

# ProTrader Software Development Kit

---

## Table of Contents

Software Development Kit .....	3
General information.....	3
Creating Component .....	4
IExternalComponent interface .....	4
ILocalizableComponent interface .....	5
ILinkableComponent interface.....	5
Quotes receiving .....	6
Trading .....	7

## Software Development Kit

SDK is created to allow developers to implement their own components and include them to ProTrader. For now there are interfaces to display component, receive quotes and trade.

### General information

Custom components appears in ProTrader at special tab **Tools->Custom panels**.

To include component to ProTrader, developer needs to place component DLL library into **[ProTrader Directory]\plug-ins\panels**.

Only **.Net** components are supported.

All above are right for **PT Multistation** too.

## Creating Component

Custom Component shall implement few simple interfaces. Every class for export shall be marked with **Exportable** attribute. It is made to simplify developer's work – if there are several classes in library developer can manage their exportability.

To access attribute and other types for SDK you need to use **Commons.dll** from ProTrader root directory.

```
namespace com.pfsoft.protrading.commonsexternal
{
    /// <summary>
    /// Use this attribute for allow/forbid export designed
    /// component to application
    /// </summary>
    [AttributeUsage(AttributeTargets.Class, AllowMultiple = false, Inherited
= false)]
    [Serializable]
    public class ExportableAttribute : Attribute
    {
        private bool FExportable;
        public bool Exportable
        {
            get { return FExportable; }
        }

        public ExportableAttribute(bool isExportable)
        {
            FExportable = isExportable;
        }
        public ExportableAttribute()
            : this(true)
        {
        }
    }
}
```

## IExternalComponent interface

Realization of **IExternalComponent** interface is a must for all exportable components.

```
/// <summary>
/// Base interface for external components.
/// All of them shall implement it.
/// </summary>
public interface IExternalComponent : IDisposable, ICaller
{
    /// <summary>
    /// Icon that will be associated with component
    /// property. If returns null no icon will be displayed.
```

```

    /// </summary>
    Icon IconImage { get; }

    /// <summary>
    /// Unique name of designed component
    /// </summary>
    string ComponentName { get; }

    /// <summary>
    /// Controls set that will be added to panel
    /// </summary>
    Panel Content { get; }
}

```

**Note:** Content shall be **Panel** type. This means no matter what is on your form or in application, ProTrader will use only controls from returned **Panel**.

## ILocalizableComponent interface

If panel supports localization, it shall realize **ILocalizableComponent** interface.

```

/// <summary>
/// If designed component supports multi-language
/// localization, it shall implement this interface
/// </summary>
public interface ILocalizableComponent : IExternalComponent
{
    /// <summary>
    /// Apply new localization environment
    /// </summary>
    /// <param name="newLocale">new locale name</param>
    void SetLocale(string newLocale);

    /// <summary>
    /// Visible name of designed component
    /// </summary>
    string VisibleName { get; }
}

```

**SetLocale** – called at language switch, **VisibleName** – component name at panel (need to be changed at language switch)

Locale is received in **newLocale** argument as string identifier. For now supported are: "en", "ru", "ar", "jp", "ch\_sm", "ch\_tr", "gr", "fr" (english, russian, arabic, japanesse, chinesse, german, francias)

## ILinkableComponent interface

If panel support instrument link, it shall realize **ILinkableComponent** interface.

```

/// <summary>
/// If designed component supports link model of ProTrader

```

```
/// it shall implement this interface
/// </summary>
public interface ILinkableComponent : IExternalComponent
{
    /// <summary>
    /// Current link symbol of the panel
    /// </summary>
    string Symbol { get; }

    /// <summary>
    /// This method will be called before link symbol will be changed
    /// </summary>
    /// <param name="newSymbol"></param>
    void OnChangeSymbol(string newSymbol);
}
```

**Property** is changed to new Instrument. **Method** is called by ProTrader at Instrument change and component shall handle it.

**Note:** Only one-way linking is supported for now.

## Quotes receiving

To receive quotes custom component shall implement [IQuoteReceiverComponent](#) and [IQuoteListener](#) interfaces and use [SubscribeEventHandler](#) events.

```
public delegate void SubscribeEventHandler(string instrumentName, int
type);
/// <summary>
/// If the designed component needs to receive quotes on the selected
instrument,
/// it shall implement this interface
/// </summary>
public interface IQuoteReceiverComponent : IExternalComponent,
IQuoteListener
{
    /// <summary>
    /// Call when your component wants to receive quotes on the selected
instrument
    /// Method newQuote will be called when new quotes are processed in
system
    /// </summary>
    event SubscribeEventHandler Subscribe;
    /// <summary>
    /// Call when your component want to stop receiving quotes after
subscribing
    /// </summary>
    event SubscribeEventHandler UnSubscribe;
}
```

instrumentName - is Instrument name (not ticker) as you see it in ProTrader: GOOG, IBM, EUR/USD, AUD/CAD (Fx instruments are used with slash).

Type - quote type. Allowed values are -

```

public const int QUOTE_LEVEL1 = 1; - Level I - best quote
public const int QUOTE_LEVEL2 = 2; - Level II - Market Depth
public const int QUOTE_TRADES = 4; - Trades
public const int QUOTE_OPTIONS = 8; - Options
    
```

You can subscribe to several types at one Instrument.

This interface is child of

```

/// <summary>
/// Interface of quotes receiver
/// </summary>
public interface IQuoteListener
{
    void newQuote(QuoteMessage message);
}
    
```

This method called at each new quote, and quote is stored in general `QuoteMessage` type.

## Trading

`ITradeComponent` is the main interface for trading.

```

/// <summary>
/// If trading allowed for the designed component,
/// it will implement this interface
/// </summary>
public interface ITradeComponent : IExternalComponent, IMessageListener
{
    /// <summary>
    /// Set or get the platform engine for trading operations
    /// It provides methods for platform functions calling
    /// </summary>
    /// <param name="platformEngine"></param>
    IPlatformEngine PlatformEngine { get;set; }
}
    
```

ProTrader will create `IPlatformEngine` class and will set it through set-accessor property.

**Note:** Trading in custom component is allowed only when Automated Trading is allowed by Broker. In other case initialization of `IPlatformEngine` will be skipped.

All messages will be available through `IMessageListener` interface.

```

/// <summary>
/// Message Listener Interface
/// </summary>
public interface IMessageListener
{
}
    
```

```
        void newMessage(Message message);  
    }
```

New message will be sent to this function with conversion to base class [Message](#). You need to analyze message content and handle if needed.

There are several methods in [IPlatformEngine](#) interface

```
public interface IPlatformEngine  
{  
    string SubmitStrategyOrder(  
        string symbol, //Instrument to open Order (name as shown in PT)  
        int buysell, //Operation  
        double quoty, //Amount  
        double price, //Limit Price (at Market is ignored)  
        double stopPrice, //Stop Price (at Market is ignored)  
        int tif, //Time In Force  
        int orderType, //Type of Order  
        string account, //Account to open Order  
        string route, //Route to Open Order  
        long timeExpired, //Expired in (not supported yet - ignored)  
        string comment, //Comment to save on Server  
        string boundTo, //For OCO orders  
        double slOffset, //Stop Loss offset  
        double tpOffset, //Take Profit offset  
        double trOffset, //Trailing Stop offset  
        int mktRange, //Market Range (in pips)  
        int magicNumber, //Alternate ID  
        bool hedge); //Hedge on server. Shall be false.  
  
    string ClosePositions(  
        string orderId, //Position number  
        double lots, //Close amount  
        string symbol, //Instrument  
        string account); //Account  
  
    bool SetSLTP(  
        string orderId, //Order ID or Position Number  
        double sl, //New SL (set 0 to remove)  
        double tp, //New TP (set 0 to remove)  
        double ts); //New Trailing Stop (set 0 to remove)  
  
    bool ReplaceOrder(  
        string orderId,  
        double lots,  
        int tif,  
        double price,  
        double sl,  
        double tp,  
        string boundTo);  
}
```

```
bool CancelOrder(  
    string orderId,  
    string symbol,  
    string account);  
}
```

## Operations:

```
VALUE_BUY = 10000; - Buy  
VALUE_SELL = 10001; - Sell
```

## TIFs

```
VALUE_DAY = 10011, - Day  
VALUE_DAYPLUS = 10009, - Extended Day  
VALUE_DAYNB = 10013, - for NY  
VALUE_GTC = 10008, - Good Till Cancel  
VALUE_GTX = 10012, - Good Till Execute  
VALUE_IOC = 10010, - Immediate or Cancel  
VALUE_FOC = 11000; - Fill or Cancel
```

## Order Types

```
VALUE_AON = 10047,  
VALUE_COND_ORD_LMT = 10036,  
VALUE_COND_ORD_MKT = 10035,  
VALUE_DISCRETIONARY = 10043,  
VALUE_HIDDEN = 10041,  
VALUE_LIMIT = 10030,  
VALUE_LIMIT_CLOSE = 10057,  
VALUE_LIMIT_OPEN = 10056,  
VALUE_LIMIT_TRAIL = 10054,  
VALUE_LIMIT_TTO = 10050,  
VALUE_MARKET = 10031,  
VALUE_MARKET_CLOSE = 10039,  
VALUE_MARKET_OPEN = 10038,  
VALUE_MARKET_TRAIL = 10055,  
VALUE_MARKET_TTO = 10051,  
VALUE_NORMAL = 10042,  
VALUE_NOW = 10048,  
VALUE_PART = 10046,  
VALUE_PNP = 10045,  
VALUE_RESERVE = 10040,  
VALUE_RSV_DISC = 10044,  
VALUE_RSV_TTO = 10052,  
VALUE_STOP_LIMIT = 10033,  
VALUE_STOP_MARKET = 10032,  
VALUE_STOP_TTO = 10053,  
VALUE_THRU = 10049,  
VALUE_TRAILING_STOP = 10034,  
VALUE_TTO_ORDER = 10037,  
VALUE_LIMIT_STOPMKT = 10064,  
VALUE_STOP_TRAIL = 10065,  
VALUE_RSV_PEGGED = 10066,  
VALUE_STOPPLMT_TTO = 10067,  
VALUE_PEGGED = 10062,  
VALUE_VWAP = 10063,  
VALUE_STOPPLMT_TRAIL = 10068,  
VALUE_STRADDLE = 10058,  
VALUE_STRANGLE = 10059,  
VALUE_OCO_ORDER = VALUE_TTO_ORDER,  
  
VALUE_MANUAL = 10070, // Manual open  
VALUE_SLTP_LIMIT = 10071, // StopLoss and TakeProfit Limit  
VALUE_SLTP_STOP = 10072, // StopLoss and TakeProfit Stop  
  
VALUE_STOP = 10073, // Stop Order  
VALUE_POSITION = 10074, // Opened position  
VALUE_TR_STOP = 10075; // Stop Order with Trailing Stop
```