo	Returns the XTAPI software version associated with an application
DebugLogLevel	Sets or returns the level of information written to the XTAPI debug log
EnableOrderSend	Gets or sets whether to allow the application to submit orders
Get	Returns a TTGate object's attributes
InstrData	Provides access to the properties of a currently-open instrument using its alias
Instruments	Returns a collection of aliased instruments in the application
IsValidGet	Returns whether an attribute name is a valid argument for the Get property
OrderProfiles	Returns a collection of order profiles with assigned aliases
OrderSetData	Provides access to the properties of a currently-open order set using its alias
OrderSets	Returns collection containing all of the order sets with assigned aliases
RapidFillDelivery	Gets or sets whether an application uses rapid fill delivery
ReadProperties	Returns information about TTGate attributes you can access through the Get property

Table 25. TTGate Class properties

Methods

Table 26 lists the `TTGate` methods.

Method	Description
AuditXTServices	Logs the current state of XT Services
DestroyDownloadedFills	Clears all fill and Start of Day (SOD) records from XTAPI memory
DisableExtendedPrices	Prevents the XTAPI from calculating implied prices
EnableServerMode	Allows the XTAPI to run in server mode
GetExchangeRate	Returns the exchange rate between two currencies
GetPrimaryCurrency	Returns the application's primary, or base, currency
OpenExchangeFills	Connects to fill servers of the specified exchange gateways
OpenExchangeOrders	Connects to order servers for the specified exchange gateways
OpenExchangePrices	Connects to price servers for the specified exchange gateways
SetExchangeLogon	Defines logon credentials for exchange gateway servers
SetExchangeLogonEx	Defines logon credentials for a specific exchange gateway server
StartNewSession	Forces a the application to perform a session rollover
StartXTrader	Starts the X_TRADER® application, if necessary
XTAPITerminate	Terminates the API application

Table 26. TTGate Class methods

Events

Table 27 lists the **`TTGate`** events.

Event	Triggered when...
OnExchangeMessage	Exchange gateway generates an informational message
OnExchangeStateUpdate	Gateway sends status updates for a connection
OnLoginFailed	Attempts to log into a TT Gateway fail
OnSessionRollMessage	New session is detected or a session roll is in progress
OnStatusUpdate	Status of X_TRADER® changes

Table 27. TTGate Class events

Property: CustomerDefaults

Purpose	Returns customer defaults selected in X_TRADER
Syntax	*customerProfiles* = myGate.CustomerDefaults
Parameters	None
Return value	*customerProfiles* — Array of customer profiles defined in X_TRADER Data type: Array of **ITTCustomerProfile** objects
Exceptions	None

Description

The `CustomerDefaults` property returns an array of customer defaults currently selected within the X_TRADER **Customer Defaults** window.

Note: The property might return the defaults in an order different than shown in the X_TRADER Customer Defaults window.

Example

The following example retrieves the customer defaults and loops through them.

```
Dim WithEvents oGate As New TTGate
Dim OCustDef(), cdObj As Object
'Fill in list of Customer Defaults
OCustDef = oGate.CustomerDefaults
  For Each cbObj In OCustDef
    Dim CustPro As ITTCustomerProfile
    CustPro = cbObj
    cmbCustDef.Items.Add(CustPro.Customer)
  Next
```

Related information

Property:Customer, on page 313

TTOrderProfile Property:CustomerDefaults, on page 315

Property: Customers

Purpose	Returns the customer defaults created in X_TRADER
Syntax	`customers = myGate.Customers`
Parameters	None
Return value	*customers* — Array of customer names defined in X_TRADER Data type: Array
Exceptions	None
Description	The `Customers` property returns the names of customer defaults currently defined within the X_TRADER Customer Defaults window. You can use this information for other operations that require customer information, such as submitting an order. **Note:** The property might return the defaults in an order different than shown in the window.
Example	The following code fragment uses the `Customers` property to loop through each set of customer defaults defined in X_TRADER.

```
Dim myGate As New TTGate
Dim Cust As Variant

For Each Cust in myGate.Customers
   ' ... omitted code to process customer defaults
Next Cust
```

Related information	**TTOrderProfile** Property:Customer, on page 313 **TTOrderProfile** Property:Customers, on page 316

Property: DebugAppInfo

Purpose	Returns the XTAPI software version associated with an application
Syntax	*appVersion* = myGate.DebugAppInfo
Parameters	None
Return value	*appVersion* — String value that contains the version information Data type: String
Exceptions	None
Description	This property returns the application name and the associated XTAPI software version. The value is returned as a string in the following format: *applicationName* using XTAPI *V*.*R*.*M*.*B* where *V*, *R*, *M*, and *B* represent the software version, release, modification, and build numbers, respectively.
Example	The following example displays the application information.

```
Sub PrintAppInfo()
   Dim myGateObj As New TTGate

    Debug.Print "XTAPI version: ", myGateObj.DebugAppInfo

    ' Code to print other information
    ' ...

End Sub
```

Related information	Property:DebugLogLevel, on page 120

Property: DebugLogLevel

Purpose

Sets or returns the level of information written to the XTAPI debug log

Syntax

```
logLevel = myGate.DebugLogLevel
myGate.DebugLogLevel = logLevel
```

Parameters

logLevel

Types of information to include in the debug log file, as a bitmask. Base hexadecimal values include:

- `LOG_ERROR (0x00)`
- `LOG_MINIMAL (0x01)`
- `LOG_MIN_API (0x03)`
- `LOG_DET_API (0x04)`
- `LOG_ORDERS (0x8)`
- `LOG_COM (0x10)`
- `LOG_COM_USE (0x20)`
- `LOG_COM_ERR (0x30)`
- `LOG_COM_EVENTS (0x40)`
- `LOG_ALL_COM (0x70)`
- `LOG_FILLS (0x100)`
- `LOG_DEBUG_WIN (0x1000)`

Data type: Integer

Return value

logLevel

Current debug log level as an integer

Data type: Integer

Exceptions

None

Description

This property specifies what types of information the XTAPI writes to a debug log file. If an application is not performing as expected, you can adjust the level of information written to the log file to help you isolate the cause.

Table 28 describes the different debug logging levels.

Enumeration	Description
LOG_ERROR	Logs only error messages
LOG_MINIMAL	Logs minimal generic debugging (low impact) information
LOG_MIN_API	Logs minimal API debugging information, such as open exchanges, server status, and so on. This value is the default setting for the XTAPI.
LOG_DET_API	Generates detailed API logging
LOG_ORDERS	Logs all orders submitted to an exchange
LOG_COM	Logs the creation and destruction of all COM objects and logs (fills and orders can generate a lot of log entries)
LOG_COM_USE	Logs each time XTAPI accesses a COM property
LOG_COM_ERR	Logs each time XTAPI creates or destroys COM objects, accesses COM object properties, or encounters COM errors.
LOG_COM_EVENTS	Logs all COM events
LOG_ALL_COM	Logs all COM information.

Table 28. Debug log levels

Enumeration	Description
LOG_FILLS	Logs all received fills
LOG_DEBUG_WIN	Logs all log information to the debug window
LOG_ALL	Logs everything **Warning:** Using this setting can overwhelm system resources and severely curtail throughput.

Table 28. Debug log levels

Note: Because writing extra information to log files can impact application performance, you should reduce the log level to the default level or lower as soon as possible, especially if you are debugging a production application.

Setting global debug log levels using an initialization file

You can also set the default debug log level in the runtime **XTAPI.ini** file for the application. The debug log level specified in this file applies globally to all objects created by the application. Several different objects contain the DebugLogLevel property for your convenience, so that you don't have to instantiate a **`TTGate`** object just to change the debug log level.

To use an initialization file for the application, create an **XTAPI.ini** file in the same directory as your application's executable file. In the file, add a line similar to the following (case-sensitive):

```
[DebugLogLevel]
  XTAPI_Application=8  # where XTAPI_Application is the application name
```

When your application starts, XTAPI reads this file and sets the debug logging level for the application. Because XTAPI reads the file only at startup, any changes you make to the DebugLogLevel property remain in effect until the application terminates or until you change the value of this property.

Location of debug log files

The XTAPI stores its debug log files in the **<root dir>:\tt\logfiles** directory with the naming convention:

```
XTAPI_YYYY-MM-DD_AppName_ProcessID.log
```

where:

- *YYYY-MM-DD* represents the file creation date
- *AppName* represents the name of the API application
- *ProcessID* is the ID associated with the process.

Format of debug logs

Each line in the debug log file contains three pipe-delimited (|) fields as follows:

```
Date | Message Source | Message
```

where:

- *Date* indicates when XTAPI wrote the message to the file
- *Message Source* identifies where the message originated

- *Message* contains the debug information

Example

The following example temporarily adds order logging for debugging purposes. It saves the current debug log level, changes the level, and finally restores the level to its original setting.

```
Sub PrintDebugInfo()
   Dim myGate As New TTGate
   Dim dLevel As Integer

   dLevel = myGate.DebugLogLevel    ' Save base log level
   Debug.Print "XTAPI version: ", myGate.DebugAppInfo
   Debug.Print "Debug Log Level: ", dLevel

   ' Change the log level to include all order information
   myGate.DebugLogLevel = dLevel + LOG_ORDERS
   Debug.Print "Debug Log Level: ", myGate.DebugLogLevel
End Sub
```

Related information

Property:DebugAppInfo, on page 119

Property: EnableOrderSend

Purpose	Gets or sets whether to allow the application to submit orders
Syntax	*allowSend* = myGate.EnableOrderSend myGate.EnableOrderSend = *allowSend*
Parameters	*allowSend* — Whether the application can submit orders to the gateway Data type: Boolean; Default: FALSE
Return value	*allowSend* — Whether the application can submit orders Data type: Boolean
Exceptions	None
Description	The `EnableOrderSend` property specifies whether an application is allowed to send orders to the connected exchanges. To limit the possibility of accidentally submitting an order, the XTAPI sets this value to FALSE.
Example	None
Related information	Method:OpenExchangeOrders, on page 146

Property: Get

Purpose

Returns a **TTGate** object's attributes

Syntax

```
attrValues = myGate.Get(attributes)
```

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **TTFillObj** object attribute

Description

The `Get` property returns the values for an arbitrary number of **TTGate** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single `ChangeMe` property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Table 29 lists the **TTGate** object attributes you can supply to the `Get` property. The Attribute column uses the following key:

- *exchg* is a connection name, as follows:
  - For single connection environments, use the gateway name (such as CME-A or Eurex).
  - For multi-connection environments, use the full connection name (such as CME·Citadel·TTORD·777·001) or at least enough of the name that uniquely identifies the connection.
- *server* is one of the following server types: Price, Order, or Fill

For example, to determine whether the Fill server on the LIFFE gateway is available, you specify **Available.LIFFE.Fill** or **Available.Eurex·Citadel·TTORD·001·001**, depending on the type of environment.

Attribute	Returned value	Return type
Available.exchg.server	Whether the server for a connection is available	Boolean
ClrMember.exchg	Clearing member for the trader associated with the specified connection	String
DefaultAcct.exchg	Default account for the trader currently logged into the specified connection	String
Exists.exchg.server	Whether a server on a connection exists	Boolean
FillServers	List of connected fill servers	Array
GatewayIP.exchg	IP address of the connection to which the application connects	String
Group.exchg	Group ID for the connection	String
HandoffTrader.exchg	Handoff trader (sometimes called the give-up trader) associated with the connection	String
HasLoginInfo.exchg	Whether the current login information is available for the indicated gateway	Boolean
IsTTTraderMode	Whether the application is accessing TT_TRADER™	Boolean
Member.exchg	Member ID for the current connection	String
Open.exchg	Whether the specified connection is currently open	Boolean
OrderServers	List of all connected order servers	Array
PriceServers	List of all connected price servers	Array
ServerUp.exchg.server	Whether a server specified connection is functional	Boolean
Trader.exchg	Trader ID for the specified connection	String
Username	Trader login. The value differs based on the trading product. For X_TRADER and XTAPI application traders, the value is the concatenation of the Member, Group, and Trader attributes.	String
WorkspacePath	Path of the current XTAPI open workspace, or an empty string or no workspace is open	String
XT.IsXTPro	Whether X_TRADER is running in Pro mode	Boolean
XTAPIServerMode	Whether XTAPI is running in server mode	Boolean
XTAPIServerMode$	Description of the XTAPI licensing status	String
XTAvailable	Whether the XTAPI can connect to X_TRADER	Boolean
XTAvailable$	Description of X_TRADER availability status	String
XTClosed	Whether X_TRADER is shut down	Boolean
XTClosed$	Description of the X_TRADER open/close status	String
XTConnected	Whether the application is currently connected to X_TRADER	Boolean
XTConnected$	Description of the X_TRADER connection status	String

Table 29. TTGate object attributes

Example

The following example checks the availability of each of the servers for a specified exchange.

```
Sub CheckServerInfo(exchSymbol as String)
   Dim myGateObj As New TTGate

   Dim attrCol as Variant
   Dim priceServer as String
   Dim orderServer as String
   Dim fillServer as String
   Dim serverList as String

   priceServer = "Available." + exchSymbol + ".Price"
   orderServer = "Available." + exchSymbol + ".Order"
   fillServer = "Available." + exchSymbol + ".Fill"

   serverList = priceServer + "," + orderServer + "," + fillServer

   attrCol = myGateObj.Get(serverList)

   Debug.Print "Price server status: ", attrCol(0)
   Debug.Print "Fill server status: ", attrCol(1)
   Debug.Print "Order server status: ", attrCol(2)
End Sub
```

Related information

Property:IsValidGet, on page 129

Property:ReadProperties, on page 136

Property: InstrData

Purpose

Provides access to the properties of a currently-open instrument using its alias

Syntax

```
propValues = myGate.InstrData(instrAlias, properties)
```

Parameters

instrAlias	Alias associated with the instrument Data type: String
properties	Array of strings representing the instrument properties Data type: Array

Return value

propValues	Array of strings representing the property values Data type: Array

Exceptions

Exception	Meaning
Invalid Instrument	Alias specified in *instrAlias* does not match a **TTInstrObj** object

Description

The `InstrData` property allows you to access the properties of any instrument (**TTInstrObj**) object currently open in the application using an alias associated with the instrument in Guardian or as set when the application created the instrument object.

The property returns an array where each element contains the value corresponding to the attribute in the same element of the input array. You must cast the returned values to the appropriate data types.

Example

The following example shows different methods of accessing instrument data using the `InstrData` property.

```
Private Sub GetInstrData()

    Dim arrayOfValues As Array
    arrayOfValues = m_TTGate.InstrData("MyFavContract","Exchange","Contract")

    Console.WriteLine("AliasName " + arrayOfValues.GetValue(0))
    Console.WriteLine("Exchange " + arrayOfValues.GetValue(1))
    Console.WriteLine("Contract " + arrayOfValues.GetValue(2))
End Sub
```

Related information

Property:Instruments, on page 128

Property: Instruments

Purpose	Returns a collection of aliased instruments in the application
Syntax	`instrs = myGate.Instruments`
Parameters	None
Return value	*instrs* — Array of instrument objects (**`TTInstrObj`**) defined in the XTAPI Data type: InstrumentCollection
Exceptions	None
Description	The `Instruments` property returns an array of all instrument objects in the application that have been assigned aliases. You can assign an alias for an instrument with the **`TTInstrObj`** object's `Alias` property.

Example

The following code fragment displays the contract associated with each instrument visible to the **`TTGate`** object.

```
Dim MyGate as New TTGate
Dim CurrInstrs As InstrumentCollection
Dim IterInstr As TTInstrObj

CurrInstrs = MyGate.Instruments

For Each IterInstr In CurrInstrs
  Debug.Print IterInstr.Contract
  'Additional Processing for Instrument
Next
End For
```

Related information

Property:InstrData, on page 127

Method:OpenExchangePrices, on page 147

InstrCollection Class, on page 29

TTInstrObj Class, on page 217

Property: IsValidGet

Purpose

Returns whether an attribute name is a valid argument for the `Get` property

Syntax

attrValues = myGate.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information

Property:Get, on page 124

Property:ReadProperties, on page 136

Property: OrderProfiles

Purpose	Returns a collection of order profiles with assigned aliases
Syntax	*profiles* = myGate.OrderProfiles
Parameters	None
Return value	*instrs* — Collection of **TTOrderProfile** objects Data type: OrderProfileCol
Exceptions	None
Description	The OrderProfiles property returns an array of all order profile objects in the application that have been assigned aliases. You can assign an alias for an order profile with the **TTOrderProfile** object's Alias property.
Example	The following code fragment returns a list of all order profiles for further processing:

```
Dim MyGateObj as New TTGate
Dim MyOrdProfiles as Variant
Dim IterOrdProf as TTOrderProfile

MyOrdProfiles=MyGateObj.OrderProfiles

For each IterOrdProf in MyOrdProfiles
 'Skipped code for processing order profiles
Next
```

Related information	None

Property: OrderSetData

Purpose

Provides access to the properties of a currently-open order set using its alias

Syntax

propValues = myGate.OrderSetData(*ordersetAlias*, *objAttributes*)

Parameters

orderSetAlias	User-defined alias associated with the order set Data type: String
objAttributes	Comma-separated string of object attributes Data type: String

Return value

propValues	Array of property values Data type: Array

Exceptions

Exception	Meaning
Invalid Order Set	Alias specified in *ordersetAlias* does not match an **OrderSet** object

Description

The OrderSetData property allows you to access the attributes of any order set (**TTOrderSet**) object currently open in the application using an alias defined by the application when the order set was instantiated.

The property returns an array where each element contains the value corresponding to the attribute in the same element of the input array. You must cast the returned values to the appropriate data types.

Example

The following subroutine uses the OrderSetData property to extract the **Buy** and **Sell** quantities of an order in the order set.

```
Private Sub GetOrderSetData()
  Dim arrayOfValues As Array

  'Returns the number of buy and sell orders in the Order Set
   arrayOfValues = m_TTGate.OrderSetData("BuyCnt,SellCnt")

  Dim doubleBuyCnt As Double = arrayOfValues.GetValue(0)
  Dim doubleSellCnt As Double = arrayOfValues.GetValue(1)

  Console.WriteLine("BuyCnt " + doubleBuyCnt.ToString())
  Console.WriteLine("SellCnt " + doubleSellCnt.ToString())

End Sub
```

Related information

None

Property: OrderSets

Purpose

Returns collection containing all of the order sets with assigned aliases

Syntax

```
orderSetCol = myGate.OrderSets
```

Parameters

None

Return value

orderSetCol	Collection of **`TTOrderSet`** objects Data type: OrderSetCol

Exceptions

None

Description

The `OrderSets` property returns an array of all order set objects in the application that have been assigned aliases. You can assign an alias for an order set with the **`TTOrderSet`** object's Alias property.

Example

The following example displays the aliases for each order set in the application.

```
Private Sub GetOrderSets()

        Dim ordersetcol As XTAPI.OrderSetCol
        ordersetcol = Me.m_TTGate.OrderSets()
        Dim orderSet As XTAPI.TTOrderSet

        For Each orderSet In ordersetcol
            Console.WriteLine("orderset Alias: " + orderSet.Alias)
        Next

End Sub
```

Related information

None

Property: RapidFillDelivery

Purpose

Gets or sets whether an application uses rapid fill delivery

Syntax

```
allowRapidFills = myGate.RapidFillDelivery
myGate.RapidFillDelivery = allowRapidFills
```

Parameters

allowRapidFills	Whether to support rapid fill delivery Data type: Boolean

Return value

allowRapidFills	Whether rapid fill delivery is enabled Data type: Boolean

Exceptions

None

Description

The `RapidFillDelivery` property indicates whether the XTAPI should receive fill information directly from the TT Gateway Order Server instead of the Fill Server. Normally, the XTAPI receives order updates from an Order Server and forwards the order update to the Fill Server as well as all interested CoreAPI clients. If the TT Gateway Order Server determines that the order update was the result of a fill, it caches a copy of the fill and forwards a complete fill update to interested CoreAPI clients

By enabling this property, your application can receive fills more quickly than going through the TT Gateways. The XTAPI uses pre-cached fill updates. Therefore, when this property is enabled, the fill updates have a null fill key (**`TTFillObj` FillKey** attribute).

Caution: Enabling this property increases the likelihood that fill updates might be received before their corresponding order acknowledgements. Consequently, your application might receive the fills more quickly, but can make it more difficult to match a fill with an order.

Because updates from a TT Gateway are asynchronous, an application cannot be assured of receiving an order acknowledgement from the gateway before receiving the fill, particularly in a busy, fast-paced market. With the `RapidFillDelivery` property disabled, a fill can still occasionally arrive before an order acknowledgement. With the property enabled, these events are more likely to occur.

Tip: If the application does not need to access information from the original order, enabling this property can update your position more rapidly. If the application accesses information from the original order after receiving a fill, you should not enable this property.

Example

The following example shows an excerpt from an initialization routine that enables and verifies the `RapidFillDelivery` property.

```
Private Sub InitializeSession()
   Dim myGateObj AS New TTGate

   myGateObj.RapidFillDelivery = TRUE   ' Enable RapidFillDelivery
```

```
    ' Display initialized settings
    Debug.Print "RapidFillDelivery: ", myGateObj.RapidFillDelivery
End Sub
```

Related information None

Property: ReadProperties

Purpose

Returns information about **TTGate** attributes you can access through the `Get` property

Syntax

attributes = myGate.ReadProperties

attrDescription = myGate.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **TTGate** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 124

Property:IsValidGet, on page 129

Method: AuditXTServices

Purpose	Logs the current state of XT Services
Syntax	`myGate.AuditXTServices`
Parameters	None
Return value	None
Exceptions	None
Description	The `AuditXTServices` method writes the current state of XT Services to the XTAPI log file (***<root_dir>*:\tt\logfiles\xtservices.exe_*YYYY-DD-MM*.log**). This method writes status information similar to the following:

```
05-05-2012 14.25:58.0837|Logging Initialized
05-05-2012 14.25:58.0837|XTServices Audit Start
05-05-2012 14.25:58.0837|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:-1
05-05-2012 14.25:58.0837|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:5488
05-05-2012 14.25:58.0837|XTServices Audit End
05-05-2012 14.27:42.0384|Registering Service: TypeName:XT TypeKey:XT_OUT2
05-05-2012 14.27:42.0400|Updating Service: TypeName:XT TypeKey:XT_OUT2 Proc:6104
05-05-2012 14.28:34.0009|Registering Client:TypeName:XT TypeKey:XT_OUT2 Proc:-1
05-05-2012 14.28:41.0400|Registering Client:TypeName:XT TypeKey:XT_OUT2 Proc:-1
05-05-2012 14.28:42.0775|Updating Client: TypeName:XT TypeKey:XT_OUT2 Proc:4924
05-05-2012 14.28:47.0728|XTServices Audit Start
05-05-2012 14.28:47.0744|Available Service: TypeName:XT TypeKey:XT_OUT2 Proc:6104
05-05-2012 14.28:47.0744|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:-1
05-05-2012 14.28:47.0744|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:5488
05-05-2012 14.28:47.0744|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:-1
05-05-2012 14.28:47.0744|Available Client: TypeName:XT:XT_OUT2 TypeKey:XT Proc:4924
05-05-2012 14.28:47.0744|XTServices Audit End
05-05-2012 14.41:13.0322|Unregistering Client: TypeName:XT TypeKey:XT_OUT2 Proc:4924
05-05-2012 14.41:13.0431|Unregistering Client: TypeName:XT TypeKey:XT_OUT2 Proc:-1
05-05-2012 14.41:18.0947|Unregistering Service: TypeName:XT TypeKey:XT_OUT2
05-05-2012 14.41:24.0197|Unregistering Client: TypeName:XT TypeKey:XT_OUT2 Proc:5488
05-05-2012 14.41:24.0322|Unregistering Client: TypeName:XT TypeKey:XT_OUT2 Proc:-1
```

Example	None
Related information	None

Method: DestroyDownloadedFills

Purpose

Clears all fill and Start of Day (SOD) records from XTAPI memory

Syntax

`myGate.DestroyDownloadedFills(`*`exchangeList`*`)`

Parameters

exchangeList	Comma-separated list of TT Gateways whose fills and SOD records you want to clear Data type: String

Return value

None

Exceptions

Exception	Meaning
Invalid exchange	One or more of the TT Gateways specified in *exchangeList* does not match a known exchange

Description

The `DestroyDownloadedFills` method removes from memory the fill and start-of-day position information received from one or more TT Gateways. You can use this method to reduce the memory footprint of your application after the fills from the gateway finish downloading. The XTAPI deletes fills for all connections associated with a gateway.

Note: After you call `DestroyDownloadedFills`, TT Gateway failover cannot be supported. Also, P&L will be incorrect if you close existing order sets.

Example

The following code fragment deletes all fills associated with the CME and LIFFE gateways:

```
Dim ExchList as String
Dim MyGateObj as New TTGate

ExchList="CME,LIFFE"

MyGateObj.DestroyDownloadedFills(ExchList)
```

Related information

None

Method: DisableExtendedPrices

Purpose

Prevents the XTAPI from calculating implied prices

Syntax

```
myGate.DisableExtendedPrices
```

Parameters

None

Return value

None

Exceptions

Exception	Meaning
Connection already initialized, cannot disable extended price calculation	The application has already registered with a price, order, or fill server, so extended prices can no longer be disabled.

Description

The `DisableExtendedPrices` method allows an application to disable automatic calculation of implied prices when not provided by an exchange. By default, the XTAPI automatically calculates implied prices and subscribes to all contracts it needs for the implieds. For example, if you subscribe to the front month LIBOR contract on the LIFFE exchange, the XTAPI also subscribes to all contracts and spreads.

Note: Calling this method must be an XTAPI application's first interaction with the XTAPI. Also, once disabled, you cannot enable extended prices later during the application session.

You might consider calling this method if you do not use implied prices, or if you want to improve performance at the expense of not having access to some prices. Implied prices provided directly from the exchanges are still available to your application.

When you disable extended prices, calling the **`TTInstrObj`** `CalculateTTImplieds` property throws an exception.

Example

None

Related information

None

Method: EnableAutoSessionRollover

Purpose

Enables or disables automatic session rollover for a TT Gateway

Syntax

`myGate.EnableAutoSessionRollover(`*`exch`*`, `*`doRollover`*`)`

Parameters

exch	Name of the exchange to rollover Data type: String
doRollover	Whether to permit automatic session rollover Data type: Boolean; Default: TRUE

Return value

None

Exceptions

Exception	Meaning
Invalid exchange	The *exch* parameter does not represent a valid TT Gateway name.

Description

Note: Currently, the XTAPI supports this functionality only for the TT LME Gateway.

The `EnableAutoSessionRollover` method determines whether the XTAPI should automatically trigger an internal XTAPI session rollover when a TT Gateway session changes. When the XTAPI starts a new session, it resets all of the positions for the TT Gateway, as if your application manually called the **`TTGate`** `StartNewSession` method.

The XTAPI enables this functionality by default. If you disable automatic rollover, the XTAPI still sends triggers the `OnSessionRollMessage` event with the message, `SESSION_ROLL_DETECTED`, to let the application know that the TT Gateway session changed.

Example

None

Related information

Method:StartNewSession, on page 153

Event:OnSessionRollMessage, on page 160

Method: EnableServerMode

Purpose

Allows the XTAPI to run in server mode

Syntax

```
myGate.EnableServerMode
```

Parameters

None

Return value

None

Exceptions

Exception	Meaning
Unable to Enable Server Mode, XTAPI Session Closed	The XTAPI could not find an available license to start in server mode.

Description

The `EnableServerMode` method allows an XTAPI application to connect to the TT Gateways when X_TRADER is not running. Server mode allows an application to run even though X_TRADER is not. When an application runs in server mode, it has access to prices, fills, and order updates; however, it cannot submit orders. You typically use this mode for applications that monitor and collect information from the gateways for storage in back-office databases.

When running in server mode:

- Each XTAPI application uses a separate X_TRADER Pro license instead of sharing the existing license used by the X_TRADER application.
- Guardian logins are not supported, so any X_ADMIN windows should be closed.
- Because X_ADMIN risk-checking is not available, XTAPI applications are not permitted to submit orders.

Note: When you enable server mode, the XTAPI automatically uses a new X_TRADER license. If you have insufficient licenses, the method fails. Also, if the application takes the last license and you later try to start X_TRADER, X_TRADER will not start.

To run in sever mode, an XTAPI application's first interaction with the XTAPI must be the `EnableServerMode` method call.

Example

Invoking the `EnableServerMode` method on a **`TTGate`** object opened `WithEvents` fails because the event notification represents the first interaction with the **`TTGate`** object. In the following example, therefore, `myGate` is declared `WithEvents`. Next, *`tmpGate`* is instantiated without `WithEvents` and is then used to invoke the `EnableServerMode` method. Finally, `tmpGate` is assigned to `myGate`.

```
Dim WithEvents myGate As TTGate

Private Sub Form_Load()
  Dim tmpGate As New TTGate

  On Error resume next

  tmpGate.EnableServerMode ' attempt to start server mode
  If err.number = 0 then ' did we succeed
    Set myGate = tmpGate
  End If
End Sub
```

Related information None

Method: GetExchangeRate

Purpose

Returns the exchange rate between two currencies

Syntax

```
rate = myGate.GetExchangeRate(fromCurrency, toCurrency)
```

Parameters

fromCurrency
toCurrency

ISO-standard symbol for the source and target currencies. Possible values include:

- ARS
- ATS
- AUD
- BEF
- BRL
- CAD
- CHF
- DEM
- ESP
- EUR
- FRF
- GBP
- HKD
- INR
- ITL
- JPY
- KRW
- NLG
- PTE
- RUR
- SGD
- USD
- XBA
- XEU

Data type: String

Note: TT Guardian manages the list of supported currencies, so might differ from the default list.

Return value

None

Exceptions

None

Description

The `GetExchangeRate` method returns the current rate of exchange between two currencies. The rate returns the factor for converting an amount from the first currency to the second.

The XTAPI retrieves the exchange rate from the Guardian currency table.

Example

The following code fragment displays the default currency in the debug window.

```
Dim myGateObj As New TTGateObj
Dim exRate As Decimal
Dim fromCurr As String
Dim toCurr As String

fromCurr = "EUR"
toCurr = "USD"

Debug.Print "Exchange Rate from ", fromCurr, " to ", toCurr, "is : ", _
  myGateObj.GetExchangeRate(fromCurr, toCurr)
```

Related information

Method:GetPrimaryCurrency, on page 144

Method: GetPrimaryCurrency

Purpose	Returns the application's primary, or base, currency
Syntax	`currency = myGateObj.GetPrimaryCurrency`
Parameters	None
Return value	*currency* — ISO-standard symbol for the source and target currencies. Possible values include: • ARS • CHF • HKD • PTE • ATS • DEM • INR • RUR • AUD • ESP • ITL • SGD • BEF • EUR • JPY • USD • BRL • FRF • KRW • XBA • CAD • GBP • NLG • XEU Data type: String **Note:** TT Guardian manages the list of supported currencies, so might differ from the default list.
Exceptions	None
Description	The `GetPrimaryCurrency` method returns the symbol representing the base currency used in the application. The XTAPI retrieves the currencies from the Guardian currency table.
Example	The following code fragment displays the default currency in the debug window. `Dim myGateObj As New TTGateObj` **`Debug.Print "Primary Currency: ", myGateObj.GetPrimaryCurrency`**
Related information	Method:GetExchangeRate, on page 143

Method: OpenExchangeFills

Purpose	Connects to fill servers of the specified exchange gateways
Syntax	`myGateObj.OpenExchangeFills(`*`exchangeList`*`)`
Parameters	*exchangeList* — Comma-separated list of exchanges whose fill servers to which you want to logon. You can specify "*" to indicate all gateways Data type: String
Return value	None
Exceptions	None
Description	The `OpenExchangeFills` method opens a connection to the fill servers on the specified gateways. You must first call the `SetExchangeLogon` or `SetExchangeLogonEx` methods to provide the login credentials for the Fill server.
Example	The following subroutine uses the `OpenExchangeFills` method to connect to the fill servers of all connected gateways during application startup.

```
Public Sub Starter_Show()
    Set MyGate = New TTGate     ' Create global access point
    With MyGate
        .DebugLogLevel = iLogLevel  ' Set debug logging
        lblAppInfo = .DebugAppInfo  ' Get application info and display it
        .EnableOrderSend = True     ' Allow orders to be send (default)
        Call .OpenExchangePrices("*")  ' Provide access to all gateways
        Call .OpenExchangeOrders("*")  ' Provide access to all gateways
        Call .OpenExchangeFills("*")   ' Provide access to all gateways
    End With
End Sub
```

Related information

Method:EnableServerMode, on page 141

Method:OpenExchangeOrders, on page 146

Method:OpenExchangePrices, on page 147

Method:SetExchangeLogon, on page 148

Method: OpenExchangeOrders

Purpose	Connects to order servers for the specified exchange gateways
Syntax	`myGateObj.OpenExchangeOrders(`*exchangeList*`)`
Parameters	*exchangeList* — Comma-separated list of exchanges whose order servers to which you want to logon. You can specify "*" to indicate all gateways Data type: String
Return value	None
Exceptions	None
Description	The `OpenExchangeOrders` method opens a connection to the order servers on the specified exchanges. You must first call the `SetExchangeLogon` or `SetExchangeLogonEx` method to provide the login credentials for the Order server.
Example	The following subroutine uses the `OpenExchangeOrders` method to connect to the order servers of all connected gateways during application startup.

```
Public Sub Starter_Show()
    Set MyGate = New TTGate     ' Create global access point
    With MyGate
        .DebugLogLevel = iLogLevel  ' Set debug logging
        lblAppInfo = .DebugAppInfo  ' Get application info and display it
        .EnableOrderSend = True     ' Allow orders to be send (default)
        Call .OpenExchangePrices("*")  ' Provide access to all gateways
        Call .OpenExchangeOrders("*")  ' Provide access to all gateways
        Call .OpenExchangeFills("*")   ' Provide access to all gateways
    End With
End Sub
```

Related information

Property:EnableOrderSend, on page 123

Method:EnableServerMode, on page 141

Method:OpenExchangeFills, on page 145

Method:OpenExchangePrices, on page 147

Method:SetExchangeLogon, on page 148

Method: OpenExchangePrices

Purpose	Connects to price servers for the specified exchange gateways
Syntax	`myGateObj.OpenExchangePrices(exchangeList)`
Parameters	*exchangeList* — Comma-separated list of exchanges whose price servers to which you want to logon. You can specify "*" to indicate all gateways Data type: String
Return value	None
Exceptions	None
Description	The `OpenExchangePrices` method opens a connection to the price servers on the specified exchanges. You must first call the `SetExchangeLogon` or `SetExchangeLogonEx` method to provide the login credentials for the Price server.
Example	The following subroutine uses the `OpenExchangePrices` method to connect to the price servers of all connected gateways during application startup.

```
Public Sub Starter_Show()
    Set MyGate = New TTGate     ' Create global access point
    With MyGate
        .DebugLogLevel = iLogLevel  ' Set debug logging
        lblAppInfo = .DebugAppInfo  ' Get application info and display it
        .EnableOrderSend = True     ' Allow orders to be send (default)
        Call .OpenExchangePrices("*")  ' Provide access to all gateways
        Call .OpenExchangeOrders("*")  ' Provide access to all gateways
        Call .OpenExchangeFills("*")   ' Provide access to all gateways
    End With
End Sub
```

Related information

Property:Instruments, on page 128

Method:EnableServerMode, on page 141

Method:OpenExchangeFills, on page 145

Method:OpenExchangeOrders, on page 146

Method:SetExchangeLogon, on page 148

Method: SetExchangeLogon

Purpose

Defines logon credentials for exchange gateway servers

Syntax

```
myGate.SetExchangeLogon(exchange, member, group, trader,
password, [clearMbr], [defAcct], [handOffGroup], [handOffMbr],
[handOffTrader])
```

Parameters

exchange	Name of desired exchange gateway Data type: String
member	Member ID assigned by Exchange or TTORD ID Information Data type: String
group	Group ID assigned by Exchange or TTORD ID Information Data type: String
trader	Trader ID assigned by Exchange or TTORD ID Information Data type: String
password	Password associated with the trader login Data type: String
clearMbr	Optional member firm of a clearing house through which trades are settled Data type: String
defAcct	Optional default account that is passed to TT Guardian during the universal login process Data type: String
handOffGroup	Optional group ID of the member for whom the trade is executed Data type: String
handOffMbr	Optional member ID of the member for whom the trade is executed Data type: String
handOffTrader	Optional trader ID of the member for whom the trade is executed Data type: String

Return value

None

Exceptions

Exception	Meaning
Invalid Exchange	The *exchange* parameter does not contain a supported exchange symbol.
Invalid Member	The member or *handOffMbr* parameters do not contain a valid member ID.

Exception	Meaning
Invalid Group	The group or *handOffGroup* parameters do not contain a valid group ID.
Invalid Trader	The trader or *handOffTrader* parameters do not contain a valid trader ID.
Invalid Password	The *password* parameter does not contain the correct password.
Unable to Set Logon Info	The XTAPI could not log in to the gateway.

Description

The `SetExchangeLogon` method establishes the logon credentials for a TT Gateway on an exchange. The XTAPI uses the credentials when opening a connection to any Price, Order, or Fill server on the gateway.

If the gateway uses separate login credentials for different servers, you must use the `SetExchangeLogonEx` method to associate the credentials with a specific server IP address.

Note: Call this method only when using the XTAPI in server mode. If X_TRADER® is currently active, calling this method when not in server mode disconnects the XTAPI from X_TRADER®.

Example

The following subroutine calls the `SetExchangeLogon` method to store the login credentials supplied in the application window

```
Private Sub cmdSetLogon_Click()
   Dim sGWInfo As String

   On Error Resume Next
   Call Gate.SetExchangeLogon(cboGW.Text, txtMember.Text, txtGroup.Text, _
                              txtTrader.Text, txtPassword.Text)
   If Err.Number = 0 Then
      sGWInfo = Left$(cboGW.Text & Space$(10), 10) & "Up  "
      Call frmStarter.GWs.Add(sGWInfo, cboGW.Text)
      Call frmStarter.cboGW.AddItem(cboGW.Text)
      Call frmStarter.cboGWinfo.AddItem(sGWInfo)
      Unload Me
   Else
      Call MsgBox("Login Error:" & vbLf & Err.Description & vbLf & _
         "encountered during login into " & cboGW.Text & " as " & -
         txtMember.Text & txtGroup.Text & txtTrader.Text, vbSystemModal, _
         "Login Error")
   End If
End Sub
```

Related information

Method:SetExchangeLogonEx, on page 150

Event:OnLoginFailed, on page 159

Method: SetExchangeLogonEx

Purpose

Defines logon credentials for a specific exchange gateway server

Syntax

myGate.SetExchangeLogonEx(*exchange*, *member*, *group*, *trader*, *password*, [*clearMbr*], [*defAcct*], [*handOffGroup*], [*handOffMbr*], [*handOffTrader*],[serverIP])

Parameters

exchange	Name of desired exchange gateway Data type: String
member	Member ID assigned by Exchange or TTORD ID Information Data type: String
group	Group ID assigned by Exchange or TTORD ID Information Data type: String
trader	Trader ID assigned by Exchange or TTORD ID Information Data type: String
password	Password associated with the trader login Data type: String
clearMbr	Optional member firm of a clearing house through which trades are settled Data type: String
defAcct	Optional default account that is passed to TT Guardian during the universal login process Data type: String
handOffGroup	Optional group ID of the member for whom the trade is executed Data type: String
handOffMbr	Optional member ID of the member for whom the trade is executed Data type: String
handOffTrader	Optional trader ID of the member for whom the trade is executed Data type: String
serverIP	IP address of the appropriate server Data type: String

Return value

None

Exceptions

Exception	Meaning
Invalid Exchange	The *exchange* parameter does not contain a supported exchange symbol.

Exception	Meaning
Invalid Member	The member or *handOffMbr* parameters do not contain a valid member ID.
Invalid Group	The group or *handOffGroup* parameters do not contain a valid group ID.
Invalid Trader	The trader or *handOffTrader* parameters do not contain a valid trader ID.
Invalid Password	The *password* parameter does not contain the correct password.
Unable to Set Logon Info	The XTAPI could not log in to the gateway.

Description

The `SetExchangeLogonEx` method establishes the logon credentials for a specific server for a TT Gateway on an exchange. The credentials are used when opening a connection to a Price, Order, or Fill server on the gateway. You use this method when a gateway requires different credentials for individual servers on the gateway. The additional *serverIP* parameter identifies the IP address of the server.

If the gateway uses the same credentials for all of the servers, you can use the `SetExchangeLogon` method to provide generic logon credentials.

Note: Call this method only when using the XTAPI in server mode. If X_TRADER® is currently active, calling this method when not in server mode disconnects the XTAPI from X_TRADER®.

Example

The following subroutine calls the `SetExchangeLogonEx` method to store the login credentials supplied in the application window

```
Private Sub cmdSetLogon_Click()
   Dim sGWInfo As String

   On Error Resume Next
   Call Gate.SetExchangeLogonEx(cboGW.Text, txtMember.Text, txtGroup.Text,_
                        txtTrader.Text, txtPassword.Text,,,,,, txtIP.Text)
   If Err.Number = 0 Then
      sGWInfo = Left$(cboGW.Text & Space$(10), 10) & "Up  "
      Call frmStarter.GWs.Add(sGWInfo, cboGW.Text)
      Call frmStarter.cboGW.AddItem(cboGW.Text)
      Call frmStarter.cboGWinfo.AddItem(sGWInfo)
      Unload Me
   Else
      Call MsgBox("Login Error:" & vbLf & Err.Description & vbLf & _
         "encountered during login into " & cboGW.Text & " as " & -
         txtMember.Text & txtGroup.Text & txtTrader.Text, vbSystemModal, _
         "Login Error")
   End If
End Sub
```

Related information

Method:SetExchangeLogon, on page 148

Event:OnLoginFailed, on page 159

Method: SetUniversalLogin

Purpose

Logs into Guardian with specified universal credentials

Syntax

```
exchList = myGate.SetUniversalLogin(username, password)
```

Parameters

username	Username for the universal login Data type: String
password	Password for the specified username Data type: String

Return value

exchList	Names of exchanges authorized for the specified `username` Data type: Array of strings

Exceptions

Exception	Meaning
Invalid user name or password	The password contains an empty value.
User must be in server mode	The XTAPI application is not running in server mode.
Login Failed - message	TT Guardian Login failed. If TT Guardian provides a message, the exception includes the message text; otherwise message contains "Username / Password not recognized".
XTAPI initialization failure	An internal error occurred.
Log in attempt timed out	The XTAPI application did not receive a response from TT Guardian within the timeout range (20 seconds).
XTAPI is already configured with a login. You can't use UL with any other run configuration	You called this method after XTAPI already initialized with another mode.

Description

The `SetUniversalLogin` method logs into TT Guardian with a universal login and then logs into each of the TT Gateways associated with the login. The method returns a list of exchange names for all TT Gateways to which it tried to connect. Also, the XTAPI triggers the `OnExchangeStateUpdate` event when connecting to each TT Gateway.

Note: Call this method only when using the XTAPI in server mode. If X_TRADER® is currently active, calling this method when not in server mode disconnects the XTAPI from X_TRADER®.

Example

None

Related information

Method:SetExchangeLogonEx, on page 150

Method: StartNewSession

Purpose

Forces a the application to perform a session rollover

Syntax

```
result = myGate.StartNewSession(exch)
```

Parameters

exch	Name of the exchange Data type: String

Return value

result	Whether the method call executes synchronously or asynchronously. Possible values include: **-1** : Error **0** : Synchronous response **1** : Asynchronous response Data type: Integer

Exceptions

Exception	Meaning
Invalid Exchange	The *exch* parameter value does not represent a valid TT Gateway name.

Description

Note: Currently, the XTAPI supports this functionality only for the TT LME Gateway.

The `StartNewSession` method forces the XTAPI application to simulate a session rollover for the specified TT Gateway. When called, this method performs the following actions for the specified TT Gateway:

- Triggers the `OnSessionRollMessage` event with the `SESSION_ROLL_START` state
- Inverts all fills
- Deletes all fills
- Re-downloads all fills
- Triggers the `OnSessionRollMessage` event with the `SESSION_ROLL_END` state

Example

None

Related information

Method:EnableAutoSessionRollover, on page 140

Event:OnSessionRollMessage, on page 160

Method: StartXTrader

Purpose	Starts the X_TRADER® application, if necessary
Syntax	`myGate.StartXTrader`
Parameters	None
Return value	None
Exceptions	None
Description	The `StartXTrader` method starts the XTAPI application if it is not currently running. For an API application to submit orders, XTAPI must be running and logged into the desired exchange. XTAPI requires manual logons. Therefore, if the `StartXTrader` method needs to start the XTAPI, someone must manually supply the logon information when the XTAPI **Logon** dialog box is displayed.
Example	None
Related information	Method:XTAPITerminate, on page 155

Method: XTAPITerminate

Purpose	Terminates the API application
Syntax	`myGate.XTAPITerminate`
Parameters	None
Return value	None
Exceptions	None
Description	The `XTAPITerminate` method terminates the current application and performs system garbage-collection cleanup operations. This method ensures that all services and objects are closed and deleted from memory. TT recommends that you close all production applications with this method instead of just ending the application. **Note:** If your API application is a .NET application, you must close the application with this method.
Example	None
Related information	Method:StartXTrader, on page 154 **MEAPI TTMarketExplorer** Method:MEAPITerminate, on page 541

Event: OnExchangeMessage

Triggered when

Exchange gateway generates an informational message

Syntax

```
Sub myGate_OnExchangeMessage(sExchange, sTimeStamp, sInfo, sText)
```

Parameters

sExchange	Name of the exchange sending the message Data type: String
sTimeStamp	Time, to the millisecond, when the message was sent Data type: String
sInfo	Type of message. Possible values include: • ALERT • INFO • ERROR • WARNING Data type: String
sText	Text of the message from the exchange Data type: String

Description

The `OnExchangeMessage` event is triggered when a message is received from an exchange gateway. The messages are generated by an exchange or TT Gateway and can represent varying situations, depending on the exchange and the different servers attached to the **`TTGate`** object.

Registering for this event allows the application to perform a set of tasks based on the message itself. As these messages are primarily informational in nature, you typically include code to output data or perform debugging tasks.

For more information about the types of messages, refer to the appropriate *TT Gateway System Administration Manual*.

Example

The following example uses the `OnExchangeMessage` event to display the exchange and time information when an exchange sends a message.

```
'm_TTGate_OnExchangeMessage - If the exchange sends out a message to alert all
'users, this event will fire and relay the message to XTAPI applications
Private Sub m_TTGate_OnExchangeMessage(ByVal sExchange As String, ByVal
sTimeStamp As String, ByVal sInfo As String, ByVal sText As String) Handles
m_TTGate.OnExchangeMessage

  'Output some of the values received from the callback
  Console.WriteLine("pStrExchange: " + sExchange + "\n" + "pStrTimestamp: " +
                    sTimeStamp + "\n")

End Sub
```

Related information

Event:OnExchangeStateUpdate, on page 157

Event: OnExchangeStateUpdate

Triggered when

Gateway sends status updates for a connection

Syntax

`Sub myGate_OnExchangeStateUpdate(exchName, srvType, isOpen, isUp)`

Parameters

exchName
Name of the connection that updated

Data type: String

srvType
Type of server. Possible values include:

- Price
- Order
- Fill

For order book downloads, the values include:

- Order (Downloading)
- Fill (Downloading)

Data type: String

isOpen
Whether the application established the connection

Data type: Boolean

isUp
Whether the server specified in `srvType` is active

Data type: Boolean

Description

The `OnExchangeStateUpdate` event allows you to execute code when some event occurs related to a connection's status. This event is passed the name of the connection and server, and information about the active server. Using the `isOpen` and `isUp` values, you can determine the status of the connection, as described in the following table.

isOpen	IsUp	Gateway status
FALSE	FALSE	Establishing a connection
FALSE	TRUE	N/A
TRUE	FALSE	Connection is down
TRUE	TRUE	Connection is active

Table 30. Gateway server statuses

Note: Your application should not attempt to place orders until all orders and fills have been downloaded and the order and fill servers are Up and Open.

Example

The following example shows how to respond to an `OnExchangeStateUpdate` event.

```
Private Sub m_TTGate_OnExchangeStateUpdate(ByVal sExchange As String, ByVal
sText As String, ByVal bOpened As Integer, ByVal bServerUp As Integer) Handles
m_TTGate.OnExchangeStateUpdate

   Dim strGatewayState As String
   strGatewayState = "Unknown status"
```

```
   'Run through all of the bOpened and bServer possible values
   'and what the values mean
   If (bOpened = 0 And bServerUp = 0) Then
      strGatewayState = " Know about GatewayServer, but not connected"
   ElseIf (bOpened = 0 And bServerUp = 1) Then
      strGatewayState = " N/A"
   ElseIf (bOpened = 1 And bServerUp = 0) Then
      strGatewayState = " Gateway Server is down"
   ElseIf (bOpened = 1 And bServerUp = 1) Then
      strGatewayState = " Gateway Server is up"
   End If

   'Output all of the values everytime this method is called
   Console.WriteLine("sExchange: " + sExchange + " sText: " + sText)
   Console.WriteLine(" bOpened: " + bOpened.ToString + "  bServerUp: " +
                     bServerUp.ToString)

   ' .
   ' . additional code omitted
   ' .
End Sub
```

Related information Event:OnStatusUpdate, on page 161

Event: OnLoginFailed

Triggered when	Attempts to log into a TT Gateway fail
Syntax	`Sub myGate_OnLoginFailed(adminExchange, adminMember, adminGroup, adminTrader, msg, loginSrc)`

Parameters

adminExchange	Name of the connection that failed Data type: String
adminMember	Administrator member ID assigned by Exchange or TTORD ID Information Data type: String
adminGroup	Administrator group ID assigned by Exchange or TTORD ID Information Data type: String
adminTrader	Administrator trader ID assigned by Exchange or TTORD ID Information Data type: String
msg	Failure message returned by the exchange Data type: String
loginSrc	Source of the login attempt. Possible values include: • **LOGIN_XT**, for attempts made on behalf of X_TRADER® • **LOGIN_XTAPI**, for attempts made by an XTAPI application Data type: `enumLoginSource`

Description

The `OnLoginFailed` event allows you to execute code when an attempt to establish a connection fails. The XTAPI throws this event when calls to `SetExchangeLogon` or `SetExchangeLogonEx` fails.

Note: The XTAPI throws stops throwing this event when the number of failed attempts matches the maximum number of login attempts specified in TT Guardian.

Example

None

Related information

Method:SetExchangeLogon, on page 148

Method:SetExchangeLogonEx, on page 150

Event: OnSessionRollMessage

Triggered when	New session is detected or a session roll is in progress
Syntax	`Sub myGate_OnSessionRollMessage(`*`sExchange`*`, `*`sMessage`*`, `*`eRollState`*`)`

Parameters

sExchange	Name of the exchange Data type: String
eMessage	Session rollover message. Possible values include: • "Gateway session roll start" • "Gateway session roll end" • "Gateway session roll was detected" Data type: String
eRollState	State of the session rollover. Possible values include: • SESSION_ROLL_START • SESSION_ROLL_END • SESSION_ROLL_DETECTED Data type: `enumSessionRollState`

Description

Note: Currently, the XTAPI supports this functionality only for the TT LME Gateway.

The `OnSessionRollMessage` event is triggered when the TT Gateway starts a new trading session or when a session roll is underway. An instrument connection is needed to guarantee the session roll is detected.

Example None

Related information

Method:EnableAutoSessionRollover, on page 140

Method:StartNewSession, on page 153

Event: OnStatusUpdate

Triggered when

Status of X_TRADER® changes

Syntax

```
Sub myGate_OnStatusUpdate(statusChange, description)
```

Parameters

statusChange

Current status of X_TRADER®. Possible values include:

1 = Connected
2 = Closed
4 = Login updated
8 = Customer defaults updated
16= Unavailable
32= Available
64= Server mode
128= Currency table updated
256= Loading a workspace
512= Saving a workspace
1024= TT_TRADER mode
2048= Closing all windows

Data type: Integer

description

Description of the status change

Data type: String

Description

The `OnStatusUpdate` event is executed when X_TRADER® receives a status update. The event provides an integer identifying the status and a description of the status change. By checking the value of the *`statusChange`* parameter, an application can take additional actions when a status change occurs. For example, if the application receives a "Customer defaults updated" event (*`statusChange`* = **8**), it could update its window interface.

The following table describes the possible statuses.

statusChange	Status	Description
1	Connected	The X_TRADER API is connected to X_TRADER®.
2	Closed	The X_TRADER API is not connected to X_TRADER®, either because X_TRADER® is not running or for some other issue like a username mismatch.
4	Login updated	X_TRADER® is logged into a Gateway
8	Customer Defaults updated	The X_TRADER API received Customer defaults; occurs the first time you log in or if the user changed the Customer defaults
16	Unavailable	The X_TRADER API is not connected X_TRADER® with a Pro License. **Note**: This status can occur when X_TRADER® has a Pro license but is currently not running, because X_TRADER API cannot determine the type of license.

statusChange	Status	Description
32	Available	The X_TRADER API is connected to an X_TRADER® with a Pro license.
64	XTAPI Server Mode	X_TRADER® API is running in server mode
128	Currency Table updated	Guardian's currency table has been updated
256	X_TRADER® Pro is loading a workspace.	X_TRADER® is currently loading a workspace from disk. The string passed is the path of the workspace.
512	X_TRADER® Pro is saving a workspace.	X_TRADER® is currently saving a workspace to disk. The string passed is the path of the workspace.
1024	XTAPI is running in TT TRADER mode	X_TRADER® is currently running in TT_TRADER mode.
2048	XT Close All Windows	X_TRADER® is in the process of closing all of its windows.

If multiple status change events occur simultaneously, the X_TRADER API writes a single message to the log file that contains the sum of the status updates, but sends a series of individual events to the client application. So, for example, if a customer's login and default information were updated, a status of 12 is written to the log file, but two `OnStatusUpdate` events of 4 and 8 are sent to the application.

The X_TRADER API automatically checks the X_TRADER product version before connecting to a running X_TRADER instance. Consequently, after the version check, the **`TTGate`** object receives an initial `OnStatusUpdate` event with a status of 18 (Closed and Unavailable). If it passes the version check, the X_TRADER API tries to connect to X_TRADER. If successful, the **`TTGate`** object receives an `OnStatusUpdate` event with a status of 33 (Connected and Available).

Note: For applications written for versions of XTAPI earlier than 7.17, you need to take note of this timing change.

Example

The following example displays the current status of X_TRADER.

```
Private Sub myGate_OnStatusUpdate(ByVal statusChange As Long, _
                                  ByVal description As String)
   Select Case statusChange
   Case 1 ' XT Connected
      Debug.Print "Found X_TRADER..."
   Case 2 ' XT Disconnected.
      Debug.Print "X_TRADER® Closed."
   Case 4 ' Logged into a gateway.
      Debug.Print "X_TRADER® Login information updated."
   Case 8 ' X_TRADER® Customer default were updated.
      Debug.Print "X_TRADER® Customer Default updated."
   Case 16 ' X_TRADER® unavailable.
      Debug.Print "X_TRADER® Pro™ is not available."
   Case 32 ' X_TRADER® available.
      Debug.Print "X_TRADER® Pro is available."
   Case 64 ' X_TRADER® API is running in server mode (No X_TRADER).
      Debug.Print "X_TRADER® API is running in server mode."
   Case 128 ' Currency table updated
      Debug.Print = "Guardian's currency table has been updated."
   End Select
End Sub
```

Related information

Event:OnExchangeStateUpdate, on page 157

14 TTHotKeyNotify Class

Chapter overview

The chapter describes the **`TTHotKeyNotify`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTHotKeyNotify class overview	164
Method:AddHotKey	165
Method:DelHotKey	167
Event:OnHotKey	168

TTHotKeyNotify class overview

Overview

The **`TTHotKeyNotify`** class allows you to define and manage key combinations to perform application related tasks. For example, you could create a function that lists all of a trader's working orders and then define the **`CTRL-SHIFT-L`** key to call that function.

Caution: A hot key affects the entire Windows session, not just your application windows. If you specify a key commonly-used in applications that might also be active, the keystrokes in that application will be executed in your XTAPI application. For example, suppose you create a spacebar hotkey to cancel an order. If you use a mail program while your XTAPI application is running on the same system, every time you press the spacebar when writing an email, your application will cancel an order. Consequently, TT recommends that you use hot keys only when your system is dedicated to your XTAPI application.

Properties

None

Methods

Table 31 lists the **`TTHotKeyNotify`** methods.

Method	Description
AddHotKey	Defines a Windows hot key that executes application functionality
DelHotKey	Deletes a hot key definition and its corresponding event

Table 31. TTHotKeyNotify methods

Events

Table 32 lists the **`TTHotKeyNotify`** events.

Event	Triggered when...
OnHotKey	User presses a predefined hot key

Table 32. TTHotKeyNotify events

Method: AddHotKey

Purpose

Defines a Windows hot key that executes application functionality

Syntax

```
myHotKey.AddHotKey(keyName, keyChar, keyMod)
```

Parameters

keyName	Name to identify the key function Data type: String; Max length: 255 characters
keyChar	ASCII representation of the desired keystroke. For non-ASCII characters, refer to the operating system key codes. Data type: String
keyMod	Bitmask representing the key modifiers. The value is a bitmask based on the following values: **1** = **ALT** key **2** = **CTRL** key **4** = **SHIFT** key **8** = **WIN** key Data type: Integer

Return value

None

Exceptions

Exception	Meaning
Unable to Add Hot Key	One or more of the parameters is invalid.

Description

The `AddHotKey` property creates a Windows operating system hot key that can perform a series of application logic. When you define a hot key, the XTAPI triggers an `OnHotKey` event when a user presses the specified key or key combination. When the event is triggered, your custom application code is executed. You can also specify one or more key modifiers for a key, such as **CTRL-SHIFT-S** or **ALT-B**.

Caution: A hot key affects the entire Windows session, not just your application windows. If you specify a key commonly-used in applications that might also be active, the keystrokes in that application will be executed in your XTAPI application. For example, suppose you create a spacebar hotkey to cancel an order. If you use a mail program while your XTAPI application is running on the same system, every time you press the spacebar when writing an email, your application will cancel an order. Consequently, TT recommends that you use hot keys only when your system is dedicated to your XTAPI application.

Example

The following code fragment uses the `AddHotKey` method to assign a **CTRL-SHIFT-D** hot key to delete orders.

```
Dim myHotKeyNotify As TTHotKeyNotify

myHotKeyNotify.AddHotKey("DeleteOrder", "D", 6) ' CTRL-Shift-D

Private Sub myHotKeyNotify_OnHotKey(ByVal keyName As String)
   ' ... omitted code to execute when user presses CTRL-Shift-D
   ' ...
```

```
End Sub
```

Related information

Method:DelHotKey, on page 167
Event:OnHotKey, on page 168

Method: DelHotKey

Purpose

Deletes a hot key definition and its corresponding event

Syntax

`myHotKey.DelHotKey(keyName)`

Parameters

keyName	Name to identify the key function Data type: String

Return value

None

Exceptions

Exception	Meaning
Unable to Add Hot Key	One or more of the parameters is invalid.

Description

The `DelHotKey` method deletes a previously defined hot key from your application. When deleted, the XTAPI no longer triggers `OnHotKey` events for the hot key.

Note: If you specify a *keyName* that has not been created, the method performs no action and returns no errors.

Example

The following code fragment deletes the previously-defined hot key named **DeleteOrder**.

```
Dim myHotKeyNotify As TTHotKeyNotify

myHotKeyNotify.DelHotKey("DeleteOrder")
```

Related information

Method:AddHotKey, on page 167

Event:OnHotKey, on page 168

Event: OnHotKey

Triggered when	User presses a predefined hot key
Syntax	`Sub myHotKey_OnHotKey(`*keyName*`)`
Parameters	*keyName* — Name of the hot key pressed Data type: String
Description	The XTAPI triggers the `OnHotKey` event when a user presses a hot key defined by the `AddHotKey` property. You can include application logic in the callback function to execute the desired code when a user presses the key.
Example	The following code fragment defines an `OnHotKey` callback to execute code when a user presses the **CTRL-SHIFT-D** key.

```
Dim hotKeys As TTHotKeyNotify

hotKeys.AddHotKey("DeleteOrder", "D", 6) ' CTRL-Shift-D

Private Sub hotKeys_OnHotKey(ByVal keyName As String)
   ' ... omitted code to execute when CTRL-Shift-D is pressed
   ' ...
End Sub
```

Related information	Method:AddHotKey, on page 165 Method:DelHotKey, on page 167

15 TTInstrNotify Class

Chapter overview

The chapter describes the **`TTInstrNotify`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTInstrNotify class overview	170
Property:DebugLogLevel	173
Property:DeliverAllPriceUpdates	176
Property:EnableDepthUpdates	178
Property:EnableOrderSetUpdates	181
Property:EnablePriceUpdates	182
Property:InstrEnableUpdates	185
Property:InstrFilter	187
Property:Instrument	190
Property:IsValidUpdateFilter	193
Property:PriceList	195
Property:UpdateFilter	197
Method:AttachInstrument	200
Method:AttachOrderSet	202
Method:DetachInstrument	204
Method:DetachOrderSet	206
Event:OnInstrumentExpiryChanged	207
Event:OnNotifyDepthData	208
Event:OnNotifyFound	210
Event:OnNotifyNotFound	211
Event:OnNotifyUpdate	212
Event:OnOrderSetUpdate	214
Event:OnPriceListUpdate	215

TTInstrNotify class overview

Overview

The **`TTInstrNotify`** class defines objects that monitor price updates from an exchange for instruments (**`TTInstrObj`**) and order sets (**`TTOrderSet`**). To register for price updates, you instantiate one or more **`TTInstrNotify`** objects and attach the instruments your application wants to monitor to the notifiers. When the XTAPI receives price updates from an exchange (CBOT, ICE, etc.), the notifiers trigger events that your application can use to perform processing logic when an instrument of interest updates.

Figure 33 illustrates how price updates are processed through the instrument notifiers. When your XTAPI application receives price updates from an exchange gateway, the instrument notify objects listen for changes in various attributes for attached instruments. Each notifier can monitor changes for one or more instruments, and multiple notifiers can listen for different changes for the same instrument.

In this example, Notifier A monitors only one instrument, Instr A, while the other notifiers monitor changes for multiple instruments. Instr C is monitored by three different notifiers (B, C, and D), each of which could be listening for changes in different instrument attributes.

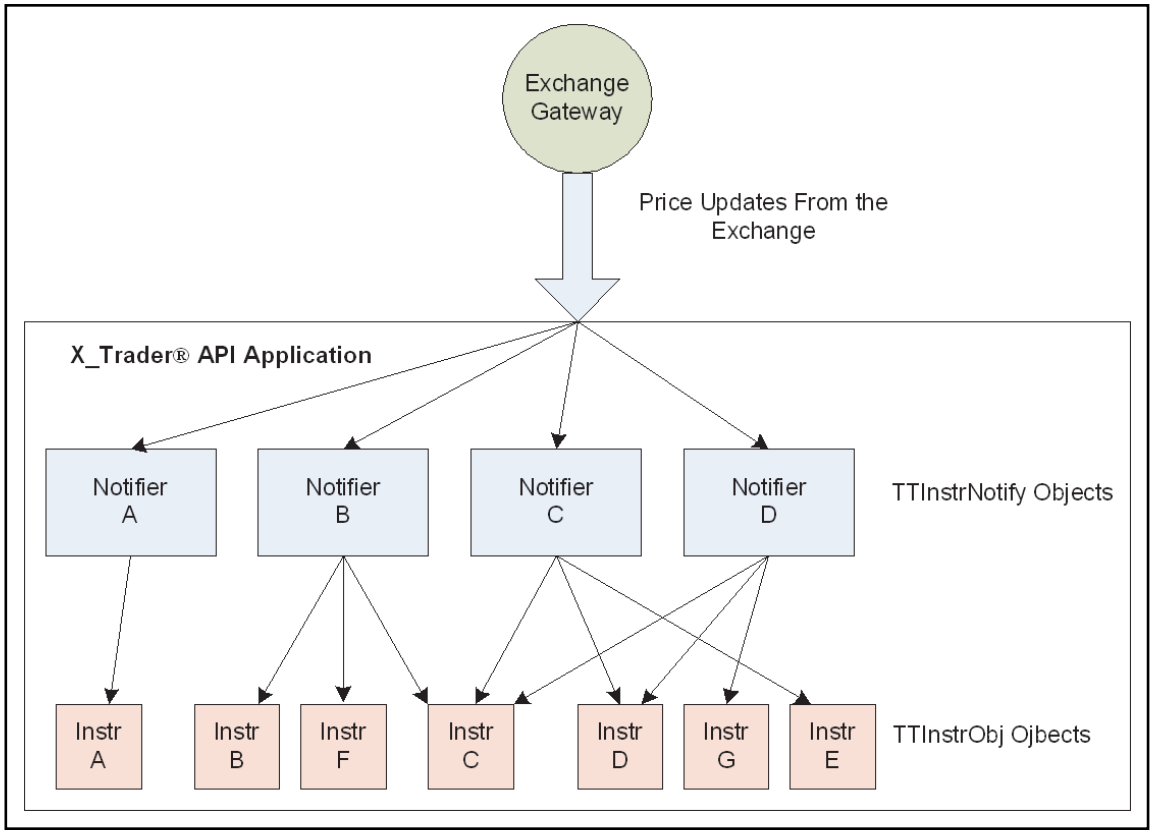


Figure 33. Processing price updates through instrument notifiers

Note: This chapter uses this illustration throughout this chapter to show how the various filter and update properties work.

Types of notifiers

When you create an instrument notifier, you can register for the types of updates for which you want to be notified. You can use a notifier to monitor the following types of price updates:

- Changes to the inside market (potentially-coalesced prices)
- Changes to the inside market (uncoalesced prices)
- Changes to the market depth

You can enable any or all of these for each notifier. However, as a practical matter, if you monitor the market depth, you need not also monitor the inside market.

Types of update filters

By default, the XTAPI delivers updates when any price-related attribute on an instrument changes. Because applications typically are not interested in all price-related changes, the XTAPI provides two types of filters to limit the events to which the application receives updates:

- Notifier filter, which allows you to specify which price updates you want the notifier to monitor for all attached instruments.
- Instrument filter, which allows you to specify which price updates to watch on an instrument-specific basis.

Properties

Table 34 lists the **`TTInstrNotify`** properties.

Property	Description
DebugLogLevel	Returns information about TTInstrNotify attributes you can access through the Get property
DeliverAllPriceUpdates	Sets or returns whether to deliver all price updates
EnableDepthUpdates	Sets or returns whether to monitor market depth changes for the instrument
EnableOrderSetUpdates	Sets or returns whether to send events when market data changes for orders in the order set
EnablePriceUpdates	Sets or returns whether to monitor inside-market price updates for the instrument
InstrEnableUpdates	Enables or disables callbacks for an instrument or order set
InstrFilter	Sets or returns a custom update filter for an instrument
Instrument	Returns the instrument associated with this notifier
IsValidUpdateFilter	Identifies whether an object attribute can be used to filter instrument updates
PriceList	Returns a collection of price updates that occurred since the PriceList was previously accessed
UpdateFilter	Identifies the criteria for filtering instrument updates

Table 34. TTInstrNotify properties

Methods

Table 35 lists the **`TTInstrNotify`** methods.

Method	Description
AttachInstrument	Attaches an instrument to the notifier
AttachOrderSet	Attaches an order set to the notifier
DetachInstrument	Detaches an instrument from a notifier
DetachOrderSet	Detaches an order set to the notifier

Table 35. TTInstrNotify methods

Events

Table 36 lists the **`TTInstrNotify`** events.

Event	Description
OnInstrumentExpiryChanged	Expiry for an instrument changes
OnNotifyDepthData	Price changes occur in an instrument's market depth
OnNotifyFound	`TTInstrObj` Open or Retrieve methods validate the existence of an instrument on the exchange
OnNotifyNotFound	`TTInstrObj Open` or `Retrieve` methods fail to validate an instrument
OnNotifyUpdate	Changes occur on the inside market for instrument attributes
OnOrderSetUpdate	Changes occur for an order set
OnPriceListUpdate	Price changes are added to the non-coalesced price list

Table 36. TTInstrNotify events

Property: DebugLogLevel

Purpose

Returns information about **TTInstrNotify** attributes you can access through the Get property

Syntax

```
logLevel = myInstrNotifer.DebugLogLevel
myInstrNotifer.DebugLogLevel = logLevel
```

Parameters

logLevel

Types of information to include in the debug log file, as a bitmask. Base hexadecimal values include:

- LOG_ERROR (0x00)
- LOG_MINIMAL (0x01)
- LOG_MIN_API (0x03)
- LOG_DET_API (0x04)
- LOG_ORDERS (0x8)
- LOG_COM (0x10)
- LOG_COM_USE (0x20)
- LOG_COM_ERR (0x30)
- LOG_COM_EVENTS (0x40)
- LOG_ALL_COM (0x70)
- LOG_FILLS (0x100)
- LOG_DEBUG_WIN (0x1000)

Data type: Integer

Return value

logLevel

Current debug log level as an integer

Data type: Integer

Exceptions

None

Description

This property specifies what types of information the XTAPI writes to a debug log file. If an application is not performing as expected, you can adjust the level of information written to the log file to help you isolate the cause.

Table 37 describes the different debug logging levels.

Enumeration	Description
LOG_ERROR	Logs only error messages
LOG_MINIMAL	Logs minimal generic debugging (low impact) information
LOG_MIN_API	Logs minimal API debugging information, such as open exchanges, server status, and so on. This value is the default setting for the XTAPI.
LOG_DET_API	Generates detailed API logging
LOG_ORDERS	Logs all orders submitted to an exchange
LOG_COM	Logs the creation and destruction of all COM objects and logs (fills and orders can generate a lot of log entries)
LOG_COM_USE	Logs each time XTAPI accesses a COM property
LOG_COM_ERR	Logs each time XTAPI creates or destroys COM objects, accesses COM object properties, or encounters COM errors.
LOG_COM_EVENTS	Logs all COM events
LOG_ALL_COM	Logs all COM information.

Table 37. Debug log levels

Enumeration	Description
LOG_FILLS	Logs all received fills
LOG_DEBUG_WIN	Logs all log information to the debug window
LOG_ALL	Logs everything **Warning:** Using this setting can overwhelm system resources and severely curtail throughput.

Table 37. Debug log levels

Note: Because writing extra information to log files can impact application performance, you should reduce the log level to the default level or lower as soon as possible, especially if you are debugging a production application.

Setting global debug log levels using an initialization file

You can also set the default debug log level in the runtime **XTAPI.ini** file for the application. The debug log level specified in this file applies globally to all objects created by the application. Several different objects contain the DebugLogLevel property for your convenience, so that you don't have to instantiate a **`TTGate`** object just to change the debug log level.

To use an initialization file for the application, create an **XTAPI.ini** file in the same directory as your application's executable file. In the file, add a line similar to the following (case-sensitive):

```
[DebugLogLevel]
  XTAPI_Application=8  # where XTAPI_Application is the application name
```

When your application starts, XTAPI reads this file and sets the debug logging level for the application. Because XTAPI reads the file only at startup, any changes you make to the DebugLogLevel property remain in effect until the application terminates or until you change the value of this property.

Location of debug log files

The XTAPI stores its debug log files in the **<root dir>:\tt\logfiles** directory with the naming convention:

```
XTAPI_YYYY-MM-DD_AppName_ProcessID.log
```

where:

- *YYYY-MM-DD* represents the file creation date
- *AppName* represents the name of the API application
- *ProcessID* is the ID associated with the process.

Format of debug logs

Each line in the debug log file contains three pipe-delimited (|) fields as follows:

```
Date | Message Source | Message
```

where:

- *Date* indicates when XTAPI wrote the message to the file
- *Message Source* identifies where the message originated

- *Message* contains the debug information

Example

The following example temporarily adds order logging for debugging purposes. It saves the current debug log level, changes the level, and finally restores the level to its original setting.

```
Sub PrintDebugInfo()
   Dim myInstrNotifer As New TTInstrNotify
   Dim dLevel As Integer

   dLevel = myInstrNotifer.DebugLogLevel    ' Save base log level
   Debug.Print "XTAPI version: ", myInstrNotifer.DebugAppInfo
   Debug.Print "Debug Log Level: ", dLevel

   ' Change the log level to include all order information
   myInstrNotifer.DebugLogLevel = dLevel + LOG_ORDERS
   Debug.Print "Debug Log Level: ", myInstrNotifer.DebugLogLevel
End Sub
```

Related information

None

Property: DeliverAllPriceUpdates

Purpose	Sets or returns whether to deliver all price updates
Syntax	`isDelivered = myInstrNotifer.DeliverAllPriceUpdates` `myInstrNotifer.DeliverAllPriceUpdates = isDelivered`
Parameters	*isDelivered* — Whether to deliver all price updates Data type: Boolean; Default: FALSE
Return value	*isDelivered* — Whether all price updates are delivered Data type: Boolean
Exceptions	None

Description

The `DeliverAllPriceUpdates` property indicates whether the notifier delivers non-coalesced price updates. Most trading applications rely on the current prices, so do not need to incur the overhead associated with tracking every price change. If you are developing an application, such a tick charting application, that needs access to all price changes, you can enable this property to get the non-coalesced prices.

By default, the XTAPI sends callbacks for registered price updates, which might be coalesced. Based on the frequency of instrument price changes on the exchange and the ability of your application to handle price updates, the normal updates might capture all of the price changes. If the instrument prices are very volatile or you must get every price change, you should enable this property.

Note: If price coalescing is enabled in the exchange gateway or if the exchange sends coalesced prices, an XTAPI application still might not receive every price update.

Also, enabling this property does not replace normal updates, so your application receives both coalesced and non-coalesced prices.

In Figure 38, Notifier C enabled the `DeliverAllPriceUpdates` property. The notifier stores non-coalesced prices and triggers `OnPriceListUpdate` events for instruments C, D, and E. When you access the `PriceList` property, the XTAPI returns an **`ITTPriceList`** object with all of the price updates received since the `PriceList` property was last checked. Also, because Instr C is attached to both Notifier B and Notifier D, your application receives callbacks from these notifiers for coalesced price updates (Notifier B) and market depth updates (Notifier D). Similarly, Notifier D generates callbacks when market depth updates occur for Instr D.

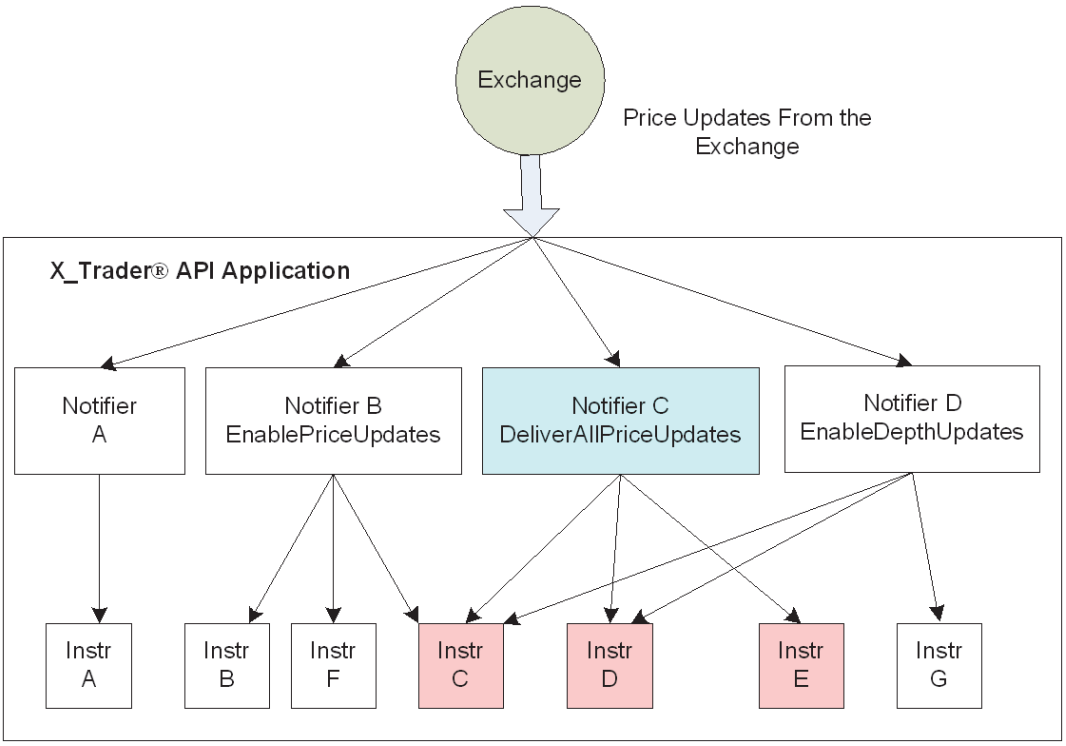


Figure 38. TTInstrNotify DeliverAllPriceUpdates processing

When this property is enabled, the XTAPI automatically instantiates an **`ITTPriceList`** object that stores all price updates that occurred since the prices were previously retrieved. The non-coalesced price updates are delivered in the `OnPriceListUpdate` event.

Caution: If no filter is defined in either the `InstrFilter` or `UpdateFilter` properties, changes in any and all instrument attributes generate callback events

If you enable this property, the instrument notifier delivers non-coalesced prices for every instrument attached to it. You cannot override delivery for individual instruments in the notifier by enabling `InstrEnableUpdates`.

Caution: If you want to disable coalesced price notification for one or more instruments, you must attach them to a separate instrument notifier with the `DeliverAllPriceUpdates` property disabled.

Example None

Related information

Property:EnableDepthUpdates, on page 178
Property:EnablePriceUpdates, on page 182
Property:InstrEnableUpdates, on page 185
Event:OnNotifyUpdate, on page 212

Property: EnableDepthUpdates

Purpose	Sets or returns whether to monitor market depth changes for the instrument
Syntax	`isEnabled = myInstrNotifer.EnableDepthUpdates` `myInstrNotifer.EnableDepthUpdates = isEnabled`
Parameters	*isEnabled* — Whether to enable depth updates Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether depth updates are enabled Data type: Boolean
Exceptions	None
Description	The `EnableDepthUpdates` property indicates whether the notifier delivers updates for price changes in the market depth for all instruments attached to the notifer.When enabled, the XTAPI monitors prices changes in the update messages from the exchange. The market depth updates are delivered in the `OnNotifyDepthData` event. In Figure 39, Notifier D has enabled the `EnableDepthUpdates` property. The notifier generates `OnNotifyDepthData` events whenever it receives price updates for instruments C,D, and G. Also, because Instr C is attached to both Notifier B and Notifier C, your application receives callbacks from these notifiers for coalesced price updates (Notifier B) and uncoalesced price updates (Notifier C). Similarly, Notifier C generates callbacks when market depth updates occur for Instr D.

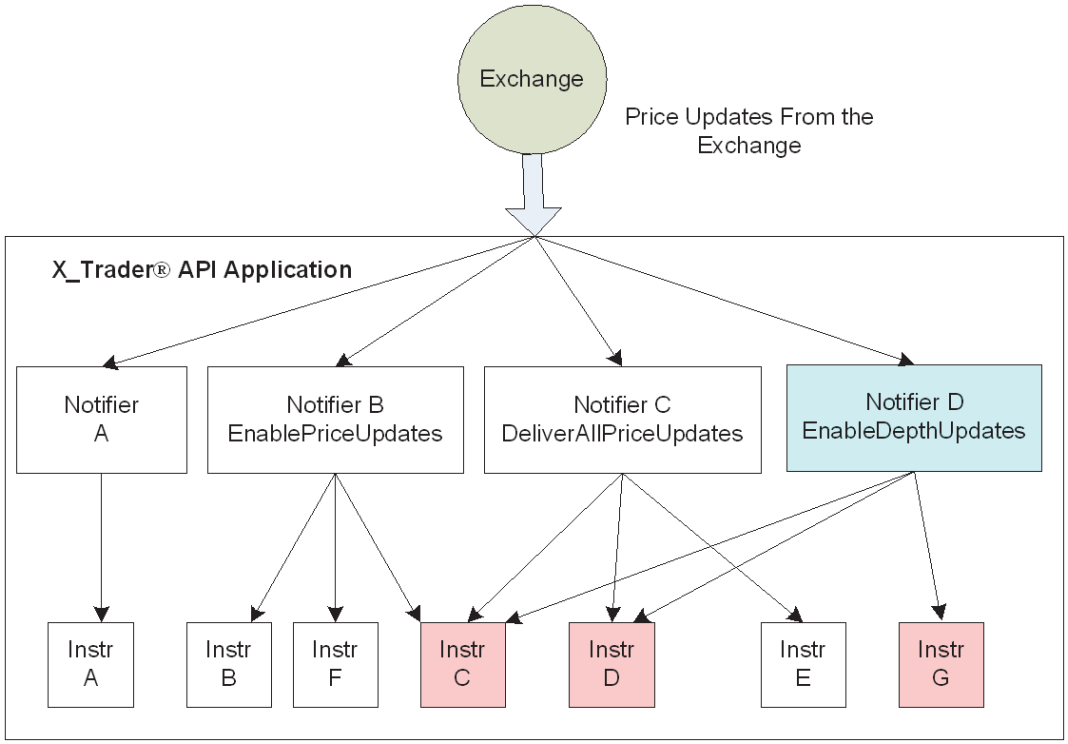


Figure 39. TTInstrNotify EnableDepthUpdates processing

The `InstrFilter` property specifies the fields that trigger a callback for your application.

If you disable this property, your application receives no depth updates, even if you have specified an update filter.

Caution: If no filter is defined in either the `InstrFilter` or `UpdateFilter` properties, changes in any and all instrument attributes generate callback events.

This property works independently of the `DeliverAllPriceUpdates` and `EnablePriceUpdates` properties. If you enable multiple properties, the XTAPI triggers events for each. So, for example, if you enable all three of these properties and a price update occurs for the **Last** price, your application will receive the three corresponding events. All of the events would contain information about the price change.

Example

The following routine uses the `EnableDepthUpdates` property to enable or disable depth updates based on a setting in an application window.

```
Private Sub enableDepthCheckBox_CheckedChanged(ByVal sender As Object,
             ByVal e As System.EventArgs) Handles
             enableDepthCheckBox.CheckedChanged
   ' Determine check box state.
   If (enableDepthCheckBox.Checked) Then
      ' Enable depth if check box is checked.
      m_TTInstrNotify.EnableDepthUpdates = TRUE
   Else
      ' Disable depth if check box is unchecked.
      m_TTInstrNotify.EnableDepthUpdates = FALSE

      ' Clear Bid and Ask list boxes.
      Me.bidDepthListBox.Items.Clear()
      Me.askDepthListBox.Items.Clear()
```

```
   End If
End Sub
```

Related information

Property:DeliverAllPriceUpdates, on page 176
Property:EnableOrderSetUpdates, on page 181
Property:EnablePriceUpdates, on page 182
Property:InstrEnableUpdates, on page 185
Event:OnNotifyDepthData, on page 208

Property: EnableOrderSetUpdates

Purpose

Sets or returns whether to send events when market data changes for orders in the order set

Syntax

```
isEnabled = myInstrNotifer.EnableOrderSetUpdates
myInstrNotifer.EnableOrderSetUpdates = isEnabled
```

Parameters

isEnabled	Whether to enable order set updates Data type: Boolean; Default: TRUE

Return value

isEnabled	Whether order set updates are enabled Data type: Boolean

Exceptions

None

Description

The `EnableOrderSetUpdates` property manages callbacks for changes for all order sets attached to the notifier. When enabled, the XTAPI monitors order set changes in the exchange update messages from the exchange.

Example

The following routine uses the `EnableOrderSetUpdates` property to enable or disable order set updates based on a setting in an application window.

```
Private Sub enableOrderSetCheckBox_CheckedChanged(ByVal sender As Object,
                              ByVal e As System.EventArgs) Handles
                              enableOrderSetCheckBox.CheckedChanged
   ' Determine check box state.
   If (enableOrderSetCheckBox.Checked) Then
      ' Enable OrderSet if check box is checked.
      m_TTInstrNotify.EnableOrderSetUpdates = TRUE
   Else
      ' Disable OrderSet if check box is unchecked.
      m_TTInstrNotify.EnableOrderSetUpdates = FALSE

      ' Clear Bid and Ask list boxes.
      Me.bidOrderSetListBox.Items.Clear()
      Me.askOrderSetListBox.Items.Clear()
   End If
End Sub
```

Related information

Property:InstrEnableUpdates, on page 185

Method:AttachOrderSet, on page 202

Event:OnNotifyUpdate, on page 212

Property: EnablePriceUpdates

Purpose	Sets or returns whether to monitor inside-market price updates for the instrument
Syntax	`isEnabled = myInstrNotifer.EnablePriceUpdates` `myInstrNotifer.EnablePriceUpdates = isEnabled`
Parameters	*isEnabled* — Whether to enable price updates Data type: Boolean; Default: TRUE
Return value	*isEnabled* — Whether price updates are enabled Data type: Boolean
Exceptions	None
Description	The `EnablePriceUpdates` property indicates whether the notifier delivers updates for inside market price changes for all instruments attached to the notifier. When enabled, the XTAPI monitors price changes in the exchange update messages from the exchange. The specific fields that trigger a callback for your application are specified in the `UpdateFilter` property. The market depth updates are delivered in the `OnNotifyUpdate` event. In Figure 40, Notifier B delivers coalesced price updates for instruments B, F, and C. It triggers an `OnNotifyUpdate` event when a price update occurs for any of these instruments. Because Instr C is also attached to Notifier C, your application also receives callbacks and uncoalesced prices when an update occurs for Instr C.

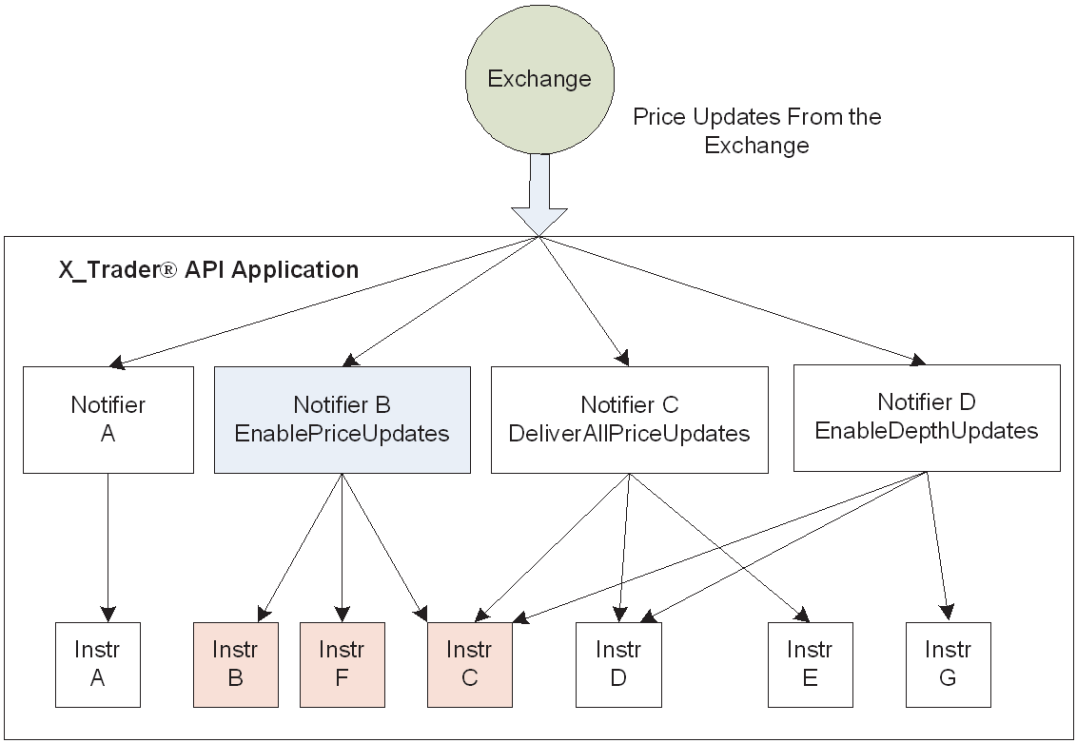


Figure 40. TTInstrNotify EnablePriceUpdates processing

If you disable this property, your application receives no inside market price updates, even if you have specified an update filter.

Caution: If no filter is defined in either the `InstrFilter` or `UpdateFilter` properties, changes in any and all instrument attributes generate callback events.

This property works independently of the `DeliverAllPriceUpdates` and `EnableDepthUpdates` properties. If you enable multiple properties, the XTAPI triggers events for each. So, for example, if you enable all three of these properties and a price update occurs for the **Last** price, your application will receive the three corresponding events. All of the events would contain information about the price change.

Example

The following routine uses the `EnablePriceUpdates` property to enable or disable depth updates based on a setting in an application window.

```
Private Sub enablePriceCheckBox_CheckedChanged(ByVal sender As Object,
                               ByVal e As System.EventArgs) Handles
                               enablePriceCheckBox.CheckedChanged
   ' Determine check box state.
   If (enablePriceCheckBox.Checked) Then
      ' Enable Price if check box is checked.
      m_TTInstrNotify.EnablePriceUpdates = TRUE
   Else
      ' Disable Price if check box is unchecked.
      m_TTInstrNotify.EnablePriceUpdates = FALSE

      ' Clear Bid and Ask list boxes.
      Me.bidPriceListBox.Items.Clear()
      Me.askPriceListBox.Items.Clear()
   End If
End Sub
```

Related information

Property:DeliverAllPriceUpdates, on page 176
Property:EnableDepthUpdates, on page 178
Property:InstrEnableUpdates, on page 185
Event:OnNotifyUpdate, on page 212

Property: InstrEnableUpdates

Purpose

Enables or disables callbacks for an instrument or order set

Syntax

```
isEnabled = myInstrNotifer.InstrEnableUpdates(obj)
myInstrNotifer.InstrEnableUpdates(obj, isEnabled)
```

Parameters

obj	Instrument or order set object Data type: `TTInstrObj` or `TTOrderSet`
isEnabled	Whether to enable updates for the instrument or order set Data type: Boolean; Default: FALSE

Return value

isEnabled	Whether instrument updates are enabled Data type: Boolean

Exceptions

None

Description

The `InstrEnableUpdates` property allows you to disable callbacks for events that occur for an individual instrument (**`TTInstrObj`**) or order set (**`TTOrderSet`**) within the instrument notifier. By default, all instruments attached to a notifier receive callbacks for the events enabled within the instrument notifier. You can use this property to disable the notifications temporarily for a specific instrument or order set. To permanently disable notifications for the instrument, use the Detach method.

When enabled, this property instructs the XTAPI to trigger the following events if they are enabled within this instrument notifier.

Object	Events
TTOrderSet	OnOrderSetUpdate
TTInstrObj	OnNotifyDepthData
	OnNotifyFound
	OnNotifyNotFound
	OnNotifyUpdate

Note: Setting this property does not affect coalesced prices notification, which is determined by the `DeliverAllPriceUpdates` property.

Example

The following subroutine toggles event notification for instrument updates.

```
'Method ChangeEnableUpdates turns off the updates for the
'instrument object and the order set object (m_TTInstrObj and
'm_TTOrderSet) provided that the string "Off" is passed to the function
'if any other string is passed then updates are enabled.  The
'ChangeEnableUpdates allows clients to turn on and off updates
'for these objects and then it outputs the current state
Private Sub ChangeEnableUpdates(ByVal pStrChange As String)
```

```
  'If the string "Off" is passed to the function then the Instrument
  'object and the Order set object updates are turned off.
  'If any other string is provided then the updates are turned on

  If (pStrChange = "Off") Then
    m_TTInstrNotify.InstrEnableUpdates(m_TTInstrObj) = False
    m_TTInstrNotify.InstrEnableUpdates(m_TTOrderSet) = False
  Else
    m_TTInstrNotify.InstrEnableUpdates(m_TTInstrObj) = True
    m_TTInstrNotify.InstrEnableUpdates(m_TTOrderSet) = True
  End If

  'True is returned if the updates are enabled
  Dim boolEnabledInstr As Boolean =
    m_TTInstrNotify.InstrEnableUpdates(m_TTInstrObj)
  Dim boolEnabledOrderSet As Boolean =
    m_TTInstrNotify.InstrEnableUpdates(m_TTOrderSet)

  'Outputs the current state of the objects
  If boolEnabledInstr Then
    Console.WriteLine("m_TTInstrObj's updates are enabled")
  Else
    Console.WriteLine("m_TTInstrObj's updates are not enabled")
  End If

  If boolEnabledOrderSet Then
    Console.WriteLine("m_TTOrderSet's updates are enabled")
  Else
    Console.WriteLine("m_TTOrderSet's updates are not enabled")
  End If

End Sub
```

Related information

Property:DeliverAllPriceUpdates, on page 176

Property:EnableDepthUpdates, on page 178

Property:EnablePriceUpdates, on page 182

Event:OnNotifyUpdate, on page 212

Property: InstrFilter

Purpose

Sets or returns a custom update filter for an instrument

Syntax

```
filter = myInstrNotifer.InstrFilter(instrObj)
myInstrNotifer.InstrFilter(instrObj, filter)
```

Parameters

instrObj	Instrument object for which you want to define a custom filter Data type: `TTInstrObj`
filter	Comma-separated list of instrument attributes Data type: String

Return value

filter	Update filter currently defined for the instrument Data type: String

Exceptions

.

Exception	Meaning
Invalid Instrument	The specified object is not a valid instrument.
Instrument Not Found	The specified instrument is not currently attached to the notifier

Description

The `InstrFilter` property sets the update filter for a specific instrument. It overrides the default notifier filter defined by the `UpdateFilter` property. When you set this filter, the XTAPI triggers update events only when the specified attributes change. Changes to other attributes, even if included in the `UpdateFilter` property, are ignored.

If you do not define an instrument-specific filter, the XTAPI uses the filter defined in the `UpdateFilter` property.

You can use one or more of the following Get property attributes as filter criteria:

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

Caution: If the `UpdateFilter` property is later modified, its filter will override all existing instrument filters in this `InstrFilter` object.

Figure 41 demonstrates how an instrument filter determines when an instrument notifier triggers update events. When the application receives the first update, Notifier B triggers an update event for Instr B because the **Ask** attribute is contained in the notifier's update filter. When the second update is received from the exchange, Notifier B triggers an update event for Instr F because the **IBid** attribute is included in the instrument filter for Instr F. Finally, when the application receives the third update, Notifier B takes no action. Although the notifier update filter includes the **Ask** attribute, the instrument filter for Instr F does not.

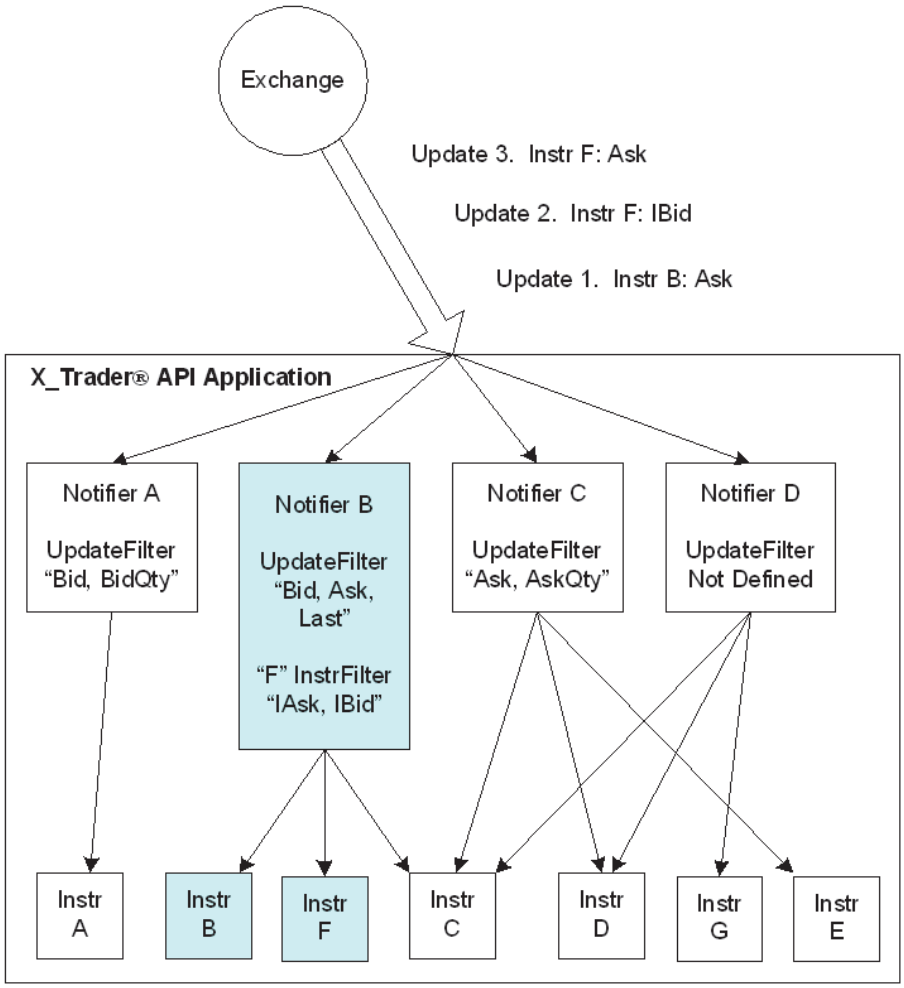


Figure 41. TTInstrNotify InstrFilter property illustration

Example

The following subroutine limits instrument updates to when the **Ask** or **AskQty** attributes change.

```
Private Sub SetFilters()

  'Set the filter to be Ask and AskQty so that updates
  'to the instrument object are only given if one of these change
  m_TTInstrNotify.InstrFilter(Me.m_TTInstrObj) = "Ask,AskQty"

  'Verify that the filter was set properly
  Dim strFilter As String
  strFilter = m_TTInstrNotify.InstrFilter(m_TTInstrObj)

  Console.WriteLine(strFilter)
```

```
End Sub
```

Related information

Property:Instrument, on page 190
Property:UpdateFilter, on page 197

Property: Instrument

Purpose

Returns the instrument associated with this notifier

Syntax

```
instrObj = myInstrNotifer.Instrument
```

Parameters

instrObj	Instrument attached to this instrument notify object Data type: `TTInstrObj`

Return value

filter	Update filter currently defined for the instrument Data type: String

Exceptions

.

Exception	Meaning
Multiple Instruments Connected	The **`TTInstrNotify`** object has more than one instrument attached to it
No Instrument Connected	No instrument is currently associated with this **`TTInstrNotify`** object.

Description

The `Instrument` property returns the **`TTInstrObj`** object attached to this notifier.

Note: You cannot use this property if the **`TTInstrNotify`** object has multiple instruments attached to it

Figure 42 shows a sample application with multiple instrument notifiers. Accessing the `Instrument` property of Notifier A returns a **`TTInstrObj`** object representing Instr A. However, accessing this property for any of the other notifiers throws an a "Multiple Instruments Connected" exception because each of them has multiple attached instruments.

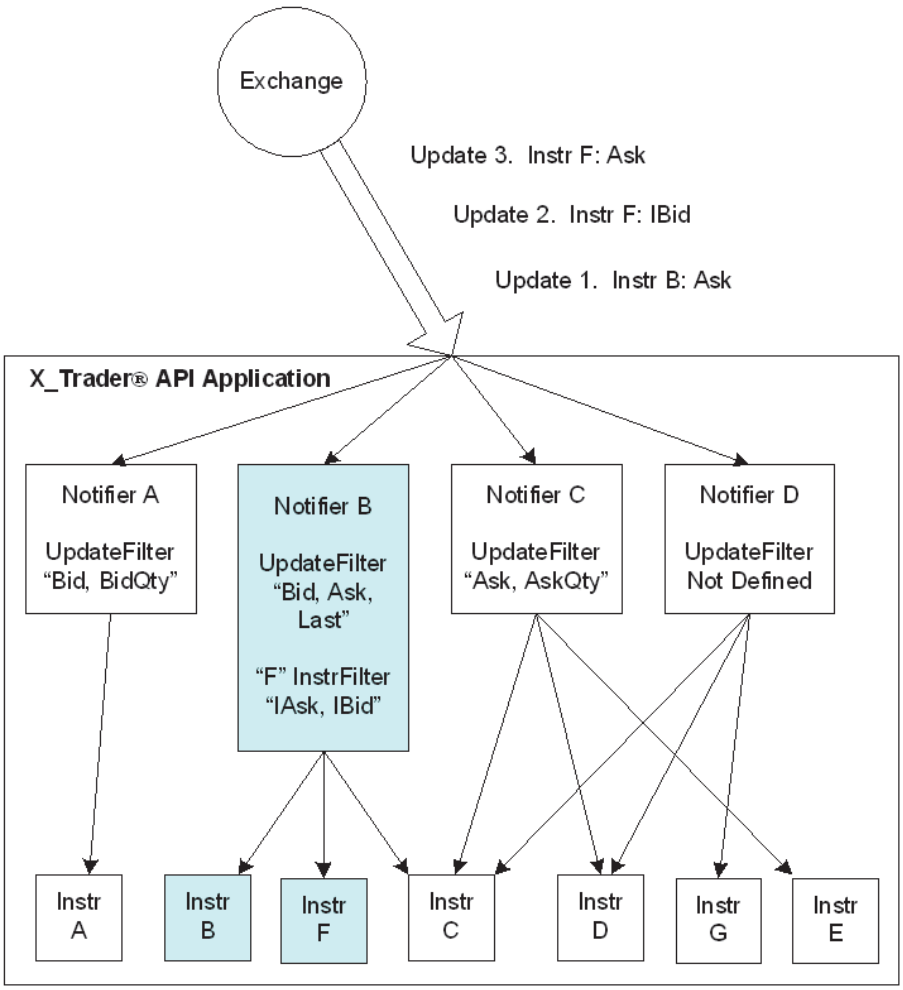


Figure 42. TTInstrNotify Instrument property illustration

Example

The following example uses the `Instrument` property to return an object attribute for the instrument attached to the instrument notify object.

```
Private Sub GetInstrument(ByRef pInstrNotify As XTAPI.TTInstrNotify,
          ByVal pGetPropOfInstr As String)

  Try
    'Create an instrument object
    Dim instr As XTAPI.TTInstrObj

    'Get the instrument object using the Instrument property
    instr = pInstrNotify.Instrument()

    'Create a string to hold the results of the instrument get property
    Dim strResults As String

    'Check to see that the string passed to the
    'fuction is a vaild get property for the instrument object
    If instr.IsValidGet(pGetPropOfInstr) Then
      strResults = instr.Get(pGetPropOfInstr)
      Console.WriteLine(strResults)
    Else
    Console.WriteLine("ERROR: invalid Get Property")
  End If
```

```
  'An exception is thrown if this notify object has more than
  'one instrument attached
  Catch ex As Exception
    Dim strExceptionMessage As String
    strExceptionMessage = ex.Message
    If strExceptionMessage = "Multiple Instruments Connected" Then
      Console.WriteLine(
        "ERROR: Multiple instruments are connected to notify object")
    End If
  End Try

End Sub
```

Related information **TTInstrObj** Property:CreateNotifyObject, on page 226

Property: IsValidUpdateFilter

Purpose

Identifies whether an object attribute can be used to filter instrument updates

Syntax

`isValidArr = myInstrNotifer.IsValidUpdateFilter(attributes)`

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

isValidArr	Array of Boolean values for each argument name supplied in attributes. Data type: Array

Exceptions

.

Exception	Meaning
Invalid Parameter	One or more of the specified attributes in not a filterable attribute

Description

The `IsValidUpdateFilter` property determines whether a specified array of object attributes represent valid attributes to use for filtering instrument notifications. The property returns an array, where each element contains a Boolean value indicating whether the name specified in the same element of the input array is a valid attribute.

You typically use this property to ensure that all of the arguments you want to pass to the `InstrFilter` property are valid. By checking for errors first, you can avoid exceptions thrown for invalid arguments.

You can use one or more of the following object attributes to identify which values to monitor for notification:

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

Example

The following example shows how to use the `IsValidUpdateFilter` property to ensure that all attributes are valid filters.

```
'Method SetFiltersSet checks that all of the filters are valid filters for the
'instrument object. The function returns a boolean value true if all of the
filters
'are valid.  If a filter is not valid then it outputs the invalid filter and
```

```
'ireturns false
Private Function CheckFilters() As Boolean

  'if all filters are valid this remains equal to true
  Dim boolCheckFiltersOK As Boolean
  boolCheckFiltersOK = True

  'string to hold the filters
  Dim strFilters As String
  strFilter = "Ask,AskQty,BidAskMid,BidQty,Change,Close,Delta"

  'boolean to hold the values in the array that is going to
  'be returned (one bool for each filter checked)
  Dim boolFilters As Boolean
  Dim arrayOfBools As Array

  'to count the interations of boolean values
  Dim i As Integer
  i = 0

  'the call passes the list of filters and returns an array of  booleans
  arrayOfBools = Me.m_TTInstrNotify.IsValidUpdateFilter(strFilters)

  'to hold the invalid property
  Dim strBadProperty As String

  Try
    For Each boolFilters In arrayOfBools

    'keeps track of which filter is being checked
    strBadProperty = strFilters.Split(",").GetValue(i)
    i = i + 1
    If boolFilters = False Then
      Console.WriteLine(strBadProperty)
      boolCheckFiltersOK = False
    End If
    Next
  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try

  'returns true if all filters
  'are valid
  Return boolCheckFiltersOK

End Function
```

Related information Property:UpdateFilter, on page 197

Property: PriceList

Purpose

Returns a collection of price updates that occurred since the PriceList was previously accessed

Syntax

priceUpdates = myInstrNotifer.PriceList

Parameters

None

Return value

priceUpdates	Collection of price update objects Data type: ITTPriceEntry

Exceptions

None

Description

The PriceList property returns all price changes that have occurred since this property was previously accessed. You use this property only for non-coalesced prices information, as it contains data only if you have enabled the DeliverAllPriceUpdates property.

By default, the XTAPI potentially coalesces price updates and returns only the most recent price for an instrument. Most traders care only about the most recent prices, so this behavior is appropriate and is more efficient. Applications that need access to each change in price, such as an instrument charting application, can access uncoalesced prices through this property.

To access the individual price updates, you iterate over the **ITTPriceList** object returned by the property when handling the OnPriceListUpdate event.

Example

The following example shows how to iterate through coalesced prices.

```
Private Sub GetPriceList()

  ' Requires m_TTInstrNotify.DeliverAllPriceUpdates = True

  'holds a collection of priceUpdates
  Dim priceList As XTAPI.ITTPriceList

  'holds a collection of priceEntries
  Dim priceUpdate As XTAPI.ITTPriceUpdate

  'holds the data for each tick
  Dim priceEntry As XTAPI.ITTPriceEntry

  'Get the price list which should contain any updates (all ticks) that
  'the application has missed since the last call to get priceList data
  priceList = Me.m_TTInstrNotify.PriceList

  Try
  'iterate through the priceUpdate collection to extract the priceEntry data
  For Each priceUpdate In priceList
  Console.WriteLine("PriceUpdate.TimeStamp= {0}", priceUpdate.TimeStamp)

    For Each priceEntry In priceUpdate
      Console.WriteLine("PriceEntry.Name= {0}", priceEntry.Name)
      Console.WriteLine("PriceEntry.toDouble = {0}", priceEntry.toDouble)
      Next
  Next
  Catch ex As Exception
```

```
    End Try

End Sub
```

Related information None

Property: UpdateFilter

Purpose

Identifies the criteria for filtering instrument updates

Syntax

```
filter = myInstrNotifer.UpdateFilter
myInstrNotifer.UpdateFilter = filter
```

Parameters

filter	Comma-separated list of filterable object attributes Data type: String

Return value

filter	Current filter criteria Data type: String

Exceptions

Exception	Meaning
Invalid Update Filter	One of more of the attributes in the filter list is not a filterable attribute.

Description

The `UpdateFilter` property limits the number of instrument-related events that generate callbacks to your application. When you set this filter, the XTAPI triggers update events only when the specified attributes change. Changes to other attributes are ignored.

Caution: If you do not define this update filter, the XTAPI generates callback events for changes in any of the filterable attributes. With no update filter defined, your application could be overwhelmed by the sheer number of updates received.

You can use one or more of the following `Get` property attributes as filter criteria:

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

Figure 43 shows how the `UpdateFilter` property determines when to trigger events for attached instruments. In this example, the application uses four instrument notifiers to register for price updates for seven different instruments. Instruments C and D are attached to multiple notifiers. When application receives the first update from the exchange gateway, the notifiers determine if the update applies to any of their attached instruments, then checks to see if the changes occurred in any of the attributes contained in their corresponding update filters.

In this example, Instr A is attached to only one notifier, Notifier A. For the first price update, only Notifier A processes the update, because the price update included a change to the **Bid** and **BidQty** fields of Instr A.

The second price update received by the application affects Instr C, which is attached to three notifiers (B, C, and D). The change occurred in the **Bid**, **High**, and **Low** attributes for the instrument. Notifier B responded to the update because its filter includes the **Bid** attribute. Because none of the changes occurred in an attribute included in its update filter, Notifier C ignored the update. Finally, because Notifier D does not have an update filter defined, it responds to changes in any of the instrument attributes. Therefore, Notifier D also responds to the price update.

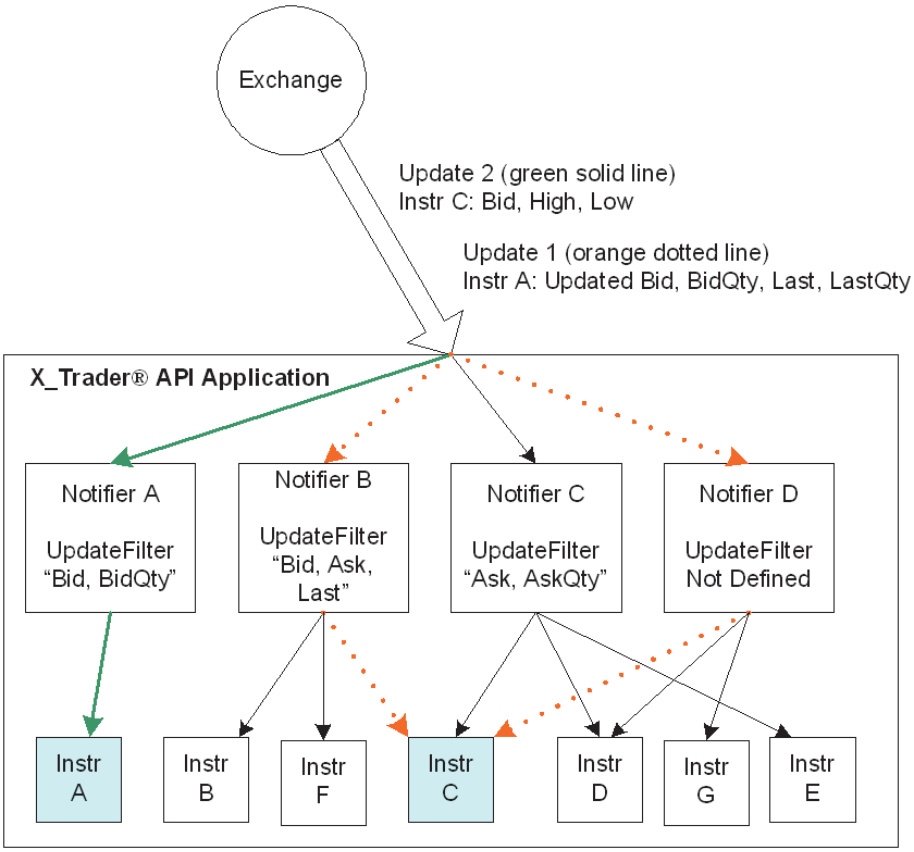


Figure 43. `TTInstrNotify` UpdateFilter property illustration

Example

The following example asks for callbacks only when the **Last** price changes.

```
Private Sub InstrumentFilter()
  Dim filterString As String

  ' Determine the state of the check boxes.
  If (BidPriceCheckBox.Checked And LastPriceCheckBox.Checked) Then
    filterString = "Bid$,Last$"
  ElseIf (BidPriceCheckBox.Checked) Then
    filterString = "Bid$"
  ElseIf (LastPriceCheckBox.Checked) Then
    filterString = "Last$"
  End If

  ' Set the UpdateFilter to the checked filter options.
  m_TTInstrNotify.UpdateFilter = (filterString)

  Application.DoEvents()
```

```
End Sub
```

Related information None

Method: AttachInstrument

Purpose

Attaches an instrument to the notifier

Syntax

`myInstrNotifer.AttachInstrument(`*instrument*`)`

Parameters

instrument	Instrument object to attach to notify object Data type: `TTInstrObj`

Return value

None

Exceptions

Exception	Meaning
Invalid Instrument	The specified instrument is not a valid TTInstrObj object.

Description

Note: You must use this method to attach the instrument to the notifier before calling the **`TTInstrObj`** `Open` method to open the instrument.

The `AttachInstrument` method attaches an instrument (**`TTInstrObj`**) object to the notifier so that your application be notified when instrument attributes change. After you attach an instrument, the instrument notifier sends callbacks to your application as changes occur in the instrument on the exchange.

When attaching instruments to receive notifications, you can use two approaches:

- You can create a single notifier and attach multiple instruments to it.
- You can create a separate notifier for each instrument by using the **`TTInstrObj`** `CreateNotifyObject` property.

Note: Attaching multiple instruments to a notifier is more efficient than creating individual notifiers for each instrument because it reduces the number of times your application must request information from the exchange gateway.

Example

The following code fragment uses the `AttachInstrument` method to attach an instrument dropped on an application window to the instrument notifier.

```
Private Sub m_TTDropHandler_OnNotifyDrop() Handles
m_TTDropHandler.OnNotifyDrop
   'Update the Status Bar text.
   statusBar1.Text = "Drag & Drop detected.  Initializing instrument..."

   Try
   ' ... omitted code to detach any existing instrument

   ' Obtain the TTInstrObj from the TTDropHandler object.
   m_TTInstrObj = m_TTDropHandler(1)

   ' Attach the TTInstrObj to the TTInstrNotify for price update events.
   m_TTInstrNotify.AttachInstrument(m_TTInstrObj)

   ' ... omitted code for further processing ...

   End Try
```

```
End Sub
```

Related information

Property:Instrument, on page 190
Method:DetachInstrument, on page 204

Method: AttachOrderSet

Purpose

Attaches an order set to the notifier

Syntax

`myInstrNotifer.AttachOrderSet(`*`orderSet`*`)`

Parameters

orderSet	Order set object to attach to the notify object Data type: `TTOrderSet`

Return value

None

Exceptions

Exception	Meaning
Invalid Order Set	The specified *`orderSet`* is not a valid **`TTOrderSet`** object.

Description

The `AttachOrderSet` property attaches an existing order set to the instrument notifier.

The XTAPI can send order set update events through the **`TTOrderSet`** or **`TTInstrNotify`** objects, depending the values of their respective `EnableOrderSetUpdates` properties. If your application deals primarily with instrument data, you might choose to write a single event handler based on the **`TTInstrNotify`** object instead of writing separate event handlers.

Note: If you enable updates for both objects, the XTAPI sends two events each time an order set update occurs.

Example

The following code attaches an order set to an instrument notify object.

```
'The purpose of this code is to show how an order set is attached to an
'instrument notify object.  A typical place to perform this is inside
Private Sub m_DropInstr_OnNotifyDrop() Handles m_DropInstr.OnNotifyDrop

   'copy the instrument to the instrument object
   m_TTInstrObj = m_DropInstr(1)

   'Creates the Instrument Notify Object
   Me.m_TTInstrNotify = New XTAPI.TTInstrNotifyClass

   'Creates the OrderSet object
   m_TTOrderSet = New XTAPI.TTOrderSet

   'Set the Net Limits to false
   m_TTOrderSet.Set("NetLimits", False)

   'Attach the Orderset to the Instrument Notify Object
    m_TTInstrNotify.AttachOrderSet(m_TTOrderSet)

   'Enable order entry, fills, order set updates
    m_TTOrderSet.EnableOrderSend = True
    m_TTOrderSet.EnableOrderFillData = True
    m_TTOrderSet.EnableOrderSetUpdates = True
    'Open the order set
    Me.m_TTOrderSet.Open(1)
```

```
    'Attach the instrument object to the instrument notify object
    m_TTInstrNotify.AttachInstrument(m_TTInstrObj)

    'Open the instrument object
    m_TTInstrObj.Open()

    'resets the drop instrument
    m_DropInstr.Reset()

End Sub
```

Related information

Property:EnableOrderSetUpdates, on page 181

Method:DetachOrderSet, on page 206

Method: DetachInstrument

Purpose

Detaches an instrument from a notifier

Syntax

`myInstrNotifer.DetachInstrument(`*`instrument`*`)`

Parameters

instrument	Instrument object to detach Data type: `TTInstrObj`

Return value

None

Exceptions

Exception	Meaning
Invalid Instrument	The specified instrument is not a valid **`TTInstrObj`** object.

Description

The `DetachInstrument` method removes an instrument from the instrument notifier. After removing the instrument, your application no longer receives update events through this notifier for changes that occur for the instrument.

Note: Detaching an instrument affects only this **`TTInstrNotify`** object. If an instrument is attached to multiple notifiers, your application continues to receive update events from the other notifiers that have the same instrument attached to them.

Figure 44 illustrates the effects of detaching instruments from a notifier with instruments attached to other notifiers. Notifier B originally had three instruments attached to it (B, F, and C). After detaching Instr B and Instr C, notifier B only monitors Instr F for price updates. Instr B is no longer monitored by any notifier, so your application no longer receives any price updates for it. However, because Instr C is attached to multiple notifiers, those notifiers still trigger events when price updates for Instr C.

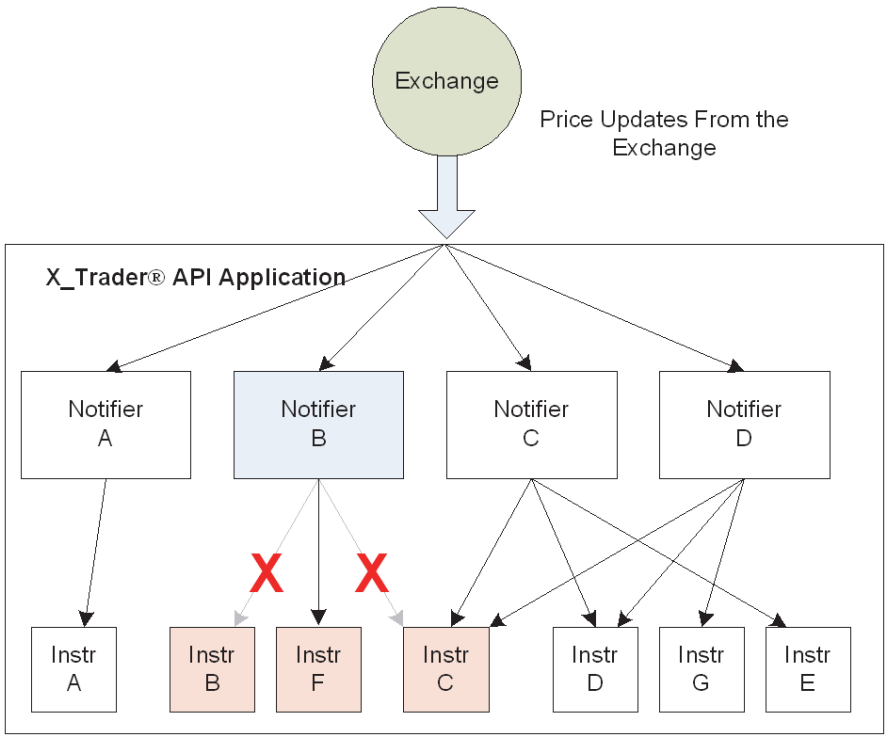


Figure 44. TTInstrNotify object: detaching an instrument

Example

The following code fragment uses the `DetachInstrument` method to an existing instrument from the notifier when an instrument is dropped on an application window.

```
Private Sub m_TTDropHandler_OnNotifyDrop() Handles
m_TTDropHandler.OnNotifyDrop
   'Update the Status Bar text.
   statusBar1.Text = "Drag & Drop detected.  Initializing instrument..."

   Try
   ' Test if a TTInstrObj currently exists.
   If Not m_TTInstrObj Is Nothing Then
   ' Detach previously attached instrument.
   m_TTInstrNotify.DetachInstrument(m_TTInstrObj)
   m_TTInstrObj = Nothing
   End If

   ' Continue processing
   ' omitted code ...
End Sub
```

Related information

Property:Instrument, on page 190

Method:AttachInstrument, on page 200

Method: DetachOrderSet

Purpose	Detaches an order set to the notifier
Syntax	`myInstrNotifer.DetachOrderSet(`*`orderSet`*`)`
Parameters	*orderSet* — Order set object to detach from the notify object Data type: `TTOrderSet`
Return value	None

Exceptions

Exception	Meaning
Invalid Order Set	The specified *`orderSet`* is not a valid **`TTOrderSet`** object.

Description

The `DetachOrderSet` property removes the specified order set from the instrument notifier. After detaching the order set, the XTAPI no longer sends events for the order set through the instrument notifier. However, if you also enabled the order set update events in the **`TTOrderSet`** object, the order set handler will still receive the updates.

Example

None

Related information

Method:AttachOrderSet, on page 202

Event: OnInstrumentExpiryChanged

Triggered when	Expiry for an instrument changes
Syntax	`Sub myInstrNotifer_OnInstrumentExpiryChanged(`*`notifier`*`, `*`instrument`*`)`
Parameters	*notifier* — Instrument notifier object that fired the event Data type: `TTInstrNotify` *instrument* — Instrument object for which the expiry changed Data type: `TTInstrObj`
Description	The `OnInstrumentExpiryChanged` event is triggered when the expiry for an instrument changes. **Note:** This event can be triggered only for instruments currently open.
Example	None
Related information	None

Event: OnNotifyDepthData

Triggered when

Price changes occur in an instrument's market depth

Syntax

`Sub myInstrNotifer_OnNotifyDepthData(`*notifier*`, `*instrument*`)`

Parameters

notifier	Instrument notifier object that fired the event Data type: `TTInstrNotify`
instrument	Instrument object for which a change occurred on the exchange Data type: `TTInstrObj`

Description

The `OnNotifyDepthData` event is triggered when some change occurs on the exchange in the market depth for an instrument attached to the instrument notifier. The specific changes that can trigger this event depend on the filters defined for the notifier (`UpdateFilter` property) or instrument (`InstrFilter` property).

Note: This event can be triggered only if the `EnableDepthData` property is enabled for this **`TTInstrNotify`** object.

Based on the filters, changes to one or more of the following instrument attributes can trigger this event

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

If the instrument notifier specifies neither an update or instrument filter, changes to any of these attributes triggers the `OnNotifyDepthData` event.

Note: The `OnNotifyUpdate` and `OnNotifyDepthData` events are designed to communicate the latest changes to an instrument's attributes and depth, respectively. These events do not guarantee current snapshots of an instrument when you open an instrument or attach a notifier to it. To ensure that you get a current snapshot, you should use the **`TTInstrObj`**`Get` property in the **`TTInstrNotify`** `OnNotifyFound` event to access current attribute values for the instrument every time it is opened.

Example

The following callback routine updates the application window every time a instrument's depth data changes.

```
Private Sub m_TTInstrNotify_OnNotifyDepthData(
                ByVal pNotify As XTAPI.TTInstrNotify,
                yVal pInstr As XTAPI.TTInstrObj)
```

```
                    Handles m_TTInstrNotify.OnNotifyDepthData
  Dim i As Integer

  ' Clear the depth list boxes.
  Me.bidDepthListBox.Items.Clear()
  Me.askDepthListBox.Items.Clear()

  ' Obtain the bid depth (Level based on user selection).
  dataArrayBid = pInstr.Get(m_bidDepthValue)   ' "BidDepth(0)   "

  ' Test if depth exists.
  If Not dataArrayBid Is Nothing Then
    ' Iterate through the depth array.
    For i = 0 To dataArrayBid.GetUpperBound(0)
      ' Break out of FOR LOOP if index value is null.
      If dataArrayBid.GetValue(i, 0) Is Nothing Then
        Exit For
      End If
      ' Update the bid depth list box.
      Me.bidDepthListBox.Items.Add("BidPrice: " & _
        dataArrayBid.GetValue(i, 0) & " | " & _
        " BidQty: " & dataArrayBid.GetValue(i, 1))
    Next i
  End If

  ' Obtain the ask depth (Level based on user selection).
  dataArrayAsk = pInstr.Get(m_askDepthValue)   ' "AskDepth(0)"

  ' Test if depth exists.
  If Not dataArrayAsk Is Nothing Then
    ' Iterate through the depth array.
    For i = 0 To dataArrayAsk.GetUpperBound(0)
      ' Break out of FOR LOOP if index value is null.
      If dataArrayAsk.GetValue(i, 0) Is Nothing Then
        Exit For
      End If

      ' Update the bid depth list box.
      Me.askDepthListBox.Items.Add(" AskPrice: " & _
        dataArrayAsk.GetValue(i, 0) & " | " & _
        " AskQty: " & dataArrayAsk.GetValue(i, 1))
    Next i
  End If

  Application.DoEvents()
End Sub
```

Related information

Property:EnableDepthUpdates, on page 178

Event:OnNotifyUpdate, on page 212

Event: OnNotifyFound

Triggered when

`TTInstrObj` Open or Retrieve methods validate the existence of an instrument on the exchange

Syntax

`Sub myInstrNotifer_OnNotifyFound(`*`notifier`*`, `*`instrument`*`)`

Parameters

notifier	Instrument notifier object that fired the event Data type: `TTInstrNotify`
instrument	Instrument object that passed validation Data type: `TTInstrObj`

Description

The `OnNotifyFound` event is triggered when the **`TTInstrObj`** `Open` or `Retrieve` method submits an instrument that exists on the specified exchange gateway.

Example

The following subroutine uses the `OnNotifyFound` method to populate fields when an instrument is dropped on an application window.

```
Private Sub m_TTInstrNotify_OnNotifyFound(ByVal pNotify As
            XTAPI.TTInstrNotify, ByVal pInstr As XTAPI.TTInstrObj)
            Handles m_TTInstrNotify.OnNotifyFound
   ' Update the Status Bar text.
   statusBar1.Text = "Instrument Found."

   ' Retrieve instrument information using the TTInstrObj Get property.
   data = pInstr.Get("Exchange,Product,ProdType,Contract")

   exchangeTextBox.Text = Convert.ToString(data.GetValue(0))
   productTextBox.Text = Convert.ToString(data.GetValue(1))
   prodTypeTextBox.Text = Convert.ToString(data.GetValue(2))
   contractTextBox.Text = Convert.ToString(data.GetValue(3))

   Application.DoEvents()

End Sub
```

Related information

None

Event: OnNotifyNotFound

Triggered when

TTInstrObj Open or Retrieve methods fail to validate an instrument

Syntax

```
Sub myInstrNotifer_OnNotifyNotFound(notifier, instrument)
```

Parameters

notifier	Instrument notifier object that fired the event Data type: TTInstrNotify
instrument	Instrument object that failed validation Data type: TTInstrObj

Description

The OnNotifyNotFound event is triggered when the **TTInstrObj** Open or Retrieve method submits an instrument that is not valid for the specified exchange gateway.

Example

The following callback function displays information about an instrument that the XTAPI could not find.

```
Private Sub m_TTInstrNotify_OnNotifyNotFound(ByVal pNotify As
XTAPI.TTInstrNotify,
             ByVal pInstr As XTAPI.TTInstrObj)
             Handles m_TTInstrNotify.OnNotifyNotFound

  Console.WriteLine("ERROR: The below instrument was not found.")
  Console.WriteLine("Contract=" + pInstr.Contract)
  Console.WriteLine("Exchange=" + pInstr.Exchange)
  Console.WriteLine("Product=" + pInstr.Product)
  Console.WriteLine("ProductType=" + pInstr.ProdType)

End Sub
```

Related information

None

Event: OnNotifyUpdate

Triggered when

Changes occur on the inside market for instrument attributes

Syntax

`Sub myInstrNotifer_OnNotifyUpdate(`*notifier*`, `*instrument*`)`

Parameters

notifier	Instrument notifier object that fired the event Data type: `TTInstrNotify`
instrument	Instrument object for which a change occurred on the exchange Data type: `TTInstrObj`

Description

The `OnNotifyUpdate` event is triggered when some change occurs on the exchange for an instrument's inside market that has been attached to an instrument notifier. The specific changes that can trigger this event depend on the filters defined for the notifier (`UpdateFilter` property) or instrument (`InstrFilter` property).

Note: The XTAPI triggers this event only when the `EnablePriceUpdates` property is enabled for this **`TTInstrNotify`** object.

Based on the filters, changes to one or more of the following instrument attributes can trigger this event:

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

If the instrument notifier specifies neither an update or instrument filter, changes to any of these attributes triggers the `OnNotifyUpdate` event.

Note: The `OnNotifyUpdate` and `OnNotifyDepthData` events are designed to communicate the latest changes to an instrument's attributes and depth, respectively. These events do not guarantee current snapshots of an instrument when you open an instrument or attach a notifier to it. To ensure that you get a current snapshot, you should use the **`TTInstrObj`** `Get` property in the **`TTInstrNotify`** `OnNotifyFound` event to access current attribute values for the instrument every time it is opened.

Example

The following routine uses the `OnNotifyUpdate` property to update bid and ask data in an application window when an instrument updates.

```
Private Sub m_TTInstrNotify_OnNotifyUpdate(ByVal pNotify As
XTAPI.TTInstrNotify,
    ByVal pInstr As XTAPI.TTInstrObj) Handles m_TTInstrNotify.OnNotifyUpdate
```

```
    ' Retrieve the instrument information using the TTInstrObj Get Properties.
    data = pInstr.Get("Bid#,BidQty$,Ask$,AskQty$,Last$,LastQty$,~LastQty$")

    bidPriceTextBox.Text = Convert.ToString(data.GetValue(0))
    bidQtyTextBox.Text = Convert.ToString(data.GetValue(1))
    askPriceTextBox.Text = Convert.ToString(data.GetValue(2))
    askQtyTextBox.Text = Convert.ToString(data.GetValue(3))
    lastPriceTextBox.Text = Convert.ToString(data.GetValue(4))
  lastQtyTextBox.Text = Convert.ToString(data.GetValue(5))
  lastQtyDeltaTextBox.Text = Convert.ToString(data.GetValue(6))

  Application.DoEvents()
End Sub
```

Related information

Property:DeliverAllPriceUpdates, on page 176
Property:EnableOrderSetUpdates, on page 181
Property:EnablePriceUpdates, on page 182
Property:InstrEnableUpdates, on page 185
Event:OnNotifyDepthData, on page 208

Event: OnOrderSetUpdate

Triggered when	Changes occur for an order set
Syntax	`Sub myInstrNotifer_OnOrderSetUpdate(`*`orderSet`*`)`
Parameters	*orderSet* — Order set object that contains updated orders Data type: TTOrderSet
Description	The `OnOrderSetUpdate` event is triggered when a change, such as an order delete, partial fill, or object attribute, occurs for an order set attached to the **`TTInstrNotify`** object. To receive this event, your application must also set the `EnableOrderSetUpdates` property to TRUE.
Example	None
Related information	Property:EnableOrderSetUpdates, on page 181 Method:AttachOrderSet, on page 202 Method:DetachOrderSet, on page 206

Event: OnPriceListUpdate

Triggered when

Price changes are added to the non-coalesced price list

Syntax

```
Sub myInstrNotifer_OnPriceListUpdate(notifier)
```

Parameters

notifier	Instrument notifier whose non-coalesced price list updated Data type: `TTInstrNotify`

Description

The `OnPriceListUpdate` event is triggered when an update occurs for the notifier's price list (`PriceList` property). When the `DeliverAllPriceUpdates` property is enabled, the XTAPI adds a price entry (**`ITTPriceEntry`** object) to the instrument notifier's `PriceList` property whenever a change occurs in an attached instrument's attributes on the exchange.

Note: This event can be triggered only if the `DeliverAllPriceUpdates` property is enabled for this **`TTInstrNotify`** object.

The `OnPriceListUpdate` event is triggered if a price update occurs for any of the instruments attached to the instrument notifier.

To access the price updates, you get the updated price list from the `PriceList` property.

Example

None

Related information

None

16 TTInstrObj Class

Chapter overview

The chapter describes the **`TTInstrObj`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTInstrObj class overview	218
Property:Alias	220
Property:CalculateTTImplieds	221
Property:Contract	222
Property:ContractFormatted	225
Property:CreateNotifyObject	226
Property:CreateOrderSelector	227
Property:DebugLogLevel	229
Property:Exchange	232
Property:GatewayType	233
Property:Get	234
Property:IsTradablePrice	254
Property:IsValidGet	256
Property:IsValidUpdateFilter	258
Property:IsVolatile	260
Property:MergeImpliedsIntoDirect	262
Property:OrderSet	264
Property:ProdType	265
Property:Product	266
Property:ReadProperties	267
Property:RoundPrice	268
Property:SeriesKey	270
Property:TickPrice	271
Property:TickPriceEx	273
Method:Open	275
Method:Retrieve	277

TTInstrObj class overview

Overview

The **TTInstrObj** class represents a tradable object. Throughout this document, tradable objects are referred to as Instruments. Your program must create a **TTInstrObj** for every instrument for which you want to receive price updates or submit orders. Attaching a **TTInstrObj** object to a **TTInstrNotify** object allows you to receive price and order updates when they occur.

An instrument is uniquely identified by the combination of the following properties, which are defined by the exchange trading the commodity:

- Exchange (i.e. CME)
- Product (i.e. ES)
- Product type (i.e. FUTURE)
- Contract (i.e. Sep13)

Note: If the `Contract` property of the instrument contains only the expiration month and year, the XTAPI modifies the value automatically when opening the instrument to contain the product name and the contract, as in **ES Sep13**.

A **TTInstrObj** object stores information to identify an instrument uniquely among all other tradable commodities and maintains price-related information about the instrument®.

Price formats

The XTAPI provides prices and quantities in several formats. When requesting a price attribute, such as the Bid price, you can append a special character to the end of the attribute name to specify the format for the price. The trailing character of the requesting field determines the format of prices and quantities as follows:

- & : Integer (typically ticks)
- # : Decimal (typically points)
- $: String (typically the X_TRADER display format)

Additionally, you can use the `TickPrice` property of the instrument to convert prices between each of these formats.

Price depth

By default, the XTAPI returns prices for the inside market. You can also register for live depth updates when opening an instrument. After you register for depth, you must also configure the **TTInstrNotify** object `EnableDepthUpdates` property to trigger event callbacks when the market depth changes.

Implied prices

For exchanges that support strategy and spread markets, such as eCBOT and LIFFE, the XTAPI can calculate implied prices for an instrument. When you enable the `CalculateTTImplieds` property for an instrument, you can trade the instrument based on implied prices.

Properties

Table 45 lists the **TTInstrObj** properties.

Property	Description
Alias	Sets or returns an alias for an instrument
CalculateTTImplieds	Identifies whether to use TT-calculated implied prices
Contract	Sets or returns the contract identifier associated with the instrument

Table 45. TTInstrObj properties

Property	Description
ContractFormatted	Returns a contract name without a Day indicator
CreateNotifyObject	Creates a notification object for an instrument
CreateOrderSelector	Creates a selector object that monitors orders only for this instrument
DebugLogLevel	Sets or returns the level of information written to the XTAPI debug log
Exchange	Sets or returns the gateway for the exchange on which the instrument is traded
GatewayType	Returns the type of the TT Gateway associated with an instrument
Get	Returns a TTInstrObj object's attributes
IsTradablePrice	Returns whether the instrument can be traded at a specified price
IsValidGet	Returns whether an attribute name is a valid argument for the Get property
IsValidUpdateFilter	Identifies whether an object attribute can be used to filter instrument updates
IsVolatile	Indicates whether an object attribute for an instrument can change
MergeImpliedsIntoDirect	Indicates whether implied prices and quantities are merged into the inside markets
OrderSet	Returns or attaches the instrument to a specified order set
ProdType	Sets or returns a product type for the instrument
Product	Sets or returns the name of the product associated with the instrument
ReadProperties	Returns information about TTInstrObj attributes you can access through the Get property
RoundPrice	Returns a valid tick price for an estimated price
SeriesKey	Sets or returns the series key associated with the instrument
TickPrice	Returns the tradable price some number of ticks from a specified price
TickPriceEx	Returns the tradable price some number of ticks from a specified price after rounding

Table 45. `TTInstrObj` properties

Methods

Table 46 lists the **`TTInstrObj`** methods.

Method	Description
Open	Opens an instrument for trading
Retrieve	Verifies that a instrument is valid

Table 46. `TTInstrObj` methods

Events

None

Property: Alias

Purpose

Sets or returns an alias for an instrument

Syntax

```
aliasName = myInstrObj.Alias
myInstrObj.Alias = aliasName
```

Parameters

aliasName	Name to associate with the instrument Data type: String

Return value

aliasName	Alias currently associated with the instrument Data type: String

Exceptions

None

Description

The `Alias` property assigns a user-defined name that can be used to reference an instrument. An alias provides a convenient method of accessing instruments from other objects, such as a **`TTGateObj`** object. By creating an alias, you also make the instrument visible to the **`TTGate`** object's `Intruments` method.

When you create an alias, the XTAPI adds an in-memory reference to the **`TTInstrObj`** object, which has the side effect of incrementing its reference count. Because the reference count is greater than 0, the XTAPI does not remove it from memory during its ongoing garbage collection activities. Consequently, creating excessive aliases can increase the memory footprint of your application.

Example

The following example creates a new instrument and sets an alias.

```
' Keep track of the instrument (contract) we are using
Dim myInstr As New TTInstrObj

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstr.Exchange = "CME"
myInstr.Product = "ES"
myInstr.ProdType = "FUTURE"
myInstr.Contract = "Sep13"

' Alias this instrument with Product and Expiry information
myInstr.Alias = "SP Mini"
```

Related information

Property:Product, on page 266

Method:Open, on page 275

`TTOrderProfile` Property:Alias, on page 312

`TTOrderSet` Property:Alias, on page 362

Property: CalculateTTImplieds

Purpose

Identifies whether to use TT-calculated implied prices

Syntax

```
calcImplieds = myInstrObj.CalculateTTImplieds
myInstrObj.CalculateTTImplieds = calcImplieds
```

Parameters

calcImplieds	Whether to use TT calculated implied prices Data type: Boolean; Default: FALSE

Return value

calcImplieds	Whether TT calculates implied prices Data type: Boolean

Exceptions

Exception	Meaning
Cannot enable TT calculated implieds after extended prices are disabled	The **TTGate** `DisableExtendedPrices` method was called during initialization, so your application cannot access TT calculated implied prices.

Description

The `CalculateTTImplieds` property allows an application to specify whether your application uses implied prices calculated by TT when not provided by the exchange. If an exchange supports trading implied markets, but does not provide the implied prices, you can enable implied price calculation for the instrument.

If the exchange already provides implied prices or does not support trading implied markets, enabling this property has no effect.

Note: This property affects only spreads and strategy markets (implied INs) for the LIFFE and eCBOT exchanges.

Example

None

Related information

Property:IsTradablePrice, on page 254

Property:MergeImpliedsIntoDirect, on page 262

Property:TickPrice, on page 271

Property:TickPriceEx, on page 273

Property: Contract

Purpose	Sets or returns the contract identifier associated with the instrument
Syntax	`contractID = myInstrObj.Contract` `myInstrObj.Contract = contractID`
Parameters	*contractID* — Exchange identifier for a contract. The ID is the series expiration or exchange-provided name where available. Data type: String
Return value	*contractID* — Contract ID associated with the instrument Data type: String
Exceptions	None

Description

The `Contract` property identifies the contract identifier associated with a product. For example, the eCBOT Treasury bond futures product (ZB) list contracts for Mar13, Jun13, and so on, such as ZB Jun13. Some exchanges, such as CME, might also provide a unique name for each contract. In the case of CME, an ES Dec13 instrument would also specify the contract as ESZ4, which corresponds to the **ContractShort** object attribute for the instrument.

Note: XTAPI does not populate the `Contract` property until after you open the corresponding instrument object, which you can verify by registering for the **`TTInstrNotify`** `OnNotifyFound` event.

Drag and drop behavior

If your application uses drag-and-drop functionality from X_TRADER, XTAPI does not populate the value of the `Contract` property until you open or retrieve the contract with the `Open` and `Retrieve` methods, respectively.

Special contract naming conventions

For cash products, like those traded on the Xetra and BrokerTec exchanges, the contract name is the same as the product name.

Note: For manually entered eCBOT option strategies, you must enter the contract names using the X_TRADER display format.

Impact of TT Gateway series name GAL setting

The value of the `TT_GAL_USE_SERIES_NAME` GAL setting on a TT Gateway can affect what values the XTAPI returns for the following ways you can retrieve contract names:

- `cName = Instr.Contract`
- `cName = Instr.Get("Contract")`
- `cName = Instr.ContractFormatted`

If the TT Gateway enables the GAL setting, the XTAPI uses the series name provided by the TT Gateway. Consequently, the XTAPI always returns the same value in *cName* in each of the cases. However, if the TT Gateway disables the GAL setting, the XTAPI might return a different *cName* value if the TT Gateway supplies a day in the name, such as **15Sep13**. In this instance, the XTAPI returns **Sep13** for the `ContractFormatted` property, but returns **15Sep13** for the other two.

Additionally, if you set the `Contract` property, the XTAPI does not reformat this property value when you use the `Open` method to open an instrument. Note that if the name you specify does not match the name expected by the TT Gateway, the open fails.

Example

The following example sets the contract name for an instrument.

```
' Keep track of the instrument (contract) we are using
Dim myInstr As New TTInstrObj

' Set instrument information (Gateway, Product, Product Type, and Expiry)
myInstr.Exchange = "CME"
myInstr.Product = "ES"
myInstr.ProdType = "FUTURE"
myInstr.Contract = "Sep13"

' Alias this instrument with Product and Expiry information
myInstr.Alias = "SP Mini"
```

Related information

Property:ContractFormatted, on page 225

Property:ProdType, on page 265

Property:Product, on page 266

Property: ContractFormatted

Purpose	Returns a contract name without a Day indicator
Syntax	*contractName* = myInstrObj.ContractFormatted
Parameters	None
Return value	*contractName* — Product name and the month and year portion of the contract ID associated with the instrument Data type: String
Exceptions	None

Description

The `ContractFormatted` property removes the Day indicator from a contract name, if present. Some contracts include a Day indicator as part of the name, such as 14Jun13. The `Contract` property returns the full name, **14Jun13**, in this case. If you only want the month and year representation, you can call this property, which would return **Jun13**.

Note: This property returns a different value than the Contract property only when the TT Gateway sets the `TT_GAL_USE_SERIES_NAME` GAL setting to FALSE.

XTAPI does not populate the `ContractFormatted` property until after you open the corresponding instrument object, which you can verify by registering for the **`TTInstrNotify`** `OnNotifyFound` event.

Drag and drop behavior

If your application uses drag-and-drop functionality from X_TRADER, XTAPI does not populate the value of the `ContractFormatted` property until you open or retrieve the contract with the `Open` and `Retrieve` methods, respectively.

Example	None
Related information	Property:Contract, on page 225

Property: CreateNotifyObject

Purpose	Creates a notification object for an instrument
Syntax	`notifyObj = myInstrObj.CreateNotifyObject`
Parameters	None
Return value	*notifyObj* — Instrument notification object Data type: `TTInstrNotify`
Exceptions	None
Description	The `CreateNotifyObject` property creates an instrument notification object. The notify object generates callbacks when changes occur for the instrument on the exchange, such as when the bid or ask prices change. When you create a notification object using this method, the **`TTInstrNotify`** object monitors events only for this instrument. By default, the notify object monitors all events for an instrument, which can generate a lot of callbacks. To limit which instrument events generate notifications, use the **`TTInstrNotify`** `UpdateFilter` property. **Caution:** If you monitor multiple instruments, using this approach for all instruments can impact performance and throughput.
Example	None
Related information	**`TTInstrNotify`** Property:Instrument, on page 190

Property: CreateOrderSelector

Purpose

Creates a selector object that monitors orders only for this instrument

Syntax

ordSelector = myInstrObj.CreateOrderSelector

Parameters

None

Return value

ordSelector	Order selector containing this instrument Data type: `TTOrderSelector`

Exceptions

Exception	Meaning
Invalid Instrument	The current instrument object is no longer a valid object.

Description

The `CreateOrderSelector` property creates an order selector object that includes only the current instrument. You can then attach the order selector to a **`TTOrderSet`** object to create an order set that filters and manages orders or fills only for this instrument.

Example

The following example shows how to create an order selector that includes only the current instrument object.

```
'The SetUpOrderSet method demonstrates how to use the instrument object to
create an
'order selector so that only orders from the instrument chosen are in the
orderset
Private Sub SetUpOrderSet()
  Try
    'Create an order set
    m_TTOrderSet = New XTAPI.TTOrderSet

    'Turn off self-imposed limits
    m_TTOrderSet.Set("NetLimits", False)

    'use an instrument that has been found by the instrument notify found
callback
    'to create an orderSelector. Only updates for this instrument will create
    'updates for this application
    Dim orderSelector As XTAPI.TTOrderSelector
    orderSelector = m_TTInstrObj.CreateOrderSelector()

    'Connect the order selector to the orderset
    Me.m_TTOrderSet.OrderSelector = orderSelector

    'Enable the orderSet to send an order
    m_TTOrderSet.EnableOrderSend = True
    m_TTOrderSet.EnableOrderSetUpdates = True
    m_TTOrderSet.EnableOrderUpdateData = True

    'Open the orderset for order entry
    m_TTOrderSet.Open(1)

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
```

```
End Sub
```

Related information TTOrderSelector Class, on page 343

Property: DebugLogLevel

Purpose

Sets or returns the level of information written to the XTAPI debug log

Syntax

```
logLevel = myInstrObj.DebugLogLevel
myInstrObj.DebugLogLevel = logLevel
```

Parameters

logLevel

Types of information to include in the debug log file, as a bitmask. Base hexadecimal values include:

- LOG_ERROR (0x00)
- LOG_MINIMAL (0x01)
- LOG_MIN_API (0x03)
- LOG_DET_API (0x04)
- LOG_ORDERS (0x8)
- LOG_COM (0x10)
- LOG_COM_USE (0x20)
- LOG_COM_ERR (0x30)
- LOG_COM_EVENTS (0x40)
- LOG_ALL_COM (0x70)
- LOG_FILLS (0x100)
- LOG_DEBUG_WIN (0x1000)

Data type: Integer

Return value

logLevel

Current debug log level as an integer

Data type: Integer

Exceptions

None

Description

This property specifies what types of information the XTAPI writes to a debug log file. If an application is not performing as expected, you can adjust the level of information written to the log file to help you isolate the cause.

Table 47 describes the different debug logging levels.

Enumeration	Description
LOG_ERROR	Logs only error messages
LOG_MINIMAL	Logs minimal generic debugging (low impact) information
LOG_MIN_API	Logs minimal API debugging information, such as open exchanges, server status, and so on. This value is the default setting for the XTAPI.
LOG_DET_API	Generates detailed API logging
LOG_ORDERS	Logs all orders submitted to an exchange
LOG_COM	Logs the creation and destruction of all COM objects and logs (fills and orders can generate a lot of log entries)
LOG_COM_USE	Logs each time XTAPI accesses a COM property
LOG_COM_ERR	Logs each time XTAPI creates or destroys COM objects, accesses COM object properties, or encounters COM errors.
LOG_COM_EVENTS	Logs all COM events
LOG_ALL_COM	Logs all COM information.

Table 47. Debug log levels

Enumeration	Description
LOG_FILLS	Logs all received fills
LOG_DEBUG_WIN	Logs all log information to the debug window
LOG_ALL	Logs everything **Warning:** Using this setting can overwhelm system resources and severely curtail throughput.

Table 47. Debug log levels

Note: Because writing extra information to log files can impact application performance, you should reduce the log level to the default level or lower as soon as possible, especially if you are debugging a production application.

Setting global debug log levels using an initialization file

You can also set the default debug log level in the runtime **XTAPI.ini** file for the application. The debug log level specified in this file applies globally to all objects created by the application. Several different objects contain the DebugLogLevel property for your convenience, so that you don't have to instantiate a **`TTGate`** object just to change the debug log level.

To use an initialization file for the application, create an **XTAPI.ini** file in the same directory as your application's executable file. In the file, add a line similar to the following (case-sensitive):

```
[DebugLogLevel]
  XTAPI_Application=8  # where XTAPI_Application is the application name
```

When your application starts, XTAPI reads this file and sets the debug logging level for the application. Because XTAPI reads the file only at startup, any changes you make to the DebugLogLevel property remain in effect until the application terminates or until you change the value of this property.

Location of debug log files

The XTAPI stores its debug log files in the **<root dir>:\tt\logfiles** directory with the naming convention:

```
XTAPI_YYYY-MM-DD_AppName_ProcessID.log
```

where:

- *YYYY-MM-DD* represents the file creation date
- *AppName* represents the name of the API application
- *ProcessID* is the ID associated with the process.

Format of debug logs

Each line in the debug log file contains three pipe-delimited (|) fields as follows:

```
Date | Message Source | Message
```

where:

- *Date* indicates when XTAPI wrote the message to the file
- *Message Source* identifies where the message originated

- *Message* contains the debug information

Example

The following example temporarily adds order logging for debugging purposes. It saves the current debug log level, changes the level, and finally restores the level to its original setting.

```
Sub PrintDebugInfo()
   Dim myInstrObj As New TTInstrObj
   Dim dLevel As Integer

   dLevel = myInstrObj.DebugLogLevel    ' Save base log level
   Debug.Print "XTAPI version: ", myInstrObj.DebugAppInfo
   Debug.Print "Debug Log Level: ", dLevel

   ' Change the log level to include all order information
   myInstrObj.DebugLogLevel = dLevel + LOG_ORDERS
   Debug.Print "Debug Log Level: ", myInstrObj.DebugLogLevel
End Sub
```

Related information

None

Property: Exchange

Purpose	Sets or returns the gateway for the exchange on which the instrument is traded
Syntax	`exchange = myInstrObj.Exchange` `myInstrObj.Exchange = exchange`
Parameters	*exchange* — Name of the exchange gateway to associate with the instrument Data type: String
Return value	*exchange* — Name of the exchange gateway currently associated with the exchange Data type: String
Exceptions	None
Description	The `Exchange` property identifies the exchange on which the instrument is traded, such as eCBOT.
Example	The following example sets the `Exchange` property to CME.

```
' Keep track of the instrument (contract) you are using
Dim myInstr As New TTInstrObj

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstr.Exchange = "CME"
myInstr.Product = "ES"
myInstr.ProdType = "FUTURE"
myInstr.Contract = "Sep13"

' Alias this instrument with Product and Expiry information
myInstr.Alias = "SP Mini"
```

Related information	None

Property: GatewayType

Purpose	Returns the type of the TT Gateway associated with an instrument
Syntax	*gwType* = myInstrObj.GatewayType
Parameters	None
Return value	*gwType* — Type of TT Gateway. Possible values include: • GATEWAY_TYPE_PFX • GATEWAY_TYPE_LEGACY • GATEWAY_TYPE_UNKNOWN Data type: enumGatewayType
Exceptions	None
Description	The GatewayType property indicates whether the TT Gateway associated with this instrument is a PFX gateway. This property returns GATEWAY_TYPE_UNKNOWN until you open the corresponding instrument object, which you can verify by registering for the **TTInstrNotify** OnNotifyFound event. If your application uses drag-and-drop functionality from X_TRADER, XTAPI does not populate the value of the GatewayType property until you open or retrieve the contract with the Open and Retrieve methods, respectively.
Example	None
Related information	**TTPriceEntry** Property:PriceID, on page 54

Property: Get

Purpose

Returns a **`TTInstrObj`** object's attributes

Syntax

```
attrValues = myInstrObj.Get(attributes)
```

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **`TTInstrObj`** object attribute
NOTE: This property can also throw the following exceptions for the PL and OpenPL attributes, which indicate reasons why the XTAPI cannot calculate P&L.	• Fill data not available; unable to calculate P&L • Invalid Parameter: 'PL' return value format is invalid; unable to calculate P&L • Instrument not attached to Orderset; unable to calculate P&L • Instrument not open; unable to calculate P&L

Description

The `Get` property returns the values for an arbitrary number of **`TTInstrObj`** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single `ChangeMe` property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 48 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 48. Price and quantity value formats

Note: Some market data, such as implied prices, might not always be available when you request it. In such cases, the XTAPI returns a null value instead of the normal data type. TT recommends that your application checks for null values before using an attribute value.

Table 49 lists the **`TTInstrObj`** object attributes you can supply to the Get property.

Attribute	Description	DataType
Ask (~Ask) Ask$ (~Ask$) Ask# (~Ask#) Ask& (~Ask&)	Best ask price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute. **Note:** When the `MergeImpliedsIntoDirects` property is enabled, the Ask price contains the better of the direct and implied prices.	As specified
AskDepth(*levels*) AskDepth#(*levels*) AskDepth&(*levels*)	Two-dimensional array of prices and the quantity associated with each (each row is price, quantity) for the specified number of depth levels.	Array of specified data types
AskDepth90(*levels*) AskDepth90#(*levels*) AskDepth90&(*levels*)	Column-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each column contains a price and a quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types

Table 49. TTInstrObj Get property attributes

Attribute	Description	DataType
AskMktQty (~Ask) AskMktQty$ (~Ask$) AskMktQty# (~Ask#) AskMktQty& (~Ask&)	Market order quantity during the pre-open phase, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
AskOrderId	Order ID for the best ask. (Currently supported only for BrokerTec)	String
AskQty (~AskQty) AskQty$ (~AskQty$) AskQty# (~AskQty#) AskQty& (~AskQty&)	Best ask quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute. **Note:** When the `MergeImpliedsIntoDirects` property is enabled, **AskQty** contains the quantity for the better of the direct and implied Ask prices.	As specified
AvgOpenPriceInDecimal	Average price of open position in decimal format. Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	Decimal
AvgOpenPriceInTicks	Average price of open position in ticks and fractions of a tick. Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	Decimal
AvgOpenPriceRnd AvgOpenPriceRnd# AvgOpenPriceRnd$ AvgOpenPriceRnd&	Average price of open position rounded to the nearest tick. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	As specified
Bid (~Bid) Bid$ (~Bid) Bid# (~Bid#) Bid& (~Bid&)	Best bid price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute. **Note:** When the `MergeImpliedsIntoDirects` property is enabled, the Bid price contains the better of the direct and implied prices.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
BidAskMid (~BidAskMid) BidAskMid$ (~BidAskMid$) BidAskMid# (~BidAskMid#) BidAskMid& (~BidAskMid&)	Average of the best bid price and the best ask price (rounded down to the nearest tick), in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
BidDepth(*levels*) BidDepth#(*levels*) BidDepth&(*levels*)	Row-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each row contains a price and a quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	As specified
BidDepth90(*levels*) BidDepth90#(*levels*) BidDepth90&(*levels*)	Column-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each column contains a price and a quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	As specified
BidMktQty (~Ask) BidMktQty$ (~Ask$) BidMktQty# (~Ask#) BidMktQty& (~Ask&)	Market order quantity during the pre-open phase, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
BidOrderId	Order ID for the best bid. (Currently supported only for BrokerTec)	String
BidQty (~BidQty) BidQty$ (~BidQty$) BidQty# (~BidQty#) BidQty& (~BidQty&)	Best bid quantity as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute. **Note:** When the `MergeImpliedsIntoDirects` property is enabled, **BidQty** contains the quantity for the better of the direct and implied Bid prices.	As specified
BuyWrk	Quantity of working buy orders. Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
CallPut	Whether the option on the instrument is a call or put: **C**: Call **P**: Put	String

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
Change (~Change) Change$ (~Change$) Change# (~Change#) Change& (~Change&)	Net price change in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Close (~Close) Close$ (~Close$) Close# (~Close#) Close& (~Close&)	Prior session's closing price for the instrument in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Contract	Default name of the instrument as displayed in the X_TRADER® **Market Grid Contract** field (example: ES Sep13).	String
ContractAlias	Alias name associated with the instrument (as assigned in Guardian), along with the contract's expiry information. If the instrument has no alias, the property returns the default instrument name and expiry information.	String
ContractShort	Short name associated with the instrument (example: ES M5)	String
Currency	ISO currency code associated with the instrument. ISO symbol for the source and target currencies.	String
Decimals	Number of implied decimals used in prices.	Integer
DeliveryUnit	Exchange-provided delivery unit for instruments in flow contracts; 1 for all others.	Integer
Exchange	Name of the exchange on which the instrument trades.	String
ExpirationDate	Instrument expiration date in the form, *MM/DD/YY*. For contracts that have no expiration date, this attribute returns NULL.	COleDateTime
Expiry	Instrument expiration, in month and year (example: Sep13).	String
FillsDownloaded	Whether all fills have been downloaded from the exchange.	Boolean
Fractional Fractional$	Ticking scheme for the instrument. Possible values include: **D**: Decimal **F**: Fractional	String
High (~High) High$ (~High$) High# (~High#) High& (~High&)	High price during the current trading session, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
IAsk (~IAsk) IAsk$ (~IAsk$) IAsk# (~IAsk#) IAsk& (~IAsk&)	Implied ask price for the instrument, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IAskDepth(*levels*) IAskDepth#(*levels*) IAskDepth&(*levels*)	Row-based, two-dimensional array of implied ask prices and corresponding quantities for the specified number of depth levels, where each row contains a price and a quantity, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	Array of specified data types
IAskDepth90(*levels*) IAskDepth90#(*levels*) IAskDepth90&(*levels*)	Column-based, two-dimensional array of implied ask prices and corresponding quantities for the specified number of depth levels, where each column contains a price and a quantity, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	Array of specified data types
IAskQty (~IAskQty) IAskQty$ (~IAskQty$) IAskQty# (~IAskQty#) IAskQty& (~IAskQty&)	Implied ask quantity, as a integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IBid (~IBid) IBid$ (~IBid$) IBid# (~IBid#) IBid& (~IBid&)	Implied bid price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IBidDepth(*levels*) IBidDepth#(*levels*) IBidDepth&(*levels*)	Row-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each row contains a price and a quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
IBidDepth90(*levels*) IBidDepth90#(*levels*) IBidDepth90&(*levels*)	Column-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each column contains a price and a quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types
IBidQty (~IBidQty) IBidQty$ (~IBidQty$) IBidQty# (~IBidQty#) IBidQty& (~IBidQty&)	Implied bid quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndAsk (~IndAsk) IndAsk$ (~IndAsk$) IndAsk# (~IndAsk#) IndAsk& (~IndAsk&)	Indicative ask price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndAskQty (~IndAskQty) IndAskQty$ (~IndAskQty$) IndAskQty# (~IndAskQty#) IndAskQty& (~IndAskQty&)	Indicative ask quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndBid (~IndBid) IndBid$ (~IndBid$) IndBid# (~IndBid#) IndBid& (~IndBid&)	Indicative bid price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndBidQty (~IndBidQty) IndBidQty$ (~IndBidQty$) IndBidQty# (~IndBidQty#) IndBidQty& (~IndBidQty&)	Indicative bid quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
IndClosePrice (~IndClosePrice) IndClosePrice$ (~IndClosePrice)$) IndClosePrice# (~IndClosePrice)#) IndClosePrice& (~IndClosePrice)&)	Indicative closing price indicating the settlement price for the session, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndCloseQty(~IndCloseQty) IndCloseQty$ (~IndCloseQty$) IndCloseQty# (~IndCloseQty#) IndCloseQty& (~IndCloseQty&)	Indicative close quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndOpenPrice (~IndOpenPrice) IndOpenPrice$ (~IndOpenPrice$) IndOpenPrice# (~IndOpenPrice#) IndOpenPrice& (~IndOpenPrice&)	Indicative opening price indicating the settlement price for the session, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndOpenQty(~IndOpenQty) IndOpenQty$ (~IndOpenQty$) IndOpenQty# (~IndOpenQty#) IndOpenQty& (~IndOpenQty&)	Indicative open quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
IndPrice (~IndPrice) IndPrice$ (~IndPrice$) IndPrice# (~IndPrice#) IndPrice& (~IndPrice&)	Indicative, or equalibrium, pre-open phase price indicating the price a market would match if it were open, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
IndSettle (~IndSettle) IndSettle$ (~IndSettle$) IndSettle# (~IndSettle#) IndSettle& (~IndSettle&)	Indicative closing price indicating the intra-day settlement price, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Instr	Default instrument name as displayed in the X_TRADER® **Market Grid Contract** field.	String
InstrStatus	Status of the connection for underlying series. Possible values include: • Connected • Found • Nonexistent • Valid • Undefined	String
IPriceDepth(*levels*) IPriceDepth#(*levels*) IPriceDepth&(*levels*)	Row-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each row contains a price, bid quantity, and ask quantity, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
IPriceDepth90(*levels*) IPriceDepth90#(*levels*) IPriceDepth90&(*levels*)	Column-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each column contains a price, bid quantity, and ask quantity, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
IsAskOwner	Whether an order belonging to the current trader is the best ask. (Currently supported only for BrokerTec)	Boolean
IsBidOwner	Whether an order belonging to the current trader is the best bid. (Currently supported only for BrokerTec)	Boolean
IsFlowContract	Whether the instrument is an energy product and can trade with quantities in flow.	Boolean
IsHit	Whether the aggressor in the workup was a seller. Valid only for BrokerTec.	Boolean
IsLTQAccum	Whether the GAL setting for LTQ accumulation is enabled.	Boolean
IsOTC	Whether the trade was an OTC trade.	Boolean
IsPrivateWorkup	Whether the market is in a private workup state. (Currently supported only for BrokerTec)	Boolean

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
IsPublicWorkup	Whether the market is in a public workup state. (Currently supported only for BrokerTec)	Boolean
IsServerDown	Whether the price server associated with the instrument is down.	Boolean
IsServerUp	Whether the price serve associated with the instrument is up.	Boolean
IsSubscribed	Whether the instrument receives live price updates.	Boolean
IsTake	Whether the aggressor in the workup was a buyer. Valid only for BrokerTec.	Boolean
IsTradedInFlow	Whether the contract is a flow contracted delivering quantities in flow. (Valid only for energy products)	Boolean
IsWorkup	Whether the market is in a public or private workup state. (Currently supported only for BrokerTec)	Boolean
IXTDepth(*levels*) IXTDepth#(*levels*) IXTDepth&(*levels*)	Row-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each row contains a bid price, bid quantity, ask price, and ask quantity, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	Array of specified data types
IXTDepth90(*levels*) IXTDepth90#(*levels*) IXTDepth90&(*levels*)	Column-based, two-dimensional array of implied prices and corresponding quantities for the specified number of depth levels, where each column contains a bid price, bid quantity, ask price, and ask quantity. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types
Key	Unique, internal ID that TT associates with the instrument. Same as **Series** and **SeriesKey** attributes.	String
Last (~Last) Last$ (~Last$) Last# (~Last#) Last& (~Last&)	Price at which the instrument last traded, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
LastQty (~LastQty) LastQty$ (~LastQty$) LastQty# (~LastQty#) LastQty& (~LastQty&)	Quantity of all trades at the last traded price, as in integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
LLimit (~LLimit) LLimit$ (~LLimit$) LLimit# (~LLimit#) LLimit& (~LLimit&)	Lower limit used during price reasonability checks, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Login.field	Information about the current user login. Possible field values include: • CLRMEMBER • DEFAULTACCT • GATEWAYIP • GROUP • HANDOFFGROUP • HANDOFFMEMBER • HANDOFFTRADER • MEMBER • TRADER	String
LotSize	Exchange-provided lot size for instruments that are flow contracts; 1 for all others.	Integer
Low (~Low) Low$ (~Low$) Low# (~Low#) Low& (~Low&)	Lowest trading price for the current trading session, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
LTPDirection	Direction of the most recently traded price for the instrument. Possible values include: **1** : Up **0** : Unchanged **-1** : Down If retrieved before price updates are received, the value returns TT_INVALID_QTY.	Integer
LTPDirection$	Direction of the most recently traded price for the instrument. Possible values include: '**+**' : Up ' ' : Unchanged '**-**' : Down If retrieved before price updates are received, the value returns an empty string.	String
Market	Name of the exchange associated with the TT Gateway; equivalent to the **Exchange** attribute.	String
MaxDecimals	Maximum number of significant digits to the right of the decimal point of a price.	Integer

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
MIAsk (~MIAsk) MIAsk$ (~MIAsk$) MIAsk# (~MIAsk#) MIAsk& (~MIAsk&)	Better of the direct best ask price and the implied ask price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
MIAskQty (~MIAskQty) MIAskQty$ (~MIAskQty$) MIAskQty# (~MIAskQty#) MIAskQty& (~MIAskQty&)	One of the following: • Direct best ask quantity if the direct best ask price is better than the implied ask price • Implied ask quantity if the implied ask price is better than the direct best ask price • Direct and implied ask quantities added together if the direct and implied ask prices are the same. The attribute returns an integer value by default. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
MIBid (~MIBid) MIBid$ (~MIBid$) MIBid# (~MIBid#) MIBid& (~MIBid&)	Better of the direct best bid price and the implied bid price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
MIBidQty (~MIBidQty) MIBidQty$ (~MIBidQty$) MIBidQty# (~MIBidQty#) MIBidQty& (~MIBidQty&)	One of the following: • Direct best bid quantity if the direct best bid price is better than the implied bid price • Implied bid quantity if the implied bid price is better than the direct best bid price • Direct and implied bid quantities added together if the direct and implied bid prices are the same. The attribute returns an integer value by default. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
MktImbalanceQty	Degree of imbalance between the number of bids and asks for the full depth of the Instrument. Positive values indicate more bids than asks; negative values indicate more asks that bids.	Integer

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
MMMYY	Contract name, using the 3-character month identifier and 2-digit year (example: Jun13). For products of type STOCK, this value returns an empty string.	String
Name	Default instrument name as displayed in the X_TRADER® **Market Grid Contract** field.	String
NetCnt	Value of the **NetCnt** attribute in the attached **`TTOrderSet`**, which is the total number of working orders (Number of Buy Orders + Number of Sell Orders). Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
NetPos	Value of the **NetPos** attribute in the attached **`TTOrderSet`**, which is the current net position (Quantity Purchased - Quantity Sold), based on fill information. Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
NetTicks	Value of the **NetTicks** attribute in the attached TTOrderSet, which is the net cost in ticks of the current position (Revenue from all Sells - Cost of all Buys). Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
NetWrk	Value of the **NetWrk** attribute in the attached TTOrderSet, which is the total quantity of working orders (Quantity of Working Buys + Working Sells). Valid only when the instrument has an attached **`TTOrderSet`** object.	Integer
NumberOfAskOrders(*priceLevel)*	Number of resting ask orders at the specified *`priceLevel`* (0 for the inside market).	Integer
NumberOfBidOrders(*priceLevel)*	Number of resting bid orders at the specified *`priceLevel`* (0 for the inside market).	Integer
Open (~Open) Open# (~Open#) Open$ (~Open$) Open& (~Open&)	Opening price of the instrument for the current trading session, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
OpenPL OpenPL& OpenPL# OpenPL$	Current value of open positions in ticks, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The **OpenPL$** value is expressed as USD currency. Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
OpenPL~ OpenPL^ OpenPL.^ OpenPL@	Current market value of open positions as: **~** : String delta format **^** : Native currency **.^** : Native currency **@** : Primary currency Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	As specified
OpenPL.[Currency]	Current market value of open positions in the currency whose code you specify (Currency). Possible values include: • ARS • CHF • HKD • PTE • ATS • DEM • INR • RUR • AUD • ESP • ITL • SGD • BEF • EUR • JPY • USD • BRL • FRF • KRW • XBA • CAD • GBP • NLG • XEU Valid only when you attach the instrument to a **TTOrderSet** object with its `EnableFillCaching` property set to **TRUE**.	String
OS.objAttribute	Value of the specified object attribute in TTOrderSet. Valid only when the instrument has an attached **`TTOrderSet`**.	As specified
OTCPrc (~OTCPrc) OTCPrc$ (~OTCPrc$) OTCPrc# (~OTCPrc#) OTCPrc& (~OTCPrc&)	Price of the last wholesale trade for the instrument in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
OTCQty (~OTCQty) OTCQty$ (~OTCQty) OTCQty# (~OTCQty) OTCQty& (~OTCQty&)	Quantity of the last wholesale trade, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
OwnAskOrderKey	Order key of the best ask if the order belongs to the current trader. Valid only when you attach the instrument to a **TTOrderSet** object. (Currently supported only for BrokerTec)	String
OwnBidOrderKey	Order key of the best bid if the order belongs to the current trader. Valid only when you attach the instrument to a **TTOrderSet** object. (Currently supported only for BrokerTec)	String
PctChange PctChange$	Percentage change from the previous close price to the current price, as a decimal value. You can use the $ modifier to return the value as a string.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
PL PL$ PL# PL&	P & L (NetPos * LTP + NetTicks), as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The PL$ value is expressed as USD currency. Valid only when you attach the instrument to a **TTOrderSet** object.	As specified
PL~ PL^ PL.^ PL@	P & L in string delta format, in one of the following formats: **~** : String delta format **^** : Native currency **.^** : Native currency **@** : Primary currency Valid only when you attach the instrument to a **TTOrderSet** object.	As specified
PL.[Currency]	P & L in the currency whose code you specify (Currency). Possible values include: • ARS • CHF • HKD • PTE • ATS • DEM • INR • RUR • AUD • ESP • ITL • SGD • BEF • EUR • JPY • USD • BRL • FRF • KRW • XBA • CAD • GBP • NLG • XEU Valid only when you attach the instrument to a **TTOrderSet** object.	String
PL.[LOTS]	P & L for instruments that trade in flow in lots; for instruments that do not trade in flow, returns in contracts. Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
PL.Z PL.Z$ PL.Z# PL.Z&	P & L since the last ClearQuotePosition call, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer Valid only when a quoting **`TTOrderSet`** is attached to the instrument.	As specified
PointValue	Value of one price point. ProductName Variable extracts this value from the TT Guardian product table for the instrument.	Double
PriceDepth(*levels*) PriceDepth#(*levels*) PriceDepth&(*levels*)	Row-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each row contains a price, bid quantity, and ask quantity. The value returns in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
PriceDepth90(*levels*) PriceDepth90#(*levels*) PriceDepth90&(*levels*)	Column-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each column contains a price, bid quantity, and ask quantity. The value returns in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types
PriceState PriceState$	Current workup state and type as string or bitmask according to the following: 0x00 : "None" 0x01 : "Workup Private" 0x02 : "Workup Public" 0x0F : "Other States" 0x10 : "Aggressor is bid" 0x20 : "Aggressor is ask" 0x40 : "Workup correction" "Other States" serves as a mask for unused values. Common values for PriceState include: 0 : "No workup" 17 : "Private workup aggressor is bid" 18 : "Public workup aggressor is bid" 33 : "Private workup aggressor is ask" 34 : "Private workup aggressor is ask" (Currently supported only for BrokerTec)	As specified
PriceType	ProductName Variable price display type. For a description of possible values, refer to "Price types" on page 555.	Integer
ProdType ProdType&	Name of (**ProdType**) or integer (**ProdType&**) representing the instrument's product type. For a list of possible product types, see "Product types" on page 557.	As specified
Product	Name of the product.	String
ProductAlias	Name of a product alias (as it is assigned in Guardian). If the product does not have an alias, **ProductAlias** returns the default name of the product.	String
ProductTitle	Product name followed by the product type in parenthesis (for example, ES (FUTURE)).	String
RoundLotQty	Size of trading quantity multiplier. Some contracts must be traded in fixed mulitples of the lot quantity. For example, a contract might require you to trade it only in mulitples of 5.	Integer
SellWrk	Quantity of working sells. Valid only when you attach the instrument to a **TTOrderSet** object.	Integer
Series	Unique, internal ID that TT associates with the instrument. Same as **Key** and **SeriesKey** attributes.	String
SeriesKey	Unique, internal ID that TT associates with the instrument. Same as **Key** and **Series** attributes.	String
SeriesName	Exchange-provided name for the instrument, if it exits; "Contract" otherwise.	String

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
Settle (~Settle) Settle$ (~Settle$) Settle# (~Settle#) Settle& (~Settle&)	Instrument's settle price, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Status Status& Status$	Status of the instrument on the exchange, such as Trading, Pre-trade, etc., as an integer. You can use the following modifiers to return the value in a specific format: & : Integer $: String For more information, refer to Product status types, on page 639.	As specified
Strike	Strike price as an integer (not ticks). Valid only for options instruments.	Integer
StrikeDecimals	Number of significant digits to the right of the decimal used for the strike price. Valid only for options instruments.	Integer
StrikeFlags	Display format for the instrument's strike price. Possible values include: D: Decimal points (default) N: Fractional tick Valid only for options instruments.	As specified
StrikePrecision	Default market precision for the instrument's strike price.	Integer
TAsk (~TAsk) TAsk$ (~TAsk$) TAsk# (~TAsk#) TAsk& (~TAsk&)	Theoretical ask price for a mass quote, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
TAskQty (~TAskQty) TAskQty$ (~TAskQty$) TAskQty# (~TAskQty#) TAskQty& (~TAskQty&)	Theoretical ask quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
TBid (~TBid) TBid$ (~TBid$) TBid# (~TBid#) TBid& (~TBid&)	Theoretical bid price for a mass quote, in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
TBidQty (~TBidQty) TBidQty$ (~TBidQty$) TBidQty# (~TBidQty#) TBidQty& (~TBidQty&)	Theoretical ask quantity, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Text	Same as the Name and Instr attributes.	String
TickIncrement	Smallest valid tick increment as an integer. The default value is 1. For reduced tick spreads, the value of underlying legs is either 2 or 4.	Integer
TPrc (~TPrc) TPrc$ (~TPrc$) TPrc# (~TPrc#) TPrc& (~TPrc&)	Theoretical mid price, as a decimal. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
ULimit (~ULimit) ULimit$ (~ULimit$) ULimit# (~ULimit#) ULimit& (~ULimit&)	Upper limit used during price reasonability check, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Version	Version of the option.	String
Vola (~Vola) Vola$ (~Vola$) Vola# (~Vola#) Vola& (~Vola&)	Theoretical volatility, as an integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified
Volume (~Volume) Volume$ (~Volume$) Volume# (~Volume#) Volume& (~Volume&)	Total quantity traded for the day, as a integer. You can also use one of the following modifiers to display the value in a specific format: **$** : String **#** : Decimal value **&** : Integer The tilde (~) prefix returns the last change (delta) in the attribute.	As specified

Table 49. TTInstrObj Get property attributes(Continued)

Attribute	Description	DataType
WorkupState	Current workup state as a string. Possible values include: • "None" • "Public" • "Private" (Currently supported only for BrokerTec)	String
WorkupState&	Current workup state as an enumeration. Possible values include: • WORKUP_TYPE_NONE • WORKUP_TYPE_PRIVATE • WORKUP_TYPE_PUBLIC (Currently supported only for BrokerTec)	Enum
WorkupType	Current workup type as a string. Possible values include: • "None" • "Hit" • "Take" (Currently supported only for BrokerTec)	Enum
WorkupType&	Current workup type as an enumeration. Possible values include: • WORKUP_TYPE_NONE • WORKUP_TYPE_HIT • WORKUP_TYPE_TAKE (Currently supported only for BrokerTec)	Enum
XTDepth(*levels*) XTDepth#(*levels*) XTDepth&(*levels*)	Row-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each row contains a bid price, bid quantity, ask price, and ask quantity. Returns prices in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types
XTDepth90(*levels*) XTDepth90#(*levels*) XTDepth90&(*levels*)	Column-based, two-dimensional array of prices and corresponding quantities for the specified number of depth levels, where each column contains a bid price, bid quantity, ask price, and ask quantity. Returns prices in X_TRADER display format. You can also use one of the following modifiers to display the value in a specific format: **#** : Decimal value **&** : Integer (Ticks)	Array of specified data types

Table 49. TTInstrObj Get property attributes(Continued)

Example

The following example uses the `Get` property to display the values for the specified **`TTInstrObj`** attributes.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidNameArr As Array
  Dim attrValuesArr As Array
  Dim isValidList As Boolean
  Dim attrName As String
  Dim i As Integer
  i = 1
```

```
  'good practice to validate list before calling Get
  isValidNameArr = Me.m_TTInstrObj.IsValidGet(objAttributes)

  ' code to verify attribute list omitted

  If isValidList = True Then
    attrValuesArr = Me.m_TTInstrObj.Get(objAttributes)

    'Display each attribute and value
    For Each attrValue in attrValuesArr
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Attribute: {0} = {1}", attrName, attrValue)
      i = i + 1
      Next
    End For

  End If
End Sub
```

Related information

Property:IsValidGet, on page 256

Property:ReadProperties, on page 267

Property: IsTradablePrice

Purpose

Returns whether the instrument can be traded at a specified price

Syntax

`isTradable = myInstrObj.IsTradablePrice(price)`

Parameters

price	Price to check. You can specify the price in the following formats: • String, for X_TRADER display format • Integer, for tick values • Double, for decimal values Data type: String, Integer, or Decimal

Return value

isTradable	Whether the instrument can be traded at the specified price Data type: Boolean

Exceptions

Exception	Meaning
Invalid Parameter	The price parameter contains an invalid price value

Description

The `IsTradablePrice` property determines whether you can trade the instrument at a specified price. The tradable price is determined by the tick size associated with the instrument. When you submit a price, the property compares it to the allowable trading prices as specified by the corresponding exchange. You can specify the price as a string, an integer, or a double. The property automatically converts the value to a tick value for comparison purposes.

You should check the `IsTradablePrice` property to verify a price before submitting an order for the instrument.

Example

The following example uses the `IsTradablePrice` property to verify that the price a user entered is valid for the instrument.

```
'SendOrder demonstrates to use the IsTradablePrice boolean value that
'checks a price for a particular instrument object
Private Sub SendOrder()

  Try
    'Create an orderprofile
    Dim orderProfile As XTAPI.TTOrderProfile
    orderProfile = New XTAPI.TTOrderProfile

    'set the needed parameters for the orderprofile
      orderProfile.Customer = "TTORDGS001001"
      orderProfile.Instrument = Me.m_TTInstrObj
      orderProfile.Set("BuySell", Me.txtBuyOrSell.Text)
      orderProfile.Set("Qty", Me.txtQty.Text)
      orderProfile.Set("OrderType", "L")

    Dim boolIsTradable As Boolean
      boolIsTradable = Me.m_TTInstrObj.IsTradablePrice(Me.txtPrice.Text)
```

```
    If (boolIsTradable) Then

      orderProfile.Set("Price", Me.txtPrice.Text)

      'Create an integer to hold the results of the ordersend function.
      Dim intOrderQty As Integer

      '.
      '. code to submit the order omitted
      '.

  Catch ex As Exception
            Console.WriteLine(ex.Message)
  End Try

End Sub
```

Related information

Property:CalculateTTImplieds, on page 221

Property:IsVolatile, on page 260

Property:MergeImpliedsIntoDirect, on page 262

Property:RoundPrice, on page 268

Property:TickPrice, on page 271

Property:TickPriceEx, on page 273

Property: IsValidGet

Purpose

Returns whether an attribute name is a valid argument for the `Get` property

Syntax

attrValues = myInstrObj.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information

Property:Get, on page 234
Property:IsValidUpdateFilter, on page 258
Property:ReadProperties, on page 267

Property: IsValidUpdateFilter

Purpose

Identifies whether an object attribute can be used to filter instrument updates

Syntax

`isValidArr = myInstrObj.IsValidUpdateFilter(attributes)`

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

isValidArr	Array of Boolean values for each argument name supplied in attributes. Data type: Array

Exceptions

Exception	Meaning
Invalid Parameter	One or more of the specified attributes in not filterable attribute

Description

The `IsValidUpdateFilter` property determines whether a specified array of object attributes represent valid attributes to use for filtering instrument notifications. The property returns an array, where each element contains a Boolean value indicating whether the name specified in the same element of the input array is a valid attribute.

You typically use this property to ensure that all of the arguments you want to pass to the `InstrFilter` property are valid. By checking for errors first, you can avoid exceptions thrown for invalid arguments.

You can use one or more of the following object attributes to identify which values to monitor for notification.

- **Ask**
- **AskQty**
- **Bid**
- **BidAskMid**
- **BidQty**
- **Change**
- **Close**
- **High**
- **HitTake**
- **IAsk**
- **IAskQty**
- **IBid**
- **IBidQty**
- **Last**
- **LastQty**
- **LLimit**
- **Low**
- **MIAsk**
- **MIAskQty**
- **MIBid**
- **MIBidQty**
- **Open**
- **OTCPrc**
- **OTCQty**
- **PL**
- **PL~**
- **PL.Z**
- **PL.[*Currency*]**
- **Settle**
- **Status**
- **TAsk**
- **TAskQty**
- **TBid**
- **TBidQty**
- **TPrc**
- **ULimit**
- **Vola**
- **Volume**

Example

The following example uses the `IsValidUpdateFilter` property to determine whether all of the specified filters represent valid update filters. If not, it displays the invalid filters.

```
Private Function CheckFilters() As Boolean

  'if all filters are valid this remains equal to true
```

```
  Dim boolCheckFiltersOK As Boolean
  boolCheckFiltersOK = True

  'string to hold the filters
  Dim strFilters As String
  strFilters = "Ask,AskQty,BidAskMid,BidQty,Change,Close,Delta"

  'boolean to hold the values in the array that is going to
  'be returned (one bool for each filter checked)
  Dim boolFilters As Boolean
  Dim arrayOfBools As Array

  'to count the interations of boolean values
  Dim i As Integer
  i = 0

  'the call passes the list of filters
  arrayOfBools = Me.m_TTInstrObj.IsValidUpdateFilter(strFilters)

  'to hold the invalid property
  Dim strBadProperty As String

  Try
    For Each boolFilters In arrayOfBools

      'keeps track of which filter is being checked
      strBadProperty = strFilters.Split(",").GetValue(i)
      i = i + 1
      If boolFilters = False Then
        Console.WriteLine(strBadProperty)
        boolCheckFiltersOK = False
      End If
    End For
  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try

  'returns true if all filters are valid
  Return boolCheckFiltersOK
End Function
```

Related information Property:IsValidGet, on page 256

Property: IsVolatile

Purpose

Indicates whether an object attribute for an instrument can change

Syntax

`isValidArr = myInstrObj.IsVolatile(attributes)`

Parameters

attributes	Array of attribute names Data type: Array

Return value

isValidArr	Array of Boolean values for each argument name supplied in attributes. Data type: Array

Exceptions

None

Description

The `IsVolatile` property returns a Boolean value indicating whether a particular `Get` attribute can change during the life of the instrument object. Some attributes, like **Bid** or **Ask**, can change after an instrument is accessed. For example, in an active market, a bid price might change while you are processing the existing price. Others, like **Market** or **Name**, are static and do not change over the life of the instrument in your application.

The property returns an array, where each element contains a Boolean value indicating whether the name specified in the same element of the input array is a valid attribute.

Example

The following example shows how to use the `IsVolatile` property to determine whether an attribute is volatile.

```
'Create the string of properties to check
Dim strAttribs As String
strAttribs = "Market,Name,Bid"

Dim boolIsVolatile As Boolean
Dim arrayOfBools As Array

'to count the interations of boolean values
Dim i As Integer
i = 0

'the call passes the list of filters and returns an array of booleans
arrayOfBools = Me.m_TTInstrObj.IsVolatile(strAttribs)

'to hold the string that is being checked if it is volatile or not
Dim strVolatile As String

Try
  For Each boolIsVolatile In arrayOfBools

    'keeps track of which filter is being checked
    strVolatile = strAttribs.Split(",").GetValue(i)
    i = i + 1

  If boolIsVolatile = False Then
    Console.WriteLine("Property is NOT volatile: {0}", strVolatile)
```

```
  ElseIf boolIsVolatile Then
    Console.WriteLine("Property is volatile: {0}", strVolatile)
  End If
  Next
Catch ex As Exception
  Console.WriteLine(ex.Message)
End Try
```

Related information Property:IsTradablePrice, on page 254

Property: MergeImpliedsIntoDirect

Purpose

Indicates whether implied prices and quantities are merged into the inside markets

Syntax

```
isMerged = myInstrObj.MergeImpliedsIntoDirect
myInstrObj.MergeImpliedsIntoDirect = isMerged
```

Parameters

isMerged	Whether to merge implied prices quantities into the direct prices and quantities Data type: Boolean

Return value

isMerged	Whether the XTAPI currently merges implied prices and quantities Data type: Boolean

Exceptions

None

Description

The `MergeImpliedsIntoDirect` property merges implied prices and quantities, when available, into the direct prices and quantities for the inside market. After the merge, the direct prices and quantities contain whichever of the implied or direct prices and quantities is the most advantageous.

The XTAPI maintains three sets of prices and quantities (direct, implied, and merged) for the following prices and quantities:

- **Ask**, **AskQty**, **Bid**, **BidQty** (direct)
- **IAsk**, **IAskQty**, **IBid**, **IBidQty** (implied)
- **MIAsk**, **MIAskQty**, **MIBid**, **MIBidQty** (merged)

For example, the instrument's `Get` property contains the following attributes for a bid price:

- **Bid** is the direct bid price
- **IBid** is the implied bid price
- **MIBid** is the merged implied price (the better of the two)

When you enable this property, the **Bid** price is overwritten with the **MIBid** price. Otherwise, the **Bid** and **MIBid** prices remain distinct. The same is true for all other sets of price-related instrument attributes as well. The following table illustrates the effect of this property for implied bids. where the bid and implied bid are 101 and 102, respectively.

Merge Implieds?	Bid	IBid	MIBid
No	101	102	102
Yes	102	102	102

Example

None

Related information

Property:CalculateTTImplieds, on page 221

Property:IsTradablePrice, on page 254
Property:TickPrice, on page 271
Property:TickPriceEx, on page 273

Property: OrderSet

Purpose

Returns or attaches the instrument to a specified order set

Syntax

```
orderSet = myInstrObj.OrderSet
myInstrObj.OrderSet = orderSet
```

Parameters

orderSet	Order set object to associate with the instrument Data type: `TTOrderSet`

Return value

orderSet	Order set to which this instrument is currently attached Data type: `TTOrderSet`

Exceptions

Exception	Meaning
Invalid OrderSet	The specified orderSet is not a valid TTOrderSet object.

Description

The `OrderSet` property identifies the order set attached to the instrument. If you attach an order set to the instrument, the instrument notifier associated with this instrument generates an `OnOrderSetUpdate` event when the order set updates.

Example

None

Related information

TTOrderSet Class, on page 357

Property: ProdType

Purpose

Sets or returns a product type for the instrument

Syntax

```
productType = myInstrObj.ProdType
myInstrObj.ProdType = productType
```

Parameters

productType — Type of product contract to associate with this instrument. Valid types include:

- FUTURE
- SPREAD
- OPTION
- STRATEGY
- STOCK
- BOND
- WARRENT
- FOREX
- ENERGY
- NDF

Data type: String

Return value

productType — Product type currently associated with this instrument

Data type: String

Exceptions

Exception	Meaning
Invalid OrderSet	Thrown when the *productType* value is missing or invalid when setting the property

Description

The `ProdType` property identifies the type of product represented by the instrument.

Example

The following example sets the `ProdType` to **FUTURE**.

```
' Keep track of the instrument (contract) we are using
Dim myInstr As New TTInstrObj

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstr.Exchange = "CME"
myInstr.Product = "ES"
myInstr.ProdType = "FUTURE"
myInstr.Contract = "Sep13"

' Open the instrument with depth
call myInstr.Open(True)
```

Related information

Property:Contract, on page 222

Property:Product, on page 266

Property: Product

Purpose

Sets or returns the name of the product associated with the instrument

Syntax

```
productName = myInstrObj.Product
myInstrObj.Product = productName
```

Parameters

productName	Name of the tradable product to associate with the instrument Data type: String

Return value

productName	Product type currently associated with this instrument Data type: String

Exceptions

Exception	Meaning
Invalid Product Name	Thrown when the *productName* value is missing or invalid when setting the property

Description

The `Product` property identifies the name of the product being traded with this instrument.

Example

The following example sets the `Product` to **ES**.

```
' Keep track of the instrument (contract) we are using
Dim myInstr As New TTInstrObj

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstr.Exchange = "CME"
myInstr.Product = "ES"
myInstr.ProdType = "FUTURE"
myInstr.Contract = "Sep13"

' Open the instrument with depth
call myInstr.Open(True)
```

Related information

Property:Alias, on page 220

Property:Contract, on page 222

Property:ProdType, on page 265

Method:Open, on page 275

Property: ReadProperties

Purpose

Returns information about **TTInstrObj** attributes you can access through the `Get` property

Syntax

attributes = myInstrObj.ReadProperties

attrDescription = myInstrObj.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **TTInstrObj** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 234

Property:IsValidGet, on page 256

Property: RoundPrice

Purpose

Returns a valid tick price for an estimated price

Syntax

```
roundPrice = myInstrObj.RoundPrice(price, roundMethod, priceFormat)
```

Parameters

price	Estimated price in ticks, points, or X_TRADER display format display format. Data type: Integer, Decimal, or String
roundMethod	Algorithm to use for finding the nearest valid tick price for the instrument. Possible values include: • ROUND_NONE • ROUND_UP • ROUND_DOWN • ROUND_NEAREST Data type: `enumRoundPriceType`
priceFormat	Format for the returned price data. Possible values include: • & (integer representing ticks) • # (decimal representing points) • $ (string representing the X_TRADER display format) • ~ (string representing the differential for inverted tick products) • @ (primary currency) • ^ (native currency) • [*cur*] (specified currency) Data type: String

Return value

roundPrice	Valid tick price that matches the rounding scheme specified in *roundMethod*. Data type: Integer

Exceptions

Exception	Meaning
Conversion Failed	XTAPI could not convert the value to a price
Invalid Enum Value	The *roundPriceType* value is not a valid enumeration.
Invalid Instrument	The instrument contains no instrument data.
Invalid Output Type	The *priceFormat* parameter contains an invalid value or data type.
Invalid Price	The *basePrice* parameter contains an invalid value or data type
Instrument Not Initialised	The instrument has not been opened or retrieved
Value overflows signed 64-bit integer	One of the values is too large.

Exception	Meaning
Value overflows signed 32-bit integer	The value of *roundPrice* is larger than can be stored in a 32-bit integer.

Description

The RoundPrice property determines the valid tick price that best matches the specified rounding criteria for a price. You can pass a price in ticks (integer), points (decimal), or XTAPI display format (string). The property interprets the price based on the data type. The following table explains how the property determines which tick price to return.

Value	Description
ROUND_NONE	Converts the specified price to the format specified by the priceFormat parameter
ROUND_UP	Returns the first valid tick price higher than the specified price
ROUND_DOWN	Returns the first valid tick price lower than the specified price
ROUND_NEAREST	Returns the valid tick price nearest to the specified price, either up or down

The *priceFormat* parameter determines the format for the returned price. The output format can differ from the input format. If you only want to convert price values from one format, such as ticks, to another, such as XTAPI display format, specify ROUND_NONE for the *roundMethod* parameter.

Example

The following example rounds an invalid price to the closest ask and bid prices and the nearest valid price.

```
Private Sub getRounds(testPrice as String)
   Dim myInstr As New TTInstrObj
   Dim isTradable As Boolean
   Dim upPrice As Integer
   Dim downPrice As Integer
   Dim nearPrice As Integer

   isTradable = myInstr.IsTradablePrice(testPrice)
   If Not isValid Then
      upPrice = myInstr.RoundPrice(testPrice, ROUND_UP, "&")
      downPrice = myInstr.RoundPrice(testPrice, ROUND_DOWN, "&")
      nearPrice = myInstr.RoundPrice(testPrice, ROUND_NEAREST, "&")
      Debug.Print "Up: ", askPrice
      Debug.Print "Down: ", bidPrice
      Debug.Print "Near: ", nearPrice
   End If
End Sub
```

Related information

Property:IsTradablePrice, on page 254

Property:TickPrice, on page 271

Property:TickPriceEx, on page 273

Property: SeriesKey

Purpose	Sets or returns the series key associated with the instrument
Syntax	`instrKey = myInstrObj.SeriesKey` `myInstrObj.SeriesKey = instrKey`
Parameters	*instrKey* — Unique, internal ID TT associates with the instrument Data type: String
Return value	*instrKey* — Unique, internal ID associated with the instrument Data type: String
Exceptions	None
Description	The `SeriesKey` property contains the unique series key that TT associates with an instrument. You can use the series key to open an instrument in place of the contract. Using the series key is particularly useful if your application saves state information in workspaces. When the application loads the workspace, it can quickly open the instruments using their `SeriesKey` property values.
Example	This callback function uses the `SeriesKey` property to get the series key for an updated instrument and creates a hash table to map the series key to the contract.
Related information	None

```
Private Sub m_TTInstrNotify_OnNotifyFound(ByVal pNotify As
XTAPI.TTInstrNotify,
           ByVal pInstr As XTAPI.TTInstrObj) Handles
m_TTInstrNotify.OnNotifyFound

  'Get the SeriesKey and the contract names from the found instrument
  Dim strSeriesKey As String
  Dim strContractName As String
  strSeriesKey = pInstr.SeriesKey
  strContractName = pInstr.Contract

  'Store the value of the contract name and the series key in
  'a hashtable that uses the contract name to retrieve the series key
  Me.hashTableOfSeriesKeys.Add(strContractName, strSeriesKey)

End Sub
```

Property: TickPrice

Purpose

Returns the tradable price some number of ticks from a specified price

Syntax

```
tPrice = myInstrObj.TickPrice(basePrice, priceLevels, [priceFormat])
```

Parameters

basePrice	Base price for the instrument in ticks, points, or X_TRADER display format. Data type: Integer, Decimal, or String
priceLevels	Number of price levels away from the specified base price Data type: Integer
priceFormat	Format for the returned price data. Possible values include: • & (integer representing ticks) • # (decimal representing points) • $ (string representing the X_TRADER display format) • ~ (string representing the differential for inverted tick products) • @ (primary currency) • ^ (native currency) • [*cur*] (specified currency) Data type: String

Return value

tPrice	Price of the instrument, formatted according to the priceFormat parameter Data type: Depends on the requested format

Exceptions

Exception	Meaning
Arithmetic Overflow	An internal error occurred.
Conversion Failed	XTAPI could not convert the value to a price.
Invalid Instrument	The instrument contains no instrument data.
Instrument Not Initialised	The instrument has not been opened or retrieved.
Invalid Output Type	The *priceFormat* parameter contains an invalid value or data type.
Invalid Price	The *basePrice* parameter contains an invalid value or data type.
Value overflows signed 64-bit integer	One of the values is too large.
Value overflows signed 32-bit integer	The value of *tPrice* is larger than can be stored in a 32-bit integer.

Description

The `TickPrice` property takes a base price and locates the tradable price nearest the specified number of price levels away. You can pass a price in ticks (integer), points (decimal), or XTAPI display format (string). The property interprets the price based on the data type. If you specify a positive priceLevels, the property returns a price higher than the base price; otherwise, it returns the next price lower than the base price.

You can also use this property to convert a price from one format to another by specifying the price level as 0.

Note: This property does not support products with variable or premium ticking. When trading these types of products or to take advantage of enhanced functionality, you should use the `TickPriceEx` property.

Example

This callback function uses the `TickPrice` property to get the series key for an updated instrument and creates a hash table to map the series key to the contract.

```
' To see this used in a larger context look at the HoldOrders property
' example in the Order Set chapter
Private Sub GetTickValue()

  Try
    ' Obtain the tick price of the instrument.
    Dim dblTickPrice As Double =
        Convert.ToDouble(Me.m_TTInstrObj.TickPrice(0, 1, "#"))

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Related information

Property:CalculateTTImplieds, on page 221

Property:IsTradablePrice, on page 254

Property:MergeImpliedsIntoDirect, on page 262

Property:RoundPrice, on page 268

Property:TickPriceEx, on page 273

Property: TickPriceEx

Purpose

Returns the tradable price some number of ticks from a specified price after rounding

Syntax

`tPrice = myInstrObj.TickPriceEx(basePrice, roundMethod, priceLevels, [priceFormat])`

Parameters

basePrice	Base price for the instrument in ticks, points, or X_TRADER display format. Data type: Integer, Decimal, or String
roundMethod	Algorithm specifying the direction to use when rounding the base price to a valid price level for the instrument. Possible values include: • ROUND_NONE • ROUND_UP • ROUND_DOWN • ROUND_NEAREST • ROUND_ERROR Data type: `enumRoundPriceType`
priceLevels	Number of price levels away from the specified base price Data type: Integer
priceFormat	Optional format for the returned price data. Possible values include: • & (integer representing ticks) • # (decimal representing points) • $ (string representing the X_TRADER display format) • ~ (string representing the differential for inverted tick products) • @ (primary currency) • ^ (native currency) • [*cur*] (specified currency) Default: Same type as basePrice Data type: String

Return value

tPrice	Price of the instrument, formatted according to the priceFormat parameter Data type: Depends on the requested format

Exceptions

Description

The `TickPriceEx` property takes a base price and returns the price at the specified number of price levels away. You can pass a price in ticks (integer), points (decimal), or X_TRADER display format display format (string). The property interprets the price based on the data type. If you specify a positive priceLevels, the property returns a price higher than the base price; otherwise, it

returns the next price lower than the base price. The property rounds the base price based on `roundMethod` and then applies the *`priceLevels`* offset.

The roundMethod parameter determines the valid tick price that best matches the specified rounding criteria for a price.The following table explains the rounding options.

Value	Description
ROUND_NONE	Returns the valid price *`priceLevels`* away from the *`basePrice`* without first rounding the *`basePrice`*.
ROUND_UP	Returns the valid price *`priceLevels`* away from the *`basePrice`* after first rounding the *`basePrice`* up to the next valid tick price.
ROUND_DOWN	Returns the valid price *`priceLevels`* away from the *`basePrice`* after first rounding the *`basePrice`* down to the next valid tick price.
ROUND_NEAREST	Returns the valid price *`priceLevels`* away from the *`basePrice`* after first rounding the *`basePrice`* up to the nearest valid tick price, without respect to direction.
ROUND_ERROR	Forces `TickPriceEx` to throw an "Invalid Price" exception if basePrice is not a valid tick price.

You can also use this property to convert a price from one format to another by specifying the price level as 0.

Example

The following example rounds up to the next valid trading price if 30 is not a valid tick price and returns the valid price two price levels away after converting it to a decimal value.

```
' To see this property used in a larger context look at the HoldOrders property
' example in the Order Set chapter
Private Sub GetExTickValue()

  Try
    ' Obtain the tick price of the instrument.
    Dim dblTickPrice As Double =
        Convert.ToDouble(Me.m_TTInstrObj.TickPriceEx(30, ROUND_UP, 2, "#"))

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Related information

Property:CalculateTTImplieds, on page 221

Property:IsTradablePrice, on page 254

Property:MergeImpliedsIntoDirect, on page 262

Property:RoundPrice, on page 268

Property:TickPrice, on page 271

Method: Open

Purpose

Opens an instrument for trading

Syntax

`myInstrObj.Open[(withDepth)]`

Parameters

withDepth	Whether to retrieve market depth information when opening the instrument Data type: Boolean; Default: FALSE

Return value

None

Exceptions

None

Description

Note: You must attach the instrument to a **`TTInstrNotify`** object before calling this method. If you do not, you will not receive callbacks.

The `Open` method establishes a connection for the instrument, validates the instrument, and registers for updates. The method first connects to the TT Gateway and verifies that the instrument is valid for the exchange, returning either a **`TTInstrNotify`** `OnNotifyFound` or `OnNotifyNotFound` event. Next, it registers for the requested price updates. By default, the method requests updates for the inside market and registers for **`TTInstrNotify`** `OnNotifyUpdate` events.

If you want to monitor prices for the market depth, you can set *`withDepth`* to **TRUE**, which enables callbacks for changes in prices at all depths. Registering for market depth price updates can generate a lot of callbacks. If you enable *`withDepth`*, the method registers for **`TTInstrNotify`** `OnNotifyDepthData` events.

Note: If you open instruments with price depth, you should define an instrument filter for the **`TTInstrNotify`** object to limit the number of price changes to those of interest.

Known Issue:

Currently, once you open an instrument and subscribe to price updates, the XTAPI continues to receive updates until your application terminates. Even if you delete the instrument notifier and its corresponding instrument object, the XTAPI continues to receive the information from the exchange. Your application would no longer receive update events, but would still incur the overhead of getting the updates.

Example

The following example opens a CME instrument with market depth.

```
' Keep track of the instrument (contract) we are using
Dim myInstrObj As New TTInstrObj
Dim myNotifier As New TTInstrNotify

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstrObj.Exchange = "CME-B"
myInstrObj.Product = "ES"
myInstrObj.ProdType = "FUTURE"
myInstrObj.Contract = "Sep13"
```

```
' Alias this instrument
myInstrObj.Alias = "SP Mini"

' Create a notifier and attach to it
myNotifier = myInstrObj.CreateNotifyObject

' Open the instrument with depth
call myInstrObj.Open(TRUE)
```

Related information

Property:Alias, on page 220

Property:Product, on page 266

Method:Retrieve, on page 277

Method: Retrieve

Purpose	Verifies that a instrument is valid
Syntax	`myInstrObj.Retrieve`
Parameters	None
Return value	None
Exceptions	None

Description

Note: You must attach the instrument to a **`TTInstrNotify`** object before calling this method. If you do not, you will not receive callbacks.

The `Retrieve` method retrieves the instrument from the exchange. It verifies whether the instrument is valid for the exchange and sends either a **`TTInstrNotify`** `OnNotifyFound` or `OnNotifyNotFound` event, as appropriate. Unlike the `Open` method, the `Retrieve` method does not register for price updates. Because it does not receive price updates, you cannot query price-related `Get` property attributes for the instrument. You typically use this method to verify that the instrument is valid before submitting an order.

If you attach the instrument to a `TTInstrNotify` object, the method triggers the `OnNotifyFound` and `OnNotifyNotFound` events, as appropriate.

Note: Once you call the `Open` method for this instrument, the XTAPI ignores all subsequent `Retrieve` calls.

Example

The following example retrieves an instrument prices with price depth.

```
' Keep track of the instrument (contract) we are using
Dim myInstrObj As New TTInstrObj
Dim myNotifier As New TTInstrNotify

' Set instrument information (Gateway, Product, Product Type and Expiry)
myInstrObj.Exchange = "CME-B"
myInstrObj.Product = "ES"
myInstrObj.ProdType = "FUTURE"
myInstrObj.Contract = "Sep13"

' Alias this instrument
myInstrObj.Alias = "SP Mini"

' Create a notifier and attach to it
myNotifier = myInstrObj.CreateNotifyObject

' Open the instrument with depth
call myInstrObj.Retrieve
```

Related information

Method:Open, on page 275

17 TTOrderObj Class

Chapter overview

The chapter describes the **`TTOrderObj`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTOrderObj class overview	280
Property:CreateOrderProfile	281
Property:GetChanged	283
Property:Get	285
Property:GetIsChanged	296
Property:HasNextOrder	297
Property:Instrument	299
Property:IsNull	300
Property:IsValidGet	301
Property:NextOrder	303
Property:ReadProperties	305
Method:DeleteOrder	306
Method:HoldOrder	307

TTOrderObj class overview

Overview

The **TTOrderObj** class is a snapshot of order information when the order is submitted. Every time that the order changes (e.g. partial fill or the trader changes the order, etc.) the XTAPI creates a new **TTOrderObj**.The **TTOrderSet** always represents the most current **TTOrderObj** for the order.

The XTAPI receives all orders associated with trader login. If a received order does not match an existing order object, the XTAPI creates a **TTOrderObj** for the order. Once the order obtains a corresponding **TTOrderObj**, it is sent through the order filter, represented by a **TTOrderSelector** in the **TTOrderSet**. The **TTOrderSet** stores pointers to all orders that match this filter.

The **TTOrderObj** object is a read-only object. You can use this object to access all order information available in X_TRADER. There are two additional things you can do using the **TTOrderObj** object. One is to delete the order that the **TTOrderObj** refers to and the other is to put the order on Hold.

Properties

Table 50 lists the **TTOrderObj** properties.

Property	Description
CreateOrderProfile	Creates an order profile to manage an order
GetChanged	Returns values for selected object attributes whose values have changed
Get	Returns a TTOrderObj object's attributes
GetIsChanged	Indicates whether specified object attribute values changed between order states
HasNextOrder	Determines whether an order has a new order state
Instrument	Returns the instrument object associated with the order
IsNull	Indicates whether this `TTOrderObj` object is actually a null object
IsValidGet	Returns whether an attribute name is a valid argument for the Get property
NextOrder	Returns the next snapshot in the order update chain
ReadProperties	Returns information about TTOrderObj attributes you can access through the Get property

Table 50. TTOrderObj properties

Methods

Table 51 lists the **TTOrderObj** methods.

Method	Description
DeleteOrder	Deletes or cancels an order
HoldOrder	Takes an order out of the market and puts it on hold

Table 51. TTOrderObj methods

Events

None

Property: CreateOrderProfile

Purpose

Creates an order profile to manage an order

Syntax

`orderProfile = myOrderObj.CreateOrderProfile`

Parameters

None

Return value

orderProfile	New order profile Data type: `TTOrderProfile`

Exceptions

.

Exception	Meaning
Invalid Customer Defaults	An error exists in the X_TRADER customer defaults.
X_TRADER is unavailable	The X_TRADER application is not currently running on your workstation.

Description

The `CreateOrderProfile` property creates a **`TTOrderProfile`** object that you can use to:

- Submit an order
- Modify an order
- Cancel an order
- Delete an order

The order profile contains the information needed to submit an order to an exchange, which is a combination of information in the **`TTOrderObj`** object and X_TRADER Customer Defaults (stored in the **`TTOrderProfile`** object).

Note: You should never reuse an order profile, as its internal values might change after an order is submitted. If you want to modify or remove an order, even if recently submitted, create a new order profile object for the order. Reusing a **`TTOrderProfile`** object could introduce a conflict with the previously submitted order.

Example

The following example shows how to use the `CreateOrderProfile` property to create an order profile from an order object.

```
Private Sub UpdateOrder()

  'Get a copy of the order
  Dim orderObj As XTAPI.TTOrderObj =
       m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

  'Create an orderprofile from the order
  Dim orderProfile As XTAPI.TTOrderProfile
  orderProfile = New XTAPI.TTOrderProfile
  orderProfile = orderObj.CreateOrderProfile()

  ' Set the new price and quantity.
  orderProfile.Set("Limit$", "550")
  orderProfile.Set("Qty", "100")

  ' Update Order as change or cancel/replace (0 - change, 1 -
```

```
  ' cancel/replace).
  m_TTOrderSet.UpdateOrder(orderProfile, 0)

End Sub
```

Related information TTOrderProfile Class, on page 309

Property: GetChanged

Purpose

Returns values for selected object attributes whose values have changed

Syntax

attrValues = myOrderObj.GetChanged(*attributes*)

Parameters

attributes	Comma-separated list of object attributes Data type: String

Return value

attrValues	Array of attribute values Data type: Array

Exceptions

.

Exception	Meaning
Invalid Parameter	One or more of the specified attributes in not valid for a **TTOrderObj** object.
No New Order	The order has been deleted or filled.

Description

The GetChanged property returns the values for an arbitrary number of **TTOrderObj** object attributes whose values on the exchange changed in a recent order update. The property returns an array of values corresponding, in order, to the specified attributes. If an attribute value changed, the array element contains the new value; otherwise the element contains an empty Variant. You must cast the returned values to the appropriate data types.

Example

The following example displays new values, if changed, for the **OrdAction**, **ExecutionType**, **Price**, **Qty**, and **Acct** order attributes.

```
Dim myOrder As XTAPI.TTOrderObj
' After an order send, get the TTOrderObj from the order profile
myOrder = CurOrderProfile.GetLastOrder

' Check to see if the order has a next state
If myOrder.HasNextOrder Then

   ' Find out which fields have changed between the 2 states

   Dim data As Variant
   data = myOrder.GetChanged("OrdAction, ExecutionType, Price, Qty, Acct")
   ' Print the changed values
   Dim changedString As String
   changedString = ""
   Dim i As Integer
   For i = 0 To UBound(data)
      If Not IsEmpty(data(i)) Then
         If Not changedString = "" Then
            changedString = changedString & ", "
         End If
         changedString = changedString & data(i)
      End If
   Next
   If Not changedString = "" Then
         Debug.Print "New values: " & changedString
```

```
   End If
End If
```

Related information

Property:Get, on page 285
Property:GetIsChanged, on page 296
Property:ReadProperties, on page 305

Property: Get

Purpose

Returns a **`TTOrderObj`** object's attributes

Syntax

attrValues = myOrderObj.Get(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **TTFillObj** object attribute

Description

The `Get` property returns the values for an arbitrary number of **`TTOrderObj`** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single `ChangeMe` property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 52 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 52. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 52. Price and quantity value formats

Table 53 lists the **`TTOrderObj`** object attributes you can supply to the `Get` property.

Attribute	Description	DataType
Acct	Customer's or trader's account number	String
AcctType AccountType	Code indicating the desired account type	String
Broker	Broker's name (multi-broker only)	String
Broker#	Broker's ID (multi-broker only)	String
BuySell BuySell$	Type of order. Possible values include: **B**: Buy **S**: Sell Specifying BuySell$ returns Buy or Sell.	String
Change Chg	Whether the order changed	Boolean
ClrMember	Clearing Member ID	String
CntrGroup	Counter Side Clearing Group ID	String
CntrMember	Counter Side Clearing Member ID	String
CntrTrader	Counter Side Clearing Trader ID	String
ColorPri ColorSec	Color to display in the MD Trader® working orders grid, where: • **ColorPri** represents the order's main color (left-side of the grid) • **ColorSec** represents the secondary color (right-side of the grid) The value is an integer value representing the RGB value for the color. The value corresponds to a Microsoft `ColorRef` data type.	Integer
Company	Broker's company name (multi-broker only)	String
Company#	Broker's company ID (multi-broker only)	String
ConnectionName	Broker's connection name (multi-broker only)	String
Contract Contract$	Instrument's name with expiration (e.g. ES Sep13)	String
ContractAlias	Instrument's alias name with expiration (e.g. E-Mini Sep13)	String
Contract.getAttribute	Get attribute for the attached instrument (TTInstrObj)	Various
Date	Date of the order in *MM*/*DD*/*YY* (e.g. 05/18/12) The value contains valid data only in the date portion of the `COleDateTime` object.	COleDateTime

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
DisclosedQty DiscQty	Quantity of the order visible in the market. Valid only for disclosed quantity orders.	Integer
Ex:OrderNo	Order number for the specified exchange (e.g. CME: 107578464)	String
ExchCred	Exchange credentials (multi-broker only)	String
ExecutionType	Current state of the order. Possible values include: **-1**: Order is in an invalid state **0**: Order is in pre-execution state (not yet sent to a TT Gateway) **1**: Order is in post-execution state and the XTAPI has received a response from the TT Gateway	Integer
ExGroup	Exchange Group ID for this order	String
ExMember	Exchange Member ID for this order	String
ExOrderID	Exchange Order ID assigned to this order. This attribute corresponds to the **OrderNo** attribute for exchanges that do not return strictly numeric order numbers. **Note:** If the exchange does not provide a value, the XTAPI populates it with the OrderNo attribute value.	String
Expiry	Expiration date for the order. Possible values include: • Null: GTC order • Today's date: GIS or GTD order • Future date: Expiration date The value contains valid data only in the date portion of the `COleDateTime` object.	COleDateTime
ExTrader	Exchange Trader ID for this order	String
ExTransNo	Unique identifier for the order transaction.	String
FFT1 FFT2 FFT3	Free-form text fields. **FFT1** typically contains the account number.	String
Fill FillQty	Order quantity already filled	Integer
GiveUp	Give-up member's clearing ID, when the fill is assigned to another member **Note:** The TT ICE Gateway does not support the **GiveUp** attribute; instead, it uses the **ClrMember** attribute for Give Up orders.	String
Group	Group name of the trader from the Member/Group/Trader ID	String
Instr Instr$	Default instrument name as displayed in the X_TRADER® **Market Grid Contract** field (e.g., ES Sep13)	String
InstrAlias	Instrument's alias name with expiration (e.g. E-Mini Sep13)	String
Instr.getAttribute	Value of the specified object attribute in the TTInstrObj object	Various
IsAON	Whether the order is an All-Or-None order	Boolean
IsAuction	Whether the order is only valid for auction	Boolean

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
IsAuto-Aggress	Whether the order is auto-aggressive (Currently supported only for BrokerTec)	Boolean
IsAutomated	Whether the order orginates from an automated program.	Boolean
IsAwayMarket	Whether an away market maker should match BBO of an away market	Boolean
IsBasis	Whether the order is a basis order	Boolean
IsBestLimit	Whether the order is a best limit order (Currently supported only for the TT TOCOM Gateway)	Boolean
IsBlock	Whether the order is a block order	Boolean
IsBuy	Whether the order is a buy order	Boolean
IsFAS	Whether the order is a fill and save order	Boolean
IsFlex	Whether the order is a flex option order	Boolean
IsFOK	Whether the order is a fill or kill order	Boolean
IsGIS	Whether the order is a good-in-session order	Boolean
IsGTC	Whether the order is a good-tll-canceled order	Boolean
IsGTD	Whether the order is a good-till-date order	Boolean
IsIceberg	Whether the order is an Iceberg order	Boolean
IsIfTouched	Whether the order requires additional action when a price is touched	Boolean
IsIOC	Whether the order is an immediate-or-cancel order	Boolean
IsLOC	Whether the order is a limit-on-close order to be filled or killed when market closes	Boolean
IsLOO	Whether the order is a limit-on-open order to be filled or killed when the market opens	Boolean
IsLSM	Whether the order is a limit-stop-market order	Boolean
IsLTM	Whether the order is a limit to market order to be converted to a market order on close	Boolean
IsMarketToLimit	Whether the order is a market-to-limit order	Boolean
IsMLM	Whether the order is a market-limit-market order	Boolean
IsMOC	Whether the order is a market-on-close order	Boolean
IsMOO	Whether the order is a market-on-open order	Boolean
IsMTL	Whether the order is a market-to-limit order	Boolean
IsMV	Whether the order is a minimum volume order	Boolean
IsNon-Leave	Whether the order is a non-leave order (Currently supported only for BrokerTec)	Boolean

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
IsOrderColorSupported	Whether the version of X_TRADER® installed on a trader's system supports receiving order color information from an XTAPI application. If color is supported, X_TRADER® uses the values in the **ColorPri** and **ColorSec** object attributes for the **`TTOrderProfile`** and **`TTOrderObj`** objects in the MD Trader® widow. You can define the colors by setting the attributes with the Set method.	Boolean
IsPartial	Whether the order is partially filled	Boolean
IsPassive	Whether the order is passive (Currently supported only for BrokerTec)	Boolean
IsSell	Whether the order is a sell order	Boolean
IsShortExempt	Whether the order is a sell short order exempt from notification	Boolean
IsStop	Whether the order is a stop order	Boolean
IsTimed	Whether the order is a timed order	Boolean
IsTriggered	Whether the order has been triggered	Boolean
IsTriggeredTime	Whether the order was triggered after a time submission	Boolean
IsVolatility	Whether the order is a volatility order	Boolean
Key	Internal static key that represents the order throughout its lifetime	String
LinkType	Type of link identifying the relationship between orders. Values include: • Algo • AlgoC • AlgoL • AS • ASQ • ASH • ASR • OCO • OMA • SSE • SSEC • SSER • Stage-BKR • Stage-INT • Stage-UNK • UNM **Note**: In some instances an order can have more than one link type, separated by a comma (e.g. SSE, OCO).	String
Limit Limit# Limit$ Limit&	Limit used during price reasonability checks. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
Location	Order sender country and region code.	String
Member	Member ID of the trader from the Member/Group/Trader ID	String
MinQty	Minimum quantity needed to be immediately available at the limit price for this order to be submitted into the market	Integer
Msg	Exchange or gateway message describing the reason for an order reject or order delete	String
OldOrderNo	Previous number used for the order, if the exchange created a new order number	String
OnHold	Whether an order is on hold (out of the market)	Boolean
OpenClose OpenClose$	Whether the order opens or closes a position. Possible values include: **O**: Open **C**: Close **F**: FIFO Specifying **OpenClose$** returns the corresponding full string. **Note:** The TOCOM Exchange changes the **OpenClose** value of a fill corresponding to a FIFO order as either **C** (Close) or **O** (Open).	String
OrdAction	Action associated with the order. Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_CHANGE` • `ORD_ACTION_DELETE` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_HOLD` • `ORD_ACTION_REPLACE` • `ORD_ACTION_UNKNOWN`	String
OrderNo	Order number assigned by the exchange. **Note:** For CME products, this property contains a different representation of the order number that matches the format used by other exchanges. To see the actual order number received from the CME, check the **ExOrderID** attribute.	String
OrderQty	Number of contracts contained in the order	Integer
OrderRestr	Restriction on the order. For a list of order restrictions, see the section called Order restrictions, on page 639.	String

Table 53. TTDropHandler Get attributes

<table>
<tr><th>Attribute</th><th>Description</th><th>DataType</th></tr>
<tr><td>OrderSource
OrderSource$</td><td>Where the order originated:
0: X_TRADER
1: Autospreader Desktop
2: Autotrader
3: XTAPI
4: Ext Source (Exchange)
5: FA Licensed (Licensed FIX Adapter)
6: FA Unlicensed (Unlicensed FIX Adapter)
7: XT Lite (X_TRADER Lite)
8: TT_TRADER
9: Autospreader Engine
10: Synthetic SE Engine
11: TT API in X_TRADER Mode
12: Algo SE
21: TT API in Universal Login Mode
22: ADL
Specifying OrderSource$ returns the corresponding string.</td><td>As specified</td></tr>
<tr><td>OrdStatus
OrdStatus&</td><td>Indication of the response to the most recent action performed on the order. Possible values include:
<table><tr><th>Integer</th><th>String</th><th>Description</th></tr><tr><td>0</td><td>Void</td><td>Default value</td></tr><tr><td>8200</td><td>New</td><td>New order released by client</td></tr><tr><td>8201</td><td>Accepted</td><td>Order accepted by the server</td></tr><tr><td>8202</td><td>OK</td><td>Order active in the market</td></tr><tr><td>8203</td><td>XT Filled</td><td>Order filled</td></tr><tr><td>8204</td><td>Hold</td><td>Order is on hold</td></tr><tr><td>8205</td><td>Deleted</td><td>Order has been deleted</td></tr><tr><td>8206</td><td>Expired</td><td>Order has expired</td></tr><tr><td>8207</td><td>Rejected</td><td>Order was rejected</td></tr><tr><td>8208</td><td>Resubmitted</td><td>Hold order was resubmitted</td></tr><tr><td>8299</td><td>Unknown</td><td>Order status could not be retrieved</td></tr></table></td><td>As specified</td></tr>
<tr><td>OrderTag</td><td>User-definable text (15 char max) containing user-defined information included in the corresponding order profile.</td><td>String</td></tr>
<tr><td>OrderTraits</td><td>Traits associated with the order as a concatenated string containing the order status history, order flag, and order restriction.</td><td>String</td></tr>
</table>

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
OrderType OrderType$	Type of order. Possible values include: **B** (OCO): OCO order **C** (CMO): CMO order **L** (Limit): Limit order **M** (Mkt): Market order **O** (BatchOrder): Batch order **Q** (Quote): Quote order **S** (BL): Best Limit order (TOCOM only) **T** (MTL): Market To Limit order **X** (Cross): Cross order The text in parenthesis indicates the value associated with the $ suffix.	String
OrigOrdAction	Action specified in the original order submission, which might differ from the current order action. For example, a Cancel/Replace order comprises two separate order actions: Delete and Add. This attribute stores the original action (Replace, in this example). Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_CHANGE` • `ORD_ACTION_DELETE` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_HOLD` • `ORD_ACTION_REPLACE` • `ORD_ACTION_UNKNOWN`	String
ParentKey	Site order key identifying the parent order.	String
Price Price# Price$ Price&	Order price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
PriceCheck	Whether price reasonability checking is enabled	String
Qty	Number of contracts contained in the order	Integer
RecoverableFailure	Whether an order was rejected temporarily, but might succeed in the future	Boolean
Sender Sender$	IP address of the machine that submitted the order, as an integer or string	As specified
SenderSubID	Unique trader (or operator) ID used to distinguish between different traders when the current MGT routes orders for multiple traders. **Note:** Some exchanges, such as CME, require clients to provide this information when routing orders on behalf of multiple traders.	String (32 character limit)
Series.getAttribute	Value of the specified attribute in the **`TTInstrObj`** object	Various
SeriesKey	Internal ID that uniquely identifies the instrument contained in the order	String
SiteKey SiteOrderKey	TT-generated static key that represents the order throughout its lifetime	String

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
Source	Where the order originated: **0**: X_TRADER **1**: Autospreader Desktop **2**: Autotrader **3**: XTAPI **4**: Ext Source (Exchange) **5**: FA Licensed (Licensed FIX Adapter) **6**: FA Unlicensed (Unlicensed FIX Adapter) **7**: XT Lite (X_TRADER Lite) **8**: TT_TRADER **9**: Autospreader Engine **10**: Synthetic SE Engine **11**: TT API in X_TRADER Mode **12**: Algo SE **21**: TT API in Universal Login Mode **22**: ADL	Integer
Stop Stop# Stop$ Stop&	Stop price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
Tag(x)	User-defined tag, where x is any integer	String
Time Time$	Time the CoreAPI sent the order to the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeExchange TimeExchange$	Time the exchange received the order. The time represents the local time at the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeLocalUpd TimeLocalUpd$	Time the XTAPI received the order, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeProcessed TimeProcessed$	Time the TT Gateway processed the order and sent it to the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified

Table 53. TTDropHandler Get attributes

Attribute	Description	DataType
TimeReceived TimeReceived$	Time the TT Gateway received the order from the XTAPI, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeRouted TimeRouted$	Time the XTAPI sent the order to X_TRADER®, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object. NOTE: These values exist only for the lifetime of your application and are not visible to other applications.	As specified
Trader	Trader ID of the trader from the Member/Group/Trader ID	String
TriggerQty	Minimum traded quantity required for a stop to trigger at the specified stop price	Integer
UserData(x)	User-defined tag, where x is any integer Valid values: 0-1023	Integer
UserTag	User-definable text (15 char max) containing user-defined information included in the corresponding order profile.	String
Username	Trader login. The value differs based on the type of credentials used: • For MGT-based logins, the value contains the concatenation of the Member, Group, and Trader attributes. • For universal logins, the value contains the trader's login ID	String
Wrk WrkQty	Order quantity that is working and has not been filled	Integer

Table 53. TTDropHandler Get attributes

Example

The following example uses the `Get` property to display the values for the specified **`TTOrderObj`** attributes.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidNameArr As Array
  Dim attrValuesArr As Array
  Dim isValidList As Boolean
  Dim attrName As String
  Dim i As Integer
  i = 1

  'good practice to validate list before calling Get
  isValidNameArr = Me.m_TTOrderObj.IsValidGet(objAttributes)

  ' code to verify attribute list omitted

  If isValidList = True Then
    attrValuesArr = Me.m_TTOrderObj.Get(objAttributes)

    'Display each attribute and value
    For Each attrValue in attrValuesArr
```

```
        attrName = objAttributes.Split(",").GetValue(i)
        Console.Writeline("Attribute: {0} = {1}", attrName, attrValue)
        i = i + 1
        Next
      End For

    End If
End Sub
```

Related information

Property:GetChanged, on page 283

Property:GetIsChanged, on page 296

Property:IsValidGet, on page 301

Property:ReadProperties, on page 305

Property: GetIsChanged

Purpose

Indicates whether specified object attribute values changed between order states

Syntax

```
changedList = myOrderObj.GetIsChanged(attributes)
```

Parameters

attributes	Comma-separated list of object attributes Data type: String

Return value

changedList	Array of Boolean values corresponding to the list of object attributes Data type: Array

Exceptions

.

Exception	Meaning
Invalid Parameter	One or more of the specified attributes in not valid for a **`TTOrderObj`** object.
No New Order	The order has been deleted or filled.

Description

The `GetIsChanged` property compares the values of the specified object attributes after an order update. The property returns an array of Boolean values corresponding, in order, to the specified attributes. If an attribute value changed, the array element contains TRUE; otherwise, it contains FALSE. You must cast the returned values to the appropriate data types.

Example

None

Related information

Property:GetChanged, on page 283

Property:Get, on page 285

Property: HasNextOrder

Purpose	Determines whether an order has a new order state
Syntax	*orderExists* = myOrderObj.HasNextOrder
Parameters	None
Return value	*orderExists* — Whether an order changed Data type: Boolean
Exceptions	None

Description

The `HasNextOrder` property determines whether a change to an order occurred, such as a partial fill or an order update. The XTAPI maintains a chain of states for each order you submit. Each order you submit has a separate and unique order chain. As an order changes over time, such as being partially-filled, a new state is added to the chain. The `HasNextOrder` property helps you traverse the order chain by telling you whether the chain has a next link.

Figure 54 illustrates a sample order chain. When you submit an order, the XTAPI creates the head of the chain. After receiving the exchange acknowledgement, it adds a new state with a working order. In this example, the order changes three additional times as partial fills are received. The `HasNextOrder` returns TRUE for each order until the end of the chain is reached.

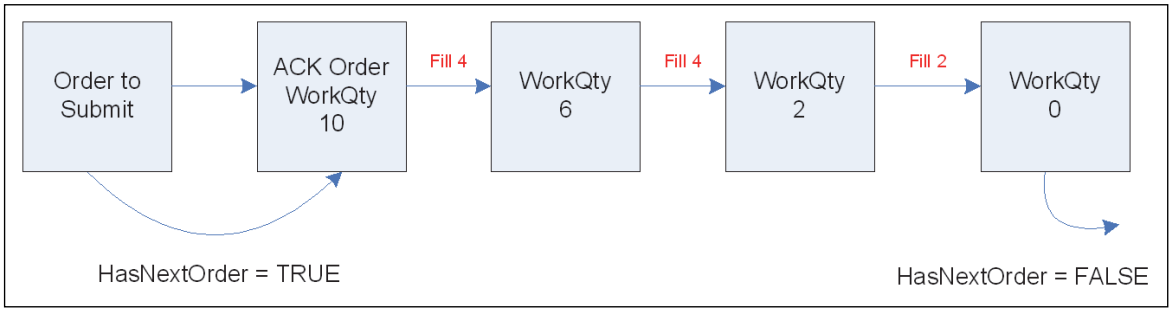


Figure 54. Using HasNextOrder to traverse an order chain

If an order has a next order in the chain, you can use the `NextOrder` property to return the next order. When traversing an order chain, your application must include logic for getting next orders and walking down the chain. You can also use the `GetChanged` and `GetIsChanged` properties to determine what changed between order states.

Note: You should always call this property before calling the `GetChanged`, or `GetIsChanged` properties. If the order has no next state, the properties throw exceptions.

Example

The following example shows how to use the `HasNextOrder` property to display changes between states in the order states.

```
Dim myOrder As XTAPI.TTOrderObj
' After an order send, get the TTOrderObj from the order profile
myOrder = CurOrderProfile.GetLastOrder
```

```
' Check to see if the order has a next state
If myOrder.HasNextOrder Then

   ' Find out which fields have changed between the 2 states

   Dim data As Variant
   data = myOrder.GetChanged("OrdAction, ExecutionType, Price, Qty, Acct")
   ' Print out the changed values
   ' If the entry in the array is not empty, then it contains the new value
   Dim changedString As String
   changedString = ""
   Dim i As Integer
   For i = 0 To UBound(data)
      If Not IsEmpty(data(i)) Then
         If Not changedString = "" Then
            changedString = changedString & ", "
         End If
         changedString = changedString & data(i)
      End If
   Next
   If Not changedString = "" Then
         Debug.Print "New values: " & changedString
   End If
End If
```

Related information

Property:IsNull, on page 300

Property:NextOrder, on page 303

Property: Instrument

Purpose

Returns the instrument object associated with the order

Syntax

```
instrObj = myOrderObj.Instrument
```

Parameters

None

Return value

instrObj	Pointer to an instrument object Data type: `TTInstrObj`

Exceptions

Exception	Meaning
COM QI Error	Some external COM error occurred
Invalid Instrument	The instrument object has yet been created. This error typically occurs when initially downloading orders during your application startup.
Invalid Order Object	The specified pointer does not reference a valid **`TTOrderObj`** object.
Unable to Create Instrument	Some internal error prevented the XTAPI from creating the **`TTInstrObj`** object.

Description

The `Instrument` property returns the instrument object associated with the order. You can use the object pointer to get information, such as the current price, about the instrument.

Example

None

Related information

None

Property: IsNull

Purpose	Indicates whether this **`TTOrderObj`** object is actually a null object
Syntax	*`nullObj`* `= myOrderObj.IsNull`
Parameters	None
Return value	*nullObj* — Whether the current object is a null object Data type: Boolean
Exceptions	None

Description

The `IsNull` property returns whether the current **`TTOrderObj`** pointer is referencing a null order object. Because orders can change rapidly and asynchronously, an order might cease to exist while your application still holds the object pointer. For example, an order might be completely filled during the time you are processing it. If you call a property that returns a **`TTOrderObj`**, but the corresponding order no longer exists, the XTAPI returns a null **`TTOrderObj`**.

Example

The following example uses the `IsNull` property to verify an order before submitting it to the exchange.

```
Private Sub UpdateOrder()

  'Get a copy of the order
  Dim orderObj As XTAPI.TTOrderObj =
      m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

  Dim boolOrderIsNull As Boolean
  boolOrderIsNull = orderObj.IsNull

  If Not (boolOrderIsNull) Then
    'Create an orderprofile from the order
    Dim orderProfile As XTAPI.TTOrderProfile
    orderProfile = New XTAPI.TTOrderProfile
    orderProfile = orderObj.CreateOrderProfile()

    ' Set the new price and quantity.
    orderProfile.Set("Limit$", "550")
    orderProfile.Set("Qty", "100")

  ' Update Order as change or cancel/replace (0 - change, 1 - cancel/replace).
    m_TTOrderSet.UpdateOrder(orderProfile, 0)
  End If

End Sub
```

Related information

Property:HasNextOrder, on page 297

Property:NextOrder, on page 303

Property: IsValidGet

Purpose

Returns whether an attribute name is a valid argument for the `Get` property

Syntax

attrValues = myOrderObj.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information Property:Get, on page 285

Property: NextOrder

Purpose

Returns the next snapshot in the order update chain

Syntax

```
orderSnapshot = myOrderObj.NextOrder
```

Parameters

None

Return value

orderSnapshot	Snapshot of the order Data type: `TTOrderObj`

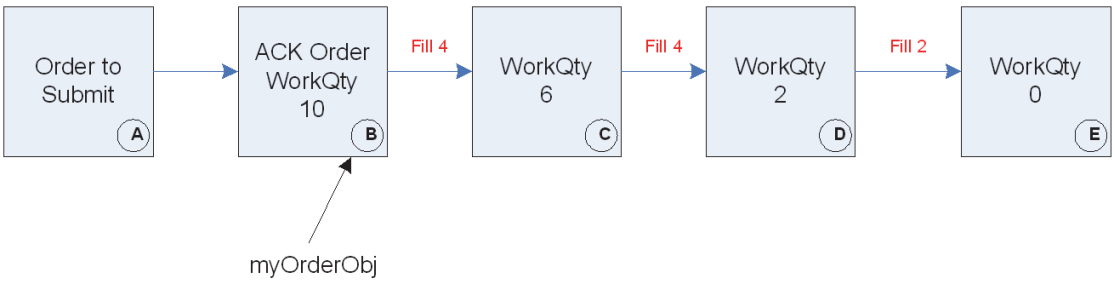
Exceptions

Exception	Meaning
Could not create order	The order does not have a next order in the chain.

Description

The `NextOrder` property returns the next state of the current order.

The following illustration shows a sample order chain with *myOrderObj* holding a pointer to B. Calling this property returns a **`TTOrderObj`** pointer to order C. If you call the `NextOrder` property when *myOrderObj* points to order E, the XTAPI returns a null **`TTOrderObj`** object.



Example

The following example uses the `NextOrder` property to step through states of an order.

```
Private Sub GetNextOrder()

  Try
    'Get a copy of the order
    Dim orderObj As XTAPI.TTOrderObj =
        m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

    While Not orderObj Is Nothing

      Console.WriteLine(orderObj.Get("OrdAction"))
      Console.WriteLine("Above was from new orderObj")
      orderObj = orderObj.NextOrder

    End While

    Catch ex As Exception
      Console.WriteLine(ex.Message)
      Console.WriteLine("Exception in GetNextOrder")
    End Try
```

```
End Sub
```

Related information

Property:HasNextOrder, on page 297
Property:IsNull, on page 300

Property: ReadProperties

Purpose

Returns information about **TTOrderObj** attributes you can access through the `Get` property

Syntax

attributes = myOrderObj.ReadProperties

attrDescription = myOrderObj.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **TTOrderObj** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 285

Method: DeleteOrder

Purpose	Deletes or cancels an order
Syntax	`myOrderObj.DeleteOrder`
Parameters	None
Return value	None
Exceptions	None
Description	The `DeleteOrder` method sends a delete message to the exchange to delete an order. When the XTAPI receives the confirmation message from the exchange, it also removes the order from the order book.

Example	The following example shows how to delete an order from an order set.
	`Dim ordKey As String`
	`Dim numDeleted As Long`
	`' code to get site key omitted`
	`Set myOrderObj = myOrderSet.SiteKeyLookup(ordKey)`
	`MyOrderObj.DeleteOrder`

Related information	Method:HoldOrder, on page 307

Method: HoldOrder

Purpose

Takes an order out of the market and puts it on hold

Syntax

```
myOrderObj.HoldOrder
```

Parameters

None

Return value

None

Exceptions

Exception	Meaning
Unable to put Order on Hold	Some condition occurred, such as an order being filled or canceled, that made putting a hold invalid.

Description

The `HoldOrder` method sends a delete message to the exchange for the order, but does not remove it from the order set. Because the order remains in the order set, you can re-submit the order at a later time. When you submit the order again, the XTAPI sends the exchange a new order with the remnants of the order you put on hold.

For example, suppose you submit an order for 50 ZB-Mar13 contracts and notice that 20 contracts are filled. Before the order is completely filled, you need to step away from your desk so you cannot monitor the order progress. You can put the order on hold until you return. The XTAPI sends a delete message for the remaining 30-contract order and leaves it in the order set. When you take the order off hold, the XTAPI submits the remaining 30 contracts as a new order.

Note: The exchanges have no concept of holding an order. Holding orders is a feature the XTAPI provides that allows you to temporarily halt trading for an order.

Example

The following example places an order on hold.

```
'Code to get an order set omitted

'Get a copy of the order
Dim orderObj As XTAPI.TTOrderObj =
     m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

'Check to make sure that the order is not null
Dim boolOrderIsNull As Boolean
boolOrderIsNull = orderObj.IsNull

If Not (boolOrderIsNull) Then
   'Place the order on hold
   orderObj.HoldOrder()
End If
```

Related information

Method:DeleteOrder, on page 306

18 TTOrderProfile Class

Chapter overview

The chapter describes the **`TTOrderProfile`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTOrderProfile class overview	310
Property:Alias	312
Property:Customer	313
Property:CustomerDefaults	315
Property:Customers	316
Property:Get	317
Property:GetLast	328
Property:GetLastOrder	330
Property:Instrument	332
Property:IsValidGet	332
Property:ReadProperties	334
Property:RoundOption	335
Property:WriteProperties	336
Method:Set	337

TTOrderProfile class overview

Overview

The **`TTOrderProfile`** class stores the information needed to submit or modify an order. A valid order contains a combination of information in the **`TTOrderProfile`** object and XTAPI Customer defaults.

The XTAPI submits orders from the **`TTOrderProfile`** object. The **`TTOrderProfile`** object uses the **`TTOrderSet`** Send method to send the orders to the TT Gateway for submission to the exchange. The XTAPI submits orders through X_TRADER, which also performs risk-checking before sending the order to the TT Gateway.

Because an order profile represents an order ready to send to the market, your application should not rely on it for any other purpose. After sending the order to the market, the XTAPI might modify or update fields in the associated order object. If your application uses or updates **`TTOrderProfile`** object attributes, the order represented by the updated order profile could differ from the order associated with the old order profile.

Note: You should never reuse an order profile, as its internal values might change after an order is submitted. If you want to modify or remove an order, even if recently submitted, create a new order profile object for the order.

TT generates a **SiteOrderKey** that uniquely identifies each submitted order. It remains intact and does not change when you change an order. You can use the **SiteOrderKey** attribute to track an order through its lifetime.

Note: You must run a Key Server somewhere in your TT Trading System network to prevent duplicate site order keys. If you do not run a Key Server, the TT system might generate duplicate keys, which can cause unpredictable errors.

Order handling differences between the XTAPI and X_TRADER

The XTAPI and X_TRADER manage working orders differently when a trader submits a new order with a decreased quantity for partially-filled orders. In X_TRADER, when a trader changes a partially-filled order to a quantity less than the filled quantity, X_TRADER removes the order from the market. For example, suppose a trader submits a 10-lot order that then gets a partial fill of 4. If the trader then changes the order with a quantity of 3, X_TRADER assumes that the desired working quantity has already been filled, so cancels the order.

The XTAPI, however, uses a different approach with changed order quantities, trying to keep the order alive in the market. When a trader changes an order quantity through an XTAPI application, the XTAPI uses the following algorithm:

- When the new quantity exceeds the partially-filled quantity, it applies the new quantity to the total quantity (New Qty = Exe Qty + Wrk Qty).
- When the new quantity is equal to or less than the partially-filled quantity, it sets the new quantity to as working quantity (New Qty = Wrk Qty).

Using the same example, the XTAPI reduces the working quantity from 6 to 3.

Properties

Table 55 lists the **`TTOrderProfile`** properties.

Property	Description
Alias	Sets or returns an alias for an order profile
Customer	Identifies which X_TRADER customer defaults to use for trading
CustomerDefaults	Sets or returns the customer defaults profile to use for this order profile
Customers	Returns the customer defaults created in X_TRADER

Table 55. TTOrderProfile properties

Property	Description
Get	Returns a TTOrderProfile object's attributes
GetLast	Returns the object attributes of the order returned by the GetLastOrder method
GetLastOrder	Returns the first order in an order chain
Instrument	Returns or defines the instrument object associated with the order profile
IsValidGet	Returns whether an attribute name is a valid argument for the Get property
ReadProperties	Returns information about TTOrderProfile attributes you can access through the Get property
RoundOption	Specifies the rounding style to use for calendar spreads
WriteProperties	Returns information about TTOrderProfile attributes you can access through the Set property

Table 55. TTOrderProfile properties

Methods

Table 56 lists the **`TTOrderProfile`** methods.

Method	Description
Set	Sets values for TTOrderProfile object's attributes

Table 56. TTOrderProfile methods

Events

None

Property: Alias

Purpose

Sets or returns an alias for an order profile

Syntax

```
aliasName = myOrderProfile.Alias
myOrderProfile.Alias = aliasName
```

Parameters

aliasName	Name to associate with the order profile Data type: String

Return value

aliasName	Name of the alias currently associated with the order profile Data type: String

Exceptions

None

Description

The `Alias` property assigns a user-defined name that can be used to reference an order profile. An alias provides a convenient method of accessing order profiles from other objects, such as a **`TTOrderSet`** object.

When you assign an alias to an order profile, the XTAPI also makes the profile available through the **`TTGate`** `OrderProfiles` property. From a global **`TTGateObj`** object, you can then access order profiles without instantiating a **`TTOrderSet`** object.

When you create an alias, the XTAPI adds an in-memory reference to the **`TTOrderProfile`** object, which has the side effect of incrementing its reference count. Because the reference count is greater than 0, the XTAPI does not remove it from memory during its ongoing garbage collection activities. Consequently, creating excessive aliases can increase the memory footprint of your application.

Example

The following code fragment shows how to create an alias for an order profile.

```
'Create an orderprofile and name its alias
Dim orderProfile As XTAPI.TTOrderProfile
orderProfile = New XTAPI.TTOrderProfile

Dim strAliasNameOfOrderProfile As String
strAliasNameOfOrderProfile = "TTSIMSIM001"
orderProfile.Alias = strAliasNameOfOrderProfile

Console.WriteLine("OrderProfileAlias {0}", orderProfile.Alias)
```

Related information

`TTInstrObj` Property:Alias, on page 220

`TTOrderSet` Property:Alias, on page 362

Property: Customer

Purpose

Identifies which X_TRADER customer defaults to use for trading

Syntax

```
customerProfile = myOrderProfile.Customer
myOrderProfile.Customer = customerProfile
```

Parameters

customerProfile	Name of X_TRADER customer profile to use Data type: String

Return value

customerProfile	Name of the active X_TRADER customer profile Data type: String

Exceptions

None

Description

The `Customer` property either returns the active X_TRADER customer defaults or selects the customer defaults to use when trading through an XTAPI application.

Note: If you set both the `Customer` and `CustomerDefaults` properties, the profile uses the value in the `CustomerDefaults` property.

Example

The following assigns the customer defaults for trading an instrument.

```
'Show setting the customer in the order profile
Dim orderProfile As XTAPI.TTOrderProfile
orderProfile = New XTAPI.TTOrderProfile

'set the needed parameters for the orderprofile
orderProfile.Customer = "<Default>"
orderProfile.Instrument = Me.m_TTInstrObj
orderProfile.Set("BuySell", Me.txtBuyOrSell.Text)
orderProfile.Set("Qty", Me.txtQty.Text)
orderProfile.Set("OrderType", "L")
orderProfile.Set("Price", Me.txtPrice.Text)

Dim intOrderQty As Integer

'intOrderQty should equal the order Qty entered
intOrderQty = Me.m_TTOrderSet.SendOrder(orderProfile)

If (Not intOrderQty = Me.txtQty.Text) Then
    Console.WriteLine("ERROR sending order")
End If
```

Related information

Property:CustomerDefaults, on page 315

Property:Customers, on page 316

TTGate Property:Customers, on page 118

Property: CustomerDefaults

Purpose

Sets or returns the customer defaults profile to use for this order profile

Syntax

```
custProfile = myOrderProfile.CustomerDefaults
myOrderProfile.CustomerDefaults(custProfile)
```

Parameters

custProfile	Customer profile object to associate with this order profile Data type: `ITTCustomerProfile`

Return value

custProfile	Customer profile currently attached to this order profile Data type: `ITTCustomerProfile`

Exceptions

Exception	Meaning
Invalid TTCustomerProfile Object, input is not of the correct type	The value sent in *custProfile* is not an **ITTCustomerProfile** object.
Invalid TTCustomerProfile Object, input is null	The input parameter is missing.

Description

The `CustomerDefaults` property sets or returns the X_TRADER Customer Defaults profile for the order submitted with this order profile.

Note: If you set both the `Customer` and `CustomerDefaults` properties, the profile uses the value in the `CustomerDefaults` property.

Example

The following example calls the `CustomerDefaults` from the **TTGate** object to retrieve all customer defaults, then calls the `CustomerDefaults` from the **TTOrderProfile** object to store a profile with specific customer default values.

```
Private Sub loadcustomerDef()
  Try
    Dim OCustDef(), cdObj As Object
    'Retrieve the list of Customer Defaults from gate
    OCustDef = oGate.CustomerDefaults
    'Select the profile which has Market = CME and Acct = 12345
    For Each cdObj In OCustDef
      Dim oCustProfile As ITTCustomerProfile
      oCustProfile = cdObj
      If oCustProfile.Market = "CME" Then
        oOrderProfile.CustomerDefaults = oCustProfile
      End If
    Next
  Catch ex As Exception
    MsgBox("lodacustomerDef: " & ex.Message)
  End Try
End Sub
```

Related information

Property:Customer, on page 313

TTGate Property:CustomerDefaults, on page 117

Property: Customers

Purpose	Returns the customer defaults created in X_TRADER
Syntax	*customers* = myOrderProfile.Customers
Parameters	None
Return value	*customers* — Array of customer default names defined in X_TRADER Data type: Array
Exceptions	None
Description	The `Customers` property returns an array containing the names of customer defaults currently defined within X_TRADER. You can use this information for other operations that require customer information, such as submitting an order.

Example

The following example displays the current customer default values.

```
Private Sub GetCustomersFromXTrader()

  'create an order profile object
    Dim orderProfile As XTAPI.TTOrderProfile
    orderProfile = New XTAPI.TTOrderProfile

  'Create an array to hold the list
  'of customers as strings
  Dim arrayOfCustomers As Array
  arrayOfCustomers = orderProfile.Customers

  'will be used to hold each customer
  Dim strCustomer As String

  'iterate through the array
  For Each strCustomer In arrayOfCustomers
       Console.WriteLine(strCustomer)
  Next
End Sub
```

Related information

Property:Customer, on page 313

TTGate Property:Customers, on page 118

Property: Get

Purpose

Returns a **`TTOrderProfile`** object's attributes

Syntax

attrValues = myOrderProfile.Get(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **TTFillObj** object attribute

Description

The `Get` property returns the values for an arbitrary number of **`TTOrderProfile`** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single `ChangeMe` property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 57 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 57. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 57. Price and quantity value formats

Table 58 lists the **`TTOrderProfile`** object attributes you can supply to the `Get` property.

Attribute	Description	DataType
Acct	Customer's or trader's account number	String
AcctType AccountType	Code indicating the desired account type	String
Broker	Broker's name (multi-broker only)	String
Broker#	Broker's ID (multi-broker only)	String
BuySell BuySell$	Type of order. Possible values include: **B**: Buy **S**: Sell Specifying BuySell$ returns Buy or Sell.	String
Change Chg	Whether the order changed	Boolean
ClrMember	Clearing Member ID	String
CntrGroup	Counter Side Clearing Group ID	String
CntrMember	Counter Side Clearing Member ID	String
CntrTrader	Counter Side Clearing Trader ID	String
ColorPri ColorSec	Color to display in the MD Trader® working orders grid, where: • **ColorPri** represents the order's main color (left-side of the grid) • **ColorSec** represents the secondary color (right-side of the grid) The value is an integer value representing the RGB value for the color. The value corresponds to a Microsoft `ColorRef` data type.	Integer
Company	Broker's company name (multi-broker only)	String
Company#	Broker's company ID (multi-broker only)	String
ConnectionName	Broker's connection name (multi-broker only)	String
Contract Contract$	Instrument's name with expiration (e.g. ES Sep13)	String
ContractAlias	Instrument's alias name with expiration (e.g. E-Mini Sep13)	String
Contract.getAttribute	Get attribute for the attached instrument (TTInstrObj)	Various
Date	Date of the order in *MM*/*DD*/*YY* (e.g. 05/18/12) The value contains valid data only in the date portion of the `COleDateTime` object.	COleDateTime

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
DisclosedQty DiscQty	Quantity of the order visible in the market. Valid only for disclosed quantity orders.	Integer
Ex:OrderNo	Order number for the specified exchange (e.g. CME: 107578464)	String
ExchCred	Exchange credentials (multi-broker only)	String
ExecutionType	Current state of the order. Possible values include: **-1**: Order is in an invalid state **0**: Order is in pre-execution state (not yet sent to a TT Gateway) **1**: Order is in post-execution state and the XTAPI has received a response from the TT Gateway	Integer
ExGroup	Exchange Group ID for this order	String
ExMember	Exchange Member ID for this order	String
ExOrderID	Exchange Order ID assigned to this order. This attribute corresponds to the **OrderNo** attribute for exchanges that do not return strictly numeric order numbers. **Note:** If the exchange does not provide a value, the XTAPI populates it with the OrderNo attribute value.	String
Expiry	Expiration date for the order. Possible values include: • Null: GTC order • Today's date: GIS or GTD order • Future date: Expiration date The value contains valid data only in the date portion of the `COleDateTime` object.	COleDateTime
ExTrader	Exchange Trader ID for this order	String
ExTransNo	Unique identifier for the order transaction.	String
FFT1 FFT2 FFT3	Free-form text fields. **FFT1** typically contains the account number.	String
Fill FillQty	Order quantity already filled	Integer
GiveUp	Give-up member's clearing ID, when the fill is assigned to another member **Note:** The TT ICE Gateway does not support the **GiveUp** attribute; instead, it uses the **ClrMember** attribute for Give Up orders.	String
Group	Group name of the trader from the Member/Group/Trader ID	String
Instr Instr$	Default instrument name as displayed in the X_TRADER® **Market Grid Contract** field (e.g., ES Sep13)	String
InstrAlias	Instrument's alias name with expiration (e.g. E-Mini Sep13)	String
Instr.getAttribute	Value of the specified object attribute in the TTInstrObj object	Various
IsAON	Whether the order is an All-Or-None order	Boolean
IsAuction	Whether the order is only valid for auction	Boolean

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
IsAuto-Aggress	Whether the order is auto-aggressive (Currently supported only for BrokerTec)	Boolean
IsAutomated	Whether the order orginates from an automated program.	Boolean
IsAwayMarket	Whether an away market maker should match BBO of an away market	Boolean
IsBasis	Whether the order is a basis order	Boolean
IsBestLimit	Whether the order is a best limit order (Currently supported only for the TT TOCOM Gateway)	Boolean
IsBlock	Whether the order is a block order	Boolean
IsBuy	Whether the order is a buy order	Boolean
IsFAS	Whether the order is a fill and save order	Boolean
IsFlex	Whether the order is a flex option order	Boolean
IsFOK	Whether the order is a fill or kill order	Boolean
IsGIS	Whether the order is a good-in-session order	Boolean
IsGTC	Whether the order is a good-tll-canceled order	Boolean
IsGTD	Whether the order is a good-till-date order	Boolean
IsIceberg	Whether the order is an Iceberg order	Boolean
IsIfTouched	Whether the order requires additional action when a price is touched	Boolean
IsIOC	Whether the order is an immediate-or-cancel order	Boolean
IsLOC	Whether the order is a limit-on-close order to be filled or killed when market closes	Boolean
IsLOO	Whether the order is a limit-on-open order to be filled or killed when the market opens	Boolean
IsLSM	Whether the order is a limit-stop-market order	Boolean
IsLTM	Whether the order is a limit to market order to be converted to a market order on close	Boolean
IsMarketToLimit	Whether the order is a market-to-limit order	Boolean
IsMLM	Whether the order is a market-limit-market order	Boolean
IsMOC	Whether the order is a market-on-close order	Boolean
IsMOO	Whether the order is a market-on-open order	Boolean
IsMTL	Whether the order is a market-to-limit order	Boolean
IsMV	Whether the order is a minimum volume order	Boolean
IsNon-Leave	Whether the order is a non-leave order (Currently supported only for BrokerTec)	Boolean

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
IsOrderColorSupported	Whether the version of X_TRADER® installed on a trader's system supports receiving order color information from an XTAPI application. If color is supported, X_TRADER® uses the values in the **ColorPri** and **ColorSec** object attributes for the **`TTOrderProfile`** and **`TTOrderObj`** objects in the MD Trader® widow. You can define the colors by setting the attributes with the Set method.	Boolean
IsPartial	Whether the order is partially filled	Boolean
IsPassive	Whether the order is passive (Currently supported only for BrokerTec)	Boolean
IsSell	Whether the order is a sell order	Boolean
IsShortExempt	Whether the order is a sell short order exempt from notification	Boolean
IsStop	Whether the order is a stop order	Boolean
IsTimed	Whether the order is a timed order	Boolean
IsTriggered	Whether the order has been triggered	Boolean
IsTriggeredTime	Whether the order was triggered after a time submission	Boolean
IsVolatility	Whether the order is a volatility order	Boolean
Key	Internal static key that represents the order throughout its lifetime	String
LinkType	Type of link identifying the relationship between orders. Values include: • Algo • AlgoC • AlgoL • AS • ASQ • ASH • ASR • OCO • OMA • SSE • SSEC • SSER • Stage-BKR • Stage-INT • Stage-UNK • UNM **Note**: In some instances an order can have more than one link type, separated by a comma (e.g. SSE, OCO).	String
Limit Limit# Limit$ Limit&	Limit used during price reasonability checks. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
Location	Order sender country and region code.	String
Member	Member ID of the trader from the Member/Group/Trader ID	String
MinQty	Minimum quantity needed to be immediately available at the limit price for this order to be submitted into the market	Integer
Msg	Exchange or gateway message describing the reason for an order reject or order delete	String
OldOrderNo	Previous number used for the order, if the exchange created a new order number	String
OnHold	Whether an order is on hold (out of the market)	Boolean
OpenClose OpenClose$	Whether the order opens or closes a position. Possible values include: **O**: Open **C**: Close **F**: FIFO Specifying **OpenClose$** returns the corresponding full string. **Note:** The TOCOM Exchange changes the **OpenClose** value of a fill corresponding to a FIFO order as either **C** (Close) or **O** (Open).	String
OrdAction	Action associated with the order. Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_CHANGE` • `ORD_ACTION_DELETE` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_HOLD` • `ORD_ACTION_REPLACE` • `ORD_ACTION_UNKNOWN`	String
OrderNo	Order number assigned by the exchange. **Note:** For CME products, this property contains a different representation of the order number that matches the format used by other exchanges. To see the actual order number received from the CME, check the **ExOrderID** attribute.	String
OrderQty	Number of contracts contained in the order	Integer
OrderRestr	Restriction on the order. For a list of order restrictions, see the section called Order restrictions, on page 639.	String

Table 58. TTDropHandler Get attributes

<table>
<tr><th>Attribute</th><th>Description</th><th>DataType</th></tr>
<tr><td>OrderSource
OrderSource$</td><td>Where the order originated:
0: X_TRADER
1: Autospreader Desktop
2: Autotrader
3: XTAPI
4: Ext Source (Exchange)
5: FA Licensed (Licensed FIX Adapter)
6: FA Unlicensed (Unlicensed FIX Adapter)
7: XT Lite (X_TRADER Lite)
8: TT_TRADER
9: Autospreader Engine
10: Synthetic SE Engine
11: TT API in X_TRADER Mode
12: Algo SE
21: TT API in Universal Login Mode
22: ADL
Specifying OrderSource$ returns the corresponding string.</td><td>As specified</td></tr>
<tr><td>OrdStatus
OrdStatus&</td><td>Indication of the response to the most recent action performed on the order. Possible values include:
<table><tr><th>Integer</th><th>String</th><th>Description</th></tr><tr><td>0</td><td>Void</td><td>Default value</td></tr><tr><td>8200</td><td>New</td><td>New order released by client</td></tr><tr><td>8201</td><td>Accepted</td><td>Order accepted by the server</td></tr><tr><td>8202</td><td>OK</td><td>Order active in the market</td></tr><tr><td>8203</td><td>XT Filled</td><td>Order filled</td></tr><tr><td>8204</td><td>Hold</td><td>Order is on hold</td></tr><tr><td>8205</td><td>Deleted</td><td>Order has been deleted</td></tr><tr><td>8206</td><td>Expired</td><td>Order has expired</td></tr><tr><td>8207</td><td>Rejected</td><td>Order was rejected</td></tr><tr><td>8208</td><td>Resubmitted</td><td>Hold order was resubmitted</td></tr><tr><td>8299</td><td>Unknown</td><td>Order status could not be retrieved</td></tr></table></td><td>As specified</td></tr>
<tr><td>OrderTag</td><td>User-definable text (15 char max) containing user-defined information included in the corresponding order profile.</td><td>String</td></tr>
<tr><td>OrderTraits</td><td>Traits associated with the order as a concatenated string containing the order status history, order flag, and order restriction.</td><td>String</td></tr>
</table>

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
OrderType OrderType$	Type of order. Possible values include: **B** (OCO): OCO order **C** (CMO): CMO order **L** (Limit): Limit order **M** (Mkt): Market order **O** (BatchOrder): Batch order **Q** (Quote): Quote order **S** (BL): Best Limit order (TOCOM only) **T** (MTL): Market To Limit order **X** (Cross): Cross order The text in parenthesis indicates the value associated with the $ suffix.	String
OrigOrdAction	Action specified in the original order submission, which might differ from the current order action. For example, a Cancel/Replace order comprises two separate order actions: Delete and Add. This attribute stores the original action (Replace, in this example). Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_CHANGE` • `ORD_ACTION_DELETE` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_HOLD` • `ORD_ACTION_REPLACE` • `ORD_ACTION_UNKNOWN`	String
ParentKey	Site order key identifying the parent order.	String
Price Price# Price$ Price&	Order price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
PriceCheck	Whether price reasonability checking is enabled	String
Qty	Number of contracts contained in the order	Integer
RecoverableFailure	Whether an order was rejected temporarily, but might succeed in the future	Boolean
Sender Sender$	IP address of the machine that submitted the order, as an integer or string	As specified
SenderSubID	Unique trader (or operator) ID used to distinguish between different traders when the current MGT routes orders for multiple traders. **Note:** Some exchanges, such as CME, require clients to provide this information when routing orders on behalf of multiple traders.	String (32 character limit)
Series.getAttribute	Value of the specified attribute in the `TTInstrObj` object	Various
SeriesKey	Internal ID that uniquely identifies the instrument contained in the order	String
SiteKey SiteOrderKey	TT-generated static key that represents the order throughout its lifetime	String

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
Source	Where the order originated: **0**: X_TRADER **1**: Autospreader Desktop **2**: Autotrader **3**: XTAPI **4**: Ext Source (Exchange) **5**: FA Licensed (Licensed FIX Adapter) **6**: FA Unlicensed (Unlicensed FIX Adapter) **7**: XT Lite (X_TRADER Lite) **8**: TT_TRADER **9**: Autospreader Engine **10**: Synthetic SE Engine **11**: TT API in X_TRADER Mode **12**: Algo SE **21**: TT API in Universal Login Mode **22**: ADL	Integer
Stop Stop# Stop$ Stop&	Stop price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
Tag(x)	User-defined tag, where x is any integer	String
Time Time$	Time the CoreAPI sent the order to the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeExchange TimeExchange$	Time the exchange received the order. The time represents the local time at the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeLocalUpd TimeLocalUpd$	Time the XTAPI received the order, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeProcessed TimeProcessed$	Time the TT Gateway processed the order and sent it to the exchange, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified

Table 58. TTDropHandler Get attributes

Attribute	Description	DataType
TimeReceived TimeReceived$	Time the TT Gateway received the order from the XTAPI, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object.	As specified
TimeRouted TimeRouted$	Time the XTAPI sent the order to X_TRADER®, in the forms: • Variant containing a `COleDateTime` structure • *HH*:*MM*:*SS*:*mmm* The value contains valid data only in the time portion of the `COleDateTime` object. NOTE: These values exist only for the lifetime of your application and are not visible to other applications.	As specified
Trader	Trader ID of the trader from the Member/Group/Trader ID	String
TriggerQty	Minimum traded quantity required for a stop to trigger at the specified stop price	Integer
UserData(x)	User-defined tag, where x is any integer Valid values: 0-1023	Integer
UserTag	User-definable text (15 char max) containing user-defined information included in the corresponding order profile.	String
Username	Trader login. The value differs based on the type of credentials used: • For MGT-based logins, the value contains the concatenation of the Member, Group, and Trader attributes. • For universal logins, the value contains the trader's login ID	String
Wrk WrkQty	Order quantity that is working and has not been filled	Integer

Table 58. TTDropHandler Get attributes

Example

The following example uses the `Get` property to display the values for the specified **`TTOrderProfile`** attributes.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidNameArr As Array
  Dim attrValuesArr As Array
  Dim isValidList As Boolean
  Dim attrName As String
  Dim i As Integer
  i = 1

  'good practice to validate list before calling Get
  isValidNameArr = Me.m_TTOrderProfile.IsValidGet(objAttributes)

  ' code to verify attribute list omitted

  If isValidList = True Then
    attrValuesArr = Me.m_TTOrderProfile.Get(objAttributes)

    'Display each attribute and value
    For Each attrValue in attrValuesArr
```

```
        attrName = objAttributes.Split(",").GetValue(i)
        Console.Writeline("Attribute: {0} = {1}", attrName, attrValue)
        i = i + 1
        Next
      End For

    End If
End Sub
```

Related information

Property:IsValidGet, on page 332

Property:ReadProperties, on page 334

Method:Set, on page 337

Property: GetLast

Purpose

Returns the object attributes of the order returned by the GetLastOrder method

Syntax

```
attrValues = myOrderProfile.GetLast(attrList)
```

Parameters

attrList	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of attribute values Data type: Array of various data types

Exceptions

Exception	Meaning
Invalid Parameters	The *attrList* parameter contains one or more invalid object attributes.

Description

The `GetLast` property returns the values for an arbitrary number of object attributes for the last order object (**`TTOrderObj`**). The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

For more information about the attributes returned, refer to the **`TTOrderObj`** Property:Get section, on page 328.

Example

The following example uses the `GetLast` property to display the values of **IsBuy** and **IsSell** attributes of the last order. It also saves the site order key for the order.

```
Private Sub SendOrder()

  Try
    'Create an order set
    m_TTOrderSet = New XTAPI.TTOrderSet

    'Turn off self-imposed limits
    m_TTOrderSet.Set("NetLimits", False)

    'Enable the orderSet to send an order
    m_TTOrderSet.EnableOrderSend = True
    m_TTOrderSet.EnableOrderSetUpdates = True
    m_TTOrderSet.EnableOrderUpdateData = True

    'Open the orderset for order entry
    m_TTOrderSet.Open(1)

    'Create an orderprofile and name its alias
    Dim orderProfile As XTAPI.TTOrderProfile
    orderProfile = New XTAPI.TTOrderProfile

    'set the needed parameters for the orderprofile
    orderProfile.Customer = "TTORDGS001001"
    orderProfile.Instrument = Me.m_TTInstrObj
    orderProfile.Set("BuySell", Me.txtBuyOrSell.Text)
    orderProfile.Set("Qty", Me.txtQty.Text)
```

```
        orderProfile.Set("OrderType", "L")
        orderProfile.Set("Price", Me.txtPrice.Text)
        Dim intOrderQty As Integer

        'intOrderQty should equal the order Qty entered
        intOrderQty = Me.m_TTOrderSet.SendOrder(orderProfile)

        'This call must be made just after sending an order
        Dim arrayOfOrderGetProperties As Array
        arrayOfOrderGetProperties = orderProfile.GetLast("IsBuy,IsSell")

        Dim strGetProperty As String
        For Each strGetProperty In arrayOfOrderGetProperties
            Console.WriteLine(strGetProperty)
        Next

        If (Not intOrderQty = Me.txtQty.Text) Then
            Console.WriteLine("ERROR sending order")
        End If

        Me.m_strSiteOrderKey = orderProfile.GetLast("SiteOrderKey")

        Catch ex As Exception
          Console.WriteLine(ex.Message)
        End Try

    End Sub
```

Related information Property:GetLastOrder, on page 330

Property: GetLastOrder

Purpose

Returns the first order in an order chain

Syntax

```
orderObj = myOrderProfile.GetLastOrder
```

Parameters

None

Return value

orderObj	First order object in the order chain Data type: Array of various data types

Exceptions

Exception	Meaning
Could not create order	No order could be found in the order profile, probably because the order was filled or cancelled.

Description

The `GetLastOrder` property returns the first order in an order chain for an order that was submitted to an exchange. You can then use the `HasNextOrder` and `NextOrder` properties to traverse the order chain to detect changes to an order.

Example

The following example returns the last state of the order.

```
Dim myOrder As XTAPI.TTOrderObj
' After an order send, get the TTOrderObj from the order profile
myOrder = myOrderProfile.GetLastOrder

' Check to see if the order has a next state
If myOrder.HasNextOrder Then

   ' Process next order
   ' ...
End If
```

Related information

Property:GetLast, on page 328

Property: Instrument

Purpose

Returns or defines the instrument object associated with the order profile

Syntax

```
instrObj = myOrderProfile.Instrument
myOrderProfile.Instrument = instrObj
```

Parameters

instrObj	Pointer to the instrument object you want to attach to the order profile Data type: `TTInstrObj`

Return value

instrObj	Instrument currently attached to the order profile Data type: `TTInstrObj`

Exceptions

Exception	Meaning
Invalid instrument	The *instrObj* parameter does not reference a **TTInstrObj** object

Description

The `Instrument` property sets or returns the instrument object associated with the order profile. You can use the object to get information, such a the current price, about the instrument.

Example

The following example sets the Instrument property from an order profile.

```
Private Sub SendOrder(Act As String)
  Dim OrdProf As New TTOrderProfile
  Dim iSent As Integer
  If Val(txtQty) > 0 And Len(txtPrice.Text) > 0 Then
   With OrdProf
     .Instrument = LiveInstr   ' Set instrument for Order
     .Set "OrderType,BuySell$,Qty,Price", Array _
      ("L", Act, txtQty.Text, txtPrice.Text)     ' Set Order info
   End With
   iSent = SubOrderSet.SendOrder(OrdProf)     ' Send the order
   Caption = "Sent " & Act & iSent & "@" & txtPrice.Text
  End If
End Sub
```

Related information

None

Property: IsValidGet

Purpose

Returns whether an attribute name is a valid argument for the `Get` property

Syntax

attrValues = myOrderProfile.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information

Property:Get, on page 317

Property:ReadProperties, on page 334

Property: ReadProperties

Purpose

Returns information about **`TTOrderProfile`** attributes you can access through the `Get` property

Syntax

attributes = myOrderProfile.ReadProperties

attrDescription = myOrderProfile.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **`TTOrderProfile`** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 317

Property:WriteProperties, on page 336

Property: RoundOption

Purpose

Specifies the rounding style to use for calendar spreads

Syntax

```
rounding = myOrderProfile.RoundOption
myOrderProfile.RoundOption = rounding
```

Parameters

rounding	Method to use for rounding calendar spreads. Possible values include: **0**: No rounding **1**: Aggressive rounding **2**: Conservative rounding Data type: Integer; Default: 2

Return value

rounding	Current rounding method Data type: Integer

Exceptions

Exception	Meaning
Invalid round option value	The rounding parameter contains an unsupported value.

Description

The `RoundOption` property defines the rounding style to use when trading calendar spreads provided by the eCBOT and EurexUS exchanges. If you submit a price that does not match the spread's ticking scheme, you can set this property to adjust the price automatically. Without a rounding method, the gateways reject orders with invalid prices.

If you want to increase the likelihood of filling orders, you can use aggressive rounding to round the price to the next tick price toward the inside market. Conversely, you can use conservative rounding scheme to round the price to the next tick away from the inside market.

Example

None

Related information

None

Property: WriteProperties

Purpose

Returns information about **TTOrderProfile** attributes you can access through the `Set` property

Syntax

```
attributes = myOrderProfile.WriteProperties
attrDescription = myOrderProfile.WriteProperties(attrName)
```

Parameters

attrName	Optional name of an object's attribute whose description you want to set Data type: String

Return value

attributes	Array of attributes you can pass to the `Set` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an *attrName* is not one of the object's attributes

Description

With no parameters, the `WriteProperties` property returns a list of all object attributes that you can pass as arguments to the Set property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `WriteProperties` property to display the descriptions for each **TTOrderProfile** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:ReadProperties, on page 334

Method:Set, on page 337

Method: Set

Purpose

Sets values for **`TTOrderProfile`** object's attributes

Syntax

`myOrderProfile.Set(attrList, attrValues)`

Parameters

attrList — Comma-separated string of object attributes

Data type: String

attrValues — Array of attribute values

Data type: Array

Note: If you specify a single attribute in attrList, you can supply the value directly, without including it in an array structure.

Return value

None

Exceptions

Exception	Meaning
Order Profile Set Failed	One or more input attribute or value is not valid.

Description

The `Set` property sets or updates the values for an arbitrary number of **`TTOrderProfile`** attributes. You must supply the values with the appropriate data types.

Note: If you need to update multiple attributes, you should specify all of them in a single IsValidGet property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 59 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 59. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 59. Price and quantity value formats

Table 60 lists the **`TTOrderProfile`** object attributes you can supply to the `Set` property.

Attribute	Description	DataType
Acct	Customer's or trader's account number. Maximum length: 15 characters.	String
AcctType AccountType	Code indicating the desired account type	String
Broker	Broker's name (multi-broker only)	String
Broker#	Broker's ID (multi-broker only)	String
BuySell BuySell$	Type of order. Possible values include: B: Buy S: Sell	String
ClrMember	Clearing Member ID	String
ColorPri ColorSec	Color to display in the MD TRADER® working orders grid, where: • ColorPri sets the order's main color (left side of the grid) • ColorSec sets the order's secondary color (right side of the grid) The value is an integer representing the RGB value of the color. The value corresponds to the Microsoft ColorRef data type.	Integer
ConnectionName	Broker's connection name (multi-broker only)	String
Destination	Exchange-specific order destination Internal use only	String
DisclosedQty DiscQty	Quantity of the order visible in the market. Valid only for disclosed quantity orders.	Integer
Expiry	Expiration date for the order. If you omit this attribute, XTAPI assigns the value based on the TIF attribute (**IsGIS**, **IsGTC**, **IsGTD**). For more information, see the section called, Handling time-in-force (TIF) attributes, on page 341.	`COleDateTime`
FFT1 FFT2 FFT3	Free-form text fields. FFT1 typically contains the account. Maximum length: 15 characters	String
GiveUp	Give-up member's clearing ID, when the fill is assigned to another member **Note:** The TT ICE Gateway does not support the GiveUp attribute; instead, it uses the ClrMember attribute for Give Up orders.	String
Group	Group name of the trader from the Member/Group/Trader ID	String
IsAON	Whether to submit the order as an All-Or-None order	Boolean
IsAuction	Whether to submit the order as only valid for auction	Boolean

Table 60. `TTOrderProfile`.Set attributes

Attribute	Description	DataType
IsAuto-Aggress	Whether to submit the order as auto-aggressive (Currently supported only for BrokerTec)	Boolean
IsAutomated	Whether the order orginates from an automated program. Default: TRUE	Boolean
IsAwayMarket	Whether an away market maker should match BBO of an away market	Boolean
IsBasis	Whether to submit the order as a basis order	Boolean
IsBestLimit	Whether to submit the order as a best limit order (Currently supported only for the TT TOCOM Gateway)	Boolean
IsBlock	Whether to submit the order as a block order	Boolean
IsBuy	Whether to submit the order as a buy order	Boolean
IsFAS	Whether to submit the order as a fill and save order	Boolean
IsFlex	Whether to submit the order as a flex option order	Boolean
IsFOK	Whether to submit the order as a fill or kill order	Boolean
IsGIS IsGTC IsGTD	Whether to submit the order as a good-in-session, good-til-close, or good-till-day order For more information, see the section called, **Handling time-in-force (TIF) attributes**, on page 341.	Boolean
IsIceberg	Whether to submit the order as an Iceberg order	Boolean
IsIfTouched	Whether the order requires additional action when a price is touched	Boolean
IsIOC	Whether to submit the order as an immediate-or-none order	Boolean
IsLOC	Whether the order is a limit-on-close order to be filled or killed when market closes	Boolean
IsLOO	Whether the order is a limit-on-open order to be filled or killed when the market opens	Boolean
IsLSM	Whether the order is a limit-stop-market order	Boolean
IsLTM	Whether the order is a limit to market order to be converted to a market order on close	Boolean
IsMarketToLimit	Whether to submit the order as a market-to-limit order	Boolean
IsMLM	Whether the order is a market-limit-market order	Boolean
IsMOC	Whether to submit the order as a market-on-close order	Boolean
IsMOO	Whether to submit the order as a market-on-open order	Boolean
IsMTL	Whether the order is a market-to-limit order	Boolean
IsMV	Whether to submit the order as a minimum volume order	Boolean
IsNon-Leave	Whether to submit the order as a non-leave order (Currently supported only for BrokerTec)	Boolean
IsPassive	Whether to submit the order as a passive order (Currently supported only for BrokerTec)	Boolean
IsSell	Whether to submit the order as a sell order	Boolean
IsShortExempt	Whether to submit the order as a sell short order exempt from notification	Boolean
IsStop	Whether to submit the order as a stop order	Boolean
IsTimed	Whether to submit the order as a timed order	Boolean

Table 60. `TTOrderProfile`.Set attributes

Attribute	Description	DataType
IsTriggered	Whether to submit the order as a triggered order	Boolean
IsTriggeredTime	Whether to submit the order as triggered after a time order	Boolean
IsVolatility	Whether to submit the order as a volatility order	Boolean
Limit Limit# Limit$ Limit&	Limit used during price reasonability checks You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
Member	Member ID of the trader from the Member/Group/Trader ID	String
MinQty	Minimum quantity needed to be immediately available at the limit price for this order to be submitted into the market	Integer
OpenClose OpenClose$	Whether the order opens or closes a position. Possible values include: **O**: Open **C**: Close **F**: FIFO Specifying **OpenClose$** returns the corresponding full string. **Note:** The TOCOM Exchange changes the **OpenClose** value of a fill corresponding to a FIFO order as either **C** (Close) or **O** (Open).	String
OnHold	Whether to submit the order directly in a HOLD state. All shared members of an order book who meet MGT/Filter criteria can see orders submitted in HOLD state.	Boolean
OrderQty	Number of contracts contained in the order	Integer
OrderRestr	Order restriction on the order. For a list of possible values, see **Order restrictions**, on page 636.	String
OrderTag	User-definable text (15 char max) to provide additional information about an order. The attribute persists with the order and is sent to the TT Gateway. An order supports only one instance of this attribute; however, complex orders, such as spreads and strategies, can use different values for each underlying leg.	String
OrderType OrderType$	Type of order. Possible values include: **B** (OCO): OCO order **C** (CMO): CMO order **L** (Limit): Limit order **M** (Mkt): Market order **O** (BatchOrder): Batch order **Q** (Quote): Quote order **S** (BL): Best Limit order (TOCOM only) **T** (MTL): Market To Limit order **X** (Cross): Cross order The text in parenthesis indicates the value associated with the $ suffix.	String

Table 60. `TTOrderProfile`.Set attributes

Attribute	Description	DataType
Price Price# Price$ Price&	Order price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
PriceCheck	Whether price checking is enabled. Possible values include Y or N.	String
Qty	Number of instruments contained in the order	Integer
SenderSubID	Unique trader (or operator) ID used to distinguish between different traders when the current MGT routes orders for multiple traders. **Note:** Some exchanges, such as CME, require clients to provide this information when routing orders on behalf of multiple traders. When trading on such an exchange, your XTAPI application is responsible for providing this value.	String (32 character limit)
Stop Stop# Stop$ Stop&	Stop price. You can also use one of the following modifiers to display the value in a specific format: **$** : X_TRADER display format **#** : Decimal value **&** : Ticks	As specified
Tag(*x*)	User-defined tag, where x is integer in the range, 0-99 or 10000-32767	String
Trader	Trader ID of the trader from the Member/Group/Trader ID	String
TriggerQty	Minimum traded quantity required for a stop to trigger at the specified stop price	Integer
UserData(x)	User-defined tag, where *x* is an integer in the range, 0-1023	String
UserTag	User-definable text (15 char max) to provide additional information about an order. The attribute persists with the order and is sent to the TT Gateway. An order supports only one instance of this attribute; however, complex orders, such as spreads and strategies, can use different values for each underlying leg.	String

Table 60. `TTOrderProfile`.Set attributes

Handling time-in-force (TIF) attributes

When specifying how long an order should be left in the market, you use the following attributes to define the time-in-force modifier for the order:

- **Expiry**, which specifies a specific date and time an order expires
- **IsGIS**, which indicates the order is valid until the end of the current trading session, as defined by the TT Gateway
- **IsGTC**, which indicates that the order is valid until specifically cancelled
- **IsGTD**, which indicates the order is valid the end of the trading day, as defined by the TT Gateway

These values represent mutually-exclusive TIF modifiers. For example, you cannot submit an order that is valid both until the end of the day (**IsGTD**) and until cancelled (**IsGTC**). If you specify precisely one of these attributes, XTAPI sends the attribute specified. Otherwise:

- If you specify none of these attributes, XTAPI enables the **IsGTD** attribute

and submits the order as a good-til-day order.

- If you specify more than one of these attributes, XTAPI uses the attribute positioned last in the parameter list and clears the other attributes.

Example

The following example shows how to use the `Set` method to set a single attribute:

```
Dim myOrderProfile As TTOrderProfile

# code to create myOrderProfile omitted
#
myOrderProfile.Set("Limit#", 32.75)

# The following example shows the preferred approach to use when setting
# multiple attribute values with different data types:
Dim myOrderProfile As TTOrderProfile

# code to create myOrderProfile omitted
#
Dim sProps As String = "Limit#,Stop$"
Dim aValues As Array = (32.75, "32.50")
TTOrderProfile.Set(sProps, aValues)
```

Related information

Property:Get, on page 317

Property:WriteProperties, on page 336

19 TTOrderSelector Class

Chapter overview

The chapter describes the **`TTOrderSelector`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTOrderSelector class overview	345
Property:AllMatchesRequired	347
Property:AllowAnyMatches	348
Property:TestMatch	349
Method:AddSelector	350
Method:AddTest	351
Method:AddTestEx	353
Method:InvertTest	354
Method:Reset	356

TTOrderSelector class overview

Overview

The **`TTOrderSelector`** class contains properties and methods to create order sets that contain a subset of the orders in your order book. An order selector contains a set of matching rules, or tests, that allow you to filter orders and fills in your order book based on the values of order or fill object attributes. Each order set is defined by an attached order selector.

Figure 61 shows an how three different order selectors can create unique order sets.

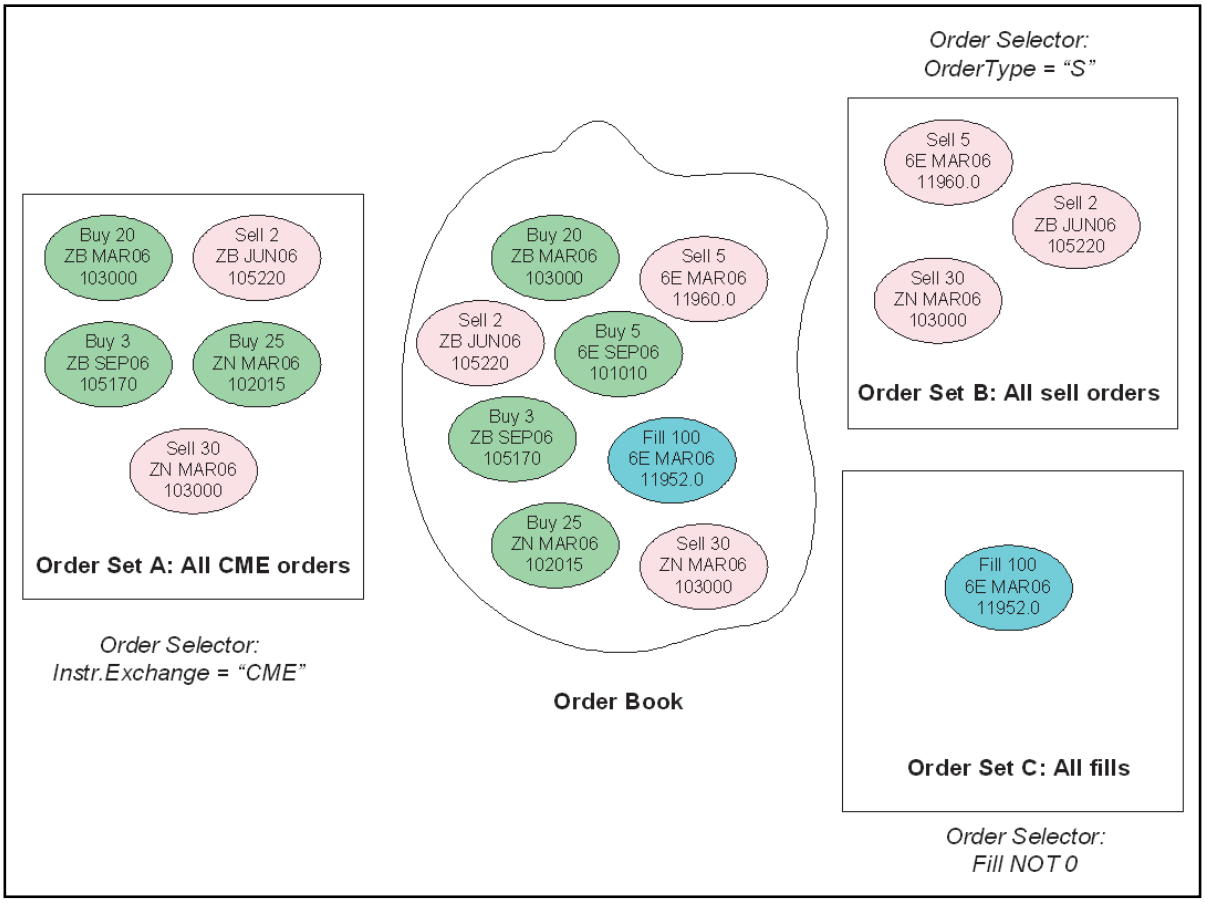


Figure 61. Sample order selectors

You can create complex selection rules by using multiple tests and nested selectors, combined with the logic operations: AND, OR, and NOT. For example, you could create a selector to include all Buy orders and all orders for Account "12." You can also create hierarchical selector structures to combine multiple selectors.

Figure 62 shows a hierarchical selector structure that loads an order set with:

- Buy orders with limit prices of 9500, 9510, or 9520
- Orders for account "123" with limit prices of 9500, 9510, or 9520

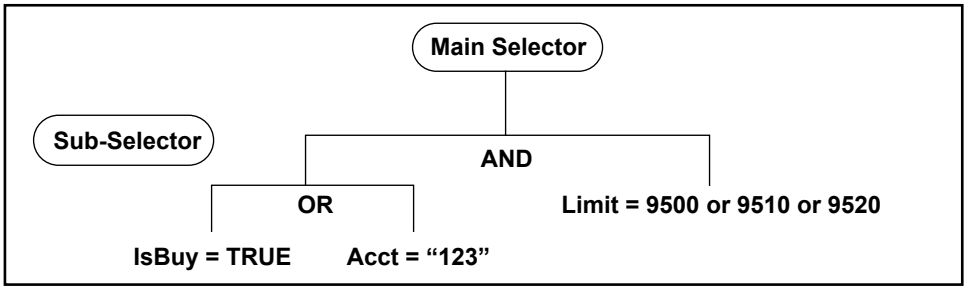


Figure 62. Sample complex order selector

Logical AND and OR operations between tests are implemented through two **`TTOrderSelector`** properties: `AllMatchesRequired` and `AllowAnyMatches`, respectively. If you do not enable either property, the XTAPI automatically enables `AllMatchesRequired` for the tests in the selector.

You can also use the `InvertTest` method to invert the logic for a selector or sub-selector. This method performs a logical NOT operation for the entire selector.

Properties

Table 63 lists the **`TTOrderSelector`** properties.

Property	Description
AllMatchesRequired	Determines whether the order selector applies AND logic to its tests
AllowAnyMatches	Determines whether the order selector applies OR logic to its tests
TestMatch	Determines whether an order or fill matches the selector test criteria

Table 63. TTOrderSelector properties

Methods

Table 64 lists the **`TTOrderSelector`** methods.

Method	Description
AddSelector	Adds a sub-selector to the current order selector or sub-selector
AddTest	Adds a test condition to an order selector
AddTestEx	Adds a test condition to an order selector
InvertTest	Inverts the logic defined for a selector
Reset	Removes all tests and selectors from the order selector

Table 64. TTOrderSelector methods

Events

None

Property: AllMatchesRequired

Purpose

Determines whether the order selector applies AND logic to its tests

Syntax

```
isAll = mySelector.AllMatchesRequired
mySelector.AllMatchesRequired = isAll
```

Parameters

isAll	Whether to apply AND logic Data type: Boolean

Return value

isAll	Whether AND logic is applied to the selectors Data type: Boolean

Exceptions

None

Description

The `AllMatchesRequired` property performs a logical AND between all sub-selectors and tests in the selector. The resulting order set contains only those orders who match all of the specified criteria. For example, you can use `AllMatchesRequired` to include only Buy orders for account 123 in an order set.

Each order selector or sub-selector can contain precisely one `AllMatchesRequired` or `AllowAnyMatches` property. If you need to test multiple sets of conditions, you can create sub-selectors for each set of conditions and then apply the logical operation to the set of conditions.

Example

The following code fragment uses the `AllMatchesRequired` property to create a sub-selector that selects only orders buy orders from account 123, then uses the property again in an order selector to refine the set further by including only orders with the specified limit prices.

```
Dim OrdSelect As New TTOrderSelector
Dim SubSelect As New TTOrderSelector

SubSelect.AllMatchesRequired = True  ' Make this an AND filter
SubSelect.AddTest "IsBuy", True   ' Check for Buys
SubSelect.AddTest "Acct", "123"   ' Check for Account # "123"
OrdSelect.AllMatchesRequired = True ' Make the main selector an AND filter
OrdSelect.AddSelector(SubSelect)   ' Add sub selector to main order selector
OrdSelect.AddTest "Limit", "9975,10000,10050"  ' Add test for limit prices
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Property:AllowAnyMatches, on page 348

Method:AddSelector, on page 350

Method:AddTest, on page 351

Method:AddTestEx, on page 353

Method:Reset, on page 356

Property: AllowAnyMatches

Purpose

Determines whether the order selector applies OR logic to its tests

Syntax

```
isAny = mySelector.AllowAnyMatches
mySelector.AllowAnyMatches = isAny
```

Parameters

isAny	Whether to apply OR logic Data type: Boolean

Return value

isAny	Whether OR logic is applied to the selectors Data type: Boolean

Exceptions

None

Description

The `AllowAnyMatches` property performs a logical OR between all sub-selectors and tests in the selector. The resulting order set contains those orders who match any of the specified criteria. For example, you can use `AllowAnyMatches` to include both ZB Mar13 and ZB Jun13 orders.

Each order selector or sub-selector can contain precisely one `AllowAnyMatches` or `AllMatchesRequired` property. If you need to test multiple sets of conditions, you can create sub-selectors for each set of conditions and then apply the logical operation to the set of conditions. For example, if you wanted to include only fills for ZB Mar13 and ZB Jun13 buys, you would create an `AllowAnyMatches` sub-selector for the two contracts and then create an `AllMatchesRequired` selector to combine the sub-selector and a test for the **IsBuy** attribute.

Example

The following code fragment uses the `AllowAnyMatches` property to define a sub-selector that matches orders for ZB Mar13 or ZB Jun13 contracts.

```
Dim OrdSelect As New TTOrderSelector
Dim SubSelect As New TTOrderSelector

SubSelect.AllowAnyMatches = True  ' Make this an OR filter
SubSelect.AddTest "Contract", "ZB Mar13"   ' Check for ZB Mar13 contracts OR
SubSelect.AddTest "Contract", "ZB Jun13"   ' ZB Jun13 contracts
OrdSelect.AllMatchesRequired = True ' Make the main selector an AND filter
OrdSelect.AddSelector(SubSelect)   ' Add sub selector to main order selector
OrdSelect.AddTest "IsBuy", TRUE  ' Add test for buy orders
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Property:AllMatchesRequired, on page 347

Method:AddSelector, on page 350

Method:AddTest, on page 351

Method:Reset, on page 356

Property: TestMatch

Purpose

Determines whether an order or fill matches the selector test criteria

Syntax

```
isMatched = mySelector.TestMatch(object)
```

Parameters

object	Order or fill object to test Data type: `TTOrderObj` or `TTFillObj`

Return value

isMatched	Whether the specified order or fill would be included in the order set Data type: Boolean

Exceptions

Exception	Meaning
Invalid Fill Object	The specified object does not reference a valid **`TTFillObj`** object
Invalid Object	The specified object is neither a **`TTFillObj`** or **`TTOrderObj`** object
Invalid Order Object	The specified object does not reference a valid **`TTOrderObj`** object

Description

The `TestMatch` property determines whether the current order selector or sub-selector would include an order (**`TTOrderObj`**) or fill (**`TTFillObj`**). You can use this property to test the selector logic to ensure it would create an order set with the desired order and fill objects.

Example

The following routine uses the `TestMatch` property to determine which orders or fills would be included in an order set created by an order selector.

```
Private Sub IsOrderIncluded(ByVal ordSelect as TTOrderSelector,_
                            ByVal order as TTOrderObj)
   Dim isIncluded as Boolean
   isIncluded = ordSelect.TestMatch(order)

   if isIncluded = TRUE then
      Debug.Print order.Get("OrderNo"), " is included."
   else
      Debug.Print order.Get("OrderNo"), " is not included."
End Sub
```

Related information

Property:AllMatchesRequired, on page 347

Property:AllowAnyMatches, on page 348

Method:AddSelector, on page 350

Method: AddSelector

Purpose

Adds a sub-selector to the current order selector or sub-selector

Syntax

`mySelector.AddSelector(`*`subSelector`*`)`

Parameters

subSelector	Order selector to include Data type: `TTOrderSelector`

Return value

None

Exceptions

Exception	Meaning
Invalid Selector	The subSelector pointer does not reference a valid **`TTOrderSelector`** object.

Description

The `AddSelector` property adds an order selector to the current order selector object as a subordinate order selector. By nesting multiple selectors, you can create complex logic an order set can use to select the orders and fills to include in the set.

Example

The following code fragment defines a sub-selector object and adds it to the order selector.

```
Dim OrdSelect As New TTOrderSelector
Dim SubSelect As New TTOrderSelector

SubSelect.AllowAnyMatches = True  ' Make this an OR filter
SubSelect.AddTest "IsBuy", True   ' Check for Buys, or
SubSelect.AddTest "Acct", "123"   ' Account # "123"
OrdSelect.AllMatchesRequired = True ' Make the main selector an AND filter
OrdSelect.AddSelector(SubSelect)   ' Add sub selector to main order selector
OrdSelect.AddTest "Limit", "9975,10000,10050"  ' Add test for limit prices
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Property:AllMatchesRequired, on page 347

Property:AllowAnyMatches, on page 348

Method:AddTest, on page 351

Method:AddTestEx, on page 353

Method:InvertTest, on page 354

Method:Reset, on page 356

Method: AddTest

Purpose

Adds a test condition to an order selector

Syntax

```
mySelector.AddTest(attrName, attrValue)
```

Parameters

attrName	Name of the object attribute to use in the condition Data type: String
attrValue	Value of the attribute to test Data type: Varies based on the attribute value

Return value

None

Exceptions

Exception	Meaning
Invalid Filter	The specified *attrName* is not a valid object attribute for the corresponding order or fill.
Invalid Property Value	The data type of the specified *attrValue* does not match the data type of the attribute.

Description

The `AddTest` property adds a single condition to an order selector to help determine which orders to include in the corresponding order set. The condition tests the equality of the attribute with the specified value or values. For example, the following code includes all Buy orders:

```
ordSel.AddTest("BuySell", 0)
```

You can call the `AddTest` property multiple times in a selector to create more specific selection criteria.

If you specify a single value, the property matches orders whose attribute has that value. If you specify multiple values for the same attribute, the property matches orders whose attribute matches any of the values (logical OR). For example, the following call matches orders for trader accounts "1234" and "banker2":

```
ordSel.AddTest("Acct", "1234", "banker2")
```

which is also equivalent to the following:

```
ordSel.AddTest("Acct", "1234")
ordSel.AddTest("Acct", "banker2")
ordSel.AllowAnyMatches = TRUE
```

Note: When developing an application with a non-Microsoft language, you must use the `AddTestEx` method instead of the this method. The underlying string data type of the `AddTest` method is incompatible with the `IDispatch` method in the Java/COM Bridge. If you chose to use Java, please be aware that TT does not provide any support for integrating a Java app with XTAPI.

Example

The following example adds two tests to a sub-selector and one test to the parent order selector.

```
Dim OrdSelect As New TTOrderSelector
Dim SubSelect As New TTOrderSelector
```

```
SubSelect.AllowAnyMatches = True  ' Make this an OR filter
SubSelect.AddTest "IsBuy", True   ' Check for Buys, or
SubSelect.AddTest "Acct", "123"   ' Account # "123"
OrdSelect.AllMatchesRequired = True ' Make the main selector an AND filter
OrdSelect.AddSelector(SubSelect)   ' Add sub selector to main order selector
OrdSelect.AddTest "Limit", "9975,10000,10050"  ' Add test for limit prices
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Property:TestMatch, on page 349

Method:AddSelector, on page 350

Method:AddTestEx, on page 353

Method:InvertTest, on page 354

Method:Reset, on page 356

Method: AddTestEx

Purpose

Adds a test condition to an order selector

Syntax

`mySelector.AddTestEx(attrName, attrValue)`

Parameters

attrName	Name of the object attribute to use in the condition Data type: String
attrValue	Value of the attribute to test Data type: Varies based on the attribute value

Return value

None

Exceptions

Exception	Meaning
Invalid Filter	The specified *attrName* is not a valid object attribute for the corresponding order or fill.
Invalid Property Value	The data type of the specified *attrValue* does not match the data type of the attribute.

Description

The `AddTestEx` property adds a single condition to an order selector to help determine which orders to include in the corresponding order set. The condition tests the equality of the attribute with the specified value or values. If you specify a single value, the property matches orders whose attribute has that value. If you specify multiple values for an attribute, the property matches orders whose attribute matches any of the values (logical OR).

Note: When developing application with any non-Microsoft programming language, you must use this method instead of the `AddTest` method. The underlying string data type of the `AddTest` method is incompatible with the `IDispatch` method in the Java/COM Bridge. If you chose to use Java, please be aware that TT does not provide any support for integrating a Java app with XTAPI.

Example

The following example adds two tests to a sub-selector and one test to the parent order selector.

```
Dim OrdSelect As New TTOrderSelector
Dim SubSelect As New TTOrderSelector

SubSelect.AllowAnyMatches = True  ' Make this an OR filter
SubSelect.AddTestEx "IsBuy", True   ' Check for Buys, or
SubSelect.AddTestEx "Acct", "123"   ' Account # "123"
OrdSelect.AllMatchesRequired = True ' Make the main selector an AND filter
OrdSelect.AddSelector(SubSelect)   ' Add sub selector to main order selector
OrdSelect.AddTestEx "Limit", "9975,10000,10050"  ' Add test for limit prices
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Method:AddTest, on page 351

Method:InvertTest, on page 354

Method: InvertTest

Purpose Inverts the logic defined for a selector

Syntax
```
mySelector.InvertTest
```

Parameters None

Return value None

Exceptions None

Description The `InvertTest` method performs a logical NOT operation on the Boolean logic of a selector or sub-selector. The inversion applies to all tests and conditions included in the selector or sub-selector.

The results are not necessarily the opposite of those returned by the uninverted logic. For example, if you create a selector that matches buy market orders (**BuySell** and **OrderType** attributes), the inverted selector does not return all orders except buy market orders. It returns any order that is a sell order or a non-market order.

The following table shows some sample selectors with various tests and sub-selectors to illustrate how the `InvertTest` method affects the selector logic.

Selector Tests	Selector Logic	Inverted Logic	Inverted returns
`ordSel.AddTest("OrderType","L")` `ordSel.InvertTest`	OrderType = "L"	OrderType NOT "L"	All non-limit orders

`ordSel.AddTest("OrderType","L")`
`ordSel.AddTest("OrderType", "M")`
`ordSel.AllowAnyMatches = TRUE`
`ordSel.InvertTest`
	(OrderType = "L") OR (OrderType = "M")	(OrderType NOT "L") AND (OrderType NOT "M")	All non-limit, non-market orders

`ordSel.AddTest("OrderType","L")`
`ordSel.AddTest("Acct", "12345")`
`ordSel.AllMatchesRequired = TRUE`
`ordSel.InvertTest`
	(OrderType = "L") AND (Acct = "12345")	(OrderType NOT "L") AND (Acct NOT "12345")	All orders except non-limit orders for Account 12345

`ordSelA.AddTest("WrkQty",0)`
`ordSelA.InvertTest`
`ordSelB.AllowAnyMatches = TRUE`
`ordSelB.AddTest("Date", "06012012")`
`ordSelB.AddTest("Date", "06022012")`
`ordSelB.InverTest`
`ordSel.AddSelector(ordSelA)`
`ordSel.AddSelector(ordSelB)`
`ordSel.AllMatchesRequired = TRUE`
	(WrkQty = 0) AND ((Date = "06/01/2012") OR (Date = "06/02/2012"))	(WrkQty NOT 0) OR ((Date NOT "06/01/2012") AND (Date NOT "06/02/2012"))	All orders that are either unfilled or that were not placed on June 1-2, 2012

Note: Because the `InvertTest` method inverts the current selector logic, calling this method twice for the same selector cancels the inversion.

Example The following example creates an order set that includes orders in your order book that are either non-limit orders or that have a status other than Hold.

```
Dim OrdSelect As New TTOrderSelector

OrdSelect.AllMatchesRequired = True  ' Make this an AND filter
```

```
OrdSelect.AddTest "OrderType", "L"  ' Include Limit orders and
OrdSelect.AddTest "OrderStatus$", "Hold"   ' Orders on hold
OrdSelect.InvertTest  ' Invert the results
myOrdSet.OrderSelector = OrdSelect  ' Assign the selector to the order set
myOrdSet.Open   ' Open the order set using the new order selector
```

Related information

Property:TestMatch, on page 349

Method:AddSelector, on page 350

Method:AddTest, on page 351

Method:AddTestEx, on page 353

Method: Reset

Purpose	Removes all tests and selectors from the order selector
Syntax	`mySelector.Reset`
Parameters	None
Return value	None
Exceptions	None
Description	The `Reset` property clears the order selector of all tests, sub-selectors, and logic operations. It also resets the `AllMatchesRequired` property to TRUE. The tests and selectors are still defined and available for other selectors in your application.
Example	The following example shows how to reset and reopen an order selector.

```
'Reset the OrderSelector
Private Sub ResetOrderSelector()

  'Create the orderselector
  Dim orderSelector As XTAPI.TTOrderSelector
  orderSelector = New XTAPI.TTOrderSelector

  'Close the orderset
  Me.m_TTOrderSet.Close()

  'Reset the OrderSelector
  orderSelector.Reset()

  'attach the orderselector to the orderset
  m_TTOrderSet.OrderSelector = orderSelector

  'open the orderset
  m_TTOrderSet.Open()

End Sub
```

Related information

Property:AllMatchesRequired, on page 347

Method:AddSelector, on page 350

Method:AddTest, on page 351

Method:AddTestEx, on page 353

20 TTOrderSet Class

Chapter overview

The chapter describes the **`TTOrderSet`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTOrderSet class overview	359
Property:Alias	362
Property:AvgOpenPriceMode	363
Property:Count	365
Property:CreateOrderSelector	366
Property:DebugLogLevel	368
Property:DeleteContracts	371
Property:DeleteOrders	373
Property:EnableFillCaching	376
Property:EnableOrderAutoDelete	377
Property:EnableOrderFillData	378
Property:EnableOrderRejectData	379
Property:EnableOrderSend	380
Property:EnableOrderSetUpdates	381
Property:EnableOrderUpdateData	382
Property:Get	385
Property:IsValidGet	383
Property:IsVolatile	389
Property:Item	390
Property:NextOrderTracker	392
Property:OrderSelector	394
Property:OrderStatusNotifyMode	397
Property:ReadProperties	398
Property:SendOrder	400
Property:SiteKeyLookup	402
Property:SiteKeyLookupEx	403
Property:WriteProperties	404
Method:Close	405
Method:HoldOrder	406
Method:HoldOrders	408
Method:Open	410
Method:Set	411
Method:SetValidExchanges	414
Method:UpdateOrder	416

Section	Page
Method:UpdateXTOrder	418
Event:OnOrderActionRejected	421
Event:OnOrderDeleted	423
Event:OnOrderFillBlockEnd	425
Event:OnOrderFillBlockStart	426
Event:OnOrderFillData	427
Event:OnOrderFilled	429
Event:OnOrderHeld	431
Event:OnOrderInstrumentChanged	432
Event:OnOrderRejected	433
Event:OnOrderSetUpdate	434
Event:OnOrderSubmitted	435
Event:OnOrderUpdated	437

TTOrderSet class overview

Overview

The **`TTOrderSet`** class defines an order set that accesses and manages specific orders in the market. An order set represents a view into the order book, a sub-section of all the orders you place in the market. When you create an order set, you create a set of selection rules, called an order selector, that identifies which of the current orders you want to include in the set. For each matching order, the **`TTOrderSet`** object adds a pointer to the corresponding **`TTOrderObj`** object. You can create multiple order sets to provide different views into the order book. For example, you might create one order set that contains all U.S. 30-Year Treasury Bond (ZB) orders and another order set that contains all buy orders. Also, individual orders can appear in multiple order sets.

Figure 65 shows three order sets that provide different views into the whole order book

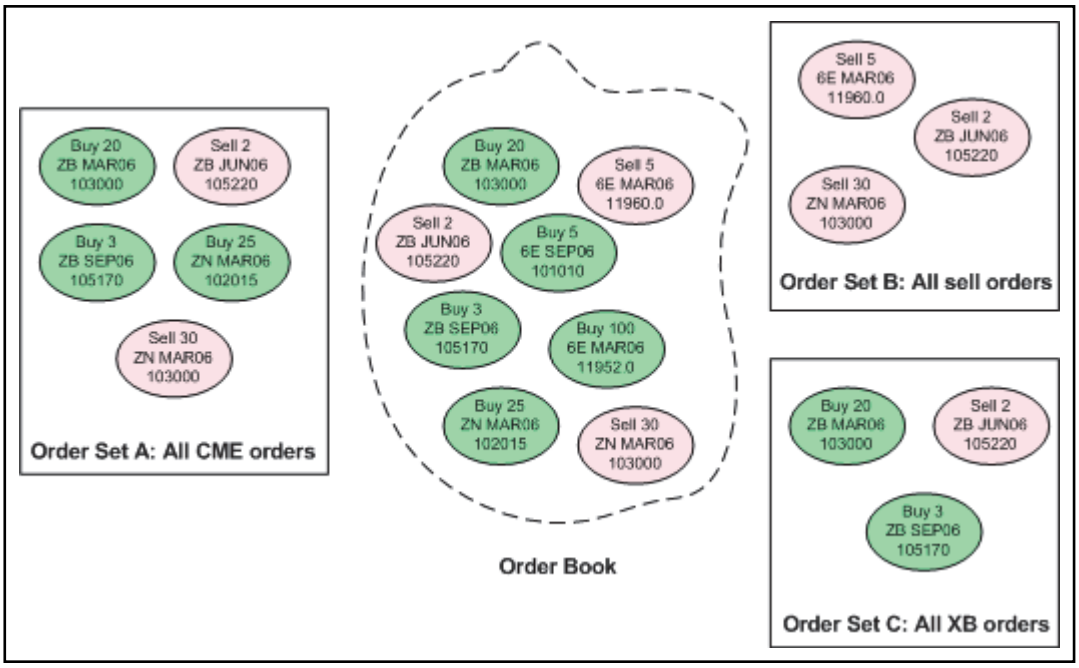


Figure 65. Order set overview

In addition to pointers to individual orders, an order set also contains position summary information based on the orders and associated fills.

Properties

Table 66 lists the **`TTOrderSet`** properties.

Property	Description
Alias	Returns or sets the user-defined name for the order set
AvgOpenPriceMode	Returns or sets the method used to calculate the average price of an open position
Count	Returns the number of orders contained in the order set
CreateOrderSelector	Creates a blank order selector for the order set
DebugLogLevel	Sets or returns the level of information written to the XTAPI debug log
DeleteContracts	Deletes orders for a specified number of contracts
DeleteOrders	Deletes orders within an order set

Table 66. TTOrderSet properties

Property	Description
EnableFillCaching	Returns or sets whether to save a history of fills for an order set
EnableOrderAutoDelete	Returns or sets whether the order set deletes working orders when closed
EnableOrderFillData	Determines whether to send notifications when receiving fills for orders in the order set
EnableOrderRejectData	Controls whether to trigger callbacks when orders are rejected
EnableOrderSend	Controls whether you can submit orders through this order set
EnableOrderUpdateData	Determines whether to add orders to the order tracker
Get	Returns a TTOrderSet object's attributes
IsValidGet	Returns a TTOrderSet object's attributes
IsVolatile	Indicates whether an object attribute for an instrument can change
Item	Returns an order from the order set
NextOrderTracker	Returns the next ordered pair from an order tracker object
OrderSelector	Returns or attaches the order selector that includes orders in the order set
OrderStatusNotifyMode	Returns or sets the level of detail included in order status events
ReadProperties	Returns information about TTOrderSet attributes you can access through the Get property
SendOrder	Submits an order to the exchange
SiteKeyLookup	Returns the order corresponding to a site order key
SiteKeyLookupEx	Returns the order corresponding to a site order key
WriteProperties	Returns information about TTOrderSet attributes you can access through the Set property

Table 66. TTOrderSet properties

Methods

Table 67 lists the `TTOrderSet` methods.

Method	Description
Close	Closes an order set
HoldOrder	Places an order in the market on temporary hold
HoldOrders	Holds orders within an order set
Open	Opens an order set for trading
Set	Sets values for TTOrderSet object's attributes
SetValidExchanges	Limits orders and fills to specific Exchange-Flavors
UpdateOrder	Updates an existing order
UpdateXTOrder	Sends the new color settings to X_TRADER

Table 67. TTOrderSet methods

Events

Table 68 lists the `TTOrderSet` methods.

Method	Description
OnOrderDeleted	Orders are taken out of the market
OnOrderFillBlockEnd	Last fill in the block is sent
OnOrderFillBlockStart	Block of fills are ready to send
OnOrderFillData	Fills are received for an order in the order set
OnOrderFilled	Orders are fully or partially filled
OnOrderHeld	A TT Gateway receives a hold order
OnOrderInstrumentChanged	Changes occur in an order's instrument
OnOrderRejected	TT Gateways or exchanges reject an order in the order set
OnOrderActionRejected	An exchange rejects an order action
OnOrderSetUpdate	Changes occur to an order set or to its associated orders and fills
OnOrderSubmitted	New order is submitted to the exchange or a held order is resubmitted
OnOrderUpdated	Existing orders change

Table 68. TTOrderSet events

Property: Alias

Purpose

Returns or sets the user-defined name for the order set

Syntax

```
osAlias = myOrderSet.Alias
myOrderSet.Alias = osAlias
```

Parameters

osAlias	Name to use as an alias for the order set Data type: String

Return value

osAlias	Alias name currently associated with the order set Data type: String

Exceptions

Exception	Meaning
Alias Already Exists	The specified osAlias already references an order set.

Description

The `Alias` property assigns a user-defined name that can be used to reference an order set. An alias provides a convenient method of accessing order sets from other objects, such as a **`TTGate`** object. By creating an alias, you also make the instrument visible to the **`TTGate`** object's `OrderSets` method.

When you create an alias, the XTAPI adds an in-memory reference to the **`TTOrderSet`** object, which has the side effect of incrementing its reference count. Because the reference count is greater than 0, the XTAPI does not remove it from memory during its ongoing garbage collection activities. Consequently, creating excessive aliases can increase the memory footprint of your application.

Example

The following example shows how to set an alias for an order set.

```
Dim myOrderSet As XTAPI.TTOrderSet

myOrderSet.Alias = "TTSIMSIM001_OrderSet"
```

Related information

`TTInstrObj` Property:Alias, on page 220

`TTOrderProfile` Property:Alias, on page 312

Property: AvgOpenPriceMode

Purpose

Returns or sets the method used to calculate the average price of an open position

Syntax

```
avgMode = myOrderSet.AvgOpenPriceMode
myOrderSet.AvgOpenPriceMode = avgMode
```

Parameters

avgMode — Calculation method. Possible values include:

- NONE
- FIFO
- LIFO

Data type: `enumAvgPriceMode`; default: NONE

Return value

avgMode — Current price averaging mode

Data type: `enumAvgPriceMode`

Exceptions

Exception	Meaning
Cannot set without fill caching enabled	The order set is open, but the `EnableFillCaching` property is not enabled.

Description

The `AvgOpenPriceMode` property specifies how to calculate the average price of an open position, as shown in Table 69.

Calculation method	Description
NONE	Do not calculate average open price
FIFO	Match buys and sells starting with the orders entered first
LIFO	Match buys and sells starting with the orders entered most recently

Table 69. Average open price calculation methods

The calculation method you use can provide different average open position prices, as follows.

Orders Placed	Using FIFO	Using LIFO
Order 1: Buy 5 @ 90	Order 1: Buy 5 @ 90	Order 1: Buy 5 @ 90
Order 2: Sell 3 @ 100	Order 2: Sell 3 @ 100	Order 2: Sell 3 @ 100
Order 3: Buy 5 @ 110	Order 3: Buy 5 @ 110	Order 3: Buy 5 @ 110
	Matches the sells against Order 1	Matches the sells against Order 3
	Average price of the position = ((2 x 90) + (5 x 110)) / 7 = 104.285	Average price of the position = ((5 x 90) + (2 x 110)) / 7 = 95.714

Note: If you enable this property before opening the order set, the `AvgOpenPriceMode` property automatically enables the `EnableFillCaching` property.

Example

The following example sets the average price calculation mode to FIFO.

```
Sub SetAvgOpenPriceMode()
   'NONE(0)
   'FIFO(1)
   'LIFO(2)
    m_OrderSet.AvgOpenPriceMode = 1 'Sets to FIFO Mode
End Sub
```

Related information

None

Property: Count

Purpose	Returns the number of orders contained in the order set
Syntax	`orderCount = myOrderSet.Count`
Parameters	None
Return value	*orderCount* — Number of orders in the order set Data type: String
Exceptions	None
Description	The `Count` property returns the number of **`TTOrderObj`** objects contained in the order set.
Example	None
Related information	Method:Open, on page 410

Property: CreateOrderSelector

Purpose

Creates a blank order selector for the order set

Syntax

```
orderSelector = myOrderSet.CreateOrderSelector
```

Parameters

None

Return value

orderSelector	New or existing order selector object Data type: `TTOrderSelector`

Exceptions

Exception	Meaning
Unable to create Selector	The **`TTOrderSelector`** object was not created successfully.

Description

The `CreateOrderSelector` property creates a new order selector and automatically attaches it to the order set (You need not call the `OrderSelector` property). The order selector applies a filter to all orders to identify specific orders to include in this order set. After you create the order selector, you use **`TTOrderSelector`** object properties and methods to define the filter criteria for the order selector.

You can also use New to create an order selector and attach it to the order set with the `OrderSelector` property.

Note: If the order set already contains an order selector, this property returns the attached **`TTOrderSelector`** object.

Example

The following example shows how to create an order selector for an order set using the `CreateOrderSelector` property.

```
Dim myOrderSet As New TTOrderSet
Dim ordSel as TTOrderSelector

' Code to define selector

ordSel = myOrderSet.CreateOrderSelector

myOrderSet.Open
```

The next example shows how to create a new order selector object and then attach it to an order set using the `OrderSelector` property.

```
Dim myOrderSet As New TTOrderSet
Dim ordSel as New TTOrderSelector

myOrderSet.OrderSelector = ordSel

' Code to define selector

myOrderSet.Open
```

Related information

Property:OrderSelector, on page 394

Method:Open, on page 410

TTOrderSelector Class, on page 343

Property: DebugLogLevel

Purpose

Sets or returns the level of information written to the XTAPI debug log

Syntax

```
logLevel = myOrderSet.DebugLogLevel
myOrderSet.DebugLogLevel = logLevel
```

Parameters

logLevel

Types of information to include in the debug log file, as a bitmask. Base hexadecimal values include:

- `LOG_ERROR (0x00)`
- `LOG_MINIMAL (0x01)`
- `LOG_MIN_API (0x03)`
- `LOG_DET_API (0x04)`
- `LOG_ORDERS (0x8)`
- `LOG_COM (0x10)`
- `LOG_COM_USE (0x20)`
- `LOG_COM_ERR (0x30)`
- `LOG_COM_EVENTS (0x40)`
- `LOG_ALL_COM (0x70)`
- `LOG_FILLS (0x100)`
- `LOG_DEBUG_WIN (0x1000)`

Data type: Integer

Return value

logLevel

Current debug log level as an integer

Data type: Integer

Exceptions

None

Description

This property specifies what types of information the XTAPI writes to a debug log file. If an application is not performing as expected, you can adjust the level of information written to the log file to help you isolate the cause.

Table 70 describes the different debug logging levels.

Enumeration	Description
LOG_ERROR	Logs only error messages
LOG_MINIMAL	Logs minimal generic debugging (low impact) information
LOG_MIN_API	Logs minimal API debugging information, such as open exchanges, server status, and so on. This value is the default setting for the XTAPI.
LOG_DET_API	Generates detailed API logging
LOG_ORDERS	Logs all orders submitted to an exchange
LOG_COM	Logs the creation and destruction of all COM objects and logs (fills and orders can generate a lot of log entries)
LOG_COM_USE	Logs each time XTAPI accesses a COM property
LOG_COM_ERR	Logs each time XTAPI creates or destroys COM objects, accesses COM object properties, or encounters COM errors.
LOG_COM_EVENTS	Logs all COM events
LOG_ALL_COM	Logs all COM information.

Table 70. Debug log levels

Enumeration	Description
LOG_FILLS	Logs all received fills
LOG_DEBUG_WIN	Logs all log information to the debug window
LOG_ALL	Logs everything **Warning:** Using this setting can overwhelm system resources and severely curtail throughput.

Table 70. Debug log levels

Note: Because writing extra information to log files can impact application performance, you should reduce the log level to the default level or lower as soon as possible, especially if you are debugging a production application.

Setting global debug log levels using an initialization file

You can also set the default debug log level in the runtime **XTAPI.ini** file for the application. The debug log level specified in this file applies globally to all objects created by the application. Several different objects contain the DebugLogLevel property for your convenience, so that you don't have to instantiate a **`TTGate`** object just to change the debug log level.

To use an initialization file for the application, create an **XTAPI.ini** file in the same directory as your application's executable file. In the file, add a line similar to the following (case-sensitive):

```
[DebugLogLevel]
  XTAPI_Application=8  # where XTAPI_Application is the application name
```

When your application starts, XTAPI reads this file and sets the debug logging level for the application. Because XTAPI reads the file only at startup, any changes you make to the DebugLogLevel property remain in effect until the application terminates or until you change the value of this property.

Location of debug log files

The XTAPI stores its debug log files in the **<root dir>:\tt\logfiles** directory with the naming convention:

```
XTAPI_YYYY-MM-DD_AppName_ProcessID.log
```

where:

- *YYYY-MM-DD* represents the file creation date
- *AppName* represents the name of the API application
- *ProcessID* is the ID associated with the process.

Format of debug logs

Each line in the debug log file contains three pipe-delimited (|) fields as follows:

```
Date | Message Source | Message
```

where:

- *Date* indicates when XTAPI wrote the message to the file
- *Message Source* identifies where the message originated

- *Message* contains the debug information

Example

The following example temporarily adds order logging for debugging purposes. It saves the current debug log level, changes the level, and finally restores the level to its original setting.

```
Sub PrintDebugInfo()
   Dim myOrderSet As New TTOrderSet
   Dim dLevel As Integer

   dLevel = myOrderSet.DebugLogLevel    ' Save base log level
   Debug.Print "XTAPI version: ", myOrderSet.DebugAppInfo
   Debug.Print "Debug Log Level: ", dLevel

   ' Change the log level to include all order information
   myOrderSet.DebugLogLevel = dLevel + LOG_ORDERS
   Debug.Print "Debug Log Level: ", myOrderSet.DebugLogLevel
End Sub
```

Related information

None

Property: DeleteContracts

Purpose

Deletes orders for a specified number of contracts

Syntax

```
numToDelete = myOrderSet.DeleteContracts(isBuy, minQty, [lowTick], [highTick], [ordSelector], [insideOut])
```

Parameters

isBuy — Which type of orders to delete. Possible values include:

TRUE: Delete buy orders
FALSE: Delete sell orders

Data type: Boolean

minQty — Minimum number of contracts to delete.

Data type: Integer

lowTick — [Optional]. Lowest tick price of orders the property can delete.

Data type: Integer; Default: 0

highTick — [Optional]. Highest tick price of orders the property can delete

Data type: Integer; Default: `MAXLONG`

ordSelector — [Optional]. Order selector that identifies the orders within the order set to delete

Data type: `TTOrderSelector`

insideOut — [Optional]. Whether to start selecting orders from the inside or outside market. Possible values include:

TRUE: Start selecting orders near the market
FALSE: Start selecting orders away from the market

Data type: Boolean

Return value

numToDelete — Number of contracts to delete

Data type: Integer

Exceptions

Exception	Meaning
Invalid Selector	The selector specified in the ordSelector parameter is not a valid order selector.

Description

The `DeleteContracts` property deletes buy or sell orders currently active in the market until it deletes a minimum number of contracts. It deletes only complete orders, so if an order contains more than the minimum number of contracts needed, it deletes the entire order. For example, if you specify 12 contracts and the first three orders contain 10, 5 and 12 contracts, respectively, the `DeleteContracts` property deletes the 10-lot and 5-lot orders.

The quantity returned in `numToDelete` represents the number of contracts the XTAPI attempts to delete, not the number actually deleted. When you call this property, the XTAPI determines which contracts match the specified criteria and

returns the number it found and sent to the exchange to be deleted. When the exchange actually deletes the orders, the XTAPI triggers the `OnOrderDeleted` event.

Example

The following subroutine deletes a minimum of 50 buy orders, starting away from the market.

```
Sub DeleteContracts()

  Dim orderSet As XTAPI.TTOrderSet
  orderSet = New XTAPI.TTOrderSet

  'turn off self limit
  orderSet.Set("NetLimits", False)

  Dim orderSelector As XTAPI.TTOrderSelector
  orderSelector = orderSet.CreateOrderSelector

  'open the orderSet
  orderSet.Open(1)

  Dim isBuy As Integer

  'isBuy TRUE (1): Delete Buy Orders
  isBuy = 1 'Delete Buy Orders

  Dim minQty As Integer
  minQty = 50

  Dim insideOut As Integer
  insideOut = 0 'Start deleting orders away from the market

  Dim lowTick As Double
  Dim highTick As Double

  ' Obtain the tick price of the instrument.
  Dim tickPrice As Double = Convert.ToDouble(m_InstrObj.TickPrice(0, 1, "$"))

  ' Obtain the tick increment of the instrument.
  Dim tickIncrement As Integer = m_InstrObj.Get("TickIncrement")

  ' Calculate the lowTick and highTick
  lowTick = Convert.ToDouble(lowPriceRangeTextBox.Text) /
       (tickPrice * tickIncrement)

  highTick = Convert.ToDouble(highPriceRangeTextBox.Text) /
       (tickPrice * tickIncrement)

  Dim deleteOrdersQuantity As Integer
  deleteOrdersQuantity = orderSet.DeleteContracts(isBuy, minQty, lowTick,
       highTick, orderSelector, insideOut)

End Sub
```

Related information

Property:DeleteOrders, on page 373

Method:HoldOrder, on page 406

Event:OnOrderDeleted, on page 423

Property: DeleteOrders

Purpose

Deletes orders within an order set

Syntax

```
numToDelete = myOrderSet.DeleteOrders([isBuy], [lowTick],
[highTick], [isInRange], [ordSelector])
```

Parameters

isBuy	[Optional]. Which type of orders to delete. Possible values include: **TRUE**: Delete buy orders **FALSE**: Delete sell orders If you omit this parameter, the property deletes both buy and sell orders. Data type: Boolean **Note**: For .NET languages, you must supply `System.Type.Missing` for this parameter if you want to delete both types of orders.
lowTick	[Optional]. Lowest tick price of orders for the delete range, inclusive. Data type: Integer; Default: 0
highTick	[Optional]. Highest tick price of orders the delete range, inclusive Data type: Integer; Default: `MAXLONG`
isInRange	[Optional]. Whether to delete orders within the specified range or orders outside the range. **TRUE**: Delete orders inside the specified range **FALSE**: Delete orders outside the specified range Data type: Boolean
ordSelector	[Optional]. Order selector that identifies the orders within the order set to delete If you omit this parameter, the property deletes all orders in the order set that match the specified criteria. Data type: `TTOrderSelector`

Return value

numToDelete	One of the following: • For lot-based contracts, the number of contracts sent • For flow-based (i.e. energy) contracts, the flow quantity sent Data type: Integer

Exceptions

Exception	Meaning
Invalid Selector	The selector specified in the ordSelector parameter is not a valid order selector.

Description

The `DeleteOrders` property deletes orders that meet a specified criteria. For example, you can delete all Buy orders or all orders outside a tick range. If you call this property with no arguments, it deletes all orders in the order set, starting from the inside market and alternating between Buy and Sell orders to minimize the chance that an order might be filled before it is deleted.

The quantity returned in *`numToDelete`* represents the number of contracts or the flow quantity the XTAPI attempts to delete, not the number actually deleted. When you call this property, the XTAPI determines which contracts match the specified criteria and returns the number it found and sent to the exchange to be deleted. When the exchange actually deletes the orders, the XTAPI triggers the `OnOrderDeleted` event.

Example

The following subroutine deletes all Buy orders whose price falls within a specified tick range.

```
Sub DeleteOrders()

   Dim orderSet As XTAPI.TTOrderSet
   orderSet = New XTAPI.TTOrderSet

   'turn off self limit
   orderSet.Set("NetLimits", False)

   Dim orderSelector As XTAPI.TTOrderSelector
   orderSelector = orderSet.CreateOrderSelector

   'open the orderSet and enable order entry
   orderSet.Open(1)

   Dim isBuy As Integer

   'isBuy TRUE (1): Delete Buy Orders
   ' FALSE (0): Delete Sell Orders
   isBuy = 1 'Delete Buy Orders

   Dim isInRange As Integer

   'isInRange TRUE (1): Delete orders inside the range
   'isInRange FALSE (0): Delete orders outside the range
   isInRange = 1

   Dim lowTick As Double
   Dim highTick As Double

   ' Obtain the tick price of the instrument.
  Dim tickPrice As Double = Convert.ToDouble(m_InstrObj.TickPrice(0, 1, "$"))

   ' Obtain the tick increment of the instrument.
   Dim tickIncrement As Integer = m_InstrObj.Get("TickIncrement")

   ' Calculate the lowTick and highTick
   lowTick = Convert.ToDouble(lowPriceRangeTextBox.Text) /
         (tickPrice * tickIncrement)

   highTick = Convert.ToDouble(highPriceRangeTextBox.Text) /
         (tickPrice * tickIncrement)
```

```
    Dim deleteOrdersQuantity As Integer
    deleteOrdersQuantity = orderSet.DeleteOrders(isBuy, lowTick, highTick,
                                    isInRange, orderSelector)

End Sub
```

Related information

Property:DeleteContracts, on page 371

Method:HoldOrder, on page 406

Event:OnOrderDeleted, on page 423

Property: EnableFillCaching

Purpose

Returns or sets whether to save a history of fills for an order set

Syntax

```
isEnabled = myOrderSet.EnableFillCaching
myOrderSet.EnableFillCaching = isEnabled
```

Parameters

isEnabled	Whether to cache fills for the order set Data type: Boolean; Default: FALSE

Return value

isEnabled	Whether the order set caches fills Data type: Boolean

Exceptions

Exception	Meaning
Cannot set after order set is open	Fill caching must be set before opening the order set.

Description

The `EnableFillCaching` property determines whether to save fill information in the order set. By default, the XTAPI acknowledges fills and adjusts order quantity and positions, but does not store the fills themselves as part of the order set. If you want to save the fills as part of the order set, you can enable this property.

You must set this value before calling the `Open` method.

Example

None

Related information

Method:Open, on page 410

Property: EnableOrderAutoDelete

Purpose	Returns or sets whether the order set deletes working orders when closed
Syntax	`isEnabled = myOrderSet.EnableOrderAutoDelete` `myOrderSet.EnableOrderAutoDelete = isEnabled`
Parameters	*isEnabled* — Whether to delete working orders automatically when closing the order set Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether the order set automatically deletes working orders Data type: Boolean
Exceptions	None
Description	The `EnableOrderAutoDelete` property lets you delete working orders automatically when you close an order set. By default, the XTAPI leaves working orders in the market when you close an order set. Because an order set is only a view of a subset of all orders and because an order can be included in multiple order sets, the XTAPI leaves orders in the market. If you set this property to TRUE, the XTAPI deletes all working orders contained in the order set when you close it. Any other order sets that include the same working orders are also impacted.
Example	None
Related information	None

Property: EnableOrderFillData

Purpose	Determines whether to send notifications when receiving fills for orders in the order set
Syntax	`isEnabled = myOrderSet.EnableOrderFillData` `myOrderSet.EnableOrderFillData = isEnabled`
Parameters	*isEnabled* — Whether to send fill notifications Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether XTAPI currently sends fill notifications Data type: Boolean
Exceptions	None
Description	The `EnableOrderFillData` property allows you to control when to send notifications when fills occur. When a partial or full fill is received for an order in the order set, the XTAPI generates a notification. You can then use the `OnOrderFillData` event to respond to the event. If you enable fill data in an order set, the XTAPI downloads all fills from the exchanges received since the start of day when you open the order set. If the fills have already been downloaded from another order set or some other mechanism, such as the **`TTGate`** `OpenExchangeFills` method, the XTAPI does not download them again, but does send notifications for all of the fills. Every time you open an order set with this property enabled, the fill notifications are generated. **Note:** If your application encounters performance or efficiency issues, you can disable this property. You can also use the `SetValidExchange` method to limit fill notifications to specific exchanges. By default, the `SetValidExchange` method is set to *****, which reacts to fills from all TT Gateways.
Example	None
Related information	None

Property: EnableOrderRejectData

Purpose	Controls whether to trigger callbacks when orders are rejected
Syntax	`isEnabled = myOrderSet.EnableOrderRejectData` `myOrderSet.EnableOrderRejectData = isEnabled`
Parameters	*isEnabled* — Whether to allow order rejection notifications Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether XTAPI currently sends order rejection notifications Data type: Boolean
Exceptions	None
Description	The `EnableOrderRejectData` property determines whether the XTAPI triggers an `OnOrderRejected` event when a gateway or exchange rejects and order attached to this order set. Enabling this property provides a more convenient to detect rejected orders. If you disable this property, your application does not receive events for rejected messages, so the application must use the **`TTOrderTrackerObj`** object or follow the order chain to determine whether an order was rejected. If you enable this property, the XTAPI throws the events immediately. If you receive a large number of rejections in succession, your application could block waiting for the events to end. To throttle the number of events the XTAPI triggers at a time, you can adjust the **MaxRejects** attribute using the Set method.
Example	None
Related information	Event:OnOrderRejected, on page 433

Property: EnableOrderSend

Purpose	Controls whether you can submit orders through this order set
Syntax	`isEnabled = myOrderSet.EnableOrderSend` `myOrderSet.EnableOrderSend = isEnabled`
Parameters	*isEnabled* — Whether to allow order submission Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether the order set can submit orders Data type: Boolean
Exceptions	None
Description	The `EnableOrderSend` property determines whether you can use the `SendOrder` property to submit orders to the exchange. As a precaution, order submission is disabled for order sets so that orders cannot be sent to the market unintentionally. To enable order submission from this order set, set `EnableOrderSend` to TRUE. If `EnableOrderSend` is FALSE, the `SendOrder` property returns a zero quantity with no error messages or exceptions.
Example	None
Related information	None

Property: EnableOrderSetUpdates

Purpose	Controls whether XTAPI sends your application events notifications when changes occur to orders in the order set
Syntax	`isEnabled = myOrderSet.EnableOrderSetUpdates` `myOrderSet.EnableOrderSetUpdates = isEnabled`
Parameters	*isEnabled* — Whether to send notifications when the order set changes Data type: Boolean; Default: TRUE
Return value	*isEnabled* — Whether notifications are sent when the order set changes Data type: Boolean
Exceptions	None
Description	The `EnableOrderSetUpdates` property allows you to control how the XTAPI sends your application events notifications when changes occur to orders in the order set. When some action occurs that affects an order set, the XTAPI generates a notification if this property is set to **TRUE**. You can then use the `OnOrderSetUpdate` event to respond to the event. Many activities can cause an order set to update, such as: • A complete or partial fill is received. • An order is submitted from the order set. • An order in the order set is deleted or put on hold. Because an order can be contained in multiple order sets, actions occurring in other order sets can cause updates in the current order set. For example, assume one order set (A) contains all eCBOT orders while another order set (B) contains all ZB orders. If a ZB order is deleted by order set B, it also causes an update to occur for order set A.
Example	None
Related information	Event:OnOrderSetUpdate, on page 434

Property: EnableOrderUpdateData

Purpose	Determines whether to add orders to the order tracker
Syntax	`isEnabled = myOrderSet.EnableOrderUpdateData` `myOrderSet.EnableOrderUpdateData = isEnabled`
Parameters	*isEnabled* — Whether to update the order tracker Data type: Boolean; Default: FALSE
Return value	*isEnabled* — Whether the order set adds orders to the order tracker object Data type: Boolean
Exceptions	None
Description	The `EnableOrderUpdateData` property controls whether order updates are monitored by the associated order tracker (**`TTOrderTrackerObj`**) object. When this property is **TRUE**, the XTAPI adds order pairs for all order updates to the order tracker list for this order set.
Example	None
Related information	Method:UpdateOrder, on page 416

Caution: The ordered pair list can grow quite large for active orders. If your client application does not clear the orders in the list, enabling this property can dramatically increase memory usage.

Property: IsValidGet

Purpose

Returns a **`TTOrderSet`** object's attributes

Syntax

attrValues = myOrderSet.IsValidGet(*attributes*)

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Array of Boolean values for each argument name supplied in attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single Boolean value.

Exceptions

None

Description

The `IsValidGet` property determines whether a specified list of strings represent valid arguments for the `Get` property. You typically use this property to ensure that all of the arguments you want to pass to the `Get` property are valid. By validating the list of attributes first, you can avoid exceptions thrown for invalid arguments when you call the `Get` property.

Note: The returned array uses a 1-based index.

Example

The following example uses the `IsValidGet` property to validate the instrument attributes passed to it and includes a stub to call the `Get` property if all attributes are valid.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidName As Boolean
  Dim isValidNameArr As Array
  Dim attrName As String
  Dim isValidList As Boolean
  isValidList = True
  Dim i As Integer
  i = 1

  isValidNameArr = Me.m_TTGate.IsValidGet(objAttributes)

  For Each isValidName in isValidNameArr
    If isValidName = False Then
      isValidList = False
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Error: Invalid Attribute: {0}", attrName)
    End If
    i = i + 1
    Next
  End For

  If isValidList = True Then
```

```
    ' code to call Get property omitted
  End If
End Sub
```

Related information

Property:Get, on page 385

Property:ReadProperties, on page 398

Property: Get

Purpose

Returns a **TTOrderSet** object's attributes

Syntax

```
attrValues = myOrderSet.Get(attributes)
```

Parameters

attributes	Comma-separated string of object attributes Data type: String

Return value

attrValues	Values for the specified object attributes. Data type: Array **Note:** If attributes contains only one attribute, the property returns a single value with the appropriate data type.

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the attributes in attributes is not a valid **TTFillObj** object attribute

Description

The `Get` property returns the values for an arbitrary number of **TTOrderSet** object attributes. The property returns an array of values corresponding, in order, to the specified attributes. You must cast the returned values to the appropriate data types.

Note: If you need to query multiple object attributes, you should specify all of them in a single `ChangeMe` property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values.

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 71 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 71. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 71. Price and quantity value formats

Table 72 lists the **`TTOrderSet`** object attributes you can supply to the `Get` property.

Attribute	Description	DataType
BuyCnt	Number of buy orders in the order set	Integer
BuyFillCnt	Number of fills for buy orders. Valid only when the `EnableFillCaching` property is TRUE.	Integer
BuyFills	Row-based, two-dimensional array of values from all filled buy orders, where each row contains an fill and each column contains an attribute value.	Variant
BuyFills90	Column-based, two-dimensional array of values from all filled buy orders, where each column contains an fill and each row contains an attribute value.	Variant
BuyLimits	Whether the order set checks self-imposed limits (**MaxBuy** attributes) when submitting buy orders	Boolean
BuyOPos	Quantity of contracts purchased based on successfully executed buy orders	Integer
BuyOrders	Row-based, two-dimensional array of values from all buy orders, where each row contains an order and each column contains an attribute value.	Variant
BuyOrders90	Column-based, two-dimensional array of values from all buy orders, where each column contains an order and each row contains an attribute value.	Variant
BuyPos	Quantity purchased based on fill information	Integer
BuyTicks	Cost in ticks of all purchased contracts	Integer
BuyWrk	Quantity of working buys	Integer
Fills	Row-based, two-dimensional array of values from all fills, where each row contains an fill and each column contains an attribute value.	Variant
Fills90	Column-based, two-dimensional array of values from all fills, where each column contains an fill and each row contains an attribute value.	Variant
MaxBuyOrderQty	Maximum quantity allowed in a single buy order	Integer
MaxBuyOrders	Maximum number of buy orders allowed in the order set	Integer
MaxBuyWorking	Maximum number of working orders allowed in the order set	Integer
MaxFills	Maximum number of fill records that can be included in one fill block	Integer
MaxOrderQty	Maximum quantity allowed in a single buy or sell order	Integer
MaxOrders	Maximum number of orders allowed in the order set	Integer
MaxOrderStatusUpdates	Maximum number of consecutive order updates allowed before allowing other events to fire	Integer
MaxPosition	Maximum long or short position (in ticks) allowed in the order set	Integer

Table 72. TTOrderSet Get attributes

Attribute	Description	DataType
MaxRejects	Maximum number of order rejection events sent in succession	Integer
MaxSellOrderQty	Maximum quantity allowed in a single sell order	Integer
MaxSellOrders	Maximum number of sell orders allowed in the order set	Integer
MaxSellWorking	Maximum number of working sell orders allowed in the order set	Integer
MaxWorking	Maximum number of working orders allowed in the order set	Integer
NetCnt	Total number of working orders (Buys + Sells)	Integer
NetFillCnt	Total number of filled orders	Integer
NetLimits	Whether the order set checks self-imposed position limits when submitting an order	Boolean
NetLongPosMax	Maximum net long position since order set was opened (quantity purchased - quantity sold), based on fill information	Integer
NetOPos	Current net position based on orders	Integer
NetPos	Current net position based on fill information	Integer
NetShortPosMax	Maximum net short position since order set was opened (quantity sold - quantity purchased), based on fill information	Integer
NetTicks	Value of the **NetTicks** attribute, which is the net cost in ticks of the current position. Valid only when you attach the instrument to a **TTOrderSet** object.	Decimal
NetWrk	Total quantity of working orders (working buys + working sells)	Integer
Orders	Row-based, two-dimensional array of values from all orders, where each row contains an order and each column contains an attribute value.	Array
Orders90	Column-based, two-dimensional array of values from all orders, where each column contains an order and each row contains an attribute value.	Array
OrderStateCorrection	TT internal use only.	Boolean
RealizedPL	Current realized P&L in ticks	Integer
SellCnt	Number of sell orders in the order set	Integer
SellFillCnt	Number of fills for sell orders. Valid only when the `EnableFillCaching` property is TRUE.	Integer
SellFills	Row-based, two-dimensional array of values from all filled sell orders, where each row contains an fill and each column contains an attribute value.	Variant
SellFills90	Column-based, two-dimensional array of values from all filled sell orders, where each column contains an fill and each row contains an attribute value.	Variant
SellLimits	Whether the order set checks self-imposed limits (**MaxSell*** attributes) when submitting sell orders	Integer
SellOPos	Quantity of contracts sold based on successfully executed sell orders	Integer
SellOrders	Row-based, two-dimensional array of values from all sell orders, where each row contains an order and each column contains an attribute value.	Array
SellOrders90	Column-based, two-dimensional array of values from all sell orders, where each column contains an order and each row contains an attribute value.	Array
SellPos	Quantity sold based on fill information	Integer

Table 72. TTOrderSet Get attributes

Attribute	Description	DataType
SellTicks	Cost in ticks of all sold contracts	Integer
SellWrk	Quantity of working sells	Integer
ThrowLimitErrors	How the XTAPI responds when a self-imposed limit is violated. Possible values include: **FALSE**, reduces order to self-imposed limit **TRUE**, throws an error	Integer

Table 72. TTOrderSet Get attributes

Example

The following example uses the `Get` property to display the values for the specified **`TTOrderSet`** attributes.

```
'Assumes objAttributes is a comma-separated string of attribute names
Private Sub GetAttributesIfValid(ByVal objAttributes As String)

  Dim isValidNameArr As Array
  Dim attrValuesArr As Array
  Dim isValidList As Boolean
  Dim attrName As String
  Dim i As Integer
  i = 1

  'good practice to validate list before calling Get
  isValidNameArr = Me.m_TTOrderSet.IsValidGet(objAttributes)

  ' code to verify attribute list omitted

  If isValidList = True Then
    attrValuesArr = Me.m_TTOrderSet.Get(objAttributes)

    'Display each attribute and value
    For Each attrValue in attrValuesArr
      attrName = objAttributes.Split(",").GetValue(i)
      Console.Writeline("Attribute: {0} = {1}", attrName, attrValue)
      i = i + 1
      Next
    End For

  End If
End Sub
```

Related information

Property:IsValidGet, on page 383

Property:ReadProperties, on page 398

Method:Set, on page 411

Property: IsVolatile

Purpose	Indicates whether an object attribute for an instrument can change
Syntax	`isValidArr = myOrderSet.IsVolatile(attrArr)`
Parameters	
attArr	Array of attribute names Data type: Array
Return value	
isValidArr	Array of Boolean values for each argument name supplied in `attrArr`. Data type: Array
Exceptions	None
Description	The `IsVolatile` property returns a Boolean value indicating whether a particular object attribute can change during the life of the order set. Some attributes, like **Bid** or **Ask**, can change after an instrument in instantiated. Others, like **Market** or **Name**, are static and do not change over the life of the instrument in your application. The property returns an array, where each element contains a Boolean value indicating whether the name specified in the same element of the input array is a valid attribute.
Example	None
Related information	None

Property: Item

Purpose

Returns an order from the order set

Syntax

```
orderObj = myOrderSet.Item([orderNum|indexNum])
```

Parameters

orderNum	Exchange-specified order number Data type: String
indexNum	Position in the order set collection Data type: Integer

Return value

orderObj	Order requested from the order set Data type: TTOrderObj

Exceptions

None

Description

The `Item` property returns the current snapshot of an order from the order set. It returns a pointer to the order object at the tail of the order chain. You can request the order either by the order number or by the index of the order in the order set.

If you know the order number, such as getting it as input from a user, you can specify the order number to retrieve the order object. The order numbers are identified by a String data type (*`orderNum`*). You can supply the order number in one of the following formats:

Format	Result
exchange:order#	Returns the order with the specified order number and exchange.
order#	Returns the first order that matches the specified order number, without regard to the exchange. If the order set contains two orders with the same order number from different exchanges, the property returns the first one it finds.

To avoid potential confusion, you should always use the exchange:order# format when connected to multiple exchanges. If the application connects to only one exchange, you can use the **order#** format to reduce the overhead of string processing.

You can also traverse the order collection in an order set by looping through them and retrieving them by their respective position indexes. When you specify an Integer data type (*`indexNum`*), the property returns the order object at the specified position in the collection.

Note: The order collection in the order set uses a 1-based index.

If the requested order does not exist on the specified exchange or is not available, the property returns a **`TTOrderObj`** object with the `IsNull` property set to TRUE.

Example

The following example shows how to get an order based on its order number.

```
Sub GetOrderObj(ByVal pStrOrderNo As String)
```

```
    'Input is the Exchange Order ID
    Dim strOrderNo As String
    strOrderNo = pStrOrderNo

    Dim myOrderObj As XTAPI.TTOrderObj
    myOrderObj = m_OrderSet.Item(strOrderNo)

    Console.WriteLine(myOrderObj.Get("Qty"))

End Sub
```

Related information None

Property: NextOrderTracker

Purpose

Returns the next ordered pair from an order tracker object

Syntax

`orderTracker = myOrderSet.NextOrderTracker`

Parameters

None

Return value

orderTracker	Order tracker with the current and next snapshot of the order Data type: TTOrderTrackerObj

Exceptions

Exception	Meaning
Unable to create order tracker	The order set does not contain another order tracker object.
Order tracking is not enabled	The `EnableOrderUpdateData` property is FALSE.

Description

The `NextOrderTracker` property returns the next order pair in the order set's **`TTOrderTrackerObj`** collection. The order set can contain multiple orders, so the tracker object returned can help you track the progress of an order over time.

Because the order tracker object contains two consecutive snapshots of an order's state, you can compare them to determine what changed in the order. If you call `NextOrderTracker` when no next tracker object exists, the property throws an exception.

Example

The following example displays the order numbers for old and new order pairs in an order set.

```
Private Sub m_TTOrderSet_OnOrderSetUpdate(ByVal pOrderSet As XTAPI.TTOrderSet)
Handles m_TTOrderSet.OnOrderSetUpdate

   Dim m_OrderTrackerObj As XTAPI.TTOrderTrackerObj
   m_OrderTrackerObj = New XTAPI.TTOrderTrackerObj

  'Get the next order tracker object from the orderset
   m_OrderTrackerObj = pOrderSet.NextOrderTracker

  'Use a while loop to iterate through the updated order objects
  While Not m_OrderTrackerObj Is Nothing

      If m_OrderTrackerObj.HasOldOrder <> 0 Then
       Console.WriteLine("Has old order " +
                    m_OrderTrackerObj.OldOrder.Get("OrderNo"))
      End If

      If m_OrderTrackerObj.HasNewOrder <> 0 Then
       Console.WriteLine("Has new order" +
                    m_OrderTrackerObj.NewOrder.Get("OrderNo"))
      End If

      'Get the next order tracker
      m_OrderTrackerObj = pOrderSet.NextOrderTracker
  End While
End Sub
```

Related information None

Property: OrderSelector

Purpose

Returns or attaches the order selector that includes orders in the order set

Syntax

```
orderSelector = myOrderSet.OrderSelector
myOrderSet.OrderSelector = orderSelector
```

Parameters

orderSelector	Order selector to attach to the order set Data type: `TTOrderSelector`

Return value

orderSelector	Order selector currently associated with the order set Data type: `TTOrderSelector`

Exceptions

Exception	Meaning
Invalid Selector	The *orderSelector* parameter is not reference a valid **TTOrderSelector** object.
Unable to create selector	Some internal error occurred.

Description

The `OrderSelector` property contains the order selector for the order set. The order selector determines which orders to include in this order set. You must attach an order selector before calling the `Open` method.

If you attach an order selector after the order set is open, or if you change the order selector, the order set automatically closes. The best way to update the order selector is to close the current order set, attach the new selector, and open the order set again.

For more information about order selectors, see Chapter 19, TTOrderSelector Class

Example

The following sample shows how to replace an existing order selector.

```
Private Sub ChangeOrderSelector()

  'Create the orderselector
  Dim orderSelector As XTAPI.TTOrderSelector
  orderSelector = New XTAPI.TTOrderSelector

  'Close the orderset
  Me.m_TTOrderSet.Close()

    'set the test so that the order set
    'filters all orders out except for market orders
      orderSelector.AddTest("OrderType", "M")

      'attache the orderselector to the orderset
      m_TTOrderSet.OrderSelector = orderSelector

      'open the orderset
      m_TTOrderSet.Open()

End Sub
```

Related information

Property:CreateOrderSelector, on page 366
TTOrderSelector Class, on page 343

Property: OrderStatusNotifyMode

Purpose	Returns or sets the level of detail included in order status events
Syntax	`myOrderSet.OrderStatusNotifyMode(`*`notifyMode`*`)` *`notifyMode`* `= myOrderSet.OrderStatusNotifyMode`
Parameters	
notifyMode	Notification mode to use for order status events. Possible values include: • ORD_NOTIFY_NONE • ORD_NOTIFY_NORMAL • ORD_NOTIFY_VERBOSE Default: ORD_NOTIFY_NONE Data type: `enumOrderStatusNotifyMode`
Return value	
notifyMode	Current notification mode Data type: `enumOrderStatusNotifyMode`
Exceptions	None
Description	The `OrderStatusNotifyMode` property sets or returns the notification mode for the order status events.
Example	None
Related information	Event:OnOrderActionRejected, on page 421 Event:OnOrderInstrumentChanged, on page 432 Event:OnOrderRejected, on page 433 Event:OnOrderSubmitted, on page 435 Event:OnOrderUpdated, on page 437

Property: ReadProperties

Purpose

Returns information about **`TTOrderSet`** attributes you can access through the `Get` property

Syntax

attributes = myOrderSet.ReadProperties

attrDescription = myOrderSet.ReadProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to return Data type: String

Return value

attributes	Array of attributes you can pass to the `Get` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an attrName is not one of the object's attributes

Description

With no parameters, the `ReadProperties` property returns a list of all object attributes that you can pass as arguments to the `Get` property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `ReadProperties` property to display the descriptions for each **`TTOrderSet`** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:Get, on page 385

Property:IsValidGet, on page 383

Property:WriteProperties, on page 404

Property: SendOrder

Purpose

Submits an order to the exchange

Syntax

```
numSent = myOrderSet.SendOrder(orderProfile)
```

Parameters

orderProfile	Order profile containing the order instructions Data type: `TTOrderProfile`

Return value

numSent	One of the following: • For lot-based contracts, the number of contracts sent • For flow-based (i.e. energy) contracts, the flow quantity sent Data type: Integer

Exceptions

Exception	Meaning
Incomplete Profile	The order profile does not contain enough information to submit an order. For example, the **Acct** attribute might have no value.
Invalid Profile	The object supplied in *orderProfile* is not a valid **TTOrderProfile** object.
Orderset Not Open	The `Open` method has not been called, or the order set has been closed.
Order Does Not Match Order Set	The order profile (**TTOrderProfile** object) for this order does not match the filter set (**TTOrderSelector** object).
Order Server Down	The TT Gateway's Order Server could not be accessed.

Description

The `SendOrder` property submits an order based on the specified order profile. The order profile contains the information about the order, such as whether it is a buy or sell order, quantity, and price. For more information about order profiles, see Chapter 18, TTOrderProfile Class.

The property returns the number of contracts or the flow quantity submitted to the exchange. Under normal circumstances, this number equals the quantity submitted. However, if you have enabled self-imposed limits in the XTAPI, the quantity submitted could be fewer than the quantity specified in the order profile.

When you send an order, the XTAPI retrieves a unique key from the Key Server defined in your network and uses it to set the value of the **SiteOrderKey** (or **SiteKey**) object attributes for the **TTOrderObj** and **TTOrderProfile** objects. The XTAPI uses this site order key to identify an order throughout its lifetime.

Example

The following example uses the `SendOrder` property to send a Buy market order.

```
Private Sub SendOrder()

  Try
    'Create an order set
    m_TTOrderSet = New XTAPI.TTOrderSet
```

```
        'Enable the orderSet to send an order
        m_TTOrderSet.EnableOrderSend = True

        'Turn off the self imposed limits
        m_TTOrderSet.Set("NetLimits", False)

        'Open the orderset for order entry
        m_TTOrderSet.Open(1)

        'Create an orderprofile
        Dim orderProfile As XTAPI.TTOrderProfile
        orderProfile = New XTAPI.TTOrderProfile

        'set the needed parameters for the orderprofile
        orderProfile.Customer = "<Default>"
        orderProfile.Instrument = Me.m_TTInstrObj
        orderProfile.Set("BuySell", "Buy")
        orderProfile.Set("Qty", "1")
        orderProfile.Set("OrderType", "M")

        'Create an integer to hold the results of the ordersend function.
        Dim intOrderQty As Integer

        'intOrderQty should equal the order Qty entered
        intOrderQty = Me.m_TTOrderSet.SendOrder(orderProfile)

        If (Not intOrderQty = 1) Then
            Console.WriteLine("ERROR sending order")
        End If

      Catch ex As Exception
        Console.WriteLine(ex.Message)
      End Try

    End Sub
```

Related information

Method:UpdateOrder, on page 416

Method:SetValidExchanges, on page 414

Property: SiteKeyLookup

Purpose

Returns the order corresponding to a site order key

Syntax

`order = myOrderSet.SiteKeyLookup(key)`

Parameters

key	Unique identifier for the order Data type: String

Return value

order	Order object associated with the specified key Data type: `TTOrderObj`

Exceptions

None

Description

The `SiteKeyLookup` property returns the order object associated with the specified site order key. When you send an order, the XTAPI assigns a string ID that uniquely identifies the order through its lifetime. The site order key differs from the order number supplied by the exchange. The site order key is stored in the **SiteOrderKey** attribute of the corresponding order and order profile objects.

The site order key allows you to retrieve orders that have been rejected, filled, modified, or deleted.

Note: When developing a Java application, you must use the `SiteKeyLookupEx` property instead of this property. The underlying string data type of this property is incompatible with the `IDispatch` method in the Java/COM Bridge. If you chose to use Java, please be aware that TT does not provide any support for integrating a Java app with XTAPI.

Example

The following code fragment shows how to return an order object by it site order key.

```
'The site order key is stored from
'a call like this one that would be in another method
'Me.m_strSiteOrderKey = orderProfile.GetLast("SiteOrderKey")
Private Sub GetOrder()

 'the order is retrieved using the stored site order key
 Dim orderObj As XTAPI.TTOrderObj =
m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

 'outputs information from the orderobj
 Console.WriteLine(orderObj.Get("BuySell"))
 Console.WriteLine(orderObj.Get("SiteOrderKey"))
End Sub
```

Related information

None

Property: SiteKeyLookupEx

Purpose

Returns the order corresponding to a site order key

Syntax

```
order = myOrderSet.SiteKeyLookupEx(key)
```

Parameters

key	Unique identifier for the order Data type: String

Return value

order	Order object associated with the specified key Data type: `TTOrderObj`

Exceptions

None

Description

The `SiteKeyLookupEx` property returns the order object associated with the specified site order key. When you send an order, the XTAPI assigns a string ID that uniquely identifies the order through its lifetime. The site order key differs from the order number supplied by the exchange. The site order key is stored in the **SiteOrderKey** attribute of the corresponding order and order profile objects.

The site order key allows you to retrieve orders that have been rejected, filled, modified, or deleted.

Note: When developing a Java application, you must use this property instead of the `SiteKeyLookup` property. The underlying string data type of the `SiteKeyLookup` property is incompatible with the `IDispatch` method in the Java/COM Bridge. If you chose to use Java, please be aware that TT does not provide any support for integrating a Java app with XTAPI.

Example

The following code fragment shows how to return an order object by it site order key.

```
'The site order key is stored from a call like this in another method
'Me.m_strSiteOrderKey = orderProfile.GetLast("SiteOrderKey")
'Use the SiteKeyLookupEx vs. the SiteKeyLookup when using java
Private Sub GetOrder()

  'the order is retrieved using the stored site order key
  Dim orderObj As XTAPI.TTOrderObj =
       m_TTOrderSet.SiteKeyLookupEx(m_strSiteOrderKey)

  'outputs information from the orderobj
  Console.WriteLine(orderObj.Get("BuySell"))
  Console.WriteLine(orderObj.Get("SiteOrderKey"))
End Sub
```

Related information

None

Property: WriteProperties

Purpose

Returns information about **`TTOrderSet`** attributes you can access through the `Set` property

Syntax

attributes = myOrderSet.WriteProperties

attrDescription = myOrderSet.WriteProperties(*attrName*)

Parameters

attrName	Optional name of an object's attribute whose description you want to set Data type: String

Return value

attributes	Array of attributes you can pass to the `Set` property. The property returns this value when you omit *attrName*. Data type: Array
attrDescription	Description of the attribute specified in the optional *attrName* parameter Data type: String

Exceptions

Exception	Meaning
Invalid Property	Value of an *attrName* is not one of the object's attributes

Description

With no parameters, the `WriteProperties` property returns a list of all object attributes that you can pass as arguments to the Set property. Each element in the array contains a string with an attribute name.

Note: The property returns the attributes in no particular order.

You can also specify an attribute to return its description.

Example

The following example uses both versions of the `WriteProperties` property to display the descriptions for each **`TTOrderSet`** object attribute.

```
Private Sub ShowGetAttrDescriptions()
   Dim myFillObj As New TTFillObj

   Dim getAttr as Variant
   Dim attrDesc as String

   For Each getAttr In myFillObj.ReadProperties     ' return attributes
      attrDesc = myFillObj.ReadProperties(getAttr)   ' get description
      Debug.Print getAttr, ": ", attrDesc
      Next getAttrtr;
End Sub
```

Related information

Property:ReadProperties, on page 398

Method:Set, on page 411

Method: Close

Purpose	Closes an order set
Syntax	`myOrderSet.Close`
Parameters	None
Return value	None
Exceptions	None
Description	The `Close` method closes an order set. Orders within the order set are unaffected. If an order in this order set also appears in another order set, the second order set still contains the order. If you enable the `EnableOrderAutoDelete` property, this method deletes the orders when it closes the order set.
Example	None
Related information	Method:Open, on page 410

Method: HoldOrder

Purpose

Places an order in the market on temporary hold

Syntax

`myOrderSet.HoldOrder(`*`order`*`, `*`doHold`*`)`

Parameters

order	Order object to hold Data type: `TTOrderObj`
doHold	Whether to hold or release the order Data type: Boolean

Return value

None

Exceptions

Exception	Meaning
Invalid Order	The order parameter does not reference a valid **`TTOrderObj`** object.
Unable to put the Order On Hold	The existing order could not be placed on hold.
Unable to Submit the Held Order	The attempt to submit the order with a HOLD status failed.

Description

The `HoldOrder` method sends a delete message to the exchange for the order, but does not remove it from the order set. Because the order remains in the order set, you can re-submit the order at a later time. When you submit the order again, the XTAPI sends the exchange a new order with the same values as the order you put on hold.

For example, suppose you submit an order for 50 ZB-Mar13 contracts and notice that 20 contracts filled. Before the order is completely filled, you need to step away from your desk so cannot monitor the order progress. You can put the order on hold until you return. The XTAPI sends a delete message for the remaining 30-contract order and leaves it in the order set. When you take the order off hold, the XTAPI submits the remaining 30 contracts as a new order.

Note: The exchanges have no concept of holding an order. Holding orders is a feature the XTAPI provides that allows you to temporarily halt trading for an order.

Example

The following example retrieves an existing order and uses the `HoldOrder` method to place a hold on the order.

```
'HoldOrder places an order on hold at the exchange
Private Sub HoldOrder()

  'the order is retrieved using the stored site order key
  Dim orderObj As XTAPI.TTOrderObj =
    m_TTOrderSet.SiteKeyLookupEx(m_strSiteOrderKey)

  'Places the order OnHold
  Me.m_TTOrderSet.HoldOrder(orderObj)

End Sub
```

Related information

Property:DeleteContracts, on page 371
Property:DeleteOrders, on page 373
Method:HoldOrders, on page 408

Method: HoldOrders

Purpose

Holds orders within an order set

Syntax

```
numToHold = myOrderSet.HoldOrders([isHold], [isBuy], [lowTick],
[highTick], [isInRange], [ordSelector])
```

Parameters

isHold	[Optional]. Whether to hold or submit the orders. Possible values include: **TRUE**: Hold orders **FALSE**: Submit orders Data type: Boolean; Default: TRUE
isBuy	[Optional]. Which type of orders to hold. Possible values include: **TRUE**: Hold buy orders **FALSE**: Hold sell orders Data type: Boolean; Default: TRUE
lowTick	[Optional]. Lowest tick price of orders for the hold range. Data type: Integer; Default: 0
highTick	[Optional]. Highest tick price of orders the hold range Data type: Integer; Default: `MAXLONG`
isInRange	[Optional]. Whether to hold orders within the specified range or orders outside the range. **TRUE**: Hold orders inside the specified range **FALSE**: Hold orders outside the specified range Data type: Boolean; Default: TRUE
ordSelector	[Optional]. Order selector that identifies the orders within the order set to hold Data type: `TTOrderSelector`

Return value

numToHold	Number of orders held or submitted Data type: Integer

Exceptions

Exception	Meaning
Invalid Selector	The specified *`ordSelector`* does not reference a valid order selector object.

Description

The `HoldOrders` method holds or submits orders that meet certain criteria from the order set. You can identify the orders based on whether:

- They are buy or sell orders
- Their prices are above or below a specific price
- Their prices fall within a specific price range

For example, you could use this method to place holds on all sell orders under 1200 ticks.

If you pass no arguments, the `HoldOrders` method places a hold on all orders in the order set.

The quantity returned in *`numToHold`* represents the number of contracts the XTAPI attempts to hold or submit, not the number actually held or submitted. When you call this property, the XTAPI determines which contracts match the specified criteria and returns the number it found and sent to the exchange to hold.

Example

The following example shows how to use the `HoldOrders` method to put a hold on all buy orders whose prices fall within a specified range.

```
'HoldOrder places all orders on hold at the exchange that fall in a range
Private Sub HoldAllOrders()

  Try

    Dim lowTick As Double
    Dim highTick As Double

    ' Obtain the tick price of the instrument.
    Dim As Double = Convert.ToDouble(Me.m_TTInstrObj.TickPrice(0, 1, "$"))

    ' Obtain the tick increment of the instrument.
    Dim tickIncrement As Integer = Me.m_TTInstrObj.Get("TickIncrement")

    'Calculate the lowTick and highTick
    lowTick = (Convert.ToDouble(lowPriceRangeTextBox.Text) / tickPrice) *
        tickIncrement
    highTick = (Convert.ToDouble(highPriceRangeTextBox.Text) / tickPrice) *
        tickIncrement

    Dim orderSelector As XTAPI.TTOrderSelector
    orderSelector = Me.m_TTOrderSet.CreateOrderSelector

    'Places the order OnHold
    Dim intHoldOrders As Integer
    intHoldOrders = Me.m_TTOrderSet.HoldOrders(1, 1, lowTick, highTick,
        1, orderSelector)

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try

End Sub
```

Related information

Property:DeleteContracts, on page 371

Property:DeleteOrders, on page 373

Method:HoldOrder, on page 406

Method: Open

Purpose	Opens an order set for trading
Syntax	`myOrderSet.Open([enableSend])`
Parameters	*enableSend* — Whether to allow orders to be submitted through this order set Data type: Boolean; Default: FALSE
Return value	None

Exceptions

Exception	Meaning
Orderset Already Open	The order set is currently open

Description

The `Open` method opens an order set for order processing and specifies whether orders can be submitted from the order set. Unless you specifically enable it, the method does not allow orders to be sent from the order set.

Example

None

Related information

Property:Count, on page 365

Property:CreateOrderSelector, on page 366

Property:EnableFillCaching, on page 376

Method:SetValidExchanges, on page 414

Method:Close, on page 405

Method: Set

Purpose

Sets values for **`TTOrderSet`** object's attributes

Syntax

`myOrderSet.Set(attrList, attrValues)`

Parameters

attrList — Comma-separated string of object attributes

Data type: String

attrValues — Array of attribute values

Data type: Array

Note: If you specify a single attribute in attrList, you can supply the value directly, without including it in an array structure.

Return value

None

Exceptions

Exception	Meaning
Manual Fill Set Failed	One or more input attribute or value is not valid.

Description

The `Set` property sets or updates the values for an arbitrary number of **`TTOrderSet`** attributes. You must supply the values with the appropriate data types.

Note: If you need to update multiple attributes, you should specify all of them in a single IsValidGet property request. Making a single request with multiple attributes provides significantly better performance than making multiple requests with single values

Some of the object attributes allow you to specify the format for price and quantity data. The XTAPI provides certain prices and quantities in several formats. When specifying an attribute that deals with prices or quantities, you can append a special character to the end of the attribute name to specify the format for the data. For example, price data can be represented in ticks or points.

Table 73 describes the different formats you can use.

Value type	Character	Description
Price	&	Ticks (integer)
	#	Points (decimal)
	$	X_TRADER display format (string)
	~	Differential for inverted tick products (string). Used for trading BrokerTec yield products, because the "$" format cannot differentiate between an actual price (LTP) or a price delta (net change).
	^	Native currency (decimal)
	@	Primary currency (decimal)

Table 73. Price and quantity value formats

Value type	Character	Description
Quantity	&	Number (integer)
	#	Number (decimal)
	$	Number (string)

Table 73. Price and quantity value formats

Table 74 lists the **`TTOrderSet`** object attributes you can supply to the `Set` property.

Attribute	Description	DataType
BuyLimits	Whether to check buy orders against buy limits	Boolean
MaxBuyOrderQty	Maximum quantity allowed in a single buy order	Integer
MaxBuyOrders	Maximum number of buy orders allowed in the order set	Integer
MaxBuyWorking	Maximum number of working orders allowed in the order set	Integer
MaxFills	Maximum number of fill records that can be included in one fill block	Integer
MaxOrderQty	Maximum quantity allowed in a single buy or sell order	Integer
MaxOrders	Maximum number of orders allowed in the order set	Integer
MaxOrderStatusUpdates	Maximum number of consecutive order updates allowed before allowing other events to fire	Integer
MaxPosition	Maximum long or short position (in ticks) allowed in the order set	Integer
MaxRejects	Maximum number of order rejection events to send in succession	Integer
MaxSellOrderQty	Maximum quantity allowed in a single sell order	Integer
MaxSellOrders	Maximum number of sell orders allowed in the order set	Integer
MaxSellWorking	Maximum number of working sell orders allowed in the order set	Integer
MaxWorking	Maximum number of working orders allowed in the order set	Integer
NetLimits	Whether the order set checks self-imposed position limits when submitting an order	Integer
SellLimits	Whether the order set checks self-imposed limits (MaxSell attributes) when submitting sell orders. Default: TRUE	Boolean
ThrowLimitErrors	How the XTAPI responds when a self-imposed limit is violated. Possible values include: **FALSE**: reduce order to self-imposed limit **TRUE**: throw an error Default: TRUE	Integer

Table 74. TTOrderSet.Set attributes

Example

The following example shows how to use the `Set` method to set a single attribute:

```
Dim myOrderSet As TTOrderSet

# code to create myOrderSet omitted
#
myOrderSet.Set("MaxOrders", 20)

# The following example shows the preferred approach to use when setting
# multiple attribute values with different data types:
```

```
Dim myOrderSet As TTOrderSet

# code to create myOrderSet omitted
#
Dim sProps As String = "MaxOrders,MaxOrderQty"
Dim aValues As Array = (20, "100")
TTOrderSet.Set(sProps, aValues)
```

Related information

Property:Get, on page 385

Property:WriteProperties, on page 404

Method: SetValidExchanges

Purpose

Limits orders and fills to specific *Exchange-Flavor*s

Syntax

`myOrderSet.SetValidExchanges(`*`exchangeList`*`, [`*`allowOrders`*`], [`*`allowFills`*`])`

Parameters

exchangeList	Comma-separated list of *Exchange-Flavor*s (such as CME, CME-B, Eurex, and LIFFE) or * for all *Exchange-Flavor*s Data type: String
allowOrders	Whether to connect to the order servers for the specified *Exchange-Flavor*s Data type: Boolean
allowFills	Whether to connect to the fill servers for the specified *Exchange-Flavor*s Data type: Boolean

Return value

None

Exceptions

Exception	Meaning
Cannot set the valid exchanges after order set is open	The order set is currently open.
Invalid Exchanges	One or more of the exchanges in `exchangeList` is not a valid *Exchange-Flavor*.

Description

The `SetValidExchanges` method determines whether the order set can connect to the order or fill servers for a set of *Exchange-Flavor*s. You can permit or deny access to the servers from this order set for one, several, or all of the *Exchange-Flavor*s. Each order set uses independent sets of exchange limits, so you can, for example, allow order access to an *Exchange-Flavor* in one order set and deny access to the same order server in another.

If your application does not use either orders or fills, you can increase the trading throughput by limiting server connections. For example, suppose you want to develop an application that focuses on submitting orders to the markets, but do not need the application to keep precise P&L information. You could use this property to stop connections to the Fill servers for the *Exchange-Flavor*s. Because the application does not receive fill information, it can submit and process more orders.

Note: You must call this method before calling the Open method for it to take effect.

Example

The following example uses the `SetValidExchanges` method to limit trading to the CME markets.

```
'SendOrder sends an order that is hardcoded as a buy market order and qty = 1
Private Sub SendOrder()
```

```
  Try
    'Create an order set
    m_TTOrderSet = New XTAPI.TTOrderSet
    'Enable the orderSet to send an order
    m_TTOrderSet.EnableOrderSend = True
    m_TTOrderSet.EnableOrderSetUpdates = True
    m_TTOrderSet.EnableOrderUpdateData = True

    'Turn off the self imposed limits
    m_TTOrderSet.Set("NetLimits", False)

    'This line prevents orders on any other exchange other than CME
    Me.m_TTOrderSet.SetValidExchanges("CME", 1, 1)

    'Open the orderset for order entry
    m_TTOrderSet.Open(1)

    'Create an orderprofile
      Dim orderProfile As XTAPI.TTOrderProfile
      orderProfile = New XTAPI.TTOrderProfile

      'set the needed parameters for the orderprofile
      orderProfile.Customer = "TTORDGS001001"
      orderProfile.Instrument = Me.m_TTInstrObj
      orderProfile.Set("BuySell", "Buy")
      orderProfile.Set("Qty", "1")
      orderProfile.Set("OrderType", "L")
      orderProfile.Set("Price", "525")

      'Create an integer to hold the results of the ordersend function.
      Dim intOrderQty As Integer

      'intOrderQty should equal the order Qty entered
      intOrderQty = Me.m_TTOrderSet.SendOrder(orderProfile)

      If (Not intOrderQty = 1) Then
        Console.WriteLine("ERROR sending order")
      End If

      'Get the last site order key for tracking
      Me.m_strSiteOrderKey = orderProfile.GetLast("SiteOrderKey")
      Console.WriteLine("ORderNumber, {0}", Me.m_strSiteOrderKey)

  Catch ex As Exception
    Console.WriteLine(ex.Message)
  End Try
End Sub
```

Related information

Property:SendOrder, on page 400

Method:Open, on page 410

Method: UpdateOrder

Purpose

Updates an existing order

Syntax

`myOrderSet.UpdateOrder(`*`ordProfile`*`, [`*`doCancelReplace`*`])`

Parameters

ordProfile	Order profile containing the updates for the order. Data type: `TTOrderProfile`
doCancelReplace	Optional. Whether to cancel and replace the order instead of changing it. Data type: Boolean

Return value

None

Exceptions

Exception	Meaning
Incomplete Profile	The specified order profile does not contain enough information to create a valid order.
Invalid Order Profile	The specified *`ordProfile`* object is not a valid **`TTOrderProfile`** object.
No Original Order To Update	The `SendOrder` property for this order set has not yet been called.
Order does not match Orderset	The order represented by *`ordProfile`* does not exist in the order set.
Unable to Cancel/Replace the Order	Replacing the order with the specified order profile would exceed the user's limits. Received when you set the **ThowLimitErrors** attribute to FALSE.
Unable to Change the Order	Changing the order with the specified order profile would exceed the user's risk limits. Received when you set the **ThowLimitErrors** attribute to FALSE.
Max Orders Limit Hit Max Order Qty Limit Hit Max Working Qty Limit Hit Max Net Orders Limit Hit Max Net Order Qty Limit Hit Max Net Working Qty Limit Hit Max Net Position Limit Hit Max Net Zero Position Limit Hit	Submitting the specified order profile would cause the trader to exceed the corresponding risk limit.

Description

The `UpdateOrder` method modifies an existing order placed through this order set. You pass in a new order profile with the updated order information. By default, the property submits the updated order as a change to an existing order. Some exchanges, however, do not support order changes and require changes to be submitted by canceling the existing order and replacing it with the new order. If you are trading products on such an exchange, you must enable the *`doCancelReplace`* parameter.

When you use the Cancel/Replace method, the exchange might create new order numbers for each changed order, which is stored in the **OrderNo** attribute for the corresponding **`TTOrderObject`**. However, the **SiteOrderKey** attribute assigned by

the XTAPI remains constant throughout the life of the order.

Example

The following example uses the `UpdateOrder` method to submit a change order.

```
'UpdateOrder changes the price and the quantity of an order by first getting
'a copy of the order that it will modify and then changing the price and
quantity.
'Lastly, it submits the change order
Private Sub UpdateOrder()

  'Get a copy of the order
  Dim orderObj As XTAPI.TTOrderObj =
      m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

  'Create an orderprofile from the order
  Dim orderProfile As XTAPI.TTOrderProfile
  orderProfile = New XTAPI.TTOrderProfile
  orderProfile = orderObj.CreateOrderProfile()

  ' Set the new price and quantity.
  orderProfile.Set("Limit$", "550")
  orderProfile.Set("Qty", "100")

  ' Update Order as change or cancel/replace (0 - change, 1 - cancel/replace).
  m_TTOrderSet.UpdateOrder(orderProfile, 0)
End Sub
```

Related information

Property:SendOrder, on page 400

Method:UpdateXTOrder, on page 418

Method: UpdateXTOrder

Purpose

Sends the new color settings to X_TRADER

Syntax

`myOrderSet.UpdateXTOrder(`*`ordProfile`*`)`

Parameters

ordProfile	Order profile containing the updates for the order Data type: `TTOrderProfile`

Return value

None

Exceptions

Exception	Meaning
Incomplete Profile	The specified order profile does not contain enough information to create a valid order.
Invalid Order Profile	The specified *`ordProfile`* object is not a valid **`TTOrderProfile`** object.
No Original Order To Update	The `SendOrder` property for this order set has not yet been called.
Order does not match Orderset	The order represented by `ordProfile` does not exist in the order set.
Unable to Change the Order	Some other error occurred.

Description

The `UpdateXTOrder` property sends the values of the **ColorPri** and **ColorSec** object attributes of an order to X_TRADER to display in the MD Trader window. If the version of X_TRADER does not support order-based colors, this method has no effect and returns no exceptions.

Note: Although this method changes only the display color, it does check the validity of the order profile. Therefore, it throws the same exceptions as the `UpdateOrder` method.

Example

The following example shows how to use the `UpdateXTOrder` method to change the primary and secondary colors displayed for an existing order in X_TRADER.

```
UpdateXTColors gets an existing order using the siteOrderKey
'that was saved to memory and then it updates the order
'giving the order colors that will show up in MD TRADER
Private Sub UpdateXTColors()

  'Get a copy of the order
  Dim orderObj As XTAPI.TTOrderObj =
    m_TTOrderSet.SiteKeyLookup(m_strSiteOrderKey)

  'Check to make sure that the order is not null
  Dim boolOrderIsNull As Boolean
  boolOrderIsNull = orderObj.IsNull

  If Not (boolOrderIsNull) Then
    Dim orderProfile As XTAPI.TTOrderProfile
    orderProfile = New XTAPI.TTOrderProfile
    orderProfile = orderObj.CreateOrderProfile()
```

```
        orderProfile.Set("ColorPri",
          System.Drawing.ColorTranslator.ToWin32(Color.Pink))
        orderProfile.Set("ColorSec",
          System.Drawing.ColorTranslator.ToWin32(Color.Purple))

        Me.m_TTOrderSet.UpdateXTOrder(orderProfile)
      End If
    End Sub
```

Related information Method:UpdateOrder, on page 418

Event: OnOrderActionRejected

Triggered when

An exchange rejects an order action

Syntax

```
Sub myOrderSet_OnOrderActionRejected(newOrderObj, oldOrderObj,
siteOrderKey, orderState, orderAction, rejectQty, orderType)
```

Parameters

newOrderObj	Order object containing the rejected order information. Data type: `TTOrderObj`
oldOrderObj	Order object containing the order information before the action was rejected. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderState	State of the order before the exchange rejected the order action. Possible values include: • `ORD_STATE_NONE`, for a new order submission • `ORD_STATE_WORKING` or `ORD_STATE_WORKING`, for an order working in the market • `ORD_STATE_UNKNOWN`, for an order on hold Additionally, if you enable verbose notifications in the `OrderStatusNotifyMode` property, the event can return other statuses for delayed update notifications. Data type: enumOrderNotifyState
orderAction	Action associated with the order rejection. Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_CHANGE` • `ORD_ACTION_DELETE` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_HOLD` • `ORD_ACTION_REPLACE` • `ORD_ACTION_UNKNOWN` Data type: `enumOrderAction`
rejectQty	Quantity of the rejected order. For working orders, the value contains the working quantity. For new or hold orders, it contains the order quantity. Data type: Integer
rejectMessage	Exchange-generated rejection message associated with the order. Data type: String

Description

The `OnOrderActionRejected` event is triggered anytime an exchange rejects an order request, including submitting new orders, updating existing orders, deleting orders, and so on.

Example	None
Related information	Property:OrderStatusNotifyMode, on page 397

Event: OnOrderDeleted

Triggered when

Orders are taken out of the market

Syntax

`Sub myOrderSet_OnOrderDeleted(`*`newOrderObj`*`, `*`oldOrderObj`*`, `*`siteOrderKey`*`, `*`orderState`*`, `*`orderAction`*`, `*`delQty`*`)`

Parameters

newOrderObj	Order object containing the order information after the exchange acknowledges the deletion. Data type: `TTOrderObj`
oldOrderObj	Order object containing order information before it was deleted. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderState	State of the order before being deleted. Possible values include: • ORD_STATE_WORKING • ORD_STATE_HOLD Data type: `enumOrderNotifyState`
orderAction	Action associated with the deleted order. Possible values include: • ORD_ACTION_DELETE, for deleted orders • ORD_ACTION_REPLACE, for orders deleted as a by-product of a cancel/replace request Data type: `enumOrderAction`
delQty	Quantity deleted for the order. Data type: Integer

Description

The `OnOrderDeleted` event is triggered when an order is deleted and is no longer working in the market. The application can receive this event for orders deleted by the `DeleteContracts` or `DeleteOrder` properties and for orders deleted as a result of changing the quantity of an existing order. If the exchange implements order changes by canceling the existing order and submitting a new order, the application receives this event with the *`orderAction`* parameter set to `ORD_ACTION_REPLACE`.

If the `OrderStatusNotifyMode` property enables verbose mode, a cancel/replace order triggers this event and an `OnOrderSubmitted` event.

The value returned in the *`delQty`* parameter depends on the fill status of the order. If the order has been partially filled, the value contains the executed quantity (**ExeQty** attribute).Otherwise, it contains the original order quantity. Also, if due to exchange timing, the XTAPI receives a full fill for the same order before it acknowledges the delete order, the *`delQty`* parameter value contains 0.

Example

None

Related information

Property:DeleteContracts, on page 371

Property:DeleteOrders, on page 373
Property:OrderStatusNotifyMode, on page 397
TTOrderObj Class, on page 279

Event: OnOrderFillBlockEnd

Triggered when	Last fill in the block is sent
Syntax	`Sub myOrderSet_OnOrderFillBlockEnd`
Parameters	None
Description	The `OnOrderFillBlockEnd` event is triggered when the last fill object in the fill block is sent. After sending this event, the XTAPI begins sending all of the events that occurred during the fill block. When your application starts downloading fills for the first time, the XTAPI sends `OnOrderFillBlockStart` and `OnOrderFillBlockEnd` events with no intervening fills.
Example	The following example displays a message on the console when a fill download completes.

```
'OnOrderFillBlockStart simply notifies your application that a block of
'fills has completed to download
Private Sub m_TTOrderSet_OnOrderFillBlockEnd() Handles
    m_TTOrderSet.OnOrderFillBlockEnd

  Console.WriteLine("OnOrderFillBlockEnd has occurred")

End Sub
```

Related information	Event:OnOrderFillBlockStart, on page 426 Event:OnOrderFillData, on page 427 Method:Set, on page 411 TTFillObj Class, on page 95

Event: OnOrderFillBlockStart

Triggered when	Block of fills are ready to send
Syntax	`Sub myOrderSet_OnOrderFillBlockStart`
Parameters	None
Description	The `OnOrderFillBlockStart` event is triggered when the XTAPI starts sending a block of cached fills. To prevent overwhelming an application during fill downloads, especially when an application initializes, the XTAPI delivers fill notifications in blocks to provide pauses your application can use to perform other tasks. After sending the `OnOrderFillBlockStart` event, the XTAPI begins sending fill events to your application and continues sending only fill events until it sends an `OnOrderFillBlockEnd` event. During this period, your application is guaranteed to receive only `OnOrderFillData` events. Each **`TTOrderSet`** object contains a **MaxFills** attribute that specifies the maximum size of a fill block. By setting this value, you can force the XTAPI to pause during large fill downloads and process other types of events, such as price updates. To set this value, call the `Set` method. When your application starts downloading fills for the first time, the XTAPI sends `OnOrderFillBlockStart` and `OnOrderFillBlockEnd` events with no intervening fills

Example	The following example displays a message on the console when a fill download starts.
	
	`'OnOrderFillBlockStart simply notifies your application that a block of`
	`'fills has completed to download`
	`Private Sub m_TTOrderSet_OnOrderFillBlockStart() Handles`
	`m_TTOrderSet.OnOrderFillBlockStart`
	
	`Console.WriteLine("OnOrderFillBlockStart has begun")`
	
	`End Sub`

Related information	Event:OnOrderFillBlockEnd, on page 425 Event:OnOrderFillData, on page 427 Method:Set, on page 411 TTFillObj Class, on page 95

Event: OnOrderFillData

Triggered when	Fills are received for an order in the order set
Syntax	`Sub myOrderSet_OnOrderFillData(`*`fillObj`*`)`

fillObj	Fill object containing the fill information Data type: `TTFillObj`

Description

The `OnOrderFillData` event is triggered when the XTAPI receives a partial or full fill message from the exchange for an order contained in the order set. The event includes a **`TTFillObj`** that contains the fill information for an order.

If your application received an `OnOrderFillBlockStart` event, the XTAPI sends an `OnOrderFillData` event for each fill in the fill block. Your application will receive no other events until the end of the fill block is signaled with an `OnOrderFillBlockEnd` event.

The XTAPI triggers this event only if the `EnableOrderFillData` property is enabled.

Example

None

Related information

Event:OnOrderFillBlockEnd, on page 425

Event:OnOrderFillBlockStart, on page 426

Event:OnOrderFilled, on page 429

Event: OnOrderFilled

Triggered when	Orders are fully or partially filled
Syntax	`Sub myOrderSet_OnOrderFilled(`*`newOrderObj`*`, `*`oldOrderObj`*`, `*`siteOrderKey`*`, `*`orderState`*`, `*`fillQty`*`)`

Parameters

newOrderObj	Order object containing the filled order information. Data type: `TTOrderObj`
oldOrderObj	Order object containing the state of the order before the fill. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderState	Status of the filled order. Possible values include: • `ORD_STATE_WORKING`, for a partial fill • `ORD_STATE_COMPLETED`, for a complete fill Data type: enumOrderNotifyState
fillQty	Fill quantity Data type: Integer

Description	The `OnOrderFilled` event is triggered when working order is completely or partially filled. The *`oldOrderObj`* and *`newOrderObj`* parameters represent the states of the order before and after the fill, respectively.
Example	None
Related information	Event:OnOrderFillData, on page 427

Event: OnOrderHeld

Triggered when	A TT Gateway receives a hold order
Syntax	`Sub myOrderSet_OnOrderHeld(`*`newOrderObj`*`, `*`oldOrderObj`*`, `*`siteOrderKey`*`, `*`orderAction`*`, `*`orderQty`*`)`

Parameters

newOrderObj	Order object containing the held order information. Data type: `TTOrderObj`
oldOrderObj	Order object containing the old order information, if applicable. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderAction	Action associated with the submitted order. Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_REPLACE` (for verbose notifications only) Data type: `enumOrderAction`
orderQty	Quantity of the held order. Data type: Integer

Description

The `OnOrderHeld` event is triggered when a TT Gateway receives and acknowledges a new order with it status set to HOLD or when a working order removed from the market and placed on hold at the TT Gateway.

This event returns the order submitted (*`newOrderObj`*) in all cases, and in most cases, returns the oldOrderObj with its **IsNull** attribute enabled. It returns a complete order object when the following conditions exist:

- An order with the same site order key exists
- The **`TTOrderSet`** `OrderStatusNofifyMode` property enables verbose notifications

Example

None

Related information

Method:HoldOrder, on page 406

Method:HoldOrders, on page 408

Event: OnOrderInstrumentChanged

Triggered when	Changes occur in an order's instrument
Syntax	`Sub myOrderSet_OnOrderInstrumentChanged(`*`orderObj`*`, `*`instrObj`*`, `*`siteOrderKey`*`, `*`instrName`*`)`
Parameters	*orderObj* — Order object representing the order. Data type: `TTOrderObj` *instrObj* — Instrument object that changed. Data type: `TTInstrObj` *siteOrderKey* — Site order key associated with the order. Data type: String *instrName* — Name of the updated instrument. Data type: String
Description	The `OnOrderInstrumentChanged` event is triggered when the instrument associated with an order changes. You can use this event to notify the application when the XTAPI resolves a contract name associated with a given order, for spread and strategy contracts, and when name resolution occurs after receiving the price table data for a product (contract).
Example	None
Related information	Property:OrderStatusNotifyMode, on page 397 TTInstrObj Class, on page 217 TTOrderObj Class, on page 279

Event: OnOrderRejected

Triggered when

TT Gateways or exchanges reject an order in the order set

Syntax

`Sub myOrderSet_OnOrderRejected(orderObj)`

Parameters

orderObj	Order object for the rejected order Data type: `TTOrderObj`

Description

The `OnOrderRejected` event is triggered when the XTAPI receives an order rejection message from the exchange for an order contained in the order set. The event includes a **`TTOrderObj`** object corresponding to the rejected order.

The XTAPI triggers this event only if the `EnableOrderRejectData` property is enabled.

Submitted orders can fail or be rejected at the XTAPI, X_TRADER, TT Gateway, or exchange level. If so:

- The program will be notified about X_TRADER and XTAPI related issues and a message might appear in the X_TRADER audit trail.
- Gateway-related issues are reported in the X_TRADER Audit Trail, but are not reported to the XTAPI application. Instead, the order times out.
- Exchange related issues are reported in the X_TRADER Audit Trail. Impacted Order sets are "informed" of the issue.

Example

The following example displays a message when the exchange rejects an order.

```
'The OnOrderRejected event is triggered if the order fails at the Exchange level
'EnableOrderRejectData must be enabled in the orderset for this event
'to work.
Private Sub m_TTOrderSet_OnOrderRejected(ByVal pRejectedOrderObj As
        XTAPI.TTOrderObj) Handles m_TTOrderSet.OnOrderRejected

  Console.WriteLine("Order Was rejected by the Exchange")
  Console.WriteLine(pRejectedOrderObj.Get("SiteOrderKey"))

End Sub
```

Related information

Property:EnableOrderRejectData, on page 379

Property:OrderStatusNotifyMode, on page 397

Event:OnOrderSubmitted, on page 435

TTOrderObj Class, on page 279

Event: OnOrderSetUpdate

Triggered when

Changes occur to an order set or to its associated orders and fills

Syntax

`Sub myOrderSet_OnOrderSetUpdate(`*orderSet*`)`

Parameters

orderSet	Pointer to the order set that updated Data type: `TTOrderSet`

Description

The `OnOrderSetUpdate` event is triggered when a change, such as an order delete, partial fill, or object attribute, occurs for the order set.

Example

The following example displays information about an order when it updates.

```
'The OnOrderSetUpdate event is where all of the processing of the Order
'Objects takes place inside the XTAPI.  If there is any update to an
'order then the Order will be sent to the TTOrderTracker object for extraction
by
'the application inside the while loop demonstrated below.
Private Sub m_TTOrderSet_OnOrderSetUpdate(ByVal pOrderSet As XTAPI.TTOrderSet)
        Handles m_TTOrderSet.OnOrderSetUpdate

  Dim ttOrderTrackerObj As XTAPI.TTOrderTrackerObj
  ttOrderTrackerObj = New XTAPI.TTOrderTrackerObj

  'Get the next order tracker object from the orderset
  ttOrderTrackerObj = pOrderSet.NextOrderTracker

  'Use a while loop to iterate through the updated order objects
  While Not ttOrderTrackerObj Is Nothing

    If ttOrderTrackerObj.HasOldOrder <> 0 Then
      Console.WriteLine("Has old order " +
        ttOrderTrackerObj.OldOrder.Get("OrderNo"))
      Console.WriteLine("Has old order " +
        ttOrderTrackerObj.OldOrder.Get("OrderType"))
    End If

    If ttOrderTrackerObj.HasNewOrder <> 0 Then
      Console.WriteLine("Has new order " +
        ttOrderTrackerObj.NewOrder.Get("OrderNo"))
      Console.WriteLine("Has new order " +
        ttOrderTrackerObj.NewOrder.Get("OrderType"))
    End If

    'Get the next order tracker
    ttOrderTrackerObj = pOrderSet.NextOrderTracker

  End While
End Sub
```

Related information

Property:EnableOrderSetUpdates, on page 381

Method:UpdateOrder, on page 416

Event: OnOrderSubmitted

Triggered when

New order is submitted to the exchange or a held order is resubmitted

Syntax

```
Sub myOrderSet_OnOrderSubmitted(newOrderObj, oldOrderObj,
siteOrderKey, orderAction, workQty, orderType, orderTraits)
```

Parameters

newOrderObj	Order object containing the newly submitted order information. Data type: `TTOrderObj`
oldOrderObj	Order object containing the held order information, when applicable. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderAction	Action associated with the submitted order. Possible values include: • `ORD_ACTION_ADD` • `ORD_ACTION_SUBMIT` • `ORD_ACTION_REPLACE` (for verbose notifications only) Data type: `enumOrderAction`
wrkQty	Working quantity for the order. Data type: Integer
orderType	XTAPI order type associated with the order. Possible values include: **B** (OCO): OCO order **C** (CMO): CMO order **L** (Limit): Limit order **M** (Mkt): Market order **O** (BatchOrder): Batch order **Q** (Quote): Quote order **S** (BL): Best Limit order (TOCOM only) **T** (MTL): Market To Limit order **X** (Cross): Cross order The text in parenthesis indicates the value associated with the $ suffix. Data type: String
orderTraits	Traits associated with the order as a concatenated string containing the order status history, order flag, and order restriction. Data type: String

Description

The `OnOrderSubmitted` event is triggered for any of the following conditions:

- When a TT Gateway receives a new order
- When an order currently in Hold status is resubmitted to a TT Gateway

- When a TT Gateway receives a new order as part of a Cancel/Replace transaction

> **Note:** Your application receives the event for this condition only if you enable verbose mode in the **`TTOrderSet`** `OrderStatusNotifyMode` property.

This event returns the order submitted (*`newOrderObj`*) in all cases, and in most cases, returns the *`oldOrderObj`* with its IsNull attribute enabled. It returns a complete order object when the following conditions exist:

- An order with the same site order key exists
- The **`TTOrderSet`** `OrderStatusNofifyMode` property enables verbose notifications

In certain circumstances, the XTAPI might receive notification that the order was fully filled before it receives the OK accepting the order (or the OK might already reflect the fact that the order has been filled). In this case, the XTAPI application receives the OnOrderFilled event. When the application receives the `OnOrderSubmitted` event, the working quantity contains 0, as the order is already filled. This case only occurs when the XTAPI receives a fully-filled antitraditional before it receives the OK for the order.

Additionally, the working quantity represents the quantity submitted to the exchange, not necessarily the quantity the trader attempted to submit. For example, if the trader submits an order for 20, but risk limits automatically reduced that quantity to 15, the `OnOrderSubmitted` **workQty** value contains 15.

Example

None

Related information

Property:OrderStatusNotifyMode, on page 397

Event:OnOrderRejected, on page 433

Event:OnOrderUpdated, on page 437

Event: OnOrderUpdated

Triggered when	Existing orders change
Syntax	`Sub myOrderSet_OnOrderUpdated(newOrderObj, oldOrderObj, siteOrderKey, orderState, orderAction, updQty, orderType, orderTraits)`

Parameters

newOrderObj	Order object containing the updated submitted order information. Data type: `TTOrderObj`
oldOrderObj	Order object containing the order information before the latest changes. Data type: `TTOrderObj`
siteOrderKey	Unique ID associated with the order during its lifetime. Data type: String
orderState	State of the updated order. Possible values include: • `ORD_STATE_WORKING` • `ORD_STATE_HOLD` Additionally, if you enable verbose notifications in the `OrderStatusNotifyMode` property, the event can return other statuses for delayed update notifications. Data type: enumOrderNotifyState
orderAction	Action associated with the submitted order. Possible values include: • `ORD_ACTION_CHANGE` • `ORD_ACTION_REPLACE` (on Cancel/Replace orders) Data type: enumOrderAction
updQty	Working quantity if the updated order was in working state or the order quantity if the order is updated while being held. Data type: Integer
orderType	XTAPI order type associated with the order. Possible values include: **B** (OCO): OCO order **C** (CMO): CMO order **L** (Limit): Limit order **M** (Mkt): Market order **O** (BatchOrder): Batch order **Q** (Quote): Quote order **S** (BL): Best Limit order (TOCOM only) **T** (MTL): Market To Limit order **X** (Cross): Cross order The text in parenthesis indicates the value associated with the $ suffix. Data type: String

orderTraits	Traits associated with the order as a concatenated string containing the order status history, order flag, and order restriction. Data type: String

Description

The `OnOrderUpdated` event is triggered when any aspect of the order changes at the exchange. For example, if the working quantity (**Wrk** attribute) changes, the XTAPI triggers this event; but if the **FFT2** attribute changes, it does not trigger this event.

If the OrderStatusNotifyMode property enables verbose mode, a cancel/replace order triggers the `OnOrderDeleted` and `OnOrderSubmitted` events instead.

Example

None

Related information

Property:OrderStatusNotifyMode, on page 397

Event:OnOrderSubmitted, on page 435

21 TTOrderTrackerObj Class

Chapter overview

The chapter describes the **`TTOrderTrackerObj`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTOrderTrackerObj class overview	440
Property:GetChanged	442
Property:GetIsChanged	443
Property:HasNewOrder	445
Property:HasOldOrder	446
Property:NewOrder	447
Property:OldOrder	449

TTOrderTrackerObj class overview

Overview

The XTAPI maintains a chain of **`TTOrderObj`** objects that represent the state changes during the life of an order. This chain of order objects comprises all updates associated with an order, such as fills, modified prices or quantities, and cancellations. You can use the **`TTOrderTrackerObj`** class when you want to keep track of each and every change for each and every order in the order set.

How order tracking works

When you use order tracking, each **`TTOrderSet`** keeps a list of **`TTOrderTrackerObj`** objects. Each tracker object contains two consecutive snapshots of an order. Each order state is represented by a **`TTOrderObj`** object, as shown in Figure 75. In this illustration, the order tracker represents two consecutive states of an order working in the market. The old order represents the state of the order before the next update, while the new order represents the state of the order after the update.

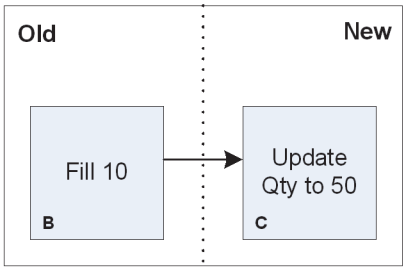


Figure 75. TTOrderTrackerObj components

During the lifetime of an order, the XTAPI uses the **`TTOrderTrackerObj`** object to track consecutive changes in an order, as shown in Figure 76. The figure shows a sample sell order with four different updates, which uses five **`TTOrderTrackerObj`** objects to track the progress.

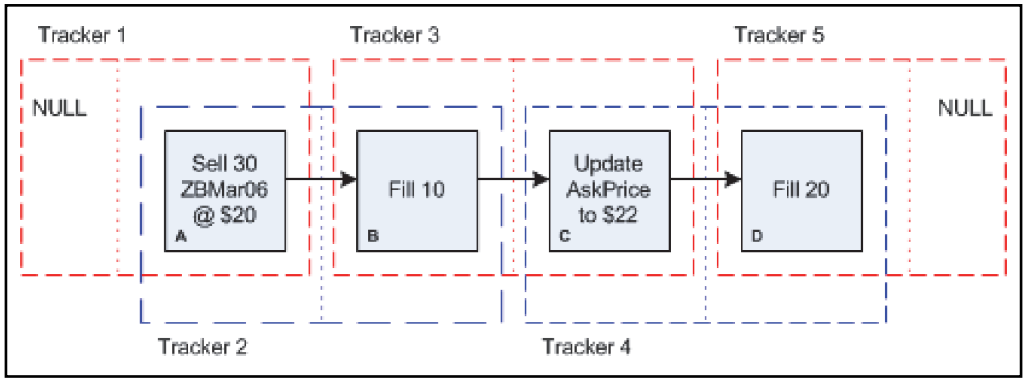


Figure 76. Order trackers in a working order

Based on what each half of an order tracker contains, you can determine where in the order lifetime a specific order update represents. Table 77 shows you what the old and new order halves of the order tracker tell you about the progress of the order.

Old	New	Meaning
N	N	Not applicable
N	Y	Initial order was received
Y	N	Order was filled, deleted, or rejected
Y	Y	Working order was updated

Table 77. TTOrderTrackerObj order states

Note: If you track an order being traded through X_TRADER, the tracker object records intermediate states indicating when an order change was sent to the exchange and a reply received from the exchange.

Tracking all changes in an order set

By default, order sets store all order state changes, but do not incur the overhead of managing **`TTOrderTrackerObj`** objects. When you instantiate a **`TTOrderTrackerObj`** object, the XTAPI starts creating the tracker objects to store the consecutive state changes.

Each order set contains a single list of **`TTOrderTrackerObj`** objects that tracks changes to all orders in the order set. Therefore, the list can contain **`TTOrderTrackerObj`** objects for different orders in the order set. However, the XTAPI guarantees that the old and new orders within a single tracker object represent the same **`TTOrderObj`** object.

For example, suppose your order set contains three different working orders that receive updates. When you retrieve an order tracker for the order set, the XTAPI returns the first **`TTOrderTrackerObj`** object in the list. When you get the next tracker object from the list, the tracker object could contain another pair of order updates for the same order or for a different order.

Properties

Table 78 lists the **`TTOrderTrackerObj`** properties.

Property	Description
GetChanged	Returns updated object attributes for an order
GetIsChanged	Indicates whether specified object attribute values changed between order states
HasNewOrder	Indicates whether the order tracker has a new order
HasOldOrder	Indicates whether the order tracker has an old order
NewOrder	Returns the order object for the new order state in the tracker
OldOrder	Returns the order object for the old order state in the tracker

Table 78. `TTOrderTrackerObj` properties

Methods

None

Events

None

Property: GetChanged

Purpose

Returns updated object attributes for an order

Syntax

`attrValues = myOrderTracker.GetChanged(attrList)`

Parameters

attrList	Comma-separated list of object attributes Data type: String

Return value

attrValues	Array of attribute values Data type: Array

Exceptions

None

Description

The `GetChanged` property returns the current snapshot of an order from the order set. It returns a pointer to the order object at the tail of the order chain. You can request the order either by the order number or by the index of the order in the order set.

Example

The following example shows how to get an order based on its order number.

```
' Get the next TTOrderTrackerObj in the list for this order set
Set myOrderTracker = myOrdSet.NextOrderTracker
' Check to see if the list was empty
If Not myOrderTracker Is Nothing Then
   ' Find out which fields have changed between the 2 TTOrderObjs
   If myOrderTracker.HasNewOrder Then
      Dim data As Variant
      data = myOrderTracker.GetChanged("OrdAction, ExecutionType, Price, Qty,_
                                        Acct")
      ' Print the changed values
     ' If the entry in the array is not empty, then it will contain the new value
      Dim changedString As String
      changedString = ""
      Dim i As Integer
      For i = 0 To UBound(data)
         If Not IsEmpty(data(i)) Then
            If Not changedString = "" Then
            changedString = changedString & ", "
            End If
            changedString = changedString & data(i)
         End If
      Next
      If Not changedString = "" Then
         Debug.Print "New values: " & changedString
      End If
   End If
End If
```

Related information

Property:GetIsChanged, on page 443

Property:HasNewOrder, on page 445

Property:OldOrder, on page 449

Property: GetIsChanged

Purpose

Indicates whether specified object attribute values changed between order states

Syntax

changedArr = myOrderTracker.GetIsChanged(*attrList*)

Parameters

attrList	Comma-separated list of object attributes Data type: String

Return value

changedArr	Array of Boolean values corresponding to the list of object attributes Data type: Array

Exceptions

Exception	Meaning
Invalid Parameters	One or more of the specified attributes in the *attrList* parameter is not a valid order attribute.

Description

The GetIsChanged property compares the values of the specified object attributes after an order update. The property returns an array of Boolean values corresponding, in order, to the specified attributes. If an attribute value changed, the array element contains **TRUE**; otherwise, it contains **FALSE**.

Example

The following example shows how to use the GetIsChanged property to determine whether specific object attributes changed when an order updated.

```
'The OnOrderSetUpdate event is where all of the processing of the Order
'Objects takes place inside the XTAPI.  If there is any update to an
'order then the Order will be sent to the TTOrderTracker object for extraction
by
'the application inside the while loop.
Private Sub m_TTOrderSet_OnOrderSetUpdate(ByVal pOrderSet As XTAPI.TTOrderSet)
     Handles m_TTOrderSet.OnOrderSetUpdate

  Dim ttOrderTrackerObj As XTAPI.TTOrderTrackerObj

  ttOrderTrackerObj = New XTAPI.TTOrderTrackerObj

  'Get the next order tracker object from the orderset
  ttOrderTrackerObj = pOrderSet.NextOrderTracker

  'Use a while loop to iterate through the updated order objects
  While Not ttOrderTrackerObj Is Nothing

    If ttOrderTrackerObj.HasOldOrder <> 0 Then
      'Get the boolean values 0 or 1 that alert if the data 'has changed or not
      Dim dataArray As Array
      dataArray = ttOrderTrackerObj.GetIsChanged("OrdAction, ExecutionType")

      Dim strName As String

      For Each strName In dataArray
          Console.WriteLine("Changed Data: " + strName)
```

```
    Next

    Console.WriteLine("Has old order " +
        ttOrderTrackerObj.OldOrder.Get("OrderNo"))

    Console.WriteLine("Has old order " +
        ttOrderTrackerObj.OldOrder.Get("OrderType"))
  End If

  If ttOrderTrackerObj.HasNewOrder <> 0 Then
    'Get the boolean values 0 or 1 that alert if the data has changed or not
    Dim dataArray As Array
    dataArray = ttOrderTrackerObj.GetIsChanged("OrdAction, ExecutionType")

    Dim strName As String

    For Each strName In dataArray
        Console.WriteLine("Changed Data: " + strName)
    Next

    Console.WriteLine("Has new order " +
      ttOrderTrackerObj.NewOrder.Get("OrderNo"))

    Console.WriteLine("Has new order " +
      ttOrderTrackerObj.NewOrder.Get("OrderType"))
  End If

  'Get the next order tracker
  ttOrderTrackerObj = pOrderSet.NextOrderTracker

 End While
End Sub
```

Related information Property:GetChanged, on page 442

Property: HasNewOrder

Purpose

Indicates whether the order tracker has a new order

Syntax

```
hasNew = myOrderTracker.HasNewOrder
```

Parameters

None

Return value

hasNew	Whether the tracker contains a new order state Data type: Boolean

Exceptions

None

Description

The `HasNewOrder` property indicates whether the order tracker object includes a new state for the order. If it contains a new order state, you can call the `NewOrder` property to retrieve the new order for processing.

Example

The following Visual Basic code illustrates how to determine if an order has been filled or deleted. It uses the `HasNewOrder` property to determine whether tracker object represents the last state of the order.

```
If Not myOrderTracker.HasNewOrder Then
   Set myOldOrder = myOrderTracker.myOldOrder
   data = myOldOrder.Get("ExecutionType, WrkQty, OrderNo")
   If data(0) = 1 Then            ' Result of a response from the gateway
      If data(1) = 0 Then         ' Working quantity of 0
        Debug.Print "Order " & data(2) & " has been removed from the order set"
      End If
   End If
End If
```

Related information

Property:GetChanged, on page 442

Property:HasOldOrder, on page 446

Property:NewOrder, on page 447

Property: HasOldOrder

Purpose	Indicates whether the order tracker has an old order
Syntax	`hasOld = myOrderTracker.HasOldOrder`
Parameters	None
Return value	*hasOld* — Whether the tracker contains an old order state Data type: Boolean
Exceptions	None
Description	The `HasOldOrder` property indicates whether the order tracker object includes an old state for the order. If it contains an old order state, you can call the `OldOrder` property to retrieve the old order for processing.
Example	The following Visual Basic code illustrates how to determine if an order has been filled or deleted. It uses the `HasOldOrder` property to determine whether tracker object represents the last state of the order.

```
' Using the TTOrderTrackerObj defined previously
Dim myOldOrder, myNewOrder As XTAPI.TTOrderObj
Dim fields, oldString, newString As String
' Fields
fields = "OrdStatus, OrdAction, Price, Qty, Acct, OrderNo, SiteOrderKey"
If myOrderTracker.HasOldOrder Then
   Set myOldOrder = myOrderTracker.OldOrder
   data = myOldOrder.Get(fields)
   For i = 0 To UBound(data)
      oldString = oldString & data(i)
      If Not i = UBound(data) Then
         oldString = oldString & " | "
      End If
   Next
End If

Debug.Print oldString
```

Related information

Property:GetChanged, on page 442

Property:HasNewOrder, on page 445

Property:OldOrder, on page 449

Property: NewOrder

Purpose

Returns the order object for the new order state in the tracker

Syntax

```
orderObj = myOrderTracker.NewOrder
```

Parameters

None

Return value

orderObj — Order object for new order state

Data type: `TTOrderObj`

Exceptions

Exception	Meaning
Could not create order	The XTAPI encountered some internal error that prevented it from creating a valid **`TTOrderObj`**.

Description

The `NewOrder` property returns an order object that represents the state of the current order after the latest change. If the order tracker contains no new order, this property returns a **`TTOrderObj`** with its `IsNull` property set to **TRUE**.

Example

The following example uses the `NewOrder` property to access the **OrderNo** and **OrderType** attribute values after the latest change.

```
'The OnOrderSetUpdate event is where all of the processing of the Order
'Objects takes place inside the XTAPI.  If there is any update to an
'order then the Order will be sent to the TTOrderTracker object for
'extraction by the application inside of the while loop.
Private Sub m_TTOrderSet_OnOrderSetUpdate(ByVal pOrderSet As XTAPI.TTOrderSet)
     Handles m_TTOrderSet.OnOrderSetUpdate

  Dim ttOrderTrackerObj As XTAPI.TTOrderTrackerObj

  ttOrderTrackerObj = New XTAPI.TTOrderTrackerObj

  'Get the next order tracker object from the orderset
  ttOrderTrackerObj = pOrderSet.NextOrderTracker

  'Use a while loop to iterate through the updated order objects
  While Not ttOrderTrackerObj Is Nothing

    If ttOrderTrackerObj.HasNewOrder <> 0 Then
      Console.WriteLine("Has new order " +
           ttOrderTrackerObj.NewOrder.Get("OrderNo"))
      Console.WriteLine("Has new order " +
           ttOrderTrackerObj.NewOrder.Get("OrderType"))
    End If

    'Get the next order tracker
    ttOrderTrackerObj = pOrderSet.NextOrderTracker

  End While
End Sub
```

Related information

Property:HasNewOrder, on page 445

Property:OldOrder, on page 449

Property: OldOrder

Purpose

Returns the order object for the old order state in the tracker

Syntax

```
orderObj = myOrderTracker.OldOrder
```

Parameters

None

Return value

orderObj	Order object for old order state Data type: `TTOrderObj`

Exceptions

Exception	Meaning
Could not create order	The XTAPI encountered some internal error that prevented it from creating a valid **`TTOrderObj`** object.

Description

The `OldOrder` property returns an order object that represents the state of the current order prior to the latest change. If the order tracker contains no old order, this property returns a **`TTOrderObj`** with its `IsNull` property set to TRUE.

Example

The following example uses the `OldOrder` property to access the **OrderNo** and **OrderType** attribute values after the latest change.

```
'The OnOrderSetUpdate event is where all of the processing of the Order
'Objects takes place inside the XTAPI.  If there is any update to an
'order then the Order will be sent to the TTOrderTracker object for
'extraction by the application inside of the while loop.
Private Sub m_TTOrderSet_OnOrderSetUpdate(ByVal pOrderSet As XTAPI.TTOrderSet)
     Handles m_TTOrderSet.OnOrderSetUpdate

  Dim ttOrderTrackerObj As XTAPI.TTOrderTrackerObj

  ttOrderTrackerObj = New XTAPI.TTOrderTrackerObj

  'Get the next order tracker object from the orderset
  ttOrderTrackerObj = pOrderSet.NextOrderTracker

  'Use a while loop to iterate through the updated order objects
  While Not ttOrderTrackerObj Is Nothing

    If ttOrderTrackerObj.HasOldOrder <> 0 Then
      Console.WriteLine("Has old order " +
           ttOrderTrackerObj.OldOrder.Get("OrderNo"))
      Console.WriteLine("Has old order " +
           ttOrderTrackerObj.OldOrder.Get("OrderType"))
    End If

    'Get the next order tracker
    ttOrderTrackerObj = pOrderSet.NextOrderTracker

  End While
End Sub
```

Related information

Property:HasOldOrder, on page 446

Property:NewOrder, on page 447

22 MEAPI Namespace

Chapter overview

This chapter provides a brief overview of the MEAPI namespace and its classes.

In this chapter

Section	Page
About the Market Explorer API	452
MEAPI classes	454

About the Market Explorer API

Overview

The Market Explorer API (MEAPI) provides direct access to product information for TT Gateways. The classes within the MEAPI encapsulate TT Gateways and their respective product and contract offerings. Using these classes you can:

- Discover and monitor the status of TT Gateways
- Download available products and product types from each TT Gateway
- Find the contracts currently trading for each product

With the MEAPI, you can create applications to provide traders with updated, real-time lists of tradable products, without having to use X_TRADER. MEAPI applications typically serve as a front ends for XTAPI applications. For example, a MEAPI application could connect to a TT Gateway and download all BOND products and display them in an application window. A trader could then select a product to find the available contracts. The trader could then use XTAPI functionality to place orders and monitor fills.

Because XTAPI and MEAPI are separate libraries, you can create separate applications that communicate with each other or combine the functionality of both into a single application.

Data flow

Figure 79 illustrates how MEAPI applications communicate with TT Gateways to retrieve product information and TT Gateway server status updates.

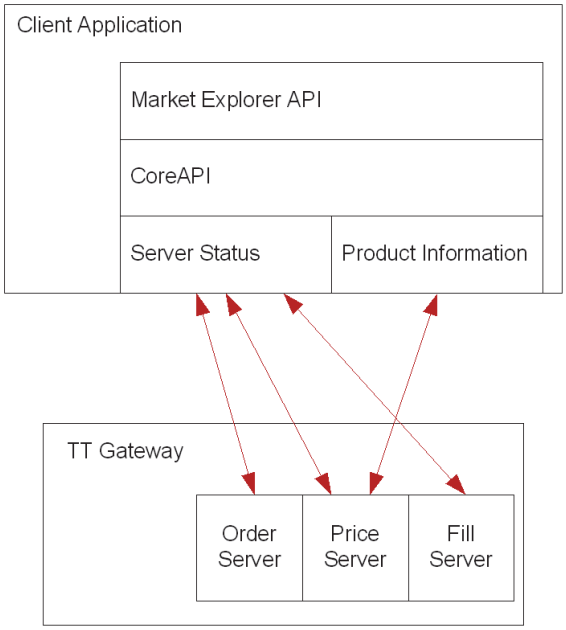


Figure 79. MEAPI data flow

Relationship between XTAPI and MEAPI applications

Figure 80 shows how MEAPI and XTAPI can exchange data. The XTAPI `TTInstrObj` object contains properties that an MEAPI objects can use to transfer product information. For example, you might use MEAPI objects to allow a trader to search for options to trade, then populate an XTAPI `TTInstrObj` object with the appropriate product information. The application could then open the instrument to request prices and submit orders for the option.

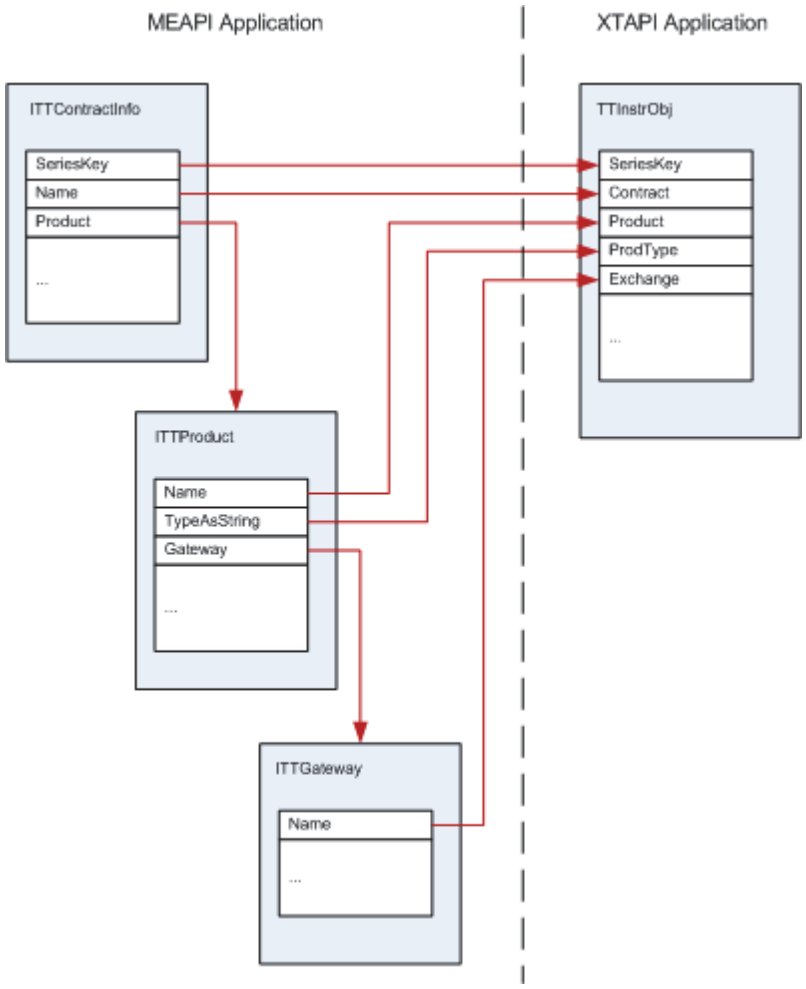


Figure 80. Relationships XTAPI and MEAPI applications

MEAPI classes

Available classes

Table 81 lists the classes available in the MEAPI namespace.

Class	Description
ITTContractCollection	Stores the contracts associated with a particular product
ITTContractInfo	Stores information about a specific contract
ITTGateway	Stores information about a TT Gateway
ITTGatewayEvents	Manages events generated by TT Gateways
ITTMarketExplorerEvents	Manages events generated by TTMarketExplorer objects
ITTProduct	Stores information about a specific product trading on a TT Gateway
ITTProductCollection	Stores the products associated with a TT Gateway
ITTProuctEvents	Manages events generated by products
TTMarketExplorer	Represents the top-level MEAPI object

Table 81. MEAPI classes

Class relationships

Figure 82 illustrates the functional representation between the different classes to help you understand how the corresponding objects interact with each other. Please note that the illustration does not imply any hierarchical structure or inheritance between the objects. The solid lines represent direct relationships, while the dashed lines represent indirect relationships. For example, an **`ITTProduct`** object has a direct relationship with an **`ITTProductEvents`** object and an indirect relationship with an **`ITTContractCollection`** object.

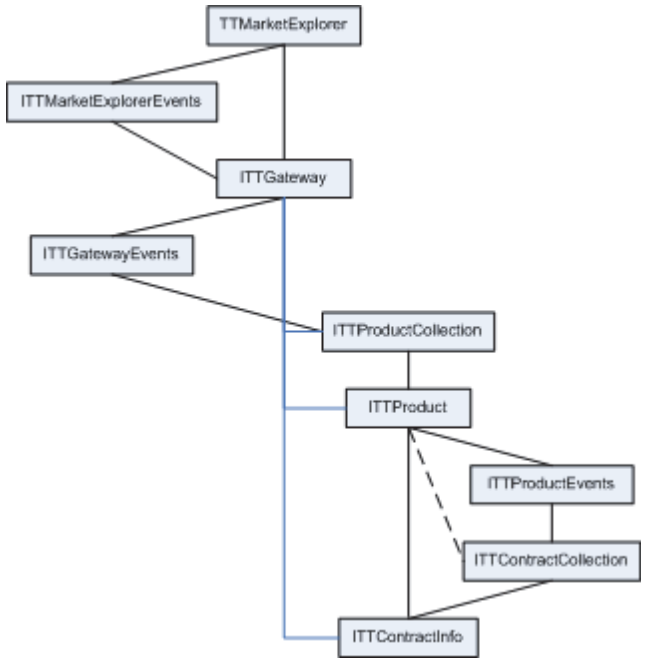


Figure 82. MEAPI class relationships

Class function types

Each class defines a set of functions you can use to create, query, or update instantiated objects and to execute code when certain events occur for an object. To help identify the basic role of the functions, the MEAPI categorizes the

functions into three areas: properties, methods, and events. Table 83 describes these function types.

Function type	Description
Property	Functions that return information about an object
Method	Functions that create or modify an object and its attributes
Event	Functions that are executed when object-specific events occur

Table 83. Class function types

About Boolean values

The MEAPI uses properties that function logically as Boolean values. These properties convert all incoming parameters into expected data types. The MEAPI conforms to the following data type conversion for non-Boolean incoming values.

Incoming value	Converted value
0	False
<>0	True
Non-integer	Undefined

Thus, if an MEAPI application expects a Boolean value for one of its parameters, it interprets any non-Boolean incoming values in Boolean terms. When the MEAPI returns values for attributes marked as `VARIANT_BOOL`; it uses 0 for FALSE and a non-zero integer for TRUE. Your application can convert these values to the Boolean representations supported by the application programming language.

The MEAPI converts data types according to the `VariantChangeType` Windows API.

About array indexing

Arrays returned from MEAPI properties, methods, and arrays use a 1-based index.

Generic data type conventions used in this guide

Because different programming languages use different data types to represent similar data, this guide uses generic terms to represent the type of data represented in parameters. For example, an integer value might declared as an int, a long, etc., depending on which programming language you use. Rather than list all the different data types for common types of information, this guide uses the generic terms described in Table 84.

Term used	Type of data
Integer	32-bit integer value (i.e. int, short, long)
Decimal	Decimal value (i.e. float, double, real)
String	Characters (i.e. string, char *)
Array	List of values of similar or different data types (i.e. array, linked list)

Table 84. Generic data type conventions

Error handling

The MEAPI classes do not throw exceptions when errors occur. Rather, the MEAPI returns the status as a return value for each property, method, or event. Based on their purposes, function return one of the following values:

- Requested information, for functions that return data if successful
- TRUE, for success when the function does not normally return information
- Error code in the form of an enumeration that identifies the error

23 ITTContractCollection Class

Chapter overview

The chapter describes the `ITTContractCollection` class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTContractCollection class overview	458
Property:Count	459
Property:Item	460

ITTContractCollection class overview

Overview

The **ITTContractCollection** class contains a collection of contract (**ITTContractInfo**) objects. This class allows you to access contracts associated with a product (**ITTProduct** object). The MEAPI returns this object as a parameter in the **ITTProduct** OnContracts event.

Properties

Table 85 lists the **ITTContractCollection** properties.

Property	Description
Count	Returns the number of contracts contained in the collection
Item	Returns the contract at the specified index in a collection

Table 85. ITTContractCollection properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of contracts contained in the collection
Syntax	`contractCount = myContractCol.Count`
Parameters	None
Return value	*contractCount* — Number of **`ITTContractInfo`** objects in the collection Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`ITTContractInfo`** objects contained in the collection.
Example	None
Related information	Property:Item, on page 460

Property: Item

Purpose	Returns the contract at the specified index in a collection
Syntax	`contract = myContractCol.Item(index)`
Parameters	*index* — Position in the collection of the desired instrument for the contract Data type: Integer or String
Return value	*contract* — Contract at the specified location Data type: `ITTContractInfo`
Description	The `Item` property returns an instrument from the **`ITTContractCollection`** collection. You can request an instrument either by specifying the index position in the collection. **Note:** The **InstrCollection** collection uses a 1-based index.
Example	None
Related information	Property:Count, on page 459 **`ITTContractInfo`** ITTContractInfo Class, on page 461

24 ITTContractInfo Class

Chapter overview

The chapter describes the **`ITTContractInfo`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTContractInfo class overview	462
Property:CallPut	463
Property:ComboCode	464
Property:Currency	465
Property:Decimals	466
Property:ExpirationDate	467
Property:LegInfo	468
Property:Name	469
Property:Precision	470
Property:PriceDisplayType	472
Property:PriceDisplayFormat	471
Property:Product	473
Property:SeriesKey	474
Property:SeriesName	475
Property:Strike	476
Property:TickPriceDenominator	477
Property:TickPriceNumerator	478
Property:TickSize	479
Method:CopyToClipboard	480
Method:ToXML	481

ITTContractInfo class overview

Overview

The `ITTContractInfo` class contains the properties associated with a contract, such as its name, currency, ticking characteristics.

Properties

Table 86 lists the `ITTContractInfo` properties.

Property	Description
CallPut	Indicates whether an option is a Call or a Put
ComboCode	Returns the series combination code for the contract
Currency	Returns the currency code associated with the contract
Decimals	Returns the contract's decimal values
ExpirationDate	Returns the contract's expiration date
LegInfo	Returns the leg information for spreads and strategies
Name	Returns the contract name
Precision	Returns the contract precision
PriceDisplayFormat	Returns a contract's price display format
PriceDisplayType	Returns the contract's price display type
Product	Returns the product object associated with the contract
SeriesKey	Returns the contract series key
SeriesName	Returns the series name for the contract
Strike	Returns an option's strike price
TickPriceDenominator	Returns the contract's tick price denominator
TickPriceNumerator	Returns the contract's tick price numerator
TickSize	Returns the contract's tick size

Table 86. ITTContractInfo properties

Methods

Table 86 lists the `ITTContractInfo` methods.

Property	Description
CopyToClipboard	Copies the contract information to the Windows clipboard
ToXML	Returns the current contract information as XML

Table 87. ITTContractInfo methods

Events

None

Property: CallPut

Purpose	Indicates whether an option is a Call or a Put
Syntax	`optionType = myContract.CallPut`
Parameters	None
Return value	*optionType* — Type of options contract. Possible values include: • Call • Put Data type: String
Exceptions	None
Description	For options contracts, the `CallPut` property returns whether the contract represents a Call or Put option. If the **`ITTContractInfo`** object does not correspond to an options contract, the property returns a NULL string.
Example	None
Related information	Property:Strike, on page 476

Property: ComboCode

Purpose	Returns the series combination code for the contract
Syntax	`cmbCode = myContract.ComboCode`
Parameters	None
Return value	*cmbCode* — Combo code associated with the contract Data type: `enumComboCode`
Exceptions	None
Description	The `ComboCode` property returns a string corresponding to the contract's combo code. For a complete listing of potential combination codes, refer to the section called ComboCode in Appendix B, MEAPI Types and Enumerations
Example	None
Related information	Property:SeriesName, on page 475 ComboCode, on page 560

Property: Currency

Purpose	Returns the currency code associated with the contract
Syntax	`currCode = myContract.Currency`
Parameters	None
Return value	*currCode* — ISO-standard symbol for the source and target currencies. Possible values include: • ARS • CHF • HKD • PTE • ATS • DEM • INR • RUR • AUD • ESP • ITL • SGD • BEF • EUR • JPY • USD • BRL • FRF • KRW • XBA • CAD • GBP • NLG • XEU Data type: String
Exceptions	None
Description	The `Currency` property returns the symbol representing the base currency in which the contract trades. The MEAPI retrieves the currencies from the Guardian currency table.
Example	None
Related information	Property:Decimals, on page 466

Property: Decimals

Purpose	Returns the contract's decimal values
Syntax	`decimals = myContract.Decimals`
Parameters	None
Return value	*decimals* — Decimals value Data type: Integer
Description	The `Decimals` property returns the number of decimals displayed in the X_Trader format version of the price for a contract.
Example	None
Related information	Property:Precision, on page 470 Property:PriceDisplayFormat, on page 471

Property: ExpirationDate

Purpose	Returns the contract's expiration date
Syntax	`expDate = myContract.ExpirationDate`
Parameters	None
Return value	*expDate* — Date the contract expires Data type: `DATE`
Description	The `ExpirationDate` property returns the expiration date associated with the contract.
Example	None
Related information	None

Note: Because the XTAPI and MEAPI use different data types for the expiration date, this property returns an invalid date (12/30/1899 12:00:00) in VB.NET applications for contracts that have no expiration date. You can use the `System.ToOADate()` function to check for a valid date.

For invalid dates, the `System.ToOADate()` function returns 0.0.

Property: LegInfo

Purpose	Returns the leg information for spreads and strategies
Syntax	`legContracts = myContract.LegInfo`
Parameters	None
Return value	*legContracts* — Collection of objects. Data type: `ITTContractCollection`
Description	The `LegInfo` property returns a collection of contracts that comprise the legs of spread or strategy contracts. If you call this property for a contract that is neither a spread or strategy, the property returns a NULL object.
Example	None
Related information	ITTContractCollection Class, on page 457

Property: Name

Purpose	Returns the contract name
Syntax	`contractName = myContract.Name`
Parameters	None
Return value	*contractName* — Name associated with the contract Data type: String
Exceptions	None
Description	The `Name` property returns the contract name.
Example	None
Related information	Property:SeriesName, on page 475

Property: Precision

Purpose	Returns the contract precision
Syntax	`precision = myContract.Precision`
Parameters	None
Return value	*precision* — Default market precision Data type: Integer
Exceptions	None
Description	The `Precision` property returns the default market precision for the contract's strike price.
Example	None
Related information	Property:Decimals, on page 466

Property: PriceDisplayFormat

Purpose	Returns a contract's price display format
Syntax	`format = myContract.PriceDisplayFormat`
Parameters	None
Return value	*format* — Price display format. Possible values include: • TT_DECIMAL • TT_FRACTION • TT_FORMAT_UNKNOWN Data type: `enumPriceDisplayFormat`
Description	The `PriceDisplayFormat` property returns the price display format associated with the contract.
Example	None
Related information	Property:PriceDisplayType, on page 472 PriceDisplayFormat, on page 562

Property: PriceDisplayType

Purpose	Returns the contract's price display type
Syntax	`displayType = myContract.PriceDisplayType`
Parameters	None
Return value	*displayType* — Price display type. Data type: `enumPriceDisplayType`
Description	The `PriceDisplayType` property returns price display type associated with the contract.
Example	None
Related information	Property:PriceDisplayFormat, on page 471

Property: Product

Purpose	Returns the product object associated with the contract
Syntax	`prodObj = myContract.Product`
Parameters	None
Return value	*prodObj* — Product object Data type: `ITTProduct`
Description	The `Product` property returns product object that corresponds to this particular contract.
Example	None
Related information	None

Property: SeriesKey

Purpose	Returns the contract series key
Syntax	`key = myContract.SeriesKey`
Parameters	None
Return value	*key* — Unique identifier for the contract Data type: String
Exceptions	None
Description	The `SeriesKey` property returns the unique series key assigned to this contract by the exchange. You can use the series key to open an instrument (**`TTInstrObj`**) in place of the product, product type, and contract. Using the series key is particularly useful if your application saves state information in workspaces. When the application loads the workspace, it can quickly open the instruments using their `SeriesKey` property values.
Example	None
Related information	Property:SeriesName, on page 475

Property: SeriesName

Purpose	Returns the series name for the contract
Syntax	`seriesName = myContract.SeriesName`
Parameters	None
Return value	*seriesName* — Name of the contract's series Data type: String
Exceptions	None
Description	The `SeriesName` property returns the series name for this contract.
Example	None
Related information	Property:Name, on page 469 Property:SeriesKey, on page 474

Property: Strike

Purpose	Returns an option's strike price
Syntax	`price = myContract.Strike`
Parameters	None
Return value	*price* — Strike price for the option Data type: Integer
Exceptions	None
Description	The `Strike` property returns the strike price of an options contract. For non-option contracts, this property returns the `INVALID_PRICE` enumeration.
Example	None
Related information	Property:CallPut, on page 463

Property: TickPriceDenominator

Purpose	Returns the contract's tick price denominator
Syntax	`tickD = myContract.TickPriceDenominator`
Parameters	None
Return value	*tickD* — Tick price denominator Data type: Integer
Description	The `TickPriceDenominator` property returns the denominator associated with the tick price of this contract.
Example	None
Related information	Property:TickPriceNumerator, on page 478 Property:TickSize, on page 479

Property: TickPriceNumerator

Purpose	Returns the contract's tick price numerator
Syntax	`tickN = myContract.TickPriceNumerator`
Parameters	None
Return value	*tickN* — Tick price numerator Data type: Integer
Description	The `TickPriceNumerator` property returns the numerator of the tick price associated with this contract.
Example	None
Related information	Property:TickPriceDenominator, on page 477 Property:TickSize, on page 479

Property: TickSize

Purpose	Returns the contract's tick size
Syntax	`tickSize = myContract.TickSize`
Parameters	None
Return value	*tickSize* — Ticking increment Data type: Integer
Description	The `TickSize` property returns tick size of the contract. The tick size indicates the interval between tradable prices for this contract.
Example	None
Related information	Property:TickPriceDenominator, on page 477 Property:TickPriceNumerator, on page 478

Method: CopyToClipboard

Purpose	Copies the contract information to the Windows clipboard
Syntax	`hWnd = myContract.CopyToClipboard(HWND hWnd)`
Parameters	*hWnd* — Windows handle for your application's main window. Data type: `HWND`
Return value	None
Description	The `CopyToClipboard` method copies the contract information to the Windows system clipboard so that you can implement drag-and-drop functionality to drag contract information from an XTAPI or MEAPI application to an another XTAPI application or to an X_TRADER window.
Example	None
Related information	None

Method: ToXML

Purpose	Returns the current contract information as XML
Syntax	`contractXML = myContract.ToXML`
Parameters	*contractXML* — Contract information in XML Data type: String
Return value	None
Description	The `ToXML` method returns the contract information as a well-formed XML structure, similar to the following:

```
<ContractInfo>
    <Name>Calendar: 1xcC Jul08:-1xcC May09</Name>
    <ProductType value="2">SPREAD</ProductType>
    <SeriesKey>XFC 02E032690S</SeriesKey>
    <SeriesName />
    <ExpirationDate />
    <Currency>GBP</Currency>
    <Precision>1</Precision>
    <Decimals>0</Decimals>
    <PriceDisplayFormat value="68">Decimal</PriceDisplayFormat>
    <PriceDisplayType value="0">TT_PRICE_DISPLAY_NORMAL</PriceDisplayType>
    <TickSize>1</TickSize>
    <TickPriceNumerator>1</TickPriceNumerator>
    <TickPriceDenominator>1</TickPriceDenominator>
    <ComboCode value="516">Calendar</ComboCode>
    <CallPut value="32" />
    <Strike>0</Strike>
    <LegList>
      <LegInfo>
          <RatioQty>1</RatioQty>
          <ContractInfo>
            <Name>cC Jul08</Name>
            <ProductType value="1">FUTURE</ProductType>
            <SeriesKey>XFC 080700000F</SeriesKey>
            <SeriesName />
            <ExpirationDate>20080716</ExpirationDate>
            <Currency>GBP</Currency>
            <Precision>1</Precision>
            <Decimals>0</Decimals>
            <PriceDisplayFormat value="68">Decimal</PriceDisplayFormat>
            <PriceDisplayType value="0">TT_PRICE_DISPLAY_NORMAL</PriceDisplayType>
            <TickSize>1</TickSize>
            <TickPriceNumerator>1</TickPriceNumerator>
            <TickPriceDenominator>1</TickPriceDenominator>
            <ComboCode value="516">Calendar</ComboCode>
            <CallPut value="32" />
            <Strike>0</Strike>
          </ContractInfo>
      </LegInfo>
      <LegInfo>
          <RatioQty>-1</RatioQty>
          <ContractInfo>
            <Name>cC May09</Name>
            <ProductType value="1">FUTURE</ProductType>
            <SeriesKey>XFC 090500000F</SeriesKey>
            <SeriesName />
            <ExpirationDate>20090513</ExpirationDate>
            <Currency>GBP</Currency>
            <Precision>1</Precision>
            <Decimals>0</Decimals>
            <PriceDisplayFormat value="68">Decimal</PriceDisplayFormat>
            <PriceDisplayType value="0">TT_PRICE_DISPLAY_NORMAL</PriceDisplayType>
            <TickSize>1</TickSize>
            <TickPriceNumerator>1</TickPriceNumerator>
            <TickPriceDenominator>1</TickPriceDenominator>
            <ComboCode value="516">Calendar</ComboCode>
            <CallPut value="32" />
            <Strike>0</Strike>
          </ContractInfo>
      </LegInfo>
    </LegList>
</ContractInfo>
```

Example None

Related information None

25 ITTGateway Class

Chapter overview

The chapter describes the **`ITTGateway`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTGateway class overview	484
Property:AvailableServices	485
Property:ID	486
Property:IsAvailable	487
Property:IsLoggedIn	489
Property:MarketName	490
Property:Name	491
Property:Products	492
Method:RegisterEventHandler	493
Method:RequestProducts	495
Method:RequestProductTypes	497
Method:UnregisterEventHandler	499

ITTGateway class overview

Overview

The **ITTGateway** class represents a TT Gateway. Through this class, you can determine the status of a TT Gateway and its servers and return the products trading on the TT Gateway.

Properties

Table 88 lists the **ITTGateway** properties.

Property	Description
AvailableServices	Returns the services available on the TT Gateway
ID	Returns the TT Gateway identifier
IsAvailable	Returns whether a user can log in to the TT Gateway
IsLoggedIn	Indicates whether the application is currently logged into a service
MarketName	Returns the market name associated with the TT Gateway
Name	Returns the name associated with the TT Gateway
Products	Returns a collection of products associated with the TT Gateway

Table 88. ITTGateway properties

Methods

Table 89 lists the **ITTGateway** methods.

Method	Description
RegisterEventHandler	Subscribes for `ITTGateway` events
RequestProducts	Requests a list of products trading on a TT Gateway
RequestProductTypes	Requests a list of product types trading on the TT Gateway
UnregisterEventHandler	Unsubscribes for `ITTGateway` events

Table 89. ITTGateway methods

Events

None

Property: AvailableServices

Purpose	Returns the services available on the TT Gateway
Syntax	`serviceMask = myGateway.AvailableServices`
Parameters	None
Return value	*serviceMask* — ID of the print service. Data type: Integer
Exceptions	None
Description	The `AvailableServices` property indicates the service ID for available services on this TT Gateway. Currently, the XTAPI supports only the Price Server.
Example	None
Related information	Property:IsAvailable, on page 487

Property: ID

Purpose	Returns the TT Gateway identifier
Syntax	`gatewayId = myGateway.ID`
Parameters	None
Return value	*gatewayId* — ID associated with this TT Gateway Data type: Integer
Exceptions	None
Description	The `ID` property returns the unique identifier associated with this **`ITTGateway`** object. Typically, the value matches the exchange ID.
Example	None
Related information	Property:Name, on page 491

Property: IsAvailable

Purpose

Returns whether a user can log in to the TT Gateway

Syntax

gatewayState = myGateway.IsAvailable(enumServiceTypes *service*)

Parameters

service	Service you want to check. Currently, the XTAPI supports only `PRICE_SERVICE`. Data type: `enumServiceTypes`

Return value

gatewayState	Whether the specified service is available. Data type: Boolean

Exceptions

None

Description

The `IsAvailable` property returns a Boolean status indicating whether a Price Server on the TT Gateway is available. If so, the application can log into the TT Gateway.

Example

The following example displays the availability of a selected service for an **`ITTGateway`** object.

```
Private Sub btnGWGetProperty_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnGWGetProperty.Click

'Declare local variables
Dim oProd As TTMarketExplorer.ITTProduct
Dim sServiceUpDown As String
Dim iServices As Integer
Dim bIsAvailable As Boolean
Dim bIsLoggedIn As Boolean
Dim iProductCount As Integer = 0

Try

 'Select the property of the Gateway Object to be displayed
  Select Case cbGWProperties.SelectedItem.ToString

  ' Other cases omitted

  'If IsAvailable is selected, display the value of this property
  Case "IsAvailable"

    'Assign the value to a temp variable
    '(sSelectedValue is a global variable storing a menu selection)
    bIsAvailable = goGateway.IsAvailable(sSelectedvalue)

    'If the IsAvailable=0 then print the server is down, else print its up
    If Not bIsAvailable = 0 Then
     WriteToMessagebox("IsAvailable ( " & sSelectedvalue & " ) " & " is TRUE")

    Else
     WriteToMessagebox("IsAvailable ( " & sSelectedvalue & " ) " & " is FALSE")

    End If
```

```
    Next

  End Select

'Display the Exception generated in the Message box
Catch ex As Exception
  'Code omitted

End Try
End Sub
```

Related information

Property:AvailableServices, on page 485

Property:IsLoggedIn, on page 489

Property: IsLoggedIn

Purpose	Indicates whether the application is currently logged into a service	
Syntax	`isIn = myGateway.IsLoggedIn(enumServiceTypes serviceType)`	
Parameters		
	serviceType	Type of service to check. Currently, the XTAPI supports only `PRICE_SERVICE`. Data type: `enumServiceTypes`
Return value		
	isIn	Whether the application is logged in to the service Data type: Boolean
Exceptions	None	
Description	The `IsLoggedIn` property returns whether the application is currently logged in to the Price Service on this TT Gateway.	
Example	None	
Related information	Property:IsAvailable, on page 487	

Property: MarketName

Purpose	Returns the market name associated with the TT Gateway
Syntax	`mktName = myGateway.MarketName`
Parameters	None
Return value	*mktName* — Market name for the TT Gateway Data type: String
Exceptions	None
Description	The `MarketName` property returns market name for this **`ITTGateway`** object. A TT Gateway with multiple exchange flavors uses distinct **`ITTGateway`** objects for each flavor, where each object uses the same market name but different gateway names. For example, if your application connects to multiple TT CME Gateways, the `MarketName` property returns the same value (CME) for each, while the `Name` property returns different values for each (CME-A, CME-B, and so on).
Example	None
Related information	Property:Name, on page 491

Property: Name

Purpose	Returns the name associated with the TT Gateway
Syntax	`gatewayName = myGateway.Name`
Parameters	None
Return value	*gatewayName* — TT Gateway name Data type: String
Exceptions	None
Description	The `Name` property returns name for this **`ITTGateway`** object. A TT Gateway with multiple flavors uses distinct **`ITTGateway`** objects for each flavor, where each object uses the same market name but different gateway names. For example, if your application connects to multiple TT CME Gateways, the `MarketName` property returns the same value (CME) for each, while the `Name` property returns different values for each (CME-A, CME-B, and so on).
Example	None
Related information	Property:MarketName, on page 490 Property:ID, on page 486

Property: Products

Purpose	Returns a collection of products associated with the TT Gateway
Syntax	`products = myGateway.Products`
Parameters	None
Return value	*products* — Collection of products available on the TT Gateway Data type: `ITTProductCollection`
Exceptions	None
Description	The `Products` property returns a list of all products currently trading on the TT Gateway. After you receive the product collection, you can retrieve all of the contracts associated with each product.
Example	None
Related information	Method:RequestProductTypes, on page 497

Method: RegisterEventHandler

Purpose

Subscribes for **ITTGateway** events

Syntax

returnCode = myGateway.RegisterEventHandler(ITTGatewayEvents *gatewayEventHandler*)

Parameters

gatewayEventHandler	Pointer to the event handler interface Data type: ITTGatewayEvents

Return value

returnCode	Status of the register request. Possible return codes include: • RR_SUCCESS • RR_INTERNAL_ERROR • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: enumRegisterReturnCodes

Description

The RegisterEventHandler method registers with the Market Explorer API for **ITTGateway** events. You must call this method if you want to be notified when **ITTGateway** events occur.

After registering an event handler, the Market Explorer API sends callbacks for the following events:

- OnGatewayAttributes
- OnServerStatus

If you want to discontinue receiving event notifications, you can call the UnregisterEventHandler method.

Example

The following example registers for gateway events. If an error occurs, the code displays the error message. The main class implements the **ITTGatewayEvents** class, so the example can use the Me identifier.

```
'Call register event handler method and check the return code
geRRCode = goGateway.RegisterEventHandler(Me)

'If the return code is RR_SUCCESS then make the label with the Gateway Name
green
'Print the return code in the message list box
If geRRCode = enumRegisterReturnCode.RR_SUCCESS Then

    'Add the name of the gateway to the list of registered gateways
    gaGatewaysRegisteredList.Add(goGateway.Name)
    'Add the name of the gateway to the Registered gateways List Box
    lbxRegisteredGateways.Items.Add(goGateway.Name)
    'Print the return code in the message list box
    WriteToMessagebox("For the Method : Gateway RegisterEventHandler: ")
    WriteToMessagebox("Return Code recieved is : " & geRRCode.ToString)
    'change the color of the label to green
    lblGatewayName.BackColor = Color.GreenYellow

Else
```

```
    'Print the return code in the message list box
    WriteToMessagebox("For the Method : Gateway RegisterEventHandler: ")
    WriteToMessagebox("Return Code recieved is : " & geRRCode.ToString)
End If
```

Related information

Method:UnregisterEventHandler, on page 499

ITTGatewayEvents Class, on page 501

Method: RequestProducts

Purpose

Requests a list of products trading on a TT Gateway

Syntax

```
returnCode = myGateway.RequestProducts(BOOL forceRequery)
```

Parameters

forceRequery	Whether to query the TT Gateway every time or to search the product list returned from previous queries. Data type: Boolean; Default: FALSE

Return value

returnCode	Status of the method call. Possible values include: • QR_SUCCESS • QR_HANDLER_NOT_REGISTERED • QR_GATEWAY_NOT_AVAIL • QR_NOT_LOGGED_IN • QR_GAL_NOT_AVAIL • QR_INVALID_PARAMETETR Data type: `enumQueryReturnCodes`

Description

The `RequestProducts` method submits a request to a TT Gateway for a list of the products it supports. If the method returns `QR_SUCCESS`, the MEAPI triggers an **`ITTGatewayEvents`** `OnProductsFound` event that contains the list of products found on the TT Gateway. Otherwise, it triggers an **`ITTGatewayEvents`** `OnProductsNotFound` event

Because the list of products changes infrequently, this method pulls the list of products directly from the cache for all calls after the first. If you want to force the MEAPI to retrieve fresh data from the TT Gateway, you can set the *`forceRequery`* parameter to TRUE.

Example

The following example calls the `RequestProducts` method in response to a user selection and ensures that it returns a fresh list of products.

```
Select Case cbGWMethods.Text

  'If RequestProducts is selected, call the method RequestProducts
  Case "RequestProducts"

    'Clear the Products Array list
    gaProductList.Clear()

    'Clear the Product Listbox and the Products count
    lbxProducts.Items.Clear()
    lblProductsNumber.Text = "0"

    'Return Code Clean up
    geQRcode = RETURN_CODE

    'Call RequestProducts method and print the return code
    geQRcode = goGateway.RequestProducts(True)

    'Insert a Blank line in the message list box
    InsertBlankLine()
```

```
    'print the return code in the message list box
    WriteToMessagebox("For the Class : Gateway , Method:  RequestProducts ")
    WriteToMessagebox("Return Code recieved is : " & geQRcode.ToString)

  'If RequestProductTypes is selected, call the method RequestProductTypes
  Case "RequestProductTypes"
    '
    ' code omitted
    '

End Select
```

Related information **`ITTGatewayEvents`** Event:OnProducts, on page 507

Method: RequestProductTypes

Purpose

Requests a list of product types trading on the TT Gateway

Syntax

```
returnCode = myGateway.RequestProductTypes
```

Parameters

None

Return value

returnCode — Status of the method call. Possible values include:

- QR_SUCCESS
- QR_HANDLER_NOT_REGISTERED
- QR_GATEWAY_NOT_AVAIL
- QR_NOT_LOGGED_IN
- QR_GAL_NOT_AVAIL
- QR_INVALID_PARAMETETR

Data type: `enumQueryReturnCodes`

Description

The `RequestProductTypes` method submits a request to a TT Gateway for a list of the product types it supports. If the method returns `QR_SUCCESS`, the MEAPI triggers an **`ITTGateway`** `OnProductTypes` event that contains the list of types found on the TT Gateway.

Example

The following example submits a request for available product types in response to a user-selection.

```
Select Case cbGWMethods.Text

  'If RequestProducts is selected, call the method RequestProducts
  Case "RequestProducts"
    '
    ' code omitted
    '


  'If RequestProductTypes is selected, call the method RequestProductTypes
  Case "RequestProductTypes"

    'Clear the ProductTypesAsString and the ProductTypes Array lists
    gaProdTypesAsStringList.Clear()
    gaProdTypesList.Clear()

    'Clear the ProductTypes Listbox and the ProductTypes count
    lbxProductTypes.Items.Clear()
    lblProdTypesNumber.Text = "0"

    'Return Code Clean up
    geQRcode = RETURN_CODE

    'Call RequestProductTypes method and print the return code
    geQRcode = goGateway.RequestProductTypes()

    'Insert a Blank line in the message list box
    InsertBlankLine()

    'print the return code in the message list box
    WriteToMessagebox("For the Class : Gateway , Method:  RequestProductTypes
")
```

```
    WriteToMessagebox("Return Code recieved is : " & geQRcode.ToString)
End Select
```

Related information

[Property:Products](), on page 492

`ITTGatewayEvents` [Event:OnProductTypes](), on page 509

Method: UnregisterEventHandler

Purpose	Unsubscribes for **`ITTGateway`** events
Syntax	*`returnCode`* `= myGateway.UnregisterEventHandler`
Parameters	None
Return value	*returnCode* — Status of the method call. Possible values include: • RR_SUCCESS • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: `enumRegisterReturnCodes`
Description	The `UnregisterEventHandler` method unregisters the **`ITTGatewayEvents`** event handler. After you call this method, the application no longer responds to TT Gateway events.
Example	None
Related information	Method:RegisterEventHandler, on page 493

26 ITTGatewayEvents Class

Chapter overview

The chapter describes the **`ITTGatewayEvents`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTGatewayEvents class overview	502
Event:OnGatewayAttributes	503
Event:OnNewContract	505
Event:OnNewProduct	506
Event:OnProducts	507
Event:OnProductTypes	509
Event:OnServerStatus	510

ITTGatewayEvents class overview

Overview

The **ITTGatewayEvents** class provides access to events generated by **ITTGateway** objects.

You must implement the class to make its members available to your application, as shown in the following code fragment.

```
'Import the MEAPI class librabry
Imports TTMarketExplorer
Imports System.IO

'**************************************************************************
'*          MAIN CLASS
'**************************************************************************
Public Class frmMEAPITestTool
    Implements TTMarketExplorer.ITTMarketExplorerEvents
    Implements TTMarketExplorer.ITTGatewayEvents
    Implements TTMarketExplorer.ITTProductEvents
```

Properties

None

Methods

None

Events

Table 90 lists the **ITTGatewayEvents** events.

Event	Description
OnGatewayAttributes	MEAPI receives the gateway attribute list (GAL settings) for the TT Gateway
OnNewContract	TT Gateways add new contracts for a product
OnNewProduct	TT Gateways receive new products from an exchange
OnProducts	TT Gateways find products that match a product request
OnProductTypes	TT Gateways return a list of product types hosted on the TT Gateway
OnServerStatus	Status of a TT Gateway server changes

Table 90. ITTGatewayEvents properties

Event: OnGatewayAttributes

Triggered when

MEAPI receives the gateway attribute list (GAL settings) for the TT Gateway

Syntax

```
Sub myMktExp_OnGatewayAttributes(ITTGateway gateway, string
attributeList, enumServiceTypes service)
```

Parameters

gateway	TT Gateway whose GAL setting changed Data type: `ITTGateway`
attributeList	Comma-separated list of GAL name-value pairs Data type: String
service	Service that started or restarted. Currently, the MEAPI supports only `PRICE_SERVICE`. Data type: `enumServiceTypes`

Description

The MEAPI triggers the `OnGatewayAttributes` event when the MEAPI receives GAL attributes from a TT Gateway in the following circumstances:

- When you register an event handler
- When one of the TT Gateway servers starts or restarts

This event converts all GAL attribute values in the attributeList variable to strings. For example, a TT Gateway defines the `TT_GAL_SUPPORTED_RESTRICTION_CODES` GAL attribute for order restriction codes as ASCII character codes (integers), not as strings. So, for an IOC order restriction, this event returns I, while the GAL setting for the TT Gateway uses the value 73 (the ASCII character code for a capital I).

For more information about the available GAL settings, refer to the appropriate TT Gateway SAM.

Note: To receive these events, you must call the **`ITTGateway`** `RegisterEventHandler` method to register for events.

Example

The following example shows how to create an event handler for the `OnGatewayAttributes` event. This sample displays the name-value pairs of the GAL settings in a message box.

```
'************************************************************************
*
'*  Name: OnGatewayAttributes
'*  Desc: Handles the event OnGatewayAttributes
'************************************************************************
*
Public Sub OnGatewayAttributes(ByVal Gateway As TTMarketExplorer.ITTGateway, _
ByVal attributeList As String, ByVal service As
TTMarketExplorer.enumServiceType) Implements
TTMarketExplorer.ITTGatewayEvents.OnGatewayAttributes

  'Declare the local variables
  Dim aKeyValue As Array
  Dim iIndex As New Integer

  Try

    'Insert a Blank line in the message list box
```

```
    InsertBlankLine()

    'Print the message to the message listbox
    WriteToMessagebox("Event OnGatewayAttributes Fired for the Gateway " & _
      Gateway.Name & " for the service " & service.ToString)

    'Insert a Blank line in the message list box
    InsertBlankLine()

    WriteToMessagebox("The GAL Attributes are " & attributeList)

    'The AttributeList is returned in a key=Value Pair sepreated by a comma
    'The key=Value Pair is split based on the comma and displayed
    'in the Message List Box
    aKeyValue = Split(attributeList, ",")

    'Insert a Blank line in the message list box
    InsertBlankLine()

    'Display the key=value Pair in the Message List Box
    For iIndex = 0 To UBound(aKeyValue)

      WriteToMessagebox(aKeyValue(iIndex))

    Next

    'Insert a Blank line in the message list box
    InsertBlankLine()

    'Print the number of GAL Attributes in the Message List Box
    WriteToMessagebox("The number of GAL attributes is " & UBound(aKeyValue)
+ 1)

  Catch ex As Exception

    'If the flag that writes all the errors to a file is checked, do
    'not display the error is a message box; instead write it to a file
    If gbErrortoFile = True Then
        gaErrors.Add("ERROR : " & "OnGatewayAttributes Event for  " & _
          " ( " & ex.Message & " ) ")
    Else
      MsgBox("There was an error in the OnGatewayAttributes Event for  " & _
        " ( " & ex.Message & " ) ")
    End If

  End Try

End Sub
```

Related information

Event:OnServerStatus, on page 510

ITTGateway Method:RegisterEventHandler, on page 493

Event: OnNewContract

Triggered when

TT Gateways add new contracts for a product

Syntax

```
Sub myMktExp_OnNewContract(ITTGateway gateway, ITTProduct product,
ITTContractInfo contract)
```

Parameters

gateway	TT Gateway on which the contract was added. Data type: `ITTGateway`
product	Product associated with the new contract. Data type: `ITTProduct`
contract	New contract. Data type: `ITTContractInfo`

Description

The MEAPI triggers the `OnNewContract` event when a TT Gateway adds a new tradable contract for the specified product.

Note: To receive these events, the application must first call the **`ITTGateway`** `RequestProducts` method.

Example

The following subroutine handles an `OnNewContract` event by displaying information about a newly-added contract.

```
'************************************************************************
*
'*  Name: OnNewContract
'*  Desc: Handles the event OnNewContract
'************************************************************************
*
Public Sub OnNewContract(ByVal Gateway As TTMarketExplorer.ITTGateway, _
ByVal Product As TTMarketExplorer.ITTProduct, _
ByVal contractInfo As TTMarketExplorer.ITTContractInfo) Implements
TTMarketExplorer.ITTGatewayEvents.OnNewContract

  'Insert a Blank line in the message list box
  InsertBlankLine()

  'Add a message to the message list box for each new contract on that gateway
  WriteToMessagebox("Event OnNewContract Fired for the contract " & _
    contractInfo.Name)
  WriteToMessagebox(" for the product " & Product.Name & " On Gateway " & _
    Gateway.Name)

End Sub
```

Related information

Event:OnNewProduct, on page 506

`ITTGateway` Method:RegisterEventHandler, on page 493

`ITTGateway` Method:RequestProducts, on page 495

Event: OnNewProduct

Triggered when

TT Gateways receive new products from an exchange

Syntax

`Sub myMktExp_OnNewProduct(ITTGateway` *gateway*`, ITTProduct` *product*`)`

Parameters

gateway	TT Gateway that received a new tradable product. Data type: `ITTGateway`
product	New product. Data type: `ITTProduct`

Description

The MEAPI triggers the `OnNewProduct` event when exchanges notify the TT Gateway of a new product available for trading or an updated product definition for an existing product. You can use the object in the product parameter to determine the contracts associated with the new product.

Note: To receive these events, the application must first call the **`ITTGateway`** `RequestProducts` method.

Example

The following example implements the `OnNewProduct` event to update the application interface when a TT Gateway offers a new product.

```
Public Sub OnNewProduct(ByVal Gateway As TTMarketExplorer.ITTGateway, ByVal
 Product As TTMarketExplorer.ITTProduct) Implements
 TTMarketExplorer.ITTGatewayEvents.OnNewProduct

  If Not lbxMessages.Items.Count = 0 Then

    If Not lbxMessages.Items(lbxMessages.Items.Count - 1) = "" Then
      lbxMessages.Items.Add("")
    End If

  End If

  lbxProducts.Items.Add(Product.Name)
  'Add a message to the message box for each gateway
  lbxMessages.Items.Add("Event OnNewProduct Fired for the Product " &
    Product.Name & "On Gateway " & Gateway.Name)

End Sub
```

Related information

Event:OnNewContract, on page 505

`ITTGateway` Method:RegisterEventHandler, on page 493

Event: OnProducts

Triggered when

TT Gateways find products that match a product request

Syntax

`Sub myMktExp_OnProducts(ITTGateway` *`gateway`*`, ITTProductCollection` *`products`*`)`

Parameters

gateway	TT Gateway containing the products. Data type: `ITTGateway`
products	Collection products returned by the TT Gateway. Data type: `ITTProductCollection`

Description

The MEAPI triggers the `OnProducts` event when you request a full list of products using the **`ITTGateway`** RequestProducts method.

You can then use the returned products collection to process the individual products.

Note: To receive this event, you must implement the **`ITTGatewayEvents`** interface and register it by calling the **`ITTGatewayEvents`** `RegisterEventHandler` method.

Example

The following example implements the `OnProducts` event to add the list of tradable products to the application window when a TT Gateway delivers its products.

```
Public Sub OnProducts(ByVal Gateway As TTMarketExplorer.ITTGateway,
  ByVal Products As TTMarketExplorer.ITTProductCollection) Implements
  TTMarketExplorer.ITTGatewayEvents.OnProductsFound

  Try

    For Each oProd In Products

      'Add all the Product objects to a Product List
      aProductList.Add(oProd)

      'Add the Product names to the List Box
      lbxProducts.Items.Add(oProd.Name)

    Next

    If Not lbxMessages.Items.Count = 0 Then

      If Not lbxMessages.Items(lbxMessages.Items.Count - 1) = "" Then
        lbxMessages.Items.Add("")
      End If
    End If

    'Add a message to the message box for each gateway
    lbxMessages.Items.Add("Event OnProductsFound Fired for the Gateway " &
      Gateway.Name)
    Count the number of gateways in the Gateway list
    iProductCount = lbxProducts.Items.Count()

    'Display the number of Products in the label
    lblProductsNumber.Text = iProductCount
```

```
    'Display the Exception generated in the Message box
  Catch ex As Exception
    MsgBox("There was an error in the OnProductsFound Event for  " &
      Gateway.Name & " ( " & ex.Message & " ) ")
  End Try

End Sub
```

Related information **`ITTGateway`** Method:RegisterEventHandler, on page 493

Event: OnProductTypes

Triggered when

TT Gateways return a list of product types hosted on the TT Gateway

Syntax

```
Sub myMktExp_OnProductTypes(ITTGateway gateway, enumProductTypes[]
productTypes, string[] productTypesAsString)
```

Parameters

gateway	TT Gateway containing the product types Data type: `ITTGateway`
productTypes	Array of product type IDs Data type: Array of `enumProductTypes`
productTypesAsString	Array of product type names Data type: Array of Strings

Description

The MEAPI triggers the `OnProductTypes` event in response to an **`ITTGateway`** RequestProductTypes method call.

Note: To receive this event, you must implement the **`ITTGatewayEvents`** interface and register it by calling the **`ITTGatewayEvents`** `RegisterEventHandler` method.

Example

None

Related information

`ITTGateway` Method:RegisterEventHandler, on page 493

`ITTGateway` Method:RequestProductTypes, on page 497

Event: OnServerStatus

Triggered when

Status of a TT Gateway server changes

Syntax

```
Sub myMktExp_OnServerStatus(ITTGateway gateway, enumServiceTypes
service, BOOL isAvailable)
```

Parameters

gateway	Gateway affected by the server status change Data type: `ITTGateway`
service	Service whose status changed. Currently, the MEAPI supports only `PRICE_SERVICE`. Data type: `enumServiceTypes`
isAvailable	Current status of the service, as: • TRUE: service started and is available • FALSE: service went down and is not currently available Data type: Boolean

Description

The MEAPI triggers the `OnServerStatus` event when the status of one of the services on a TT Gateway changes. You also receive this event for each of the services as a by-product of registering the event handler so that you know the initial states of the TT Gateway services.

Note: To receive this event, you must implement the **`ITTGatewayEvents`** interface and register it by calling the **`ITTGatewayEvents`** `RegisterEventHandler` method.

Example

The following subroutine handles an `OnServerStatus` event by displaying information about the server status.

```
'**************************************************************************
*
'*  Name: OnServerStatus
'*  Desc: Handles the event OnServerStatus
'**************************************************************************
*
Public Sub OnServerStatus(ByVal Gateway As TTMarketExplorer.ITTGateway, _
ByVal service As TTMarketExplorer.enumServiceType, ByVal IsAvailable As
Integer, _
ByVal IsLoggedIn As Integer) Implements
TTMarketExplorer.ITTGatewayEvents.OnServerStatus

  'Insert a Blank line in the message list box
  InsertBlankLine()

  'Add a message to the message list box for each OnServerStatus Callback
  WriteToMessagebox("Event OnServerStatus Fired for the Gateway " &
Gateway.Name)
  WriteToMessagebox("The Service " & service.ToString & " is " & IsAvailable)

End Sub
```

Related information

Event:OnGatewayAttributes, on page 503

`ITTGateway` Method:RegisterEventHandler, on page 493

27 ITTMarketExplorerEvents Class

Chapter overview

The chapter describes the **`ITTMarketExplorerEvents`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTMarketExplorerEvents class overview	512
Event:OnGateway	513

ITTMarketExplorerEvents class overview

Overview

The **ITTMarketExplorerEvents** class provides access to market explorer events.

You must implement the class to make its members available to your application, as shown in the following code fragment.

```
'Import the MEAPI class librabry
Imports TTMarketExplorer
Imports System.IO

'**********************************************************************
'*          MAIN CLASS
'**********************************************************************
Public Class frmMEAPITestTool
    Implements TTMarketExplorer.ITTMarketExplorerEvents
    Implements TTMarketExplorer.ITTGatewayEvents
    Implements TTMarketExplorer.ITTProductEvents
```

Properties

None

Methods

None

Events

Table 91 lists the **ITTMarketExplorerEvents** events.

Property	Description
OnGateway	TT Gateways are found in response to a RequestGateways method call or when new TT Gateways become available

Table 91. ITTMarketExplorerEvents properties

Event: OnGateway

Triggered when

TT Gateways are found in response to a RequestGateways method call or when new TT Gateways become available

Syntax

`Sub myMktExp_OnGateway(ITTGateway gateway)`

Parameters

gateway	TT Gateway object found by the MEAPI. Data type: `ITTGateway`

Description

The MEAPI triggers the `OnGateway` event in the following circumstances:

- In response to a **`TTMarketExplorer`** RequestGateways method call
- When new TT Gateways become available

Example

The following example shows how to handle an `OnGateway` event.

```
Public Sub OnGateway(ByVal gatewayObject As TTMarketExplorer.ITTGateway)
 Implements TTMarketExplorer.ITTMarketExplorerEvents.OnGateway

  'Declare the local variables
  Dim igatewayCount As Integer

  Try

    'Add all the Gateway names to a GatewayNames Array List
    gaGatewayNameList.Add(gatewayObject.Name)

    'Add all the Gateway Objects to a GatewayObjects Array List
    gaGatewayList.Add(gatewayObject)

    'Add the Gateway names to the List Box
    lbxGateways.Items.Add(gatewayObject.Name)

    'Update the Gateway ListBox
    lbxGateways.Update()

    'Sleep to add the gateway objects to the gateway Listbox
    System.Threading.Thread.Sleep(100)

    'Add a message to the message list box for each gateway
    WriteToMessagebox("Event OnGateway Fired for the Gateway " _
      & gatewayObject.Name)

    'Count the number of gateways in the Gateway list
    igatewayCount = lbxGateways.Items.Count()

    'Display the number of gateways in the label
    lblGatewaysNumber.Text = igatewayCount

    'Display the Exception generated in the Message box
  Catch ex As Exception

    'If the flag that writes all the errors to a file is checked,
    'do not display the error in a message box; instead write it to a file
    If gbErrortoFile = True Then
      gaErrors.Add("ERROR : " & "OnGateway Event for  " & gatewayObject.Name
& _
        " ( " & ex.Message & " ) ")
    Else
```

```
            MsgBox("There was an error in the OnGateway Event for  " & _
              gatewayObject.Name & " ( " & ex.Message & " ) ")
        End If

        End Try

    End Sub
```

Related information

TTMarketExplorer Method:RegisterEventHandler, on page 542

TTMarketExplorer Method:UnregisterEventHandler, on page 513

28 ITTProduct Class

Chapter overview

The chapter describes the **`ITTProduct`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTProduct class overview	516
Property:Alias	517
Property:Description	518
Property:Gateway	519
Property:Name	520
Property:Type	521
Property:TypeAsString	522
Method:RegisterEventHandler	523
Method:RequestContracts	524
Method:UnregisterEventHandler	525

ITTProduct class overview

Overview

The **ITTProduct** class stores information about a tradable product. An **ITTProduct** object contains the name and type of the product, the TT Gateway on which it trades, and the contracts available for the product.

Properties

Table 92 lists the **ITTProduct** properties.

Property	Description
Alias	Returns an alias associated with the product
Description	Returns the description of the product, if made available by the exchange
Gateway	Returns the `ITTGateway` object associated with this product
Name	Returns the product name
Type	Returns the product type as an enumeration
TypeAsString	Returns the product type as a string

Table 92. ITTProduct properties

Methods

Table 92 lists the **ITTProduct** methods.

Property	Description
RegisterEventHandler	Subscribes for `ITTProduct` events
RequestContracts	Requests all contract associated with a product
UnregisterEventHandler	Unsubscribe to `ITTProduct` events

Table 93. ITTProduct properties

Events

None

Property: Alias

Purpose	Returns an alias associated with the product
Syntax	*prodAlias* = myProduct.Alias
Parameters	None
Return value	*prodAlias* — Alias associated with the product Data type: String
Description	The `Alias` property returns the alias associated with the product, if one exists. For example, the CME ES FUTURE product could use an alias like SP Mini.
Example	None
Related information	Property:Name, on page 517

Property: Description

Purpose	Returns the description of the product, if made available by the exchange
Syntax	`gwId = myProduct.Description`
Parameters	None
Return value	*description* — Exchange-supplied description for the product, or an empty string. Data type: String
Exceptions	None
Description	The `Description` property returns the description for the product, if the exchange supplies it.
Example	None
Related information	Property:Name, on page 520

Property: Gateway

Purpose	Returns the **`ITTGateway`** object associated with this product
Syntax	*gw* = myProduct.Gateway
Parameters	None
Return value	*gw* — Object representing the TT Gateway on which the product trades Data type: ITTGateway
Exceptions	None
Description	The Gateway property returns the **`ITTGateway`** object associated with this **`ITTProduct`** object.
Example	None
Related information	None

Property: Name

Purpose	Returns the product name
Syntax	`prodName = myProduct.Name`
Parameters	None
Return value	*prodName* — Name of the product Data type: String
Description	The `Name` property returns the name assigned to the product by the exchange.
Example	None
Related information	Property:Alias, on page 517 Property:Description, on page 518

Property: Type

Purpose

Returns the product type as an enumeration

Syntax

```
prodTypeInt = myProduct.Type
```

Parameters

None

Return value

prodTypeInt	Product type Data type: `enumProductTypes`

Exceptions

None

Description

The `Type` property returns as an enumeration value. The property returns one of the following enumerations.

- TT_PROD_FUTURE
- TT_PROD_SPREAD
- TT_PROD_OPTION
- TT_PROD_STRATEGY
- TT_PROD_STOCK
- TT_PROD_BOND
- TT_PROD_SWAP
- TT_PROD_WARRANT
- TT_PROD_ACTIVE
- TT_PROD_AUCTION
- TT_PROD_WI
- TT_PROD_ROLL
- TT_PROD_SWAP_ANCHOR
- TT_PROD_YIELD_CURVE
- TT_PROD_BILL
- TT_PROD_UNDER_2YR
- TT_PROD_2YR
- TT_PROD_2YR_SWAP
- TT_PROD_2YR_BASIS
- TT_PROD_5YR
- TT_PROD_5YR_SWAP
- TT_PROD_5YR_BASIS
- TT_PROD_10YR
- TT_PROD_10YR_SWAP
- TT_PROD_10YR_BASIS
- TT_PROD_30YR
- TT_PROD_30YR_SWAP
- TT_PROD_30YR_CALLABLE
- TT_PROD_30YR_BASIS
- TT_PROD_30YR_OTR_SWAP
- TT_PROD_ENERGY
- TT_PROD_FOREX
- TT_PROD_NDF
- TT_PROD_ALL

Example

None

Related information

Property:TypeAsString, on page 522

Property: TypeAsString

Purpose	Returns the product type as a string
Syntax	`prodTypeInt = myProduct.TypeAsString`
Parameters	None
Return value	*prodTypeInt* — Product type Data type: String
Exceptions	None

Description

The `TypeAsString` property returns the product type as a string value. For example, while a FUTURES product corresponds to the `TT_PROD_FUTURE` product type enumeration; this property returns the type as "FUTURE". The property returns one of the following strings:

- FUTURE
- SPREAD
- OPTION
- STRATEGY
- STOCK
- BOND
- SWAP
- WARRANT
- ACTIVE
- WHEN ISSUE
- ROLL
- SWAP_ANCHOR
- YIELD_CURVE
- BILL
- UNDER_2YR
- 2YR
- 2YR_SWAP
- 2YR_BASIS
- 5YR
- 5YR_SWAP
- 5YR_BASIS
- 10YR
- 10YR_SWAP
- 10YR_BASIS
- 30YR
- 30YR_SWAP
- 30YR_CALLABLE
- 30YR_BASIS
- 30YR_OTR_SWAP
- ENERGY
- FOREX

Example	None
Related information	Property:Type, on page 521

Method: RegisterEventHandler

Purpose	Subscribes for **`ITTProduct`** events
Syntax	*returnCode* = myProduct.RegisterEventHandler(ITTProductEvents *productEventHandler*)
Parameters	*productEventHandler* — Pointer to the event handler interface Data type: ITTProductEvents
Return value	*returnCode* — Status of the register request. Possible return codes include: • RR_SUCCESS • RR_INTERNAL_ERROR • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: `enumRegisterReturnCodes`
Description	The `RegisterEventHandler` method registers with the Market Explorer API for **`ITTProduct`** events. You must call this method if you want to be notified when **`ITTProduct`** events occur. After registering an event handler, the Market Explorer API sends callbacks for the following events: • OnContracts • OnNewContract If you want to discontinue receiving event notifications, you can call the `UnregisterEventHandler` method.
Example	None
Related information	Method:UnregisterEventHandler, on page 525

Method: RequestContracts

Purpose	Requests all contract associated with a product
Syntax	`returnCode = myProduct.RequestContracts()`
Parameters	None
Return value	*returnCode* — Status of the method call. Possible values include: • QR_SUCCESS • QR_HANDLER_NOT_REGISTERED • QR_GATEWAY_NOT_AVAIL • QR_NOT_LOGGED_IN • QR_GAL_NOT_AVAIL • QR_INVALID_PARAMETETR Data type: `enumQueryReturnCodes`
Description	The `RequestContracts` method submits an asynchronous query for all contracts available for this product. **Note:** To receive this event, you must implement the **InstrCollection`Events`** interface and register it by calling the **InstrCollection** `RegisterEventHandler` method.
Example	None
Related information	None

Method: UnregisterEventHandler

Purpose	Unsubscribe to **`ITTProduct`** events
Syntax	*`returnCode`* `= myProduct.UnregisterEventHandler()`
Parameters	None
Return value	*returnCode* — Status of the method call. Possible values include: • RR_SUCCESS • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: `enumRegisterReturnCodes`
Description	The `UnregisterEventHandler` method unregisters the **`ITTProductEvents`** event handler. After you call this method, the application no longer responds to `OnContracts` or `OnNewContract` events.
Example	None
Related information	Method:RegisterEventHandler, on page 523

29 ITTProductCollection Class

Chapter overview

The chapter describes the **`ITTProductCollection`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
ITTProductCollection class overview	528
Property:Count	529
Property:Item	530

ITTProductCollection class overview

Overview

The `ITTProductCollection` class contains a collection of product (`ITTProduct`) objects.

Properties

Table 94 lists the `ITTProductCollection` properties.

Property	Description
Property:Count	Returns the number of product objects contained in the collection
Property:Item	Returns the product at the specified index in a collection

Table 94. ITTProductCollection properties

Methods

None

Events

None

Property: Count

Purpose	Returns the number of product objects contained in the collection
Syntax	`prodCount = myProductCol.Count`
Parameters	None
Return value	*prodCount* — Number of products in the collection Data type: Integer
Exceptions	None
Description	The `Count` property returns the number of **`ITTProduct`** objects contained in the collection.
Example	None
Related information	Property:Item, on page 530

Property: Item

Purpose	Returns the product at the specified index in a collection
Syntax	`prodObj = myProductCol.Item(VARIANT index)`
Parameters	*index* — Position in the collection of the desired product Data type: Integer or String
Return value	*prodObj* — Product object at the specified location Data type: ITTProduct
Description	The `Item` property returns a product from the **`ITTProductCollection`** collection. **Note:** The **InstrCollection** collection uses a 1-based index.
Example	None
Related information	Property:Count, on page 529

30 ITTProductEvents Class

Chapter overview

The chapter describes the `ITTProductEvents` class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
Event:OnContracts	533
Event:OnNewContract	535
Event:OnRequestContractsTimeout	537

ITTProductEvents class overview

Overview

The **ITTProductEvents** class contains

You must implement the class to make its members available to your application, as shown in the following code fragment.

```
'Import the MEAPI class librabry
Imports TTMarketExplorer
Imports System.IO

'**********************************************************************
'*          MAIN CLASS
'**********************************************************************
Public Class frmMEAPITestTool
    Implements TTMarketExplorer.ITTMarketExplorerEvents
    Implements TTMarketExplorer.ITTGatewayEvents
    Implements TTMarketExplorer.ITTProductEvents
```

Properties

None

Methods

None

Events

Table 95 lists the **ITTProductEvents** events.

Property	Description
OnContracts	TT Gateways deliver a list of contracts associated with the product
OnNewContract	Exchanges lists new contracts on the TT Gateway
OnRequestContractsTimeout	Contract download times out

Table 95. ITTProductEvents properties

Event: OnContracts

Triggered when

TT Gateways deliver a list of contracts associated with the product

Syntax

```
Sub myProdEvents_OnContracts(ITTGateway gateway, ITTProduct product,
ITTContractCollection contracts)
```

Parameters

gateway	Object representing the TT Gateway on which the product contracts trade. Data type: `ITTGateway`
product	Object representing the product associated with the contracts. Data type: `ITTProduct`
contracts	Collection of contracts available for the associated product. Data type: `ITTContractCollection`

Description

The XTAPI triggers the `OnContracts` event in response to an **`ITTProduct`** RequestContracts method call. This event returns a list of all contracts a TT Gateway lists for a product. You can use the gateway and product parameters to associate this event with its corresponding request call.

Note: To receive this event, you must implement the **`ITTProductEvents`** interface and register it by calling the **`ITTProduct`** `RegisterEventHandler` method.

Example

The following shows a sample callback for the `OnContracts` event. It fills a list box in the application.

```
Public Sub OnContracts(ByVal Gateway As TTMarketExplorer.ITTGateway, ByVal
Product As TTMarketExplorer.ITTProduct, ByVal contracts As
TTMarketExplorer.ITTContractCollection) Implements
TTMarketExplorer.ITTProductEvents.OnContracts

  'Declare Local variables
  Dim oContract As TTMarketExplorer.ITTContractInfo

  For Each oContract In contracts
    'Add all the Contract objects to a Contract Array List
    gaContractsList.Add(oContract)

    'Add the name of the contract to the contracts List box
    lbxContracts.Items.Add(oContract.Name)
  Next

  'Count the number of Contracts in the Contracts listbox, show number in the
label
  giContractsCount = lbxContracts.Items.Count()
  lblContractsNumber.Text = giContractsCount

End Sub
```

Related information

Event:OnNewContract, on page 535

ITTContractCollection Class, on page 457

`ITTProduct` Method:RegisterEventHandler, on page 523

Event: OnNewContract

Triggered when

Exchanges lists new contracts on the TT Gateway

Syntax

```
Sub myProdEvents_OnNewContract(ITTGateway gateway, ITTProduct
product, ITTContractInfo contractInfo)
```

Parameters

gateway	Object representing the TT Gateway on which the contract's associated product trades. Data type: `ITTGateway`
product	Object representing the product associated with the contract. Data type: `ITTProduct`
contractInfo	Object representing the new contract. Data type: `ITTContractInfo`

Description

The XTAPI triggers the `OnNewContract` event when an exchange adds a new contract for one of its products. You can access the properties for the *`contractInfo`* object to get details about the new contract.

You can use the gateway and product parameters to associate this event with its corresponding request call.

Note: To receive this event, you must implement the **`ITTProductEvents`** interface and register it by calling the **`ITTProduct`** `RegisterEventHandler` method.

Example

The following example shows a sample callback for the `OnNewContract` event.

```
Public Sub OnNewContract(ByVal Gateway As TTMarketExplorer.ITTGateway, ByVal
Product As TTMarketExplorer.ITTProduct, ByVal contractInfo As
TTMarketExplorer.ITTContractInfo) Implements
TTMarketExplorer.ITTGatewayEvents.OnNewContract

  'Add a message to the message list box for each new contract on that gateway
  WriteToMessagebox("Event OnNewContract Fired for the contract " &
 contractInfo.Name)
  WriteToMessagebox(" for the product " & Product.Name & " On Gateway " &
 Gateway.Name)

  'Code to update window omitted

End Sub
```

Related information

Event:OnContracts, on page 533

ITTContractInfo Class, on page 461

`ITTProduct` Method:RegisterEventHandler, on page 523

Event: OnRequestContractsTimeout

Triggered when	Contract download times out
Syntax	`Sub myProdEvents_OnRequestContractsTimeout(ITTGateway gateway, ITTProduct product)`
Parameters	*gateway* — Object representing the TT Gateway on which the contract's associated product trades. Data type: `ITTGateway` *product* — Object representing the product associated with the contract. Data type: `ITTProduct`
Description	The XTAPI triggers the `OnRequestContractsTimeout` event when a timeout occurs while downloading contracts for the specified product. This event would be triggered, for example, if the Price Server went down while your application is downloading contracts for a product. **Note:** When a Price Server goes down, the TT Gateway notifies the XTAPI, which then waits for the timeout period. Consequently, your application might take over a minute to receive this event when a Price Server goes down. You can use the gateway and product parameters to associate this event with its corresponding request call. **Note:** To receive this event, you must implement the **`ITTProductEvents`** interface and register it by calling the **`ITTProduct`** `RegisterEventHandler` method.
Example	None
Related information	Event:OnContracts, on page 533 ITTContractInfo Class, on page 461 **`ITTProduct`** Method:RegisterEventHandler, on page 523

31 TTMarketExplorer Class

The chapter describes the **`TTMarketExplorer`** class. It provides an overview of the class and includes detailed information about its methods and events.

In this chapter

Section	Page
TTMarketExplorer class overview	540
Method:MEAPITerminate	541
Method:RegisterEventHandler	542
Method:RequestGateways	544
Method:UnregisterEventHandler	545

TTMarketExplorer class overview

Overview

The **`TTMarketExplorer`** class represents the top-level class in the MEAPI namespace and serves as the entry point for a market explorer application. This class provides access to TT Gateways for request product information.

Properties

None

Methods

Table 96 lists the **`TTMarketExplorer`** methods.

Property	Description
MEAPITerminate	Terminates the Market Explorer API application
RegisterEventHandler	Subscribes for `TTMarketExplorer` events
RequestGateways	Requests the TT Gateways currently visible to the application
UnregisterEventHandler	Unsubscribes to `TTMarketExplorer` events

Table 96. TTMarketExplorer properties

Events

None

Method: MEAPITerminate

Purpose	Terminates the Market Explorer API application
Syntax	`myMktExp.MEAPITerminate()`
Parameters	None
Return value	None
Description	The `MEAPITerminate` method terminates the current Market Explorer application and performs system garbage-collection cleanup operations. This method ensures that all services and objects are closed and deleted from memory. TT recommends that you close all production applications with this method instead of just ending the application.
Example	None
Related information	**XTAPI TTGate** Method:XTAPITerminate, on page 155

Method: RegisterEventHandler

Purpose

Subscribes for **`TTMarketExplorer`** events

Syntax

returnCode = myMktExp.RegisterEventHandler(TTMarketExplorerEvents *mktExplEventHandler*)

Parameters

mktExplEventHandler	Pointer to the implementation of the **`ITTMarketExploreEvents`** interface. Data type: `TTMarketExplorerEvents`

Return value

returnCode	Status of the method call. Possible values include: • RR_SUCCESS • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: `enumRegisterReturnCodes`

Description

The `RegisterEventHandler` method registers with the Market Explorer API for **`TTMarketExplorer`** events. You must call this method if you want to be notified when events occur.

If you want to discontinue receiving event notifications, you can call the `UnregisterEventHandler` method.

Example

The following example registers for gateway events. If an error occurs, the code displays the error message. The main class implements the **`ITTMarketExplorerEvents`** class, so the example can use the Me identifier.

```
Public Class myMarketExplorerEventHandler
               Implements TTMarketExplorer.ITTMarketExplorerEvents

  Public Sub OnGateway(ByVal gateway As TTMarketExplorer.ITTGateway)
            Implements TTMarketExplorer.ITTMarketExplorerEvents.OnGateway
    ' <do something here>
  End Sub
End Class

oMyMarketExplorerEventHandler = myMarketExplorerEventHandler

geRRCode =
oMarketExplorer.RegisterEventHandler(oMyMarketExplorerEventHandler)

'Call register event handler method and check the return code
'If the return code is RR_SUCCESS then make the label with the Gateway Name
green
'Print the return code in the message list box
If geRRCode = enumRegisterReturnCode.RR_SUCCESS Then
  'Print the return code in the message list box
  MessageBox.Show "RegisterEventHandler was successful."
Else
  MessageBox.Show "RegisterEventHandler returned " + geRRCode.ToString()
End If
```

Related information

Method:RequestGateways, on page 544

Method:UnregisterEventHandler, on page 545

`ITTMarketExplorerEvents` Event:OnGateway, on page 513

Method: RequestGateways

Purpose	Requests the TT Gateways currently visible to the application
Syntax	`returnCode = myMktExp.RequestGateways()`
Parameters	None
Return value	*returnCode* — Status of the method call. Possible values include: • QR_SUCCESS • QR_HANDLER_NOT_REGISTERED • QR_GATEWAY_NOT_AVAIL • QR_NOT_LOGGED_IN • QR_GAL_NOT_AVAIL • QR_INVALID_PARAMETETR Data type: `enumQueryReturnCodes`
Description	The `RequestGateways` method submits an asynchronous query for all TT Gateways visible to the XTAPI application. As each TT Gateway is found, the MEAPI fires an `OnGateway` event that contains the identity and status of the TT Gateway. **Note:** To receive this event, you must implement the **InstrCollection`Events`** interface and register it by calling the **InstrCollection** `RegisterEventHandler` method.
Example	None
Related information	Method:RegisterEventHandler, on page 542 Event: OnGatewayFound, on page 550

Method: UnregisterEventHandler

Purpose	Unsubscribes to **`TTMarketExplorer`** events
Syntax	*`returnCode`* `= myMktExp.UnregisterEventHandler(TTMarketExplorerEvents` *`mktExplEventHandler`*`)`
Parameters	*`mktExplEventHandler`* — Pointer to the implementation of the **`ITTMarketExploreEvents`** interface. Data type: `TTMarketExplorerEvents`
Return value	*`returnCode`* — Status of the method call. Possible values include: • RR_SUCCESS • RR_ALREADY_REGISTERED • RR_HANDLER_NOT_REGISTERED • RR_INVALID_PARAMETER Data type: `enumRegisterReturnCodes`
Description	The `UnregisterEventHandler` method unregisters the **`ITTMarketExplorerEvents`** event handler. After you call this method, the application no longer responds to `OnGateway` events.
Example	None
Related information	Method:RegisterEventHandler, on page 542 **`ITTMarketExplorer`** Event:OnGateway, on page 513

A XTAPI Enumerations and Codes

Appendix overview

This appendix describes the XTAPI namespace enumerations and codes used by the class properties, methods, and events.

In this chapter

Section	Page
Constants and enumerations	548
Codes, types, and restrictions	552

Constants and enumerations

Constants

The XTAPI namespace defines the following constants:

- TT_INVALID_PRICE
- TT_INVALID_QTY

Enumerations

The XTAPI namespace defines the following enumerations.

- AvgOpenPriceMode
- DebugLogLevels
- GateStatus
- LoginSource
- OrderAction
- OrderNotifyState
- OrderStatusNotifyMode
- PriceID
- RoundOptions
- RoundPriceType
- SessionRollState
- WorkupState
- WorkupType

When using an enumeration value, you must specify the full enumeration (as in *`enumPriceID.value)`*. For brevity sake, this manual omits the `enum` prefix in the section headings.

Note: Some languages, such as C#, require you to qualify enumeration values with the namespace, such as XTAPI.enumPriceID.value.

AvgOpenPriceMode

Table 97 lists the values that determine how to calculate the average price of an open instrument.

Enumeration	Value
NONE	0
FIFO	1
LIFO	2

Table 97. enumAvgOpenPriceMode values

DebugLogLevels

Table 98 lists the values you can use to control the level of debugging information generated by the application.

Enumeration	Value
LOG_MINIMAL	1
LOG_MIN_API	3
LOG_DET_API	4
LOG_ORDERS	8
LOG_COM	0x10
LOG_COM_USE	0x20
LOG_COM_ERR	0x30
LOG_COM_EVENTS	0x40

Table 98. enumDebugLogLevels values

Enumeration	Value
LOG_ALL_COM (LOG_COM | USE | ERR | EVENTS)	0x70
LOG_FILLS	0x100
LOG_DEBUG_WIN	0x1000
LOG_ALL	0xFFFFFFFF

Table 98. enumDebugLogLevels values(Continued)

GateStatus

Table 99 lists the values that represent the state of a TT Gateway.

Enumeration	Value
GATE_INIT	0
GATE_XT_CONNECT	1
GATE_XT_CLOSED	2
GATE_XT_EXCHANGE	4
GATE_XT_CUSTLIST	8
GATE_XT_UNAVAILABLE	16
GATE_XT_AVAILABLE	32
GATE_XTAPI_SERVERMODE	64
GATE_CURRENCY_TABLE	128
GATE_LOAD_WORKSPACE	256
GATE_SAVE_WORKSPACE	512
GATE_TT_TRADER_MODE	1024
GATE_CLOSE_ALL_WINDOWS	2048

Table 99. enumGateStatus values

GatewayType

Table 100 lists the values that indicate whether a TT Gateway operates in a PFX environment.

Enumeration	Description
GATEWAY_TYPE_PFX	PFX Gateway
GATEWAY_TYPE_LEGACY	Non-PFX Gateway
GATEWAY_TYPE_UNKOWN	Instrument not yet opened

Table 100. enumGatewayType values

LoginSource

Table 101 lists the values that indicate how a trader logged into a TT Gateway.

Enumeration	Description
LOGIN_XTAPI	Logged into service mode through the XTAPI application
LOGIN_XT	X_TRADER login propagated to the XTAPI application

Table 101. enumLoginSource values

OrderAction

Table 102 lists the order action enumerations.

Enumeration	Description
ORD_ACTION_NONE	Internal initializer
ORD_ACTION_ADD	Order added to the market
ORD_ACTION_CHANGE	Some aspect of the order changed
ORD_ACTION_DELETE	Order deleted from the market
ORD_ACTION_SUBMIT	Held order was resubmitted
ORD_ACTION_HOLD	Working order was put on hold
ORD_ACTION_REPLACE	Cancel/Replace (delete and re-add) order
ORD_ACTION_UNKNOWN	Invalid order action

Table 102. enumOrderAction values

OrderNotifyState

Table 103 lists the possible states returned with an order status callback.

Enumeration	Description
ORD_STATE_NONE	Order does not yet exist or has not been acknowledged by the exchange
ORD_STATE_WORKING	Order is active in the market
ORD_STATE_HOLD	Order has been put on hold
ORD_STATE_COMPLETED	Order life cycle completed; order has been fully filled or deleted
ORD_STATE_UNKNOWN	Order could not be retrieved

Table 103. enumOrderNotifyState values

OrderStatusNotifyMode

Table 104 lists the possible modes for order status updates.

Enumeration	Description
ORD_NOTIFY_NONE	Order status callback not enabled
ORD_NOTIFY_NORMAL	Callbacks for common order status changes
ORD_NOTIFY_VERBOSE	Callbacks for all order status changes

Table 104. enumOrderStatusNotifyMode values

PriceID

Table 105 lists the values for the different PriceID enumerations.

Enumeration	Value	Enumeration	Value	Enumeration	Value
ttUnknown	-1	ttOpenPrice	10	ttOTCTradePrice	31
ttLastTradedPrice	5	ttClosePrice	11	ttOTCTradeQty	32
ttTotalTradedQty	6	ttSettlePrice	12	ttTradeState	34
ttLastTradedQty	7	ttSeriesStatus	17	ttIndicativeSettle	41
ttHighPrice	8	ttUpperLimitPrice	18	ttHitTake	42
ttLowPrice	9	ttLowerLimitPrice	19	ttIndicativePrice	43

Table 105. enumPriceID values

RoundOptions

Table 106 lists the values you can use to set the approach to use when rounding prices.

Enumeration	Value
NO_ROUND	0
AGGRESSIVE_ROUND	1
CONSERVATIVE_ROUND	2

Table 106. enumRoundOptions values

RoundPriceType

Table 107 lists the values for specifying which direction to use when rounding prices.

Enumeration	Value
ROUND_NONE	0
ROUND_UP	1
ROUND_DOWN	2
ROUND_NEAREST	3

Table 107. enumRoundPriceType values

SessionRollState

Table 108 lists the enumSessionRollState enumeration values.

Value	Description
SESSION_ROLL_UNKNOWN	Internal error occurred
SESSION_ROLL_DETECTED	Session roll detected when the gateway session ID changed
SESSION_ROLL_START	Session roll started
SESSION_ROLL_END	Session roll completed

Table 108. SessionRollState enumeration

WorkupState

Table 109 lists the valid workup state values.

Enumeration	Value
WORKUP_STATE_NONE	0
WORKUP_STATE_PRIVATE	1
WORKUP_STATE_PUBLIC	2

Table 109. enumWorkupState values

WorkupType

Table 110 lists the valid workup type values.

Enumeration	Value
WORKUP_STATE_NONE	0
WORKUP_STATE_HIT	1
WORKUP_STATE_TAKE	2

Table 110. enumWorkupType values

Codes, types, and restrictions

Overview

The XTAPI namespace uses the following codes types, and restrictions for object attribute values:

- Account types
- Combination codes
- Currency codes
- Order restrictions
- Order status
- Order types
- Price types
- Product status types
- Product types

Account types

Table 111 lists the AccountType (AcctType) values.

Value	Description	Value	Description
A1	First Agent	M1	First Market Maker
A2	Euronext Products: Customer/ Automatic Allocation	M2	Second Market Maker
A3	Euronext Products: Customer/ Give-up or System Allocation	M3	Market Maker/Give-up or System Allocation
A4	Fourth Agent	P1	First Principal
A5	Fifth Agent	P2	Second Principal
A6	Sixth Agent	P3	Euronext Products: House/Give-up or System Allocation
A7	Seventh Agent	U1	Unallocated
A8	Eighth Agent	U2	Unallocated/Automatic
A9	Ninth Agent	U3	Unallocated/System
G1	Pre-Designated Give-up Trade		
G2	Designated Give-up Trade/ Automatic Allocation		
G3	Give-up System Allocation		

Table 111. AccountTypes codes

Combination codes

Table 112 lists the ComboCode values.

Value	Description
""	Fill for an outright order
C	Calendar spread fill
F	All other spread fills

Table 112. ComboCodes codes

Currency codes

Table 113 lists the Currency codes.

Value	Currency	Value	Currency	Value	Currency
ARS	Argentina Peso	ESP	Spanish Peseta	KRW	South Korean Won
ATS	Austrian Shilling	EUR	Euro	NLG	Netherlands Gilder
AUD	Austrian Dollar	FRF	French Franc	PTE	Portugese Escudo
BEF	Belgian Franc	GBP	British Pound	RUR	Russian Ruple
BRL	Brazilian Real	HKD	Hong Kong Dollar	SGD	Singapore Dollar
CAD	Canadian Dollar	INR	Indian Rupee	USD	US Dollar
CHF	Swiss Franc	ITL	Italian Lira	XBA	European Composite Unit
DEM	Deutsche Mark	JPN	Japanese Yen	XEU	European Currency Unit

Table 113. Currency codes

Order restrictions

Table 114 lists the OrderRestr values.

Value	String	Name	Description
" "	<space>	None	Type of order that specifies the order can only be activated if the full order will be filled.
A	Auction	Auction Order	Auction order entered during the pre-market opening period for execution at the Calculated Opening Price (COP). If the order is not filled on the open, the order is re-submitted as a limit order with the limit price set to the COP or the best bid/ask after the market opens.
B	Block	Block Order	Order, usually quite large in size, that is negotiated between two parties. Such parties prefer receiving a single, negotiated price, even if the price is slightly discounted from the current market, to avoid the risk of receiving multiple, unknown fills to execute their large futures order.
E	Iceberg	Iceberg (hidden qty)	Order where only a portion of the total order quantity is sent to the market at a time. Each disclosed order quantity is equal. Iceberg orders allow you to set the quantity parameter as a number or percentage.
F	FOK	Fill or Kill	Market or limit-price order to be executed in its entirety as soon as it is represented in the market; if not executed in full, the order is canceled.
H	FAS	Fill and Save	Limit orders
I	IOC	Immediate or Cancel	Order executed against any existing orders at the stated price or better, up to the volume of the IOC order. Any additional volume from the IOC order is cancelled.
M	MOC	Market on Close	Order to trade stocks, options, or futures as close as possible to the market close. An MOC order instructs the broker to fill the order at the market during the close. The order must be filled at a price that falls within the closing range.
N	AON	All or None	Order that specifies the order can only be activated if the full order will be filled.

Table 114. OrderRestr codes

Value	String	Name	Description
O	MOO	Market on Open	Order to trade stocks, options, or futures as close as possible to the market open. A MOO order instructs the broker to fill the order at the market during the open. The order must be filled at a price that falls within the opening range.
S	Stop	Stop Order	Order that becomes a market order if and when the market reaches a designated price.
T	Trigger	Trigger Order	Order entered at a trigger price.
V	MV	Minimum Volume	Order that is executed only if a minimum volume available, at the stated price or better. If not, the whole order is cancelled. Any additional volume from a partially executed minimum volume order is retained in the central order book.
X	Flex	Flex Option	Order where the buyer and seller are permitted to negotiate the exact terms of the option contract. The order can include the exercise price, the expiration date, and the terms of exercise.
Y	Volatility	Volatility	Order that allows you to trade volatility by calculating the limit price as the function of the implied volatility.
Z	Basis	Basis Order	Order based on the price difference between the cash price and the futures price observed in the market. It is used to determine capital gains or losses for tax purposes when the stock is sold.
a	LOO	Limit Order on Open	Limit order that will either be filled or killed when the market opens.
b	LOC	Limit Order on Close	Limit order that will either be filled or killed when the market closes.
c	LTM	Limit to Market Order on Close	Limit order that converts to a market order when the market closes.

Table 114. OrderRestr codes(Continued)

Order status

Table 115 lists the OrdStatus values.

Value	String	Description	Value	String	Description
0	Void	Default value	8205	Deleted	Order deleted on the exchange
8200	New	New order released by client	8206	Expired	Order expired
8201	Accepted	Order accepted by the server	8207	Rejected	Order rejected by the exchange
8202	OK	Order active in the market	8208	Resubmitted	Hold order resubmitted
8203	XT Filled	Order filled	8299	Unknown	Order status could not be retrieved
8204	Hold	Order put on hold			

Table 115. OrdStatus codes

Order types

Table 116 lists the OrderType codes.

Value	Description	Value	Description
B	OCO Order	M	Market Order
BL	Best Limit order	O	Batch Order
C	CMO Order	Q	Quote Order
L	Limit Order	X	Cross Order

Table 116. OrderType codes

Price types

Table 117 lists the PriceType codes.

Code	Description	Code	Description
0	Precision and decimals values, which is the normal Price Display.	28	N/A
1	Thirty-seconds (32nds)	29	N/A
2	Yield price (format 1)	30	BrokerTec 1/64: 1.00, 1.00+, 1.01, 1.01+... 1.31, 1.31+, 2.00
3	Yield price (format 2)	31	BrokerTec 2/8: 1, 1.2, 1+, 1.6, 2
4	Discount price	32	BrokerTec 2/8 of 1/32: 1.00, 1.002, 1.00+, 1.006, 1.01.....1.31, 1.312, 1.31+, 1.316, 2
5	Decimal price	33	BrokerTec 1/8 of 1/32: 1.00, 1.001, 1.002, 1.003, 1.00+, 1.005, 1.006, 1.007, 1.01
6	Basis price	34	BrokerTec Both 1/100 and 1/8 (104 ticks per point): 1.000, 1.010, 1.020...1.120, 1.125, 1.130
7	Roll price	35	BrokerTec Normal - Uses precision and decimals values, but with series instead of prod if there is no tick table
8	Spread price	36	BrokerTec Points.FFQ FF=00..63 Q=1,+,3
9	512th price	37	BrokerTec 32nds.Q, Q=1,2,3,+,5,6,7
10	Half-cents (last digit is either 0 or 5)	38	Quarter thirty-seconds 4x - Price is in quarter thirty-seconds but ticks 4 at a time
11	Fourths (last digit is either 0, 2, 4, or 6)	39	Quarter thirty-seconds 2x - Price is in quarter thirty-seconds but ticks 2 at a time
12	Modified fourths (last digit is either 0, 2, 5, or 7)	40	BrokerTec Yield - Price has 400 ticks per point, displays with 4 digits after decimal point
13	Eighths	41	Half-cents 2x - Price is in half cents (last digit 0, 5), but ticks 2 at a time
14	Modified decimal fourths (last digit is either 0, 2, 5, or 7)	42	Normal 2x - Normal type, but tick 2 at a time
15	Thirty-seconds (Differs from 1 in that it does not use + or decimal places)	43	Normal 5x - Normal type, but ticks 5 at a time

Table 117. PriceType codes

Code	Description	Code	Description
16	Sixty-fourths (64ths)	44	Normal 100x - Normal type, but ticks 100 at a time
17	Half thirty-seconds	45	Five one-thousandths, but ticks 2 at a time
18	Quarter thirty-seconds	46	Five one-thousandths, but ticks 5 at a time
19	Half sixty-fourths	47	Five one-thousandths, but ticks 10 at a time
20	Fives (1/20 points)	48	Ten-hundredths of a cent, but ticks 2 at a time
21	Half points	49	Ten-hundredths of a cent, but ticks 5 at a time
22	Five one-thousandths	50	Ten-hundredths of a cent, but ticks 10 at a time
23	One-hundredths	51	One-hundredths, but ticks 1/4 of 100th at a time
24	Tenths (dimes)	52	Normal 10x - Normal type, but ticks 10 at a time
25	Ten-hundredths of a cent	53	Normal 50x - Normal type, but ticks 50 at a time
26	No decimals	54	Normal 500x - Normal type, but ticks 500 at a time
27	Modified 1/40 points (last digit is 0, 2, 5, or 7)	55	Normal 25x - Normal type, but ticks 25 at a time

Table 117. PriceType codes(Continued)

Product status types

Table 118 lists the possible product statuses.

Value	Status	Description
0	UKNOWN	Unknown status
1	PRE-TRADING	Pre-trading phase
2	TRADING	Normal trading
3	FAST-MARKET	Fast market or volatility interruption
4	AUCTION	Product in auction
5	POST-TRADING	Post-trading phase
6	CLOSED	Market closed
7	NOT-TRADEABLE	Product not tradeable (used for price feeds)
8	PRE-OPEN	Pre-open phase
9	FREEZE	Market frozen
10	EXPIRED	Series expired
11	OPENING AUCTION	Product in day open auction
12	CLOSING AUCTION	Product in day close auction
13	RESERVE	Market in reserve mode (CME only)

Table 118. Product status types codes

Value	Status	Description
14	PRICE DISC	Market in price discovery mode, similar to pre-open
15	LEVEL	Market in leveling phase, which determines the indicative market opening price

Table 118. Product status types codes(Continued)

Product types

Table 119 lists the ProdType codes.

Value	String	Description	Value	String	Description
0	Invalid	Use for enumerating values	16	2YR	Two-year
1	Future	Future	17	2YR Swap	Two-year swap
2	Spread	Spread	18	2YR Basis	Two-year basis
3	Option	Option	19	5YR	Five-year
4	Strategy	Strategy	20	5YR Swap	Five-year swap
5	Stock	Stock	21	5YR Basis	Five-year basis
6	Bond	Bond	22	10YR	Ten-year
7	Swap	Swap	23	10YR Swap	Ten-year swap
8	Warrant	Warrant	24	10YR Basis	Ten-year basis
9	Active	Garban products	25	30YR	Thirty-year
10	When Issue	When Issue	26	30YR Swap	Thirty-year swap
11	Roll	Roll	27	30YR Callable	Thirty-year callable
12	Swap Anchor	Swap anchor	28	30YR Basis	Thirty-year basis
13	Yield Curve	Yield curve	29	30YR OTR Swap	Thirty-year OTR swap
14	Bill	Bill	30	Energy	Energy
15	Under 2YR	Under two-year	31	FOREX	FOREX
			33	NDF	NDF

Table 119. ProdType codes

B MEAPI Types and Enumerations

Appendix overview

This appendix describes the MEAPI namespace enumerations and constants used by its class properties, methods, and events.

In this chapter

Section	Page
Constants and Enumerations	560

Constants and Enumerations

Constants

The MEAPI namespace defines the constant, `TT_INVALID_PRICE`.

Enumerations

The MEAPI namespace defines the following enumerations.

- CallPutCode
- ComboCode
- LogLevel
- PriceDisplayFormat
- PriceDisplayType
- ProductType
- QueryReturnCode
- RegisterReturnCode
- ServiceType

When using an enumeration value, you must specify the full enumeration (as in `enumCallPutCode.`*`value`*). For brevity sake, this manual omits the enum prefix in the section headings.

Note: Some languages, such as C#, require you to qualify enumeration values with the namespace, such as MEAPI.enumPriceID.value.

CallPutCode

Table 120 lists the call and put codes used for options contracts.

Enumeration	Description
TT_CALL	Call option
TT_PUT	Put option
TT_NO_CALL_PUT_CODE	Other

Table 120. enumCallPutCode enumeration

ComboCode

Table 121 lists the codes used for combination contracts.

Enumeration
TT_NO_COMB_ID
TT_STRANGLE_COMB_ID
TT_STRADDLE_COMB_ID
TT_GUTS_COMB_ID
TT_LADDER_COMB_ID
TT_VOLA_COMB_ID
TT_IRON_BUTTERFLY_COMB_ID
TT_BOX_COMB_ID
TT_COMBO_COMB_ID
TT_2X1RATIO_COMB_ID
TT_CALLPUT_COMB_ID
TT_CONV_REVERS_COMB_ID
TT_DIAG_CALENDAR_COMB_ID
TT_STRAD_CALENDAR_COMB_ID

Table 121. enumComboCode enumeration

Enumeration
TT_DIAG_STRAD_CAL_COMB_ID
TT_INTERPROD_COMB_ID
TT_CALENDAR_COMB_ID
TT_CONDOR_COMB_ID
TT_BUTTERFLY_COMB_ID
TT_STRIP_COMB_ID
TT_PACK_COMB_ID
TT_BUNDLE_COMB_ID
TT_SYNTH_CONV_REVERS_COMB_ID
TT_CALL_SPREAD_VERS_PUT_COMB_ID
TT_PUT_SPREAD_VERS_CALL_COMB_ID
TT_STRAD_VERS_CALLPUT_COMB_ID
TT_HEDGED_CALLPUT_COMB_ID
TT_HEDGED_STRAD_COMB_ID
TT_HEDGED_CALL_SPREAD_VERS_PUT_COMB_ID
TT_HEDGED_PUT_SPREAD_VERS_CALL_COMB_ID
TT_HEDGED_CALLPUT_LADDER_COMB_ID
TT_HEDGED_COMBO_COMB_ID
TT_HEDGED_2X1RATIO_COMB_ID
TT_HEDGED_CALENDAR_COMB_ID
TT_JELLY_ROLL_COMB_ID
TT_IRON_CONDOR_COMB_ID
TT_OPTION_STRIP_COMB_ID
TT_REDUCED_TICK_SPREAD_COMB_ID
TT_HEDGED_STRANGLE_COMB_ID
TT_FUTURE_SPREAD_COMB_ID
TT_HEDGED_DIAG_CALENDAR_COMB_ID
TT_HEDGED_STRAD_CALENDAR_COMB_ID
TT_HEDGED_DIAG_STRAD_CAL_COMB_ID
TT_HEDGED_BUTTERFLY_COMB_ID
TT_HEDGED_IRON_CONDOR_COMB_ID
TT_HEDGED_CONDOR_COMB_ID
TT_HEDGED_IRON_BUTTERFLY_COMB_ID
TT_HEDGED_GUTS_COMB_ID
TT_CABINET_COMB_ID
TT_HEDGED_CONV_REVERS_COMB_ID
TT_BUY_WRITE_COMB_ID
TT_VERTICAL_COMB_ID
TT_CUSTOM_MARKET_COMB_ID
TT_COVERED_COMB_ID

Table 121. enumComboCode enumeration(Continued)

Enumeration
TT_HORIZONTAL_COMB_ID
TT_XMAS_TREE_COMB_ID
TT_RISK_REVERSAL_COMB_ID
TT_HORIZONTAL_STRADDLE_COMB_ID
TT_DOUBLE_COMB_ID
TT_COND_CURVE_COMB_ID
TT_STRADDLE_STRIP_COMB_ID
TT_3WAY_COMB_ID
TT_1X2RATIO_COMB_ID
TT_1X3RATIO_COMB_ID
TT_2X3RATIO_COMB_ID
TT_PACK_BUTTERFLY_COMB_ID
TT_DOUBLE_BUTTERFLY_COMB_ID
TT_MONTH_PACK_COMB_ID
TT_CRACK_COMB_ID
TT_HEDGED_VERTICAL_COMB_ID
TT_HEDGED_HORIZONTAL_COMB_ID
TT_HEDGED_XMAS_TREE_COMB_ID
TT_HEDGED_STRIP_COMB_ID
TT_HEDGED_RISK_REVERSAL_COMB_ID
TT_HEDGED_HORIZONTAL_STRADDLE_COMB_ID
TT_HEDGED_DOUBLE_COMB_ID
TT_HEDGED_COND_CURVE_COMB_ID
TT_HEDGED_STRADDLE_STRIP_COMB_ID
TT_HEDGED_3WAY_COMB_ID
TT_HEDGED_BOX_COMB_ID
TT_TAILOR_MADE_COMB_ID
TT_ICS_COMB_ID

Table 121. enumComboCode enumeration(Continued)

LogLevel

Table 122 lists the possible levels of detatil to write to log files.

Enumeration	Description
LOG_ERROR	Log error messages
LOG_MIN_API	Log API events with minimal detail
LOG_MAX_API	Log API events with more detail
LOG_COM	Write log messages to the system console
LOG_DEBUG_WIN	Write log messages to the debug window
LOG_ALL	Log everything

Table 122. enumLogLevel enumeration

PriceDisplayFormat

Table 123 lists the possible price type display formats.

Enumeration	Description
TT_DECIMAL	Display price as a decimal value
TT_FRACTION	Display price in a fractional format
TT_FORMAT_UNKNOWN	Invalid format

Table 123. enumPriceDisplayFormat enumeration

PriceDisplayType

Table 124 lists the possible price display types.

Enumeration	Description
TT_PRICE_DISPLAY_32	Thirty-seconds (32nds)
TT_PRICE_DISPLAY_YIELD1	Yield price (format 1)
TT_PRICE_DISPLAY_YIELD2	Yield price (format 2)
TT_PRICE_DISPLAY_DISCOUNT	Discount price
TT_PRICE_DISPLAY_DECIMAL	Decimal price
TT_PRICE_DISPLAY_BASIS	Basis price
TT_PRICE_DISPLAY_ROLL	Roll price
TT_PRICE_DISPLAY_SPREAD	Spread price
TT_PRICE_DISPLAY_512TH	512th price
TT_PRICE_DISPLAY_HALVES	Half-cents (last digit is either 0 or 5)
TT_PRICE_DISPLAY_FOURTHS	Fourths (last digit is either 0, 2, 4, or 6)
TT_PRICE_DISPLAY_MODIFIED_FOURTHS	Modified fourths (last digit is either 0, 2, 5, or 7)
TT_PRICE_DISPLAY_EIGHTHS	Eighths
TT_PRICE_DISPLAY_MODIFIED_DECIMAL_FOURTHS	Modified decimal fourths (last digit is either 0, 2, 5, or 7)
TT_PRICE_DISPLAY_THIRTY_SECONDS	Thirty-seconds (Differs from TT_PRICE_DISPLAY_32 in that it does not use + or decimal places)
TT_PRICE_DISPLAY_SIXTY_FOURTHS	Sixty-fourths (64ths)
TT_PRICE_DISPLAY_HALF_THIRTY_SECONDS	Half thirty-seconds
TT_PRICE_DISPLAY_QUARTER_THIRTY_SECONDS	Quarter thirty-seconds
TT_PRICE_DISPLAY_HALF_SIXTY_FOURTHS	Half sixty-fourths
TT_PRICE_DISPLAY_FIVES	Fives (1/20 points)
TT_PRICE_DISPLAY_HALF_POINTS	Half points
TT_PRICE_DISPLAY_FIVE_ONE_THOUSANDTHS	Five one-thousandths
TT_PRICE_DISPLAY_ONE_HUNDREDTHS	One-hundredths
TT_PRICE_DISPLAY_TENTHS	Tenths (dimes)
TT_PRICE_DISPLAY_TEN_HUNDREDTHS	Ten-hundredths of a cent
TT_PRICE_DISPLAY_NO_DECIMALS	No decimals
TT_PRICE_DISPLAY_MODIFIED_ONE_FORTIETHS	Modified 1/40 points (last digit is 0, 2, 5, or 7)
TT_PRICE_DISPLAY_EIGHTH_THIRTY_SECONDS_NOT_IMPLEMENTED	N/A
TT_PRICE_DISPLAY_ISLAND_NOT_IMPLEMENTED	N/A

Table 124. enumPriceDisplayType enumeration

Enumeration	Description
TT_PRICE_DISPLAY_BTEC_1_8_OF_1_32	BrokerTec 1/8 of 1/32: 1.00, 1.001, 1.002, 1.003, 1.00+, 1.005, 1.006, 1.007, 1.01
TT_PRICE_DISPLAY_BTEC_1_100_AND_1_8	BrokerTec Both 1/100 and 1/8 (104 ticks per point): 1.000, 1.010, 1.020...1.120, 1.125, 1.130
TT_PRICE_DISPLAY_BTEC_NORMAL	BrokerTec Normal - Uses precision and decimals values, but with series instead of prod if there is no tick table
TT_PRICE_DISPLAY_BTEC_1_4_OF_1_64	BrokerTec 1/64: 1.00, 1.00+, 1.01, 1.01+... 1.31, 1.31+, 2.00
TT_PRICE_DISPLAY_BTEC_WHOLE_1_8_OF_1_32	BrokerTec 32nds.Q, Q=1,2,3,+,5,6,7
TT_PRICE_DISPLAY_QUARTER_THIRTY_SECONDS_4X	Quarter thirty-seconds 4x - Price is in quarter thirty-seconds but ticks 4 at a time
TT_PRICE_DISPLAY_QUARTER_THIRTY_SECONDS_2X	Quarter thirty-seconds 2x - Price is in quarter thirty-seconds but ticks 2 at a time
TT_PRICE_DISPLAY_BTEC_YIELD	BrokerTec Yield - Price has 400 ticks per point, displays with 4 digits after decimal point
TT_PRICE_DISPLAY_HALVES_2X	Half-cents 2x - Price is in half cents (last digit 0, 5), but ticks 2 at a time
TT_PRICE_DISPLAY_NORMAL_2X	Normal 2x - Normal type, but tick 2 at a time
TT_PRICE_DISPLAY_NORMAL_5X	Normal 5x - Normal type, but ticks 5 at a time
TT_PRICE_DISPLAY_NORMAL_10X	Normal 10x - Normal type, but ticks 10 at a time
TT_PRICE_DISPLAY_NORMAL_50X	Normal 50x - Normal type, but ticks 50 at a time
TT_PRICE_DISPLAY_NORMAL_25X	Normal 25x - Normal type, but ticks 25 at a time
TT_PRICE_DISPLAY_NORMAL_100X	Normal 100x - Normal type, but ticks 100 at a time
TT_PRICE_DISPLAY_NORMAL_500X	Normal 500x - Normal type, but ticks 500 at a time
TT_PRICE_DISPLAY_FIVE_ONE_THOUSANDTHS_2X	Five one-thousandths, but ticks 2 at a time
TT_PRICE_DISPLAY_FIVE_ONE_THOUSANDTHS_5X	Five one-thousandths, but ticks 5 at a time
TT_PRICE_DISPLAY_FIVE_ONE_THOUSANDTHS_10X	Five one-thousandths, but ticks 10 at a time
TT_PRICE_DISPLAY_TEN_HUNDREDTHS_2X	Ten-hundredths of a cent, but ticks 2 at a time
TT_PRICE_DISPLAY_TEN_HUNDREDTHS_5X	Ten-hundredths of a cent, but ticks 5 at a time
TT_PRICE_DISPLAY_TEN_HUNDREDTHS_10X	Ten-hundredths of a cent, but ticks 10 at a time
TT_PRICE_DISPLAY_QUARTER_ONE_HUNDREDTHS	One-hundredths, but ticks 1/4 of 100th at a time
TT_PRICE_DISPLAY_UNKNOWN	Invalid price display type

Table 124. enumPriceDisplayType enumeration

ProductType

Table 125 lists the supported product types.

Enumeration	Description
TT_PROD_FUTURE	Future
TT_PROD_FSPREAD	Spread
TT_PROD_OPTION	Option
TT_PROD_OSTRATEGY	Strategy
TT_PROD_STOCK	Stock
TT_PROD_BOND	Bond
TT_PROD_SWAP	Swap
TT_PROD_WARRANT	Warrant
TT_PROD_AUCTION	Garban products
TT_PROD_ACTIVE	Garban products
TT_PROD_WI	When Issue
TT_PROD_ROLL	Roll
TT_PROD_SWAP_ANCHOR	Swap anchor
TT_PROD_YIELD_CURVE	Yield curve
TT_PROD_BILL	Bill
TT_PROD_UNDER_2YR	Under two-year
TT_PROD_2YR	Two-year
TT_PROD_2YR_SWAP	Two-year swap
TT_PROD_2YR_BASIS	Two-year basis
TT_PROD_5YR	Five-year
TT_PROD_5YR_SWAP	Five-year swap
TT_PROD_5YR_BASIS	Five-year basis
TT_PROD_10YR	Ten-year
TT_PROD_10YR_SWAP	Ten-year swap
TT_PROD_10YR_BASIS	Ten-year basis
TT_PROD_30YR	Thirty-year
TT_PROD_30YR_CALLABLE	Thirty-year swap
TT_PROD_30_SWAP	Thirty-year callable
TT_PROD_30YR_BASIS	Thirty-year basis
TT_PROD_30YR_OTR_SWAP	Thirty-year OTR swap
TT_PROD_ENERGY	Energy
TT_PROD_FOREX	Forex
TT_PROD_NDF	NDF
TT_PROD_TYPES_ALL	All product types

Table 125. enumProductType enumeration

QueryReturnCode

Table 126 lists the return codes from the various query (Request*) methods.

Enumeration	Description
QR_SUCCESS	The method completed successfully.
QR_HANDLER_NOT_REGISTERED	The application did not register the event handler.
QR_GATEWAY_NOT_AVAIL	The specified TT Gateway is unavailable.
QR_NOT_LOGGED_IN	The application is not connected to the TT Gateway.
QR_GAL_NOT_AVAIL	The specified GAL setting is invalid.
QR_INVALID_PARAMETER	The parameter passed to the method is invalid for the method.

Table 126. enumQueryReturnCode enumeration

RegisterReturnCode

Table 127 lists the return codes from RegisterEventHandler or UnregisterEventHandler methods.

Enumeration	Description
RR_SUCCESS	The method completed successfully.
RR_ALREADY_REGISTERED	The application previously registered the specified event handler.
RR_HANDLER_NOT_REGISTERED	The specified event handler cannot be unregistered because it was never registered.
RR_INVALID_PARAMETER	The parameter passed to the method is invalid for the method.

Table 127. enumRegisterReturnCode enumeration

ServiceType

Table 128 lists the possible server types.

Enumeration	Description
PRICE_SERVICE	Price server

Table 128. enumServiceType enumeration

C Order Rejections Not Exposed to Clients

Overview

This appendix lists some common cases in which the gateway or exchange rejects an order but does not notify an XTAPI client application.

Situations in which your application does not receive notice of an order rejection

The following lists some situations in which your client application might not be notified of a rejected order:

- Submit requests are routed through X_TRADER, while delete requests are directly routed to the gateway.

  Normally when a request could not be submitted for some reason, the order functions return 0 (the number of contracts affected by the order). However, external situations, such as brief network or gateway outages, can prevent the reject from being returned.

- The order request could not be submitted to the gateway, or the gateway could not submit it to the exchange.

  In this case, the XTAPI does not return 0 right away and might even return a positive value to signify the successful request submission. However, in such cases, the request mighjt time-out in XTAPI and display the same value in the X_TRADER Audit Trail. In these situations, the XTAPI does not send callbacks.

- A delete request might time-out at the gateway, but the XTAPI does not know about it. This request is not visible in X_TRADER Audit Trail, and XTAPI client applications are not notified. The client might need to resubmit the order in such cases.

Send Us Your Comments

XTAPI Class Reference

7.7.9

Trading Technologies® welcomes your comments and suggestions on the accuracy and usefulness of this publication. Your input is important and valuable in revising our documentation and helps ensure a constantly improving level of quality.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- Which features did you find particularly useful?
- What did you like most about this manual or document?

If you encounter any errors in this document or would like to share other suggestions you might have for improving this document, send comments to: documentation.dept@tradingtechnologies.com.

If possible, please indicate the chapter, section, and page number relevant to your feedback.

Index

A

Acct property 35
AcctType property 36
AddHotKey method 165
AddSelector property 350
AddTest property 351
AddTestEx property 353
Alias property 517
    TTInstrObj 220
    TTOrderProfile 312
    TTOrderSet 362
Aliases
    instrument 220
    order set 362
AllMatchesRequired property 347
AllowAnyMatches property 348
Attach
    instrument 200
    order set 202
AttachInstrument method 200
AttachOrderSet method 202
AvailableServices property 485
AvgOpenPriceMode property 363

C

CalculateTTImplieds property 221
Call 463
CallPut property 463
Classes
    InstrCollection 29
    ITTContractCollection 457
    ITTContractInfo 461
    ITTCustomerProfile 33
    ITTGateway 483
    ITTGatewayEvents 501
    ITTMarketExplorerEvents 511
    ITTPriceEntry 51
    ITTPriceList 59
    ITTPriceUpdate 65
    ITTProduct 515
    ITTProductCollection 527
    ITTProductEvents 531
    OrderProfileCol 73
    OrderSetCol 77
    TTDropHandler 81
    TTFillObj 95
    TTGate 111
    TTHotKeyNotify 163
    TTInstrNotify 169
    TTInstrObj 217
    TTMarketExplorer 539
    TTOrderObj 279
    TTOrderProfile 309
    TTOrderSelector 343
    TTOrderSet 357
    TTOrderTrackerObj 439
Close method 405
Collections
    instrument 29
Combo codes 464
ComboCode property 464
Confirm method 110
Contract
    new 505
Contract property 222
ContractFormatted property 225
Contracts
    combo codes 464
    currency 465
    decimals 466
    expiration 467
    legs 468
    name 469
    options 463
    precision 470
    price display 472
    product 473
    series key 474
    series name 475
    strike price 476
    tick size 479
    XML 481
CopyToClipboard method 480
Count property 365, 397
    InstrumentCollection 31, 63, 67
    ITTContractCollection 459
    ITTProductCollection 529
    OrderProfileCol 75
    OrderSetCol 79
    TTDropHandler 83
CreateNotifyObject property 226
CreateOrderProfile property 281
CreateOrderSelector property 227, 366
Currency 144
    contracts 465
Currency property 465
Customer profiles 33
Customer property 37, 313
CustomerDefaults property 117, 315
Customers property 118, 316

D

Dates
    expiration 467
DebugAppInfo property 119
DebugLogLevel property
    TTGate 120
    TTInstrNotify 173

TTInstrObj 229, 368
Decimals property 466
DeleteContracts property 371
DeleteOrder method
TTOrderObj 306
DeleteOrders property 373
DelHotKey method 167
DeliverAllPriceUpdates property 176
Depth updates 178
Description property 518
DestroyDownloadedFills method 138
Detach
instrument 204
order set 206
DetachInstrument method 204
DetachOrderSet method 206
DisableExtendedPrices method 139
DropEffect property 84
DropKeyState property 86
DropX property 88
DropY property 89

E

EnableAutoSessionRollover method 140
EnableDepthUpdates property 178
EnableFillCaching property
Fill caching 376
EnableOrderAutoDelete property 377
EnableOrderFillData property 378
EnableOrderRejectData property 379
EnableOrderSend property 123, 380
EnableOrderSetUpdates property 381
EnableOrderUpdateData property 382
EnablePriceUpdates property 182
EnableServerMode method 141
Event handlers
ITTGateway 493, 499
ITTProduct 523
TTMarketExplorer 542, 545
Events
ITTGatewayEvents 502
ITTMarketExplorerEvents 512
ITTProductEvents 532
TTDropHandler 82
TTGate 115
TTHotKeyNotify 164
TTInstrNotify 172
TTOrderSet 361
Exchange
failed logins 159
fills 145
logging into 148, 150, 152
message events 156, 160
orders 146
prices 147
state updates 157
Exchange property 232
Exchanges 414
ExpirationDate property 467
Expiry
notification 207

F

FFT2 property 38
FFT3 property 39
Fills
caching 376
data 427
exchange 145
fill blocks 425, 426
order set notifications 378
filter 258
Filters
update 258

G

Gate
attributes 124
Gateway
identifier 486
logging in 478, 489
request available 544
Gateway property 519
Gateways 414
legacy 233
PFX 233
products 519
types 233
GatewayTypes property 233
Get property
TTFillObj 97
TTGateObj 124
TTInstrObj 234
TTOrderObj 285
TTOrderProfile 317
TTOrderSet 385
GetChanged property 283, 442
GetExchangeRate method 143
GetIsChanged property 296, 443
GetLast property 328, 330
GetPrimaryCurrency method 144
GiveUp property 40
GuardServer 487

H

HasNewOrder property 445
HasNextOrder property 297
HasOldOrder property 446
HoldOrder method 406
TTOrderObj 307
HoldOrders method 408
Hot keys 163
adding 165
cautions 164, 165
deleting 167

I

Id property 486
Implied prices 218
InstrCollection class 29
  properties, list of 30
InstrCollection properties
  Item 32, 64, 70
InstrData property 127
InstrEnableUpdates 185
InstrFilter property 187
Instrument
  attach 200
  collection 29
  detach 204
  filter 187
  notifiers 169
  property 190
  updates 185
Instrument property 68, 106, 299, 331, 335
InstrumentCollection properties
  Count 31, 63, 67
Instruments
  associated order 299
  attributes 234
  expiry 207
  for order profile 331, 335
  notifiers 226
  opening 275
  order selector 227
  retrieving 277
  series key 270
  verifying 277
  volatility 389
Instruments property 128
InvertTest property 354
IsAvailable property 487
IsLoggedIn property 478, 489
IsNull property 300
IsTradablePrice property 254
IsValidGet property
  TTFillObj 107
  TTGate 129
  TTInstrObj 256
  TTOrderObj 301
  TTOrderProfile 332
  TTOrderSet 383
IsValidUpdateFilter property 193, 258
IsVolatile property 260, 262, 389
Item property 90, 390
  InstrCollection 32, 64, 70
  ITTProductCollection 530
  OrderProfileCol 76
  OrderSetCol 80
  TTContractCollection 460
ITT Gateway methods
  RequestProducts 495
ITTContractCollection class 457
  overview 458
  properties, list of 458
ITTContractCollection properties
  Count 459
ITTContractInfo class 461
  overview 462
  properties, list of 462
ITTContractInfo methods
  CopyToClipboard 480
  ToXML 481
ITTContractInfo properties
  CallPut 463
  ComboCode 464
  Currency 465
  Decimals 466
  ExpirationDate 467
  LegInfo 468
  Name 469
  Precision 470
  PriceDisplayFormat 471
  PriceDisplayType 472
  Product 473
  SeriesKey 474
  SeriesName 475
  Strike 476
  TickPriceNumerator 478
  TickSize 479
ITTContractInfo propertiesTickPriceDenominator
  TickPriceDenominator 477
ITTCustomerProfile class 33
  properties, list of 34
ITTCustomerProfile properties
  Acct 35
  AcctType 36
  Customer 37
  FFT2 38
  FFT3 39
  GiveUp 40
  Market 41
  MaxOrderQty 42
  OpenClose 43
  OrderRestr 44
  OrderType 45
  ProdType 46
  Product 47
  TIF 48
  UseMaxQty 49
ITTGateway class 483
  methods, list of 484
  overview 484
  properties, list of 484
ITTGateway method
  RequestProducts 495
ITTGateway methods
  RegisterEventHandler 493
  RequestProductTypes 497
  UnRegisterEventHandler 499
ITTGateway properties
  AvailableServices 485
  Id 486
  IsAvailable 487
  IsLoggedIn 489
  MarketName 490
  Name 491

Products 492
ITTGatewayEvents
OnGatewayAttributes 503
OnNewContract 505
OnNewProduct 506
OnProductsFound 507
OnProductTypes 509
OnServerStatus 510
ITTGatewayEvents class 501
events, list of 502
overview 502
ITTGatewayEvents events
OnProducts 507
ITTMarketExploreEvents class 511
ITTMarketExplorer events
OnGateway 513
ITTMarketExplorerEvents class
events, list of 512
ITTPriceEntry class 51
overview 34, 52
properties, list of 52
ITTPriceEntry methods
toDouble 56
toString 55
toTicks 57
ITTPriceEntry properties
Name 53
PriceID 54
ITTPriceList class 59
ITTPriceUpdate class 65
overview 66
properties, list of 66
ITTPriceUpdate properties
Instrument 68
TimeStamp 71, 72
ITTProduct class 515
methods, list of 516
overview 516
properties, list of 516
ITTProduct events
OnContractsFound 533
OnNewContract event 535, 537
ITTProduct methods
RegisterEventHandler 523
RequestContracts 524
UnregisterEventHandler 525
ITTProduct properties
Alias 517
Description 518
Gateway 519
Name 520
Type 521
TypeAsString 522
ITTProductCollection class 527
overview 528
properties, list of 528
ITTProductCollection properties
Count 529
Item 530
ITTProductEvents class 531
events, list of 532
overview 532

K

Key modifiers 165

L

LegInfo property 468
Legs 468
Login
TTGateway 478, 489
Logins
failed 159
Logon
SetExchangeLogon property 148, 150
SetUniversalLogin property 152

M

Market
name 490
Market property 41
MarketName property 490
MaxOrderQty property 42
MEAPI
event handlers 493, 499, 523, 542, 545
terminate 541
MEAPI classes
ITTContractCollection 457
ITTContractInfo 461
ITTGateway 483
ITTGatewayEvents 501
ITTMarketExplorerEvents 511
ITTProduct 515
ITTProductCollection 527
ITTProductEvents 531
TTMarketExplorer 539
MEAPITerminate method 541
Methods
ITTGateway 484
ITTProduct 516
TTDropHandler 82
TTFillObj 96
TTGate 114
TTHotKeyNotify 164
TTInstrNotify 171
TTInstrObj 219
TTMarketExplorer 540
TTOrderObj 280
TTOrderProfile 311
TTOrderSelector 346
TTOrderSet 360

N

Name property 53, 520
ITTContractInfo 469

ITTGateway 491
NewOrder property 447
NextOrder property 303
NextOrderTracker property 392
Notifiers
instrument 226
Null objects 300

O

Object attributes
gate 124
order 285
order profile 317, 334, 336, 337, 398, 404
OldOrder property 449
OnContractsFound event 533
OnExchangeMessage event 156, 160
OnExchangeStateUpdate event 157
OnGateway event 513
OnGatewayAttributes event 503
OnHotKey event 168
OnInstrumentExpiryChange event 207
OnLoginFailed event 159
OnNewContract event 505, 535, 537
OnNewProduct event 506
OnNotifyDepthData event 208
OnNotifyDrop event 93
OnNotifyFound event 210
OnNotifyNotFound event 211
OnNotifyUpdate event 212
OnOrderChanged event 432
OnOrderDeleted event 423
OnOrderFillBlockEnd event 425
OnOrderFillBlockStart event 426
OnOrderFillData event 427
OnOrderHeld event 431
OnOrderRejected event 433
OnOrderSetUpdate event 214, 434
OnOrderSubmitted event 435
OnOrderUpdated event 421, 429, 437
OnPriceListUpdate event 215
OnProducts event 507
OnProductsFound event 507
OnProductTypes event 509
OnServerStatus event 510
OnStatusUpdate event 161
Open method 275, 410
OpenClose property 43
OpenExchangeFills method 145
OpenExchangeOrders method 146
OpenExchangePrices method 147
Options
call/put 463
strike price 476
Order profiles
alias 312
associated instrument 331, 335
attributes 317, 334, 336, 337, 398, 404
create 281
Order selectors
add sub-selector 350
add test 351, 353
all matches 347
any matches 348
invert test 354
order set 394
reset 356
testing 349
Order set
updates 181
Order sets
alias 362
attach 202
attached to instruments 264
average prices 363
changed orders 432
closing 405
delete orders 371, 373, 377
deleted orders 423
detach 206
enable orders 380
fill blocks 425, 426
fill data 427
fill notifications 378
held orders 431
holding orders 406, 408
instrument volatility 389
matching rules 347, 348
opening 410
order selector 394
order tracker 392
orders in 365, 397
readable attributes 385
rejected orders 379, 433
selectors 366
send order 400
setting attributes 411
site order keys 402, 403
submit order 400
submit orders 380
submitted orders 435
testing orders 349
track orders 382
updated orders 421, 429, 437
updates 381
updating 434
updating order colors 418
updating orders 416
valid exchagnes 414
working orders 377
Order trackers
changed attributes 442, 443
new orders 445, 447
old orders 446, 449
OrderProfileCol class 73
overview 74, 512
properties, list of 74
OrderProfileCol properties
Count 75
Item 76
OrderProfiles property 131
OrderRestr property 44

Orders
attributes 285, 305
canceling 307
changed 296, 432
changed attributes 283, 442, 443
changing color in X_TRADER 418
create 281
delete automatically 377
deleted 423
deleting 306
exchange 146
get last attributes 328, 330
held 431
holding 406, 408
next 303
next state 297
null 300
rejected 379, 433
selectors 227
submitted 435
submitting 380, 400
tracking 382, 392
updated 421, 429, 437
updating 416
OrderSelector property 394
OrderSet property 264
OrderSetCol class 77
overview 78
properties, list of 78
OrderSetCol properties
Count 79
Item 80
OrderSetData property 132
OrderSets property 133
OrderType property 45

P

PFX gateways 233
Precision property 470
Price
entry 51
list 59
update 65
Price updates 182
PriceDisplayFormat property 471
PriceDisplayType property 472
PriceID property 54
PriceList property 195
Prices
average open 363
calculating implieds 221
calculation mode 363
depth 218
display format 471
exchange 147
formats 218
implied 218
implieds 221
rounding 268, 273
strike 476
tick 271, 273
tick denominator 477
tick numerator 478
tradable 254
update filter 197
updates 176
Primary currency 144
ProdType property 46, 265
Product
new contract 505
Product property 47, 266, 473
Products 492
alias 517
contracts 524
description 518
event handler 525
gateway 519
name 520
request 495
requesting 495
type 521, 522
types 497
Products property
ITTGateway 492
Properties
InstrCollection 30
ITTContractCollection 458
ITTContractInfo 462
ITTCustomerProfile 34
ITTGateway 484
ITTPriceEntry 52
ITTPriceUpdate 66
ITTProduct 516
ITTProductCollection 528
OrderProfileCol 74
OrderSetCol 78
TTDropHandler 82
TTFillObj 96
TTGate 113
TTInstrNotify 171
TTInstrObj 218
TTOrderObj 280
TTOrderProfile 310
TTOrderSelector 346
TTOrderSet 359
TTOrderTrackerObj 441
property 477
Put 463

R

RapidFillDelivery property 134
ReadProperties property
TTFillObj 109, 136, 267
TTOrderObj 305
TTOrderProfile 334, 398
RegisterDropWindow method 91
RegisterEventHandler method
ITTGateway 493

ITTProduct 523
TTMarketExplorer 542
RequestContracts method 524
RequestGateways method 544
RequestProducts method 495
RequestProductTypes method 497
Reset method 92
Reset property 356
Retrieve property 277
Rollover, session
see Session rollover 153
RoundPrice property 268

S

SendOrder property 400
Series key 270
SeriesKey property 270
ITTContractInfo 474
SeriesName property 475
Services
available 485
Session rollover
starting 153
Set method
TTOrderProfile 337
TTOrderSet 411
SetExchangeLogon method 148, 150
SetUniversalLogin method 152
SetValidExchanges method 414
Shut down
MEAPITerminate 541
Site order keys 402, 403
SiteKeyLookup property 402
SiteKeyLookupEx property 403
StartNewSession method 153
StartXTrader method 154
Strike property 476

T

TestMatch property 349
TickPrice property 271
TickPriceEx property 273
TickPriceNumerator property 478
Ticks
denominator 477
price numerator 478
size 479
TickSize property 479
TIF property 48
TimeStamp property 71, 72
toDouble method 56
toString method 55
toTicks method 57
ToXML method 481
TT Gateways
market name 490
name 491
product request 495
product types 497
products 492
requesting products 495
services 485
TTContractCollection properties
Item 460
TTDropHandler class 81
events, list of 82
methods, list of 82
overview 82
properties, list of 82
TTDropHandler events
OnNotifyDrop 93
TTDropHandler methods
RegisterDropWindow 91
Reset 92
TTDropHandler properties
Count 83
DropEffect 84
DropKeyState 86
DropX 88
DropY 89
Item 90
TTFillObj class 95
methods, list of 96
overview 96
properties, list of 96
TTFillObj methods
Confirm 110
TTFillObj properties
Get 97
Instrument 106
IsValidGet 107
ReadProperties 109, 136, 267
TTGate class 111
events, list of 115
methods, list of 114
properties, list of 113
TTGate events
OnExchangeMessage 156, 160
OnExchangeStateUpdate 157
OnLoginFailed 159
OnStatusUpdate 161
TTGate method
OpenExchangePrices 147
TTGate methods
DestroyDownloadedFills 138
DisableExtendedPrices 139
EnableAutoSessionRollover 140
EnableServerMode 141
GetExchangeRate 143
GetPrimaryCurrency 144
OpenExchangeFills 145
OpenExchangeOrders 146
SetExchangeLogon 148, 150
SetUniversalLogin 152
StartNewSession 153
StartXTrader 154
XTAPITerminate 137, 155
TTGate properties
CustomerDefaults 117

Customers 118
DebugAppInfo 119
DebugLogLevel 120
EnableOrderSend 123
InstrData 127
Instruments 128
IsValidGet 129
OrderProfiles 131
OrderSetData 132
OrderSets 133
RapidFillDelivery 134
TTGateObj properties
Get 124
TTGateway properties
IsLoggedIn 478
TTHotKeyNotify class 163
events, list of 164
methods, list of 164
overview 113, 164, 170
TTHotKeyNotify events
OnHotKey 168
TTHotKeyNotify methods
AddHotKey 165
DelHotKey 167
TTInstrNotify class 169
events, list of 172
methods, list of 171
properties, list of 171
TTInstrNotify events
OnInstrumentExpiryChange 207
OnNotifyDepthData 208
OnNotifyFound 210
OnNotifyNotFound 211
OnNotifyUpdate 212
OnOrderSetUpdate 214
OnPriceListUpdate 215
TTInstrNotify methods
AttachInstrument 200
AttachOrderSet 202
DetachInstrument 204
DetachOrderSet 206
TTInstrNotify properties
DebugLogLevel 173
DeliverAllPriceUpdates 176
EnableDepthUpdates 178
EnableOrderSetUpdates
EnableOrderSetUpdates property 181
EnablePriceUpdates 182
InstrEnableUpdates 185
InstrFilter 187
Instrument 190
IsValidUpdateFilter 193
PriceList 195
UpdateFilter 197
TTInstrObj class 217
methods, list of 219
overview 218, 280
properties, list of 218
TTInstrObj methods
Open 275
Retrieve 277
TTInstrObj properties
Alias 220
CalculateTTImplieds 221
Contract 222
ContractFormatted 225
CreateNotifyObjet 226
CreateOrderSelector 227
DebugLogLevel 229, 368
Exchange 232
GatewayTypes 233
Get 234
IsTradablePrice 254
IsValidGet 256
IsValidUpdateFilter 258
IsVolatile 260, 262
OrderSet 264
ProdType 265
Product 266
RoundPrice 268
SeriesKey 270
TickPrice 271
TickPriceEx 273
TTMarketExplorer class
methods, list of 540
overview 540
TTMarketExplorer methods
MEAPITerminate 541
RegisterEventHandler 542
RequestGateways 544
UnregisterEventHandler 545
TTMarketExplorerclass 539
TTOrderObj class 279
methods, list of 280
properties, list of 280
TTOrderObj methods
DeleteOrder 306
HoldOrder 307
TTOrderObj properties
CreateOrderProfile 281
Get 285
GetChanged 283
GetIsChanged 296
HasNextOrder 297
Instrument 299
IsNull 300
IsValidGet 301
NextOrder 303
ReadProperties 305
TTOrderProfile class 309
methods, list of 311
overview 310
properties, list of 310
TTOrderProfile methods
Set 337
TTOrderProfile properties
Alias 312
CustomeDefaultsr 315
Customer 313
Customers 316
Get 317
GetLast 328, 330

IsValidGet 332
ReadProperties 334, 398
WriteProperties 336, 404
TTOrderSelector class 343
methods, list of 346
overview 345
properties, list of 346
TTOrderSelector properties
AddSelector 350
AddTest 351
AddTestEx 353
AllAnyMatches 348
AllMatchesRequired 347
InvertTest 354
Reset 356
TestMatch 349
TTOrderSet class 357
events, list of 361
methods, list of 360
overview 359
properties, list of 359
TTOrderSet events
OnOrderDeleted 423
OnOrderFillBlockEnd 425
OnOrderFillBlockStart 426
OnOrderFillData 427
OnOrderHeld 431
OnOrderInstrumentChanged 432
OnOrderRejected 433
OnOrderSetUpdate 434
OnOrderSubmitted 435
OnOrderUpdated 421, 429, 437
TTOrderSet methods
Close 405
HoldOrder 406
HoldOrders 408
Open 410
Set 411
SetValidExchanges 414
UpdateOrder 416
UpdateXTOrder 418
TTOrderSet properties
Alias 362
AvgOpenPriceMode 363
Count 365, 397
CreateOrderSelector 366
DeleteContracts 371
DeleteOrders 373
EnableFillCaching 376
EnableOrderAutoDelete 377
EnableOrderFillData 378
EnableOrderRejectData 379
EnableOrderSend 380
EnableOrderSetUpdates 381
EnableOrderUpdateData 382
Get 385
IsValidGet 383
IsVolatile 389
Item 390
NextOrderTracker 392
OrderSelector 394
SendOrder 400
SiteKeyLookup 402
SiteKeyLookupEx 403
TTOrderTrackerObj class 439
overview 440
properties, list of 441
TTOrderTrackerObj properties
GetChanged 442
GetIsChanged 443
HasNewOrder 445
HasOldOrder 446
NewOrder 447
OldOrder 449
TTTOrderProfile properties
Instrument 331, 335
Type property 521
TypeAsString property 522

U

Universal logins 152
UnRegisterEventHandler method
ITTGateway 499
UnregisterEventHandler method
ITTProduct 525
TTMarketExplorer 545
UpdateFilter property 197
UpdateOrder method 416
Updates 258
depth 178
instrument 185
order set 181
price 182
prices 176
UpdateXTOrder method 418
UseMaxQty property 49

W

WriteProperties property
TTOrderProfile 336, 404

X

X_TRADER, starting 154
XTAPI classes
InstrCollection 29
ITTPriceEntry 51
ITTPriceList 59
ITTPriceUpdate 65
OrderProfileCol 73
OrderSetCol 77
TTDropHandler 81
TTFillObj 95
TTGate 111
TTHotKeyNotify 163
TTInstrNotify 169
TTInstrObj 217
TTOrderObj 279
TTOrderProfile 309

TTOrderSelector 343
TTOrderSet 357
TTOrderTrackerObj 439
XTAPITerminate method 137, 155