东海证券 DONGHAI SECURITIES

TradeStation

EasyLanguage 自学教程

EasyLanguage Home Study Course

作者：TradeStation团队，TradeStation Technologies, Inc.

译者：东海证券量化团队

修订日期

2018年3月22日

重要信息与免责声明：

TradeStation Securities, Inc. 致力于为机构交易者以及活跃交易者提供服务。请注意，活跃交易并不适合资源有限、投资额有限或交易经验有限、风险承受能力低或不愿承受至少 50000 美元资本风险的投资者。

本教程讨论 TradeStation EasyLanguage 如何帮助您开发和实施客户指标与交易策略。但 TradeStation Technologies 及其附属机构并不提供或建议任何指标或交易策略。我们为您提供独有的工具，帮助您设计自己的策略并了解如果这些策略在过往时段应用可能的表现。我们相信这些信息非常有价值，同时我们也提醒您：一个交易策略的历史模拟交易结果并不能保证其将来的表现或成功。另外，我们也不推荐或招揽任何特定证券或招揽衍生产品的买卖。本书提及的任何证券仅用于举例说明，并非推荐。

本书讨论自动化电子订单与执行。请注意，虽然 TradeStation 旨在自动执行您的交易策略并及时提交、发送和执行订单，但偶尔会有延迟，甚至会因为市场波动性、报价延迟、系统和软件错误、互联网通信问题、停电或其他因素而失败。

TradeStation 中的全部专有技术由 TradeStation Securities, Inc. 下属公司 TradeStation Technologies, Inc. 拥有。可从 TradeStation 内访问的订单执行服务由 TradeStation Securities, Inc. 根据其下属公司的技术许可以及其作为注册自营券商和期货代理商的权力提供。TradeStation® 和 EasyLanguage® 的所有其他特性与功能均为 TradeStation Technologies, Inc.（下文简称为 "TradeStation"）的注册商标。

期权风险披露

期权交易风险很高。期权的买家和卖家应熟悉期权交易理论和定价以及所有相关风险因素。请参阅 Options Clearing Corporation 网站提供的 Characteristics and Risks or Standardized Options（标准化期权特征与风险）一文 (http://www.optionsclearing.com/publications/risks/riskstoc.pdf)；您也可以致函 TradeStation Securities，地址为 8050 SW 10 Street, Suite 2000, Plantation, FL 33324。

TradeStation Securities Inc. 出版

尽管在准备这本书的过程中，采取了所有防范措施，东海证券和TradeStation Securities不为使用此书包含的信息所产生的错误、遗漏或任何损害负责。

下载教程材料

www.tradestation.com/education/downloads/ELOBJECTS

目录

关于本书

感谢购买《*EasyLanguage Objects* 自学教程》(*EasyLanguage Objects - Home Study Course*)。

本教程的目标是帮助您更好的熟悉 EasyLanguage 对象以及了解如何用其来扩展您已经熟悉的现有 EasyLanguage 代码。您可以通过使用对象创建和检查一系列独特的 EasyLanguage 指标来实现这个目标。

本书全书采用简单一致的格式：首先介绍一组 EasyLanguage 元素和概念，然后是与概念相关的一个或多个示例练习。

请创建并输入每个练习，然后将您的作品应用到图表或雷达图（如果适用）。

预备知识

在尝试本教程的练习前，强烈建议您亲自参加为期 2 天的现场或在线 EasyLanguage Boot Camp（EasyLanguage 新兵训练营）教程。阅读编程手册或浏览网络上的面向对象编程概念也可能会对您有所裨益。

本书旨在介绍用于在现有 EasyLanguage 框架内访问市场数据并下单的 EasyLanguage 对象增强功能。我们推荐用户在开始本教程前先了解和熟悉 TradeStation。其中包括创建和管理图表分析窗口以及普通文件、窗口、工作区和桌面的管理指标等各项内容。TradeStation 教育中心的 TradeStation 教育资源是很好的起点，在 TradeStation 平台的帮助菜单或下列网址可以访问这些资源：

www.tradestation.com/education。

您可能是在上完现场 EasyLanguage 教程后购买本教程作为复习，也可能是因为未能亲自听课而购买的。无论是什么原因，您现在拥有了一套透彻的高级教程，可以教会您如何使用 TradeStation 的 EasyLanguage 来理解和编写包含对象、属性与方法的自定义指标与交易策略。

TradeStation 团队

EasyLanguage 对象简介

自推出以来，EasyLanguage 作为编程语言一直在不断发展，它比传统编程语言能更有效的帮助交易者分析交易想法和实现其自己的交易策略。EasyLanguage 的最新发展是加入了对象，并提供了一系列增强的语言元素和编辑工具，在扩展 EasyLanguage 的强大功能与灵活性的同时，可以轻松集成到现有代码。

本书将介绍一些新词汇与工具，它们将帮助您开始在 EasyLanguage 中使用对象。但是，本书并不打算对面向对象编程进行总体介绍。通过一系列的代码示例，您将会了解到如何结合您早已熟悉的 EasyLanguage 语句来使用 EasyLanguage 对象。无论您是否打算使用对象增强功能开发新的 EasyLanguage 代码，请注意现有的指标和策略将会保持有效，无需任何修改。

学会驾驭

从基本要素开始是学习一个科目的一种方法。例如，我们可以通过学习内燃机、转向联动装置和其他复杂的机械科目去了解汽车——或者，我们也可以直接将汽车作为精密的机械，然后学习如何驾驶。编程对象的学习与此很相似，我们可以从学习多态、封装、继承和其他复杂的编程概念开始学习对象——或者，我们也可以直接接受“对象是代表包含属性和方法的复杂程序”这个事实，然后学会驾驭它们。

学习开车的第一步是熟悉我们的座驾。在这里，我们要熟悉的是 TradeStation 开发环境和几个很有用的面向对象编辑工具，其中包括：

EasyLanguage 编辑器
工具箱
组件盘
属性编辑器
EasyLanguage 字典

在使用 TradeStation 开发环境和 EasyLanguage 对象时，您需要熟悉几个新词汇。我们在后面将会详细介绍对象术语，这里是一些有助于起步的基本概念：

组件 (Component) - 工具箱中可拖放的对象
对象 (Object) - 组件或其他对象的副本
类 (Class) - 对象的蓝图，其中描述了对象的特征与行为
属性 (Property) - 对象内的数据与信息
点运算符 - 在代码中用于获取对象的属性和方法
方法 (Method) - 相当于对象的 EasyLanguage 函数
事件 (Event) - 告知有变化发生的通知
集合 (Collection) - 对象与对象数据的数据结构

EasyLanguage 代码编辑器

EasyLanguage 开发环境是您在 EasyLanguage 文件中输入与编辑 EasyLanguage 语句与代码注释的编辑器。

当您在编辑器中开始输入时，将会弹出自动完成窗口，根据正在输入的内容为您提供可能的选择。除了显示常用的保留字与函数外，自动完成窗口也有助于您选择已创建对象的属性。在本书中将涉及很多新的对象相关属性。

例如，在代码编辑器中打开 EasyLanguage 文件，输入字符 "va"，会出现自动完成窗口，其中 "value1" 一词高亮显示，因为这是以 "va" 开头的第一个词。再输入字母 "r"，高亮显示的词就变成了 "var"。继续输入字母 "i"，高亮显示的词就转到了 "variable"。任何时候按下回车键就会将高亮显示的词条完整插入您的文档。

工具箱

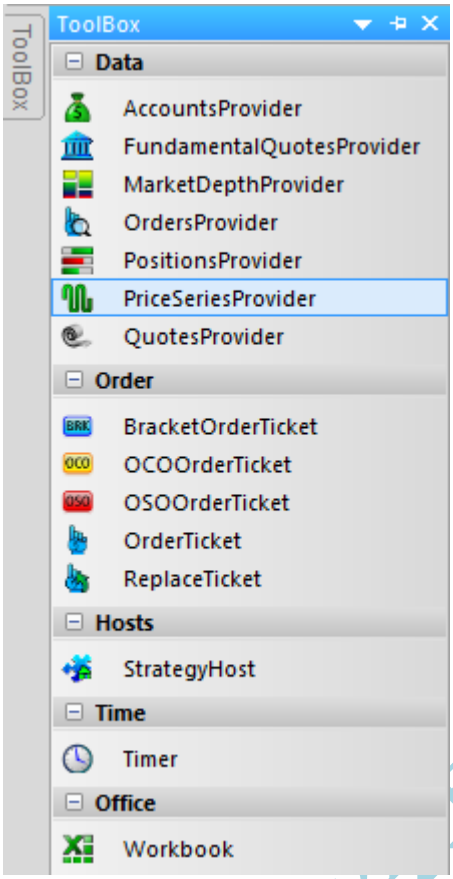
工具箱选项卡位于代码编辑器的左侧，里面显示了一系列可用的组件，这些组件可以作为对象加入到您的 EasyLanguage 文件中。对象把数据和程序结合成为方便的程序包，可以在 EasyLanguage 代码中作为属性和方法使用。工具箱组件旨在让您无需太多编程即可方便的将对象添加到指标、策略和函数中。

可用组件（在工具箱中）

价格、基本面、报价以及市场深度 (Level II) 数据
账户数据
订单与仓位数据
订单（下单）
定时器
Workbook（与 Excel 表集成）

非组件对象（不在工具箱中）

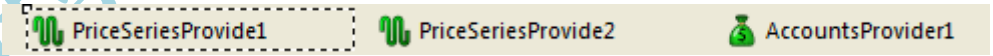
全局变量
向量（数据集合）
自定义 EL 窗口设计
XML 数据库
DateTime
敬请期待…



选中组件（名称高亮显示）然后双击（或拖放）到代码编辑器中，即可将组件添加到现有 EasyLanguage 文件中（请参阅“组件盘”）。在后面的很多示例中，我们将使用工具箱来向我们的 EasyLanguage 文件添加组件对象。

组件盘

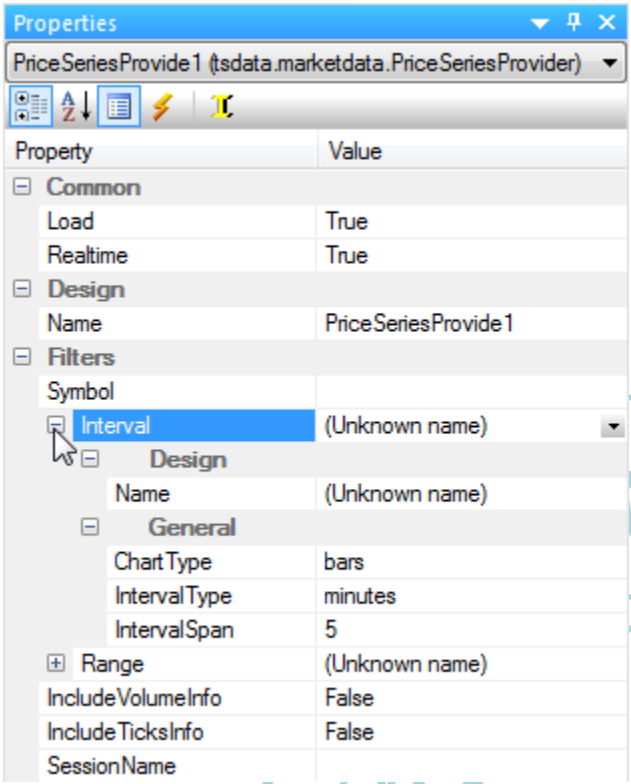
组件盘位于代码编辑器底部，显示已从工具箱添加到 EasyLanguage 文件的组件对象的名称。默认情况下，该名称由组件名称和实例编号组成。



一个文件可以添加不同的组件或者同一组件的多个实例。我们将在后面的示例中对此详细讨论。

属性编辑器

属性选项卡位于代码编辑器右侧，用于访问属性编辑器窗格（见下），在这里可以设置或检查文件中特定组件的属性。为表明要在属性编辑器中编辑属性的组件，请点击组件盘上的组件名称或从属性面板顶部的下拉列表选择其名称。



属性编辑器工具条包含数个不同的图标，可用于选择属性窗格、事件窗格以及我们后面会遇到的其他编辑工具。

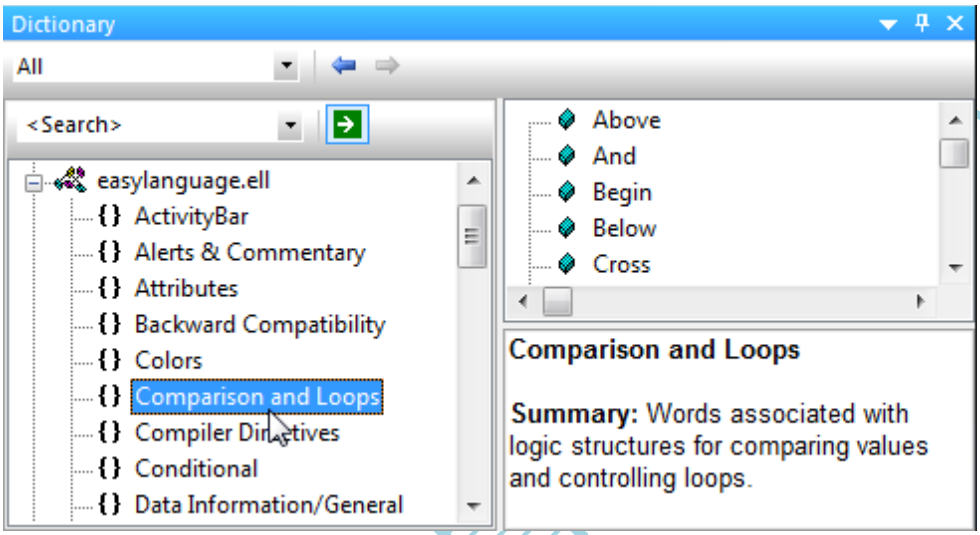
在默认情况下显示的是属性面板。Property（属性）栏显示了按类别分组的属性名称，Value（值）栏显示了其设置。类别与子类别可以点击 ⊞ 和 ⊟ 符号展开或折叠。事件窗格包含 Event（事件）栏与 Value（值）栏。

默认情况下，在您点击面板以外位置时，面板会自动关闭并再次显示为选项卡。要在修改时保持面板始终打开，请点击面板顶部的 图标。完成该面板操作后，再次点击 即可将其关闭。

注意：不要将对象属性与指标属性混淆，后者用于设置指标的样式、颜色和刻度。

EasyLanguage 字典

字典选项卡也位于代码编辑器的右侧，用于访问字典面板。在字典面板可搜索 EasyLanguage 的保留字和函数的概要，还可查询 EasyLanguage 代码中对象所代表的类和类成员的信息（属性、方法、事件等）。



默认情况下，对象窗格（位于字典面板左侧）会分级显示包含 EasyLanguage 类和保留字的库和命名空间。点击 ⊞ 和 ⊟ 符号可浏览各级别的内容。点击对象窗格中的对象可在右上的成员窗格内看到相关条目。在成员窗格内点击条目可在右下的描述窗格内显示该条目的概要。图标有助于识别对象或成员的类型（有关图标的更多信息，请参阅 TradeStation 开发环境的帮助）。

例如（见上），在对象窗格的老版本 easyLanguage.ell 部分选择Comparison and Loops，在成员窗格会显示属于该类别的一系列保留字，在描述窗格内会显示该类别的概要。

您也可在字典面板左上的搜索框输入一个或多个词来搜索字典中的条目。搜索结果会显示在对象窗格内，选择条目，在描述窗格内会显示对应概要。

当您完成本教程，更加了解对象和类之后，不妨花些时间浏览字典中的各种命名空间，看看不同的类及其属性的相互关系。字典包含了 EasyLanguage 支持的所有类、属性、方法、事件和其他对象特征的参考资料。很多说明也带有链接，可以跳转到概括了语言功能及相关对象元素的 EasyLanguage 帮助主题。

EasyLanguage 中的对象

本教程旨在介绍对象在 EasyLanguage 中的用法，所以显然第一个问题就是，“为什么要使用对象？”简单来说，对象可以在 EasyLanguage 中实现一些之前无法做到的事情，而对于您在 EasyLanguage 中早已熟悉的事情，至少提供了灵活性更高、结构更优秀的替代方法。

那么，对象是什么？简而言之，对象代表特定类型的值，在 EasyLanguage 代码中可以通过用户定义的名称来访问。等等……听上去很像变量的定义，不是吗？事实上，就是这样！对象和变量很相似，不过，不是我们此前使用 EasyLanguage 事已经遇到过的传统变量（保存单个数值、字符串或布尔值），对象变量代表的是一个集成的程序包，包含值以及对值操作的相关元素。

在对象中，值称为“属性”，可以看作是具体对象的保留字与变量。对象元素称为“方法”（类似于 EasyLanguage 的函数），可搭配运算符和事件等其他元素执行特定计算以及管理对象任务。

按照程序包的概念，属性和方法的名称总是与所属对象写在一起（详见后节）。例如，要使用由名为 myObject 的变量所代表对象的属性，需要在对象变量名后加上一个 ″.″ 和所需的属性名，如下所示：

```
Value1 = myObject.PropertyName;
```

因为对象是独立模块，将数据和程序结合成为方便的程序包，所以无需知道对象内部的任何代码即可使用对象支持的属性值与方法。因此在访问复杂的数据结构（例如价格数据流）或操作复杂程序元素（例如窗口控件和事件）时，对象就是个理想选择。使用对象时，对象变量的类型与对象的类以及独特对象变量名（称为“实例”）代表的特定值数据包关联。

对象是什么？

组件的一个副本，知道如何访问、返回以及操作数据
对象数据通过属性返回
对象具有和变量一样的描述性名称

初始组件设置

首先使用工具箱中的组件向 EasyLanguage 文件中添加对象，并用属性编辑器进行设置。

大部分组件都需要一些初始设置后，才能在您的指标或策略中用于收集数据或执行所

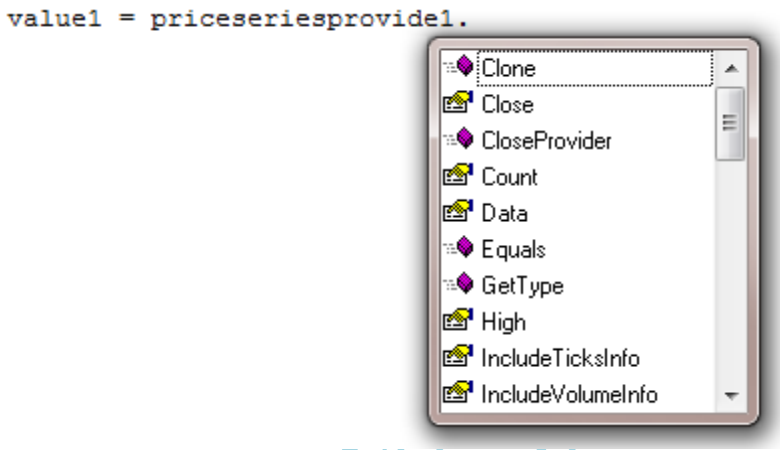
需操作。例如，要通过 PriceSeriesProvide 对象引用价格时需要先指定 Symbol；要通过 OrderTicket 对象下单时需要先指定 AccountID 号。

在本书中，教程示例会在“组件和属性编辑器”标题下列出示例中用到的每个组件需要输入的设置。当您开始在自己的指标与策略中使用组件时，这应该有助于您了解需要注意的属性。

在代码中访问属性

组件对象添加到文件并用属性编辑器完成设置后，就可以从 EasyLanguage 代码访问对象值了。具体方法是在想要引用的属性或方法名与对象名之间使用点运算符 (".")。对于组件，对象的名称通常以组件名称后接实例序号来表示（如 PriceSeriesProvider1）。在对象名称后输入 "." 字符后，在自动完成窗口中将显示可用属性 和方法 的列表。输入更多属性名称字符，列表选择范围会更准确。

在代码编辑器中，在名称 PriceSeriesProvide1 后输入 "."，自动完成窗口会出现，提供一系列属性供选择。



例如，下列语句读取一个 PriceSeriesProvider 对象的 Count 属性并将之赋值给一个变量。Count 属性返回历史周期（K 线）的数量，该数值由名为 PriceSeriesProvider1 组件对象的具体实例表示。

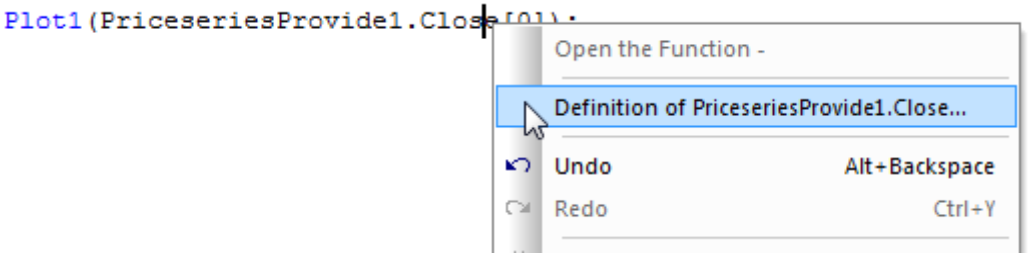
```
Value1 = PriceSeriesProvide1.Count ;
```

PriceSeriesProvider 的另一个属性是 Close 属性。但是，与指向单个值的 Count 不同，Close 属性代表了一系列收盘价格，范围从当前 K 线开始，向后到价格系列范围的开始位置为止。要访问某个具体收盘价，需要使用方括号 [ ] 索引运算符和索引值。在本例中，索引代表相对于对象周期设置的 K 线数量，EasyLanguage 中标准 Close 保留字与之很相似，后者在图表中代表相对于 K 线周期设置往回几根 K 线。这种类型代表一系列值的属性称为“集合”，其中的值使用括号索引值访问。下节对此有详细介绍。

```
Plot1(PriceSeriesProvide1.Close[0]);
```

在本例中，"." 后的 Close[0] 是名为 PriceSeriesProvide1 对象的一个属性，代表该对象可用收盘价集合中 0 根 K 线前的收盘价。注意，与 EasyLanguage 中相似的保留字 Close 不同，在此类集合属性中引用具体收盘价时，在属性标识符中必须指定索引 [0]。

在 EasyLanguage 代码中添加了属性引用后，可以点击该属性名称并从弹出的快捷菜单点击“…（在本例中为 PriceseriesProvide1.Close）的定义”来阅读所选对象和相关属性的帮助主题。



PriceSeriesProvider

PriceSeriesProvider 组件是一个对象，它包含了在一个历史数据的范围内，指定交易代码在某个区间中的实时和历史价格值的集合。每个 PriceSeriesProvider 对象都可以视为是一个能访问价格属性的虚拟图表，如 High（高）、Low（低）、Open（开盘）、Close（收盘）、Volume（量）、Time（时间）和 OpenInterest（未平仓合约）等大家熟悉的名称，其中具体的价格可使用代表“几根 K 线之前”的索引值来引用。但是，与图表中的数据不同，PriceSeriesProvider 对象让您可以预先定义指标需要访问的交易代码和交易周期，且不受图表中的交易代码和交易周期的限制。下面列出了 PriceSeriesProvider 的一些重要特征：

访问任意交易代码的 K 线数据点
历史值与 Data1 一致
可与某些标准 EL 函数一起使用
交易代码、区间和范围独立于 Data1 交易代码

在一个指标中可以包含多个 PriceSeriesProvider 组件来支持多个数据分析，而无需在图表或表格中以数据流方式包含组件的交易代码。这就意味着您可以用对象来创建在雷达屏中使用的多数据指标，从而使图表内设置多数据分析更简单。此外，还可以选择不依赖图表和表格交易周期设置访问 Tick 和 Volume 相关值，包括涨跌点交易笔数与涨跌成交量。

PriceSeriesProvider 也支持 Updated 事件，在 EasyLanguage 代码中，当一个标的价格变动时会触发该事件。

※ 教程示例 #1

目标：（PriceSeriesProvider 收盘价指标）

将 PriceSeriesProvider 组件拖到文件中

用属性编辑器设置组件对象属性

根据组件对象绘制一个属性值的图形

理解 PriceSeriesProvider 数据和图表数据的不同

指标：$01_DailyBarClose

本例用一个 PriceSeriesProvider 组件在 30 分钟周期图上访问并绘制每日收盘价图形。

工作区：$01_DailyBarClose

构建图表

创建：30 分钟周期图

插入指标：$01_DailyBarClose

组件和属性编辑器

PriceSeriesProvide1:

Symbol:	*symbol*	（在 Filters 类别下）
IntervalTyp	*daily*	（在 Filters-Interval-General 类别下）
IntervalSpan	*1*	〃〃
Type	*years*	（在 Filters-Range-General 类别下）
Years	*2*	〃〃

指标属性

刻度值： 与标的物数据同轴

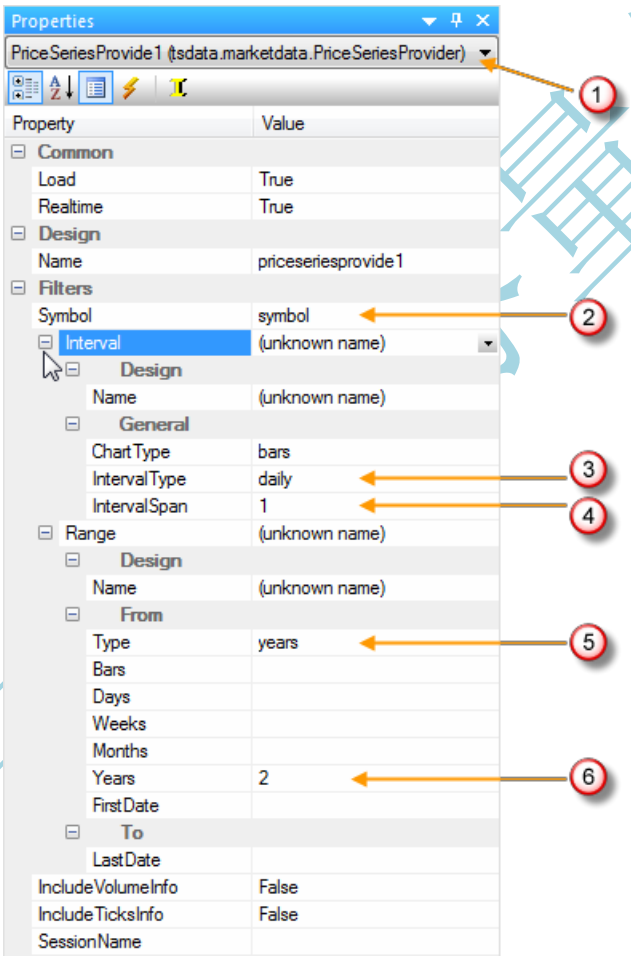
指标练习 #1:'$01_DailyBarClose'

样式约定说明：每个练习中需要输入的 EasyLanguage 代码以 courier 字体表示。从其他练习复制或由组件自动插入的代码以浅灰色背景表示。遇到 EasyLanguage 保留字或函数时，将会用单引号表示（如 'EntryPrice'）。课程练习输入值或变量以斜体表示，如 myInput。

创建新指标并命名为：#01_DailyBarClose。我们使用 # 命名符号来区分教程练习与教程的可下载示例（以 $ 开头）。

首先我们要点击代码编辑器窗口左侧的工具箱选项卡，打开工具箱面板。找到 PriceSeriesProvider 组件，然后点击该组件名称并拖动到代码编辑器。默认名称 PriceSeriesProvide1 将显示在代码编辑器窗口底部的组件盘中。

现在，点击代码编辑器右侧的属性选项卡属性选项卡打开属性面板，确保 PriceSeriesProvider 是属性编辑器顶部的指定组件。



在 Filters 类别下，展开 Interval 和 Range 部分，以便设置 PriceSeriesProvide1 组件的初始值来指定要收集的价格数据。首先，在 Symbol 属性旁输入 'symbol' 。在 Interval 下，将 IntervalType 修改为 daily ，将 IntervalSpan 的值改为 1 。在 Range 下，将 Type 修改为 years ，将Years 的值改为 2 。注意：Type 属性位于 Years 属性上方数

行，所以请确保在正确的行内输入值。这个组件现在已设置为获取 symbol 过去两年内的每日价格数据。

现在，在使用刚才输入的周期（Interval）和范围（Range）设置的图表中，就可以用 PriceSeriesProvide1 组件作为对象在代码中获取当前交易代码的价格了。

在本例中，我们将添加 Plot 语句在图表的 K 线上显示 PriceSeriesProvide1 对象的最后收盘价。

要在代码中访问一个对象的属性，在该对象名称后输入点运算符 '.'，后面接对象属性的名称，在本例中为 '.Close'。因为我们要显示该属性的当前收盘价，所以要在属性名称后添加'[0]'来指定当前 K 线的收盘价。

```
plot1(PriceSeriesProvide1.Close[0],"Daily Close");
```

现在将编辑光标放在 EasyLanguage 文件的空白处，右键单击打开快捷菜单并选择底部的“属性”。在指标属性对话框下的刻度轴选项卡上，将坐标轴设置改为刻度位置：与标的物数据相同，以便在分割条上方显示该图。

验证指标。如果没有显示，应该从视图-工具栏-输出菜单打开 EasyLanguage 输出栏。这可以让您看到任何校验错误以及纠正错误的有用信息。

对象与函数

在分析市场数据时，经常会使用以一系列价格为基础的计算结果，例如在一定数量 K 线范围内的收盘价移动平均数。在老版本的 EasyLanguage 中，您可能会用一个函数加一个价格来实现：

```
Value1 = Average(Close,10);
```

当前图表的 10 K 线的平均收盘价赋值给了 Value1。然后您可以绘制 Value1，或者将其用于其他计算或表达式。这是可行的，因为保留字 Close 并不只代表当前的收盘价，而是包含从当前 K 线到 MaxBarsBack 的一系列 K 线收盘价。

与之类似，对象属性 PriceSeriesProvide1.Close 在未附带索引时，代表的是基于 PriceSeriesProvide 指定周期与范围的收盘价格的集合。下列语句将 PriceSeriesProvide1 对象的 10 K 线平均收盘价赋值给 Value2：

```
Value2 = Average(PriceSeriesProvide1.Close,10);
```

通常，包含一系列价格数值的对象或属性可以搭配 EasyLanguage numericseries 类型的函数输入值使用。在代码中右键点击函数名并选择“打开函数 - 名称”，可查看任何函数的输入类型。

例如 Average 函数的输入值（如下所示），Price 输入值是 numericseries 类型，除了可以接受带价格集合的对象，还可以接受包含历史值的保留字（例如 High、Low、Open、Close 等）。

Average 函数，前三行代码：

```
Inputs:
   Price( numericseries ),
   Length( numericsimple ) ;
```

价格集合

集合是一种对象（或属性），与 EasyLanguage 的数组相似，能保存多个值，通过索引访问具体的值。以价格属性为例，比如 PriceSeriesProvider 对象的 .Close 属性可保存对该组件指定的周期范围内的一系列收盘价。如前例，可以在属性名称后加方括号 [] 和索引值引用某个具体的收盘价，比如 PriceSeriesProvide1.Close[2] 从名为 PriceSeriesProvide1 的组件对象获取两个周期前的收盘价。

但是，如果去掉方括号，PriceSeriesProvide1.Close 属性引用的是组件的整个收盘价集合，并可以作为参数传递给任何接受数字序列的标准 EasyLanguage 函数。例如，下面的语句

计算并绘制 PriceSeriesProvide1 对象最近 15 个周期的平均收盘价。

```
Plot1(Average(PriceSeriesProvide1.Close,15);
```

后面我们会详细讨论集合，目前您只需知道价格集合（如 PriceSeriesProvider 中的价格集合）可以使用方括号加索引值（表示几根 K 线之前）来返回具体的价格值，也可以不使用方括号将价格序列传递给函数。

PriceSeriesProvider – Count 属性

PriceSeriesProvider 组件由一个交易代码在指定周期范围内的一系列价格组成。Count 属性说明在 PriceSeriesProvider 集合中，当指标在图表或雷达屏行遍历时，相对于当前 K 线有多少根 K 线的历史价格可用。

为确保在 PriceSeriesProvider 中有足够的历史价格来执行价格集合的计算，应该在执行计算代码前检查 Count 属性的值。例如，如果要确保有足够价格数据来计算指定的移动平均线，可以按下例使用 Count 属性：

```
Input:MovAvgLen(10);
If PriceSeriesProvide1.Count > MovAvgLen then
   Plot2(Average(PriceSeriesProvide1.Close,MovAvgLen));
```

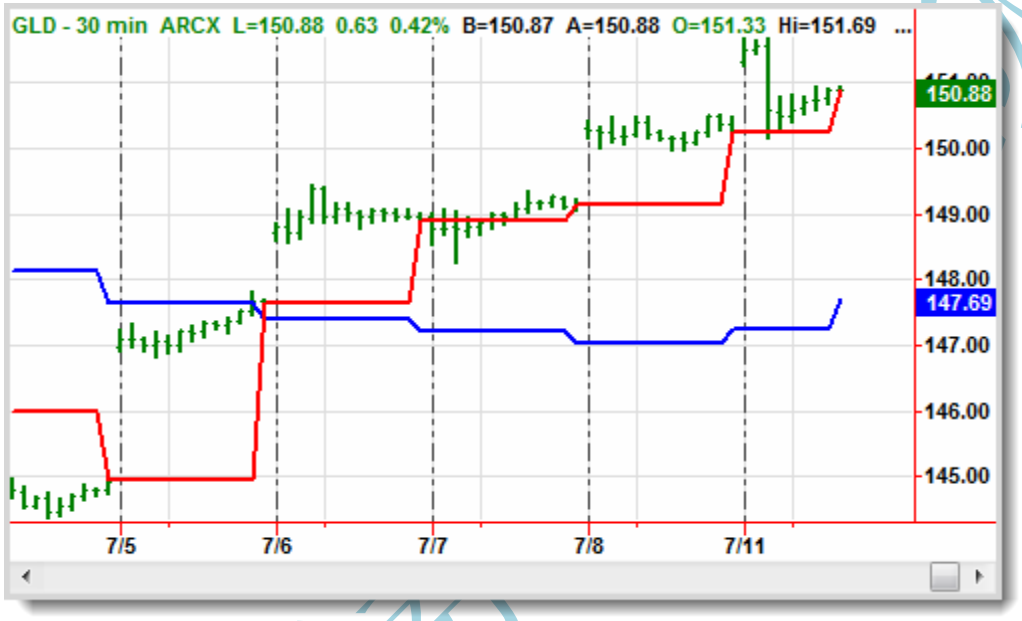
※ 教程示例 #2

目标： （MovAvg + Close 指标）

搭配 EasyLanguage 'Average' 函数使用 PriceSeriesProvider 对象

指标： '$02_DailyAvgClose'

这个指标使用一个 PriceSeriesProvider 组件访问并在 K 线上绘制 10 天移动平均线。



工作区：$02_DailyAvgClose

构建图表：

创建：30 分钟日内图
插入指标：$02_DailyAvgClose

组件和属性编辑器设置：

PriceSeriesProvide1:

Symbol:	*symbol*	（在 Filters 类别下）
IntervalTyp	*daily*	（在 Filters-Interval-General 类别
IntervalSpa	*1*	″ ″
Type	*years*	（在 Filters-Range-General 类别下）
Years	*2*	″ ″

指标练习 #2: '$02_DailyAvgClose'

样式约定说明：每个练习中需要输入的 EasyLanguage 代码以 courier 字体表示。从其他练习复制或由组件自动插入的代码以浅灰色背景表示。遇到 EasyLanguage 保留字或函数时，将会用单引号表示（如 'EntryPrice'）。课程练习输入值或变量以斜体表示，如 myInput。

在本例中，我们将从教程示例 #1 中创建的 #01_DailyBarClose 开始。将之保存为 #02_DailyAvgClose。再说一遍，我们两使用 # 命名符号来区分教程练习和教程的可下载示例。

因为 PriceSeriesProvide1 组件已经是刚刚创建的指标的一部分，不用再次添加。但如果从头开始，则需要将工具箱中的 PriceSeriesProvider 组件拖到文件内，并按照上方"组件和属性编辑器设置"用属性编辑器设置组件的 Symbol（交易代码）、Interval（周期）和 Range（范围）值。

上一示例的绘图语句显示的是一条线，连接每个相邻的 PriceSeriesProvide1 K 线的最新收盘价。记住，这与在相同的 K 线上绘制老版本保留字 Close 几乎一样，只是对象属性要求在属性名后面用 '[0]' 来指定当前收盘价。

现在，在原来的绘图语句前，我们要加一个输入声明，指定移动平均线长度：

```
input:MovAvgLen(10);

plot1(PriceSeriesProvide1.Close[0],"Daily Close");
```

接下来，我们将要加入第二个图，显示 PriceSeriesProvide1 组件集合内收盘价的移动平均线。我们将检查组件的 .Count 属性来确保它包含计算所需的足够价格数据。

Average 函数可接受两个参数，第一个代表一系列价格，第二个代表对往回多少根 K 线的价格计算平均数。使用 Close 属性（不加方括号）来引用基于 PriceSeriesProvide1 周期和范围设置的整个收盘价集合。第二个参数表示从当前价格开始往回用于计算计算移动平均数的周期数量。

```
If PriceSeriesProvide1.Count > MovAvgLen then
    plot2(Average(PriceSeriesProvide1.Close,MovAvgLen),"MovAvgLen");
```

验证指标。

输入与属性

除了在属性编辑器中将初始组件属性设置为特定值外，还可以设置属性使用 EasyLanguage 输入作为其值。在图标或表格上应用指标或策略并由组件使用该值时，这可以支持输入值的修改。

例如，当设置一个组件对象（比如 PriceSeriesProvide1）的 Symbol 属性时，假设输入 "SPY" 作为交易代码值，那么每次运行您的指标时，组件将用属性编辑器中指定的周期设置来访问 SPY 的价格。假设您想要能通过输入来修改这个交易代码，在您编写标准 EasyLanguage 代码时也会遇到这样的情况。点击 Symbol 右侧，然后点击属性编辑器顶部的  图标。此操作会将 iSymbol1 作为 Symbol 属性的值插入，并在您的 EasyLanguage 代码中插入下列输入语句，并将输入的交易代码名称设置为默认输入。

```
Input: string iSymbol1( "SPY" );
```

请注意，如果此前未在属性编辑器中设置交易代码值，iSymbol 的初始输入值会被设为 ""。

常见 Provider 属性

工具箱中有不少组件的名称都以 'provider'（提供器）一词结尾。 使用属性编辑器可以看到，它们都包含 'Common' 类别下的 Load（加载）和 Realtime（实时）属性。这些属性用于控制 Provider 组件的初始状态，默认情况下两者都设置为 True（真）。如果为真，当指标或策略初次加载时，Load 属性会告诉 Provider 自动建立相关数据的连接。例如，AccountsProvider 会连接到与指定账户相关的数据，PriceSeriesProvider 会连接到指定交易代码的价格数据流，等等。Realtime 属性为真时，Provider 会不断从关联数据请求实时数据更新。例如，PositionsProvider 会根据所请求交易代码的实时价格变化计算新仓位值，MarketDepthProvider 会报告买卖行情的实时变化等。

大部分情况下这些设置都不需要修改；但是，如果希望设置一个 Provider 组件，当某个代码中的条件出现时可以被激活，则可以在属性编辑器里设置 Load 和 Realtime 属性为 false，之后在代码中合适的地方再将它们设为 true。另外，如果想在代码中改变一个已加载 Provider 的 filter （筛选）属性设置，需要暂时将 Load 属性设为 false 来关闭数据连接，给 filter 赋新值，然后再把 Load 设回为 true，以便根据新的 filter 设置重新建立数据连接。

※ 教程示例 #3

目标：（ RelStrengthRS ）

添加 Symbol 属性输入

在 RadarScreen®（雷达屏）上执行多数据分析

指标：'$03_RelStrengthRS'

这个指标用一对 PriceSeriesProvider 组件来计算指定天数内两个交易代码的百分比变化。

	Symbol	Interval	Last	Low	High	Volume Today	$03_RelStrengthRS		
							Rel Perf	Sym Perf	Bench Perf
1	SPY	Daily	132.13	131.84	133.18	110,731,079	0.00	3.55	3.55
2	CSCO	Daily	15.41	15.35	15.62	26,750,312	(1.70)	1.85	3.55
3	IBM	Daily	174.84	174.61	176.15	2,632,484	3.60	7.15	3.54
4	USO	Daily	37.19	36.94	37.63	6,295,587	(8.35)	(4.81)	3.53
5	GLD	Daily	150.88	150.20	151.69	13,284,890	(2.44)	1.10	3.54
6	AAPL	Daily	354.65	353.34	359.77	11,475,256	5.28	8.82	3.54
7	GE	Daily	18.57	18.50	18.78	27,835,150	(2.19)	1.36	3.55

工作区：$03_RelStrengthRS

建立雷达屏：

创建：雷达屏窗口，交易代码设为日周期

指标属性/表格样式：设置每个指标根据 Number 绘图，精确到小数点后两位

插入指标：$03_RelStrengthRS

组件和属性编辑器设置：

PriceSeriesProvide1:

Symbol:	*iSymbol1*	（作为输入添加）
IntervalTyp	daily	（在 Filters-Interval-General 类别下）
IntervalSpa	1	〃 〃
Type	years	（在 Filters-Range-General 类别下）
Years	2	〃 〃

指标练习 #3:'$03_RelStrengthRS'

在这个指标中，我们将把雷达屏中各行当前交易代码的表现和一个基准交易代码进行比较。在引入对象概念前，这种类型的多数据分析在雷达屏中是无法实现的。各行当前交易代码的表现用标准的 Close 保留字计算，但是基准表现将会用来自某个参考代码的 PriceSeriesProvider 组件对象的收盘价集合来计算。

创建新指标并命名：'#03_ RelStrengthRS'。

打开工具箱，找到 PriceSeriesProvider 组件，然后将其拖到代码编辑器。在代码编辑

器窗口底部的组件盘中将显示默认名称 PriceSeriesProvide1。

打开属性编辑器。点击 Symbol 右侧的空文本框，然后输入交易代码名 "SPY"（含引号）作为参考交易代码。当我们将该指标应用于图表时，有可能会想更改参考代码，因此我们将点击属性编辑器顶部的  工具栏图标按钮，把这个代码值变为输入值。在属性编辑器内，iSymbol1 出现在 Symbol 旁，并且应该可以看到下列代码插入到 EasyLanguage 代码的顶部，这里采用了默认交易代码 "SPY" 作为初始值。

```
Input: string iSymbol1( "SPY" );
```

还是在属性编辑器内，找到 Filters 并展开 Interval 和 Range 部分，以便设置 PriceSeriesProvide1 组件的初始值来指定要收集的价格数据。在 Interval 下，把 IntervalType 改为 daily，IntervalSpan 值改为 1。然后在 Range 下，把 Type 修改为 years，把 Years 的值改为 2。注意：Type 属性位于 Years 属性上方数行，所以请确保在正确的行内输入值。这个组件现在已设置为获取 PriceSeriesProvide1 交易代码过去两年内的每日价格数据。

在代码内，再声明一个输入值，设置用于表现回顾值的回溯 K 线默认数量。

```
Input:LookBack( 20 );
```

声明三个变量来保存百分比变化的计算结果（用于衡量各行当前交易代码表现）、基准参考交易代码以及它们之间的相对差。

```
Vars:  SymbolPerf ( 0 ), BenchPerf( 0 ), RelPerf( 0 );
```

接下去，我们将计算基础（当前）交易代码以及基准交易代码的百分比变化值。

当前行交易代码的表现使用保留字 Close 计算，结果作为参数与 K 线范围一起传递到 PercentChange 函数，然而计算出百分比变化。

```
SymbolPerf= PercentChange(Close,LookBack);
```

基准交易代码表现使用来自 PriceSeriesProvide1 对象的收盘价集合进行计算。

```
BenchPerf = PercentChange(PriceSeriesProvide1.Close,LookBack);
```

相对表现是两个交易交易代码百分比变化之间的差异，需计算得出。

```
Relperf = (SymbolPerf-BenchPerf);
```

最后，绘制计算结果值。参考值要乘以 100，以便显示为百分比。

```
plot1(RelPerf * 100,"Rel Perf");
plot2(SymbolPerf * 100, "Sym Perf");
plot3(BenchPerf * 100, "Bench Perf");
```

验证指标。

将指标插入到交易代码周期为日的雷达屏窗口内。

※ 教程示例 #4

目标：(RelStrengthFixed)

使用多个 PriceSeriesProvider 组件

指标：'$04_ RelStrengthFixed'

这个指标用一对 PriceSeriesProvider 组件来计算指定天数内两个交易代码的百分比变化。

工作区：$04_RelStrengthFixed

构建图表：

创建：30 分钟日内图
插入指标：$04_RelStrengthFixed

组件和属性编辑器设置：

PriceSeriesProvide1:

Symbol:	*iSymbol1*	（在 Filters 类别下）
IntervalType:	*daily*	（作为输入添加）
IntervalSpan:	*1*	（在 Filters-Interval-General 类别下）
Type	*years*	〃 〃
Years	*2*	（在 Filters-Range-General 类别下） 〃 〃

PriceSeriesProvide2:

Symbol:	*symbol*	（图表或行中的交易代码）

IntervalType:	*daily*	（在 Filters-Interval-General 类别下）
IntervalSpan:	*1*	〃 〃
Type	*years*	（在 Filters-Range-General 类别下）
Years	*2*	〃 〃

指标练习 #4:'$04_ RelStrengthFixed'

这个指标类似于之前的例子，只是本指标旨在使用一对周期均为日的 PriceSeriesProvider 组件在图表上绘制相对表现。这可以实现将当前交易代码的每日表现与基准交易代码的每日表现的比较，而无需考虑图表的周期设置。

在本例中，我们将从教程示例 #3 中创建的 #03_DailyAvgClose 指标开始。将之保存为 #04_RelStrengthFixed。同样，我们使用 # 命名符号来区分教程练习与教程的可下载示例（以 $ 开头）。

因为 PriceSeriesProvide1 组件已经是刚刚创建的指标的一部分，不用再次添加。但如果从头开始，则需要将工具箱中的 PriceSeriesProvider 组件拖到文件内，并按照上方“组件和属性编辑器设置”用属性编辑器设置 PriceSeriesProvide1 组件的 Symbol、Interval和Range值。

接下来，我们从工具箱内添加第二个组件。找到 PriceSeriesProvider 组件，然后点击该组件名称并拖动到代码编辑器。采用默认名称 PriceSeriesProvide2 的第二个组件将显示在代码编辑器窗口底部的组件盘中。

选中 PriceSeriesProvide2 组件，转到属性编辑器。在 Symbol 属性旁，输入 symbol 来将该属性设置为图表的当前交易代码。在 Filters 类别下，展开 Interval 和 Range 部分，以便设置 PriceSeriesProvide1 组件的初始值来指定要收集的价格数据。在 Interval 部分，把 IntervalType 改为 daily，IntervalSpan 值改为 1。然后在 Range 下，把 Type 修改为 years，把 Years 的值改为 2。第二个组件现在已设置为获取 symbol 过去两年内的每日价格数据。

我们将使用与练习 #3 中相同的输入名称；不过，让我们把初始 LookBack 值改为 '8'。

```
Input: string iSymbol1( "SPY" );
Input:LookBack( 8 );
```

变量名称也会基于练习 #3，不过，让我们把当前交易代码表现变量的名称改为 SymbolPerf。

```
Vars:  SymbolPerf( 0 ), BenchPerf( 0 ), RelPerf( 0 );
```

计算基础（当前）交易代码以及基准（参考）交易代码的百分比变化值。我们将把表现计算放在一个 "if" 语句内，在每天的第一根 K 线计算每日表现。记住，我们将从刚刚创建的第二个 PriceSeriesProvider 组件获取参考交易代码日周期的收盘价，这样，图表周期不同，表现也会总是基于两个 Provider 的日周期。

当前交易代码表现用 PriceSeriesProvide2 组件对象的收盘价集合计算，在计算百分比变化时，PriceSeriesProvide2 与回顾的 K 线数量一起作为参数传递给 PercentChange 函数。基准交易代码表现的计算使用的是 PriceSeriesProvide1（基于输入值 iSymbol1）的收盘价

集合以及相同的 PercentChange（百分比变化）回顾长度。

```
If Date <> Date[1] then
begin
    SymbolPerf = PercentChange(PriceSeriesProvide2.Close, LookBack) ;
    BenchPerf = PercentChange(PriceSeriesProvide1.Close, LookBack);
end;
```

计算基本和参考百分比变化值之间的相对表现差距。

```
RelPerf = SymbolPerf - BenchPerf;
```

绘制三个计算值，这次删除 *100语句，因为我们不需要在图表中将这些值绘制为百分数。添加一条零基准线作为第四个图形，用来比较正负表现值。

```
Plot1(SymbolPerf  , "Symbol Perf");
Plot2(BenchPerf  , "Bench Perf");
Plot3(RelPerf    , "Rel Perf");
Plot4(0);
```

右键单击指标文件，选择右键菜单底部的“属性”来访问“指标属性”对话框。选择“刻度轴”选项卡并确认“坐标轴”设定为右侧坐标轴 (Scale On: Right Axis)。

验证指标。

方法

方法是一种包含 EasyLanguage 语句的代码结构，仅在所在的 EasyLanguage 文件内有效（本地）。在代码中引用方法的方式与引用函数非常相似，但因为方法是本地的，所以运行速度更快，并且支持直接读写指标或策略中的其他变量。虽然和函数一样，方法可以接受输入参数并返回值，但是使用 void 方法，可以执行一组语句而不要求返回值，从而可以让方法像一些保留字一样在代码中独立存在，而不需要被赋值给变量。下面是方法特点的总结：

方法可以返回值
方法可以有一个或多个参数
方法可以修改标准的变量
方法可以声明只在该方法内可见的本地变量
方法通常比标准 EasyLanguage 函数更快

声明一个方法要使用保留字 method 后接返回值类型，方法名称以及方法的参数。方法参数用英文括号表示，用英文逗号分隔。即便没有指定参数，括号也是必需的。方法的主体包含一个 begin/end 块，其中在 EasyLanguage 语句前面是任何本地变量声明（如有）。

```
method void myMethod(int param1)
var: double localVar1, int localVar2
begin
  { EasyLanguage statements }
  localVar1 = myObject.Close[0];
  localVar2 = 10;
  if param1<localVar2 then
       plot1(localVar1);
end;
```

void 返回值意味着该方法不返回值，空括号表示该方法不需要参数。

EasyLanguage 方法中的代码仅在该方法被调用时执行，并非每次评估指标或策略时都会执行。在下例中，当 Condition1 为真时将在您的文件中执行前面方法里的语句。这和外部函数很相似，但是方法是文件本地的。当在一个文件中需要多次调用代码时，经常会使用方法，但是在将 EasyLanguage 代码组织为命名的模块时也很有用。此外，方法也用于作为事件处理器并与对象的事件属性关联。

```
Value1 = 5;
If Condition1=True then
    myMethod(Value1);
```

EasyLanguage 文件中的本地方法必须始终跟在 Inputs、Vars 和 Arrays 的 EasyLanguage 声明后，在指标代码主体之前。

事件

事件是对象在其内部发生了关注情况时为客户程序（您的 EasyLanguage 代码）提供通知的一种方法。事件最常见的用法之一是图形控件的编程，例如按钮，当用户点击按钮时程序会收到通知。另一个例子是定时器控件，当指定倒计时结束时会通知调用该控件的程序。下面是事件的一些特征：

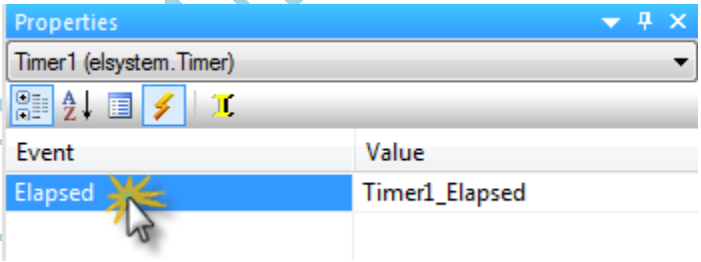
事件会监听要发生或改变的事
被触发后，事件会调用事件处理器方法
处理事件要执行事件处理器方法内的代码
在属性编辑器内，事件与事件处理器方法关联

不过，事件并不仅限于图形界面或定时器用途。EasyLanguage 中的很多数据提供器组件也支持在与对象相关的数据发生变化时，以事件发出通知，例如价格、仓位或账户状态更新。在过去，您可能必须要写一段轮询循环代码来在进入数据中查找变化，但是采用对象事件后，在您的指标或策略中添加一个事件处理器方法即可自动完成这一目标。

事件处理器

在 EasyLanguage 中，事件处理器是在对象内相应事件发生时被调用的方法。在事件处理器方法中，您可以自行编写 EasyLanguage 语句来设定指标、函数或策略如何响应事件。

事件处理器方法必须要与事件关联后才能使用。在组件对象中，可使用属性编辑器将方法与指定事件关联。例如，假设在您的 EasyLanguage 文件中有一个 Timer 组件，属性编辑器的 Event 部分会显示一个空白 Elapsed 属性。



双击 Elapsed 将会自动在您的 EasyLanguage 代码中创建一个事件处理器方法，并将该方法（在本例中为 Timer1_Elapsed）与 Elapsed 事件关联。现在，每次 Timer1 的 Elapsed 事件被触发时，就会执行您添加到关联方法中的 EasyLanguage 代码。如下所示，事件处理器方法通常有一个 Sender 和一个 Args 参数。这些参数用于接收与事件相关的信息，不应该自行编辑。

```
method void Timer1_Elapsed( elsystem.Object sender,
elsystem.TimerElapsedEventArgs args )
begin
```

```
    { Insert your EasyLanguage statements below }
  end;
```

对于组件，事件处理器方法的名称通过属性编辑器与事件相关联。

对于非组件对象，可使用下列格式在代码中将事件处理器方法分配给对象事件属性：

```
Object.EventName += Method_Name;
```

设计器代码

除了使用 EasyLanguage 编辑器编写的代码外，您的分析文件也可以包含关于每个组件的信息，当创建组件和编辑属性时，会自动生成此类信息。这些信息保存在一个特殊的称为设计器代码 (Designer Generated Code) 的只读页上，可通过代码编辑器的视图菜单访问。

设计器代码由系统创建，每当输入或改变组件属性时，会在属性编辑器内更新。这些代码在代码编辑器中不能直接修改。

当用属性编辑器创建一个事件时（例如 Timer 对象的 Elapsed 事件），就会将一个 EasyLanguage 语句添加到设计器代码块，该语句会将事件处理器方法（位于 EasyLanguage 代码的主体部分）的名称指派给组件的事件。

```
timer1.elapsed += timer1_elapsed;
```

在上面的例子中，每当 Timer1 组件对象触发 Elapsed 事件时，就会调用代码中的 ’timer1_elapsed’ 方法并执行方法内的语句。

暂时来说，您无需理解组件生成的设计器代码，但这是很有趣的资源，能够让我们逐渐了解 EasyLanguage 对象是如何创建和操作的。

定时器

Timer 组件会创造一个对象，让您可以设置一个定时器来按指定毫秒数倒计时，并在到时间后调用事件处理器方法。定时器可以被设置为每次到期后自动重新开始，这样事件处理器可以不断被调用。

※ 教程示例 #5

目标：（Bar Countdown 指标）

用工具箱组件创建一个定时器对象

用属性编辑器创建一个事件

在出现事件通知时运行特定 EasyLanguage 代码

指标：'$05_BarPcntLeft'

该指标使用一个 Timer 组件来设置一个定时器，每隔 1000 毫秒不断更新当日时间。1000 毫秒 = 1 秒。

工作区：$05_BarPcntLeft

构建图表：

创建：1 分钟图表

插入指标：$05_BarPcntLeft

组件和属性编辑器设置：

Timer1:

- Interval :*1000*（在 General 类别下）
- AutoReset:*True* 〃 〃
- Enable: *True* 〃 〃
- Elapsed:Timer1_Elapsed

指标练习 #5:'$05_BarPcntLeft'

创建新指标并命名为：'#05_BarPcntLeft'。我们使用 # 命名符号来区分教程练习与教程的可下载示例（以 $ 开头）。

点击代码编辑器窗口左侧的工具栏选项卡，将 Timer 组件拖到代码编辑器中。默认名称 Timer1 将显示在代码编辑器窗口底部的组件盘中。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 Timer1 是属性编辑器顶部的指定组件。

在 Property（属性） 窗格的 General 类别下，将 Interval 值设置为 1000。这个值的单位是毫秒，因此实际是设置定时器在 1 秒钟内倒计时到零。在本例中，AutoReset 属性设为 True，这样定时器每过一秒都会自动重置并重新倒计时。另外，将 Enable 设为 True，这样只要指标应用到分析窗口，定时器就会开始运行。

然后点击 图标切换到属性编辑器的 Event（事件）窗格。可以看到 Event 栏中列出的 Elapsed 值为空白。我们只要在 EasyLanguage 文件中希望调用事件处理器方法的地方加入定时器，当定时器倒计时到零时，Elapsed 事件将由 Timer1 对象触发。只需双击事件名称 Elapsed 在代码编辑器内创建方法，新方法名会添加到 Value 栏。

现在，名为 Timer1_Elapsed 的事件已添加到文件。注意，事件处理器括号中的参数是自动生成的，请勿删除或自行修改。您的 EasyLanguage 代码应插入到 begin 和 end 语句之间。

```
method void Timer1_Elapsed( elsystem.Object sender, elsystem.TimerElapsedEventArgs
    args )
begin
  { Insert your EasyLanguage statements below }
end;
```

在 Timer1_Elapsed 方法语句的前一行，输入三个变量声明。第一个是 intrabarpersist 变量，用于在当前 K 线对发生的定时器事件计数。这是 K 线内的持续变量，它需要在每笔交易更新后保留每个新值，而不是像普通变量一样重置。第二个变量保存最大计数器数值。第三个变量保存最后形成的 K 线的 K 线编号。

```
var: intrabarpersist iCounter(60),iMax(60),barnumb(0);
```

注意：变量声明语句必须放在任何方法之前。

在 Timer1_Elapsed 方法的 begin 和 end 语句中间，输入一个 if 语句，用于在出现定时器事件时减少 iCounter 的值，另外输入另一个 if 语句绘制 K 线计数器的直方图。

```
method void Timer1_Elapsed( elsystem.Object sender, elsystem.TimerElapsedEventArgs a
      rgs )
begin
    If iCounter > 0 then
        iCounter = iCounter-1;
    If currentbar > barnumb then
        PlotValues();
end;
```

再添加一个名为 PlotValues() 的方法，用于在定时器事件发生时调用。第一个图显示了剩余 K 线事件百分比的直方图。第二和第三个图显示保持指标为同一大小的参考线。

```
Method void PlotValues()
begin
   plot1((iCounter/iMax)*100, "PctLeft");
end;
```

代码的最后部分检查是否图表中的最后一根 K 线已经结束，并在 K 线周期的分钟数后重置计数器与最大计数器值。同时保存结束 K 线编号，这样下一个绘图动作将只会在下一笔 K 线交易后发生。

```
Once If Bartype = 1 then
              iMax = Barinterval*60;

if LastBarOnChart then
begin
    if BarStatus( 1 ) = 2 then
       begin
           plot1(0, "PctLeft");
           iCounter = iMax;
           barnumb = currentbar;
       end;
end;

    plot2(100, "100");
    plot3(0, "zero");
```

验证指标。

AccountsProvider

AccountsProvider 组件用于创建组件来引用真实或模拟的 TradeStation 账户。

AccountsProvider 对象属性编辑器中的 Accounts filter（账户筛选）属性用于指定希望访问数据的（一个或多个）账户。AccountsProvider 对象会使用您指定的筛选标准建立一个账户集合，并将 Count 属性设置为该集合中的元素个数。默认情况下，空白的 Accounts filter 会从全部有效账户收集账户信息。

AccountsProvider 可以搭配 Updated 事件使用，这样当与任何被引用账户的关联值改变时，代码就会得到通知。如果您想要在指标或策略中知道某个交易代码并不处于您的图标或表格图中的账户的变化，这个搭配就特别有用。

在您的代码中，在尝试从账户集合访问索引元素前，通常使用 Count 属性来确定是否有 AccountsProvider 找到的任何账户。

AccountsProvider1 的 Account 属性代表一个或多个账户的集合（实际上就是 Account 对象）以及可从每个账户读取的一系列属性。例如，您可以通过在 provider 组件的 Account 属性后添加带方括号的索引标识符来访问集合中的账户，（在本例中为 [0]，访问首个账户）。

```
AccountsProvider1.Account[0]
```

引用了特定的 Account 后，即可在索引的 Account[0] 标识符后添加 '.' 和属性名来添加关注账户的属性，例如 .BDAccountNetWorth（表示集合中首个账户的首日净值）。

```
Value1 = AccountsProvider1.Account[0].BDAccountNetWorth;
```

如果在属性编辑器中对 Accounts 属性只指定一个账户，那么集合将包含 Account[0] 的数据。但是，如果从三个账户请求数据（例如 SIM12345、SIM67890、SIM1357X），第二个账户就是 Account[1]，第三个是 Account[2]。下列语句从包含至少三个账户的 AccountsProvider1 对象读取 Account[2]（第三个账户） 的 UnsettledFund（未结算资金）属性的值。

```
Value1 = AccountsProvider1.Account[2].UnsettledFund;
```

集合

组件对象通常要操作项目集合。例如，AccountsProvider 对象管理的是您可以访问的所有账户的集合（由 Accounts filter 指定）。

下列语句显示集合中首个账户（索引值为 0）的账号：

```
Plot1(AccountsProvider1.Account[0].AccountID);
```

Count 属性告诉您在对象集合列表中有多少个条目，以便浏览整个集合列表。如果 Count 为 0，则集合中没有条目。

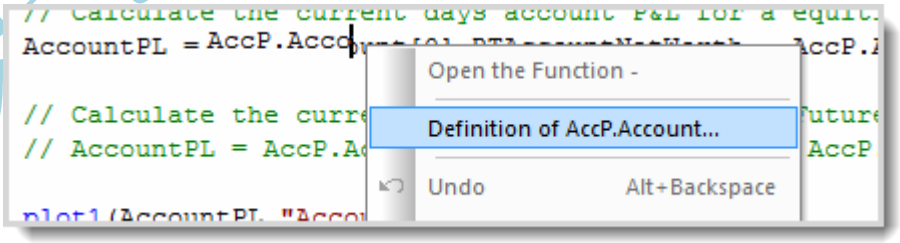
```
Value1 = AccountsProvider1.Count;
```

AccountsProvider 对象在属性编辑器中列出的 Accounts filter 用于指定希望从哪个或哪些账户访问数据。AccountsProvider 对象会使用您指定的筛选标准建立一个账户集合，并将 Count 属性设置为该集合中的元素个数。默认情况下，空白的 Accounts filter 会从全部有效账户收集账户信息。

查询定义与帮助

要更多了解一个组件的属性和方法有几种方法。例如在属性编辑器中，选择一个组件属性后，编辑器下部的面板会显示字典内该属性的简短说明。在描述窗格上方是一个写着 Help about Property（关于属性的帮助）的链接，该链接会跳转到组件所属类的帮助主题。帮助主题通常包含指定类的一系列属性、方法和事件，并提供相关类和属性的链接。

在代码编辑器中，可以右键单击代码引用的对象，然后在右右键菜单中选择“对象的定义(D)...”。这样将会直接跳转到该对象类的帮助主题。



顺便说一句，如果右键单击属性名称，“属性的定义...”菜单项将跳转到包含该属性的类。但是，请注意属性可能与组件处于不同的类，因为很多属性是从其他的类继承的，其定义包含在基类中。比如 .RTAccountNetWorth 属性，它从 AccountsProvider 组件中

访问，但其实是 Account 类的成员，与组件的 Account[0] 属性关联。

※ 教程示例 #6

目标：（账户损益指标）

创建一个可访问您的权益账户信息的对象

使用 Count 属性来检查是否有账户

用更短的名称重命名 provider

指标：'$06_AccountPL'

本指标用一个 AccountProvider 组件访问指定权益账户的值。账户的 Beginning-Day Net Worth（首日净值）和 Real-Time Net Worth（实时净值）属性值的差值经计算后绘制为直方图。如果账户利润大于 0，图的颜色是绿色；如果净值差小于 0，则为红色。

构建图表：

创建：带有 5 天价格数据的 5 分钟图

插入指标：$06_AccountPL

组件和属性编辑器设置：

AccP:

Name:	AccP	（修改组件名称）
Accounts:	*iAccounts1*	（作为输入添加）
Updated:	AccP_Updated	

指标练习 #6:'$06_AccountPL'

创建新指标并命名：'#06_AccountPL'。我们使用 # 命名符号来区分教程练习与教程的可下载示例（以 $ 开头）。

点击代码编辑器左侧的工具箱选项卡，将 AccountsProvider 组件拖到代码编辑器中。默认名称 AccountsProvider1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 AccountsProvider1 是属性编辑器顶部的指定组件。

在属性编辑器顶部，选择 AccountsProvider1，找到 Name 属性，将值改为 AccP。这样就缩短了组件对象的名称，写代码时可以少打点字。但组件对象的特征及其属性保持不变。注意在代码编辑器底部的组件盘中现在显示的是修改的名称。

点击属性名称 Accounts 右侧的空文本框，然后点击属性编辑器顶部的 图标。现在，iAccounts1 应该会在属性编辑器中显示在 Accounts 旁。另外，在您的 EasyLanguage 代码顶部将会插入下行代码。

```
Input: string iAccounts1( "" );
```

在两个引号中间，输入一个您模拟账户的账号，该账号将用于 AccP 组件对象。当您请求 AccP 的账户属性时，这就是被引用的账号。

```
Input: string iAccounts1( "SIM00000" );
```

接下去我们要声明另一个输入，指定该账户的损益警告线。

```
Input:PLAlert( 500 );
```

我们将声明一个变量，用于在绘图前保存计算得到的账户利润。

```
var:AccountPL( 0 );
```

接下去，点击 图标切换到属性编辑器的事件窗格。可以看到 Event 栏中列出的 Updated 值为空白。当您账户中的任何值变化时，Updated 事件将由 AccountsProvider1 对象触发，并会调用我们 EasyLanguage 文件中的一个事件处理器方法。只需双击事件名称 Updated 即可创建新的事件处理器方法名称。

名为 AccP_Updated 的事件处理器方法会自动添加到您的文件。

```
method void AccP_Updated( elsystem.Object sender, elsystem.TimerElapsedEventArgs arg
    s )
begin
    { Insert your EasyLanguage statements below }
end;
```

将 begin 语句后自动生成的注释替换为对名为 PlotValues() 方法的调用。

```
method void AccP_Updated( elsystem.Object sender, tsdata.trading.AccountUpdatedEve
    ntArgs args )
begin
    PlotValues();
end;
```

现在，创建一个方法，用来在每次账户更新事件发生时计算并绘制账户损益。这个语句只会在 Provider 从指定账户收集到数据时绘制，并只会绘制在图表的最后 K 线上。

```
Method void PlotValues()
begin
    If LastBarOnChart and AccP.Count > 0 then
begin
        AccountPL = AccP.Account[0].RTAccountNetWorth -
          AccP.Account[0].BDAccountNetWorth;
        plot1(AccountPL,"AccountPL");
```

下面的语句根据净值差来设定颜色，零为浅灰色，正负账户利润值则分别为绿色或红色。

```
setplotcolor(1,LightGray);
If AccountPL > 0 then
          setplotcolor(1,green);
    If AccountPL < 0 then
          setplotcolor(1,red);
    end;
end;

plot2(0,"Zero");
plot3(PLAlert);
plot4(-PLAlert);
```

在文件属性的 Chart Style（图表样式）选项卡将 AccountPL 图像修改为 Histogram（直方图）。

验证指标。

注意：计算期货或外汇交易账户损益的代码为：

```
{ AccountPL = AccP.Account[0].RTAccountEquity - AccP.Account[0].BDAccountEquity; }
```

当您将指标应用到图表时，应检查输入设置，并修改账号以及利润警告值（改为适合您图表交易代码的值）。例如，如果您要绘制某个股票交易代码，需要将 iAccounts1 输入修改为您的模拟股票账户，并根据您账户的当前净值修改利润警告值。

Filter 属性（TokenList）

很多情况下，属性编辑器的 Filter 属性包含的属性需要一个包含一个或多个值（以英文逗号隔开）组成的字符串，这个字符串也称为 Token List（符号列表）。

例如，PositionsProvider 组件包含三个 filter 属性，用于限定 Provider 组件的可能选择。假设您希望收集特定交易代码的仓位信息。我们用下面的代码设置 Symbol 属性收集您账户中 MSFT、AAPL 和 CSCO 的仓位。

```
Accounts
Symbols        "msft,aapl,csco"
Types
```

PositionsProvider 对象将会返回您账户中这三个指定交易代码的仓位，但不会包含其他交易代码。此外，留空的 Types 和 Accounts 属性表示这三个指定交易代码在任何账户中任何类型的仓位都要返回结果。但是，也完全可以轻松添加一组特定账户或仓位类型，结果就将会限制在仅包括符合全部筛选条件的这些条目。例如，下列筛选设置将只会找到两个账户中指定交易代码的多头持仓。

```
Accounts       "SIM12345,SIM67890"
Symbols        "msft,aapl,csco"
Types          "long"
```

除了输入指定代码的名称，也可以输入保留字 symbol（无引号），让代码筛选返回当前图表或表格内代码的仓位，如下所示：

```
Symbols        symbol
```

+= 加等号赋值运算符

与对象一起新加入 EasyLanguage 的还有一些面向对象编程语言常用的运算符。其中一个运算符可以在一个变量上加上一个表达式的值，而无需在赋值语句两侧重复变量名。也可以说，这个表达式“加给”结果。

例如，下列表达式：

```
result += 1
```

也可以写作

```
result = result + 1
```

差别只是 result 仅指定并求值一次。在后面，您将会在事件处理器的名称被赋值或加给一个事件属性时看到这个运算符。

PositionsProvider

PositionsProvider 组件用于创建对象来引用指定交易代码或交易代码列表的仓位值。您也可以筛选结果，只包含某些账户或仓位类型。PositionsProvider 对象会建立一个符合您标准（您指定的 Symbol、Account 和 Type 等筛选标准）的仓位集合，并将 Count 属性设置为该集合中的元素个数。如果不指定任何筛选标准，PositionsProvider 将构建一个包含所有交易代码 (Symbol)、账户 (Account) 和类型 (Type) 的仓位集合。

PositionsProvider 关联 Updated 事件可以让与任何被引用仓位的关联值改变时，代码就会得到通知。如果您想要在指标或策略中知道某个交易代码并不处于您的图标或表格图中的仓位的变化，这个搭配就特别有用。

在您的代码中，在尝试从仓位集合访问索引元素前，通常使用 Count 属性来确定是否有 PositionsProvider 找到的任何仓位。

例如下列代码显示 PositionsProvider 集合中首个仓位元素各种仓位属性的一部分：

```
Plot1(PosProvider1.Position[0].OpenPL, "P/L");
Plot2(PosProvider1.Position[0].Quantity, "PosSize");
Plot3(PosProvider1.Position[0].AveragePrice, "AvgPx");
Plot4(PosProvider1.Position[0].InitialMargin, "I Margin");
Plot5(PosProvider1.Position[0].RequiredMargin, "R Margin");
```

※ 教程示例 #7

目标：（仓位值指标）

用工具箱组件创建一个能读取仓位信息的对象。

用属性编辑初始化组件对象属性

绘制一个已绘制交易代码的仓位值的图形

指标：'$07_PosValue'

本指标使用 PositionsProvider 组件访问雷达屏或图表内某个交易代码的仓位信息。如果无仓位，则会显示 "0"。

	Symbol	Interval	Last	Low	High	Volume Today	$07_PosValue		
							Mkt Value	Qty	P/L
1	SPY	5 Min	132.55	132.42	133.15	38,818,835	13,255	100	$57.00
2	CSCO	5 Min	15.68	15.58	15.77	12,060,641	1,568	100	$31.00
3	IBM	5 Min	183.25	183.01	184.42	1,984,059	91,625	500	$4,160.00
4	USO	5 Min	38.40	38.30	38.52	1,213,427			
5	GLD	5 Min	154.80	154.12	154.81	5,450,070	15,480	100	$393.99
6	AAPL	5 Min	387.18	386.47	396.27	16,204,528	38,718	100	$3,291.60
7	GE	5 Min	18.62	18.60	18.82	13,450,374			

工作区：$07_PosValue

建立雷达屏窗口：

创建：任意周期

插入指标：$07_PosValue

组件和属性编辑器设置：

PosP:

- Name: *PosP*　（修改组件名）
- Symbol: *symbol*　（在 Filters 类别下）
- Updated:PosP_Updated

指标练习 #7:'$07_PosValue'

创建新指标并命名为：'#07_PosValue'。

点击代码编辑器左侧的工具箱选项卡，将 PositionsProvider 组件拖到代码编辑器中。默认名称 PositionsProvider1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 PositionsProvider1 是属性编辑器顶部的指定组件。

将组件的 Name 属性改成更短的名称 PosP。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样 Provider 将只返回当前交易代码的仓位。将 Accounts和 Type 属性留空，这样 Provider 将在所有相关账户（例如不同的融资账户或现金账户）中寻找当前交易代码的仓位。

现在，就可以用 PosP 组件作为对象在代码中获取表格或图表中当前交易代码的仓位值了。

接下去，点击  图标切换到属性编辑器的事件窗格.您将看到 Event 栏中的 Updated 为空值。双击事件名称 Updated，在您的 EasyLanguage 文件中创建一个事件处理器，并将新方法名与属性关联。

现在，名为 PosP_Updated 的方法已添加到文件。注意，事件处理器括号中的参数是自动生成的，请勿删除或自行修改。

```
method void PosP_Updated( elsystem.Object sender, tsdata.trading.PositionUpdatedEve
     ntArgs args )
begin
    PlotValues();
end;
```

在 PosP_Updated 方法的 begin 和 end 语句之间插入对 PlotValues() 的调用。

本指标不要求输入或变量。

接着，创建 PlotValues() 方法。

```
Method void PlotValues() Begin
```

begin 后的首个语句是一个 if 条件语句，读取组件的 Count 属性以确认是否有找到当前交易代码的任何仓位。如果不存在当前交易代码的任何仓位，表格窗口中该交易代码的值将以空白开始。如果存在当前交易代码的任何仓位，则指标将绘制三个仓位值。

```
if (PosP.Count > 0) then
    begin
        Plot1(PosP.Position[0].MarketValue, "Mkt Value");
        Plot2(PosP.Position[0].Quantity, "Qty");
        Plot3(PosP.Position[0].OpenPL, "P/L");
```

期初 P/L 曲线的颜色在正值时变为绿色，负值时变为红色。

```
if PosP.Position[0].OpenPL >= 0 then
      SetPlotColor(3,Green)
else
       SetPlotColor(3,Red);
end
else
    begin
```

如果不存在当前交易代码的任何仓位，则下列代码将显示总数 '0'。

```
        Plot1(0, "Mkt Value");
        Plot2(0, "Qty");
        Plot3(0, "P/L");
    end;
end
```

验证指标

本指标针对表格分析窗口设计，例如雷达屏，并会显示交易代码的任何仓位。但是，它在图表的子图中也会绘制仓位值。读取 Quantity（数量）、Market Value（市值）以及 P/L 值（依此顺序排列）时，指标状态线非常有用。

方法变量

在函数里声明变量，该变量只在该函数内可见/可用。与之类似，可以在一个方法内声明方法的本地变量，不能从指标的其他部分访问。这些本地变量在方法每次运行时创建，在方法中相应代码执行后消失。本地变量很有用，对事件处理器更新的值进行临时计算或像计数器以及索引这种仅在方法内需要的值都可以使用本地变量。在一个方法中，本地变量与指标变量可以混合使用，两个都可以包含在计算中。但是，在退出方法后只有标准的指标变量值会被保存。

您已经知道如何在指标或策略开头使用标准变量的声明语句，方法中的本地变量声明语句在这点上与标准变量声明语句很相似。但是，有两个重要的不同：1）本地变量语句必须在变量名前指定变量类型（int、double、bool 或 string）；2）不能在变量名称后为本地变量指定初始值。另外，由于本地变量不能有初始值，因此可以在使用本地变量进行计算前，在方法代码的 begin 和 end 之间给本地变量赋值。

```
method void myMethod()
var: int myIndex, double priceTotal;
begin
  {EasyLanguage statements and calculations}
end;
```

变量类型复习：

Int = Integer（整数型） - 没有小数部分的正数或负数
Double = Double Float（双精度浮点数） - 有小数部分的正数或负数
Bool = Boolean（布尔值）- 真或假的条件
String = Text（文本） - 字母数字序列

NumToStr(num, dec)

用保留字 NumToStr 可以将数值转化为绘图用的字符串。当希望在数字的文本版中控制小数点后的位数时，这很有用。

例如，如果希望绘制 Order 集合中第一个订单的 LimitPrice 属性并且只显示小数点后两位，可以使用 NumToStr 保留字，引用数字属性值作为第一个参数，指定小数点后位数作为第二个参数。

```
Plot1(numtostr(ordersprovider1.Order[0].LimitPrice,2),"Limit");
```

注意，在表格应用中将数值转换为字符串意味着该列所得值将不会按数值大小排序，并且小数点后位数的设置将不会被应用。

※ 教程示例 #8

目标：（仓位和账户值指标）

组合不同组件

使用工具箱组件，创建读取仓位和账户信息的对象

用属性编辑初始化组件对象属性

指标：'$08_PosValueAcct'

本指标使用 PositionsProvider 和 AccountProvider 来计算和显示某个股票仓位占总体账户净值的百分比。如果无仓位，则会显示 "0"。

	Symbol	Interval	Low	High	Volume Today	$08_PosValueAcct			
						Mkt Value	Qty	P/L	% of Acct
1	SPY	5 Min	132.42	133.15	39,078,751	13,262	100	63.90	4.336 %
2	CSCO	5 Min	15.58	15.77	12,172,557	1,568	100	31.15	0.513 %
3	IBM	5 Min	183.01	184.42	1,992,024	91,630	500	4,165.00	29.960 %
4	USO	5 Min	38.30	38.52	1,223,117				
5	GLD	5 Min	154.12	154.81	5,474,550	15,480	100	394.00	5.061 %
6	AAPL	5 Min	386.47	396.27	16,261,089	38,722	100	3,295.60	12.660 %
7	GE	5 Min	18.60	18.82	13,581,523				

工作区：$08_PosValueAcct

建立雷达屏窗口：

创建：任意周期

注意： 请列出希望监视仓位的交易代码

插入指标：$08_PosValueAcct

组件和属性编辑器设置：

PosP:

- Name: *PosP* （修改组件名）
- Symbol: *symbol* （在 Filters 类别下）
- Updated: PosP_Updated

AccP:

- Name: *AccP* （修改组件名称）

指标练习 #8:'$08_PosValueAcct'

在本例中，我们将从教程示例 #7 中创建的 #07_PosValue 指标开始。将之保存为 #08_PosValueAcct。

PosP 组件已经是前面指标的一部分，因此无需再次添加。但如果从头开始，则需要将工具箱中的 PositionsProvider 组件拖到文件内，并按照上方“组件和属性编辑器设置”用属性编辑器设置组件的 Symbol（交易代码）与 Updated 事件的值。

点击代码编辑器左侧的工具箱选项卡，将 AccountsProvider 组件拖到代码编辑器中。默认名称 AccountsProvider1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 AccountsProvider1 是属性编辑器顶部的指定组件。

将组件的 Name 属性改成更短的名称 AccP。

提醒一下，灰色的代码是从练习 #7 复制的。

```
method void PosP_Updated( elsystem.Object sender, tsdata.trading.PositionUpdatedEve
    ntArgs args )
begin
   PlotValues();
end;
```

要添加的第一行 EasyLanguage 代码是方法头部与 begin 语句之间的变量声明。您要声明两个本地变量，仅在 PlotValues 方法内部可见。记住，本地方法变量声明要求您指定一个类型，且不接受默认值。

第一个变量用于计算当前交易代码仓位所占账户净值的百分比，第二个用于保存仓位的 AccountID。

```
Method void PlotValues()

var: double PcntOfAccount, string PosAccountID;

begin
   if (PosP.Count > 0) then
      begin
         Plot1(PosP.Position[0].MarketValue, "Mkt Value");
         Plot2(PosP.Position[0].Quantity, "Qty");
         Plot3(PosP.Position[0].OpenPL, "P/L");

         if PosP.Position[0].OpenPL>=0 then
            SetPlotColor(3,Green)
         else
            SetPlotColor(3,Red);
```

下列代码首先检查 AccP 组件的 Count 属性来确认是否有任何可用账户,然后计算并以百分比显示总体账户仓位市值。注意我们是先从 PositionProvider 的当前仓位获取账号 (ID),然后使用账号引用实时净值数据。

```
            PosAccountID = PosP.Position[0].AccountID;
              If AccP.Count>0 then
              begin
                  Value1 = PosP.Position[0].MarketValue;
                  Value2 = AccP.Account[PosAccountID].RTAccountNetWorth;
                  PcntOfAccount = Value1/Value2;
                  Plot4(numtostr(PcntOfAccount*100,3)+" %","% of Acct");

              end;

        end

    else

          begin
              Plot1(0, "Mkt Value");
              Plot2(0, "Qty");
              Plot3(0, "P/L");
             Plot4("0","% of Acct");
          end;
    end;
```

验证指标。

本指标为表格分析窗口（如雷达屏）设计，将显示交易代码的任何仓位并基于市值显示每个仓位占账户的百分比。

ToString()

toString() 方法是将对象值单属性转换为字符串的方便方法。通常为了能让属性的数字或日期值使用画图或打印语句显示，而无需考虑属性类型时，就可以使用这个方法。

例如，下列绘图语句将生成运行时错误，因为 Plot1 并不知道如何显示所引用的 CurrentTime 属性（DateTime 对象类型）：

```
Plot1(elsystem.datetime.CurrentTime);
```

但只要添加 .toString() 到属性应用的末尾就会自动将 CurrentTime 属性转换为可显示的字符串，可以用 plot 或 print 语句绘制。

```
Plot1(elsystem.datetime.CurrentTime.toString());
```

OrdersProvider

OrdersProvider 让用户
可以指定筛选条件从一组规定的 TradeStation 账户访问实时和历史订单信息。这些信息与 TradeManager（交易管理器）内 Orders（订单）选项卡上的信息相似。Order 类描述一个指定订单可用的属性（见下）。

您也可以通过筛选使结果中只包含特定账户、日期范围、订单编号以及订单状态（如已接收、已执行、已拒绝等）。OrdersProvider 会建立一个符合您标准（您指定的 Symbol、Account From/To dates 和 Order States 等筛选标准）的订单对象集合，并将 Count 属性设置为该集合中的元素个数。 如果不指定任何筛选标准，OrdersProvider 将构建一个包含所有交易代码、账户、范围和状态订单的集合。

OrdersProvider 关联 Updated 事件可以让与任何被引用订单的关联值改变时，代码就会得到通知。例如，OrdersProvider 的特定实例管理的订单状态发生任何改变，都将触发一个通知事件，从而可以让您绘制改变的状态。如果您想要在指标或策略中了解某个交易代码并不处于您的图标或表格图中的订单的变化，这个搭配也很有用。

在尝试从订单集合访问索引元素前，先在代码中使用 Count 属性来确定是否有 OrdersProvider 找到的任何账户，这样做肯定没错。

例如，以下代码显示 OrdersProvider 集合中第一个元素的多种 Orders 属性：

```
Plot1(OrdersProvider1.Order[0].AvgFilledPrice, "AvgFillPx");
Plot2(OrdersProvider1.Order[0].FilledQuantity, "FillQTY");
Plot3(OrdersProvider1.Order[0].State.ToString(), "State");
Plot4(OrdersProvider1.Order[0].LimitPrice, "LimitPx");
Plot5(OrdersProvider1.Order[0].Duration, "Duration");
```

注意：订单集合按时间顺序排序，从 0（最新）到最早。

※ 教程示例 #9

目标：（订单状态指标）

用工具箱组件创建一个能读取订单状态信息的对象。

用属性编辑初始化组件对象属性

绘制某个绘制交易代码的订单数量、订单类型和订单状态

指标：'$09_OrderStatus'

本指标使用一个 OrdersProvider 组件访问以图表或雷达屏中交易代码提交的最后一个订单的信息。如果一个交易代码没有任何下单活动，雷达屏将显示空行。

	Symbol	Interval	Last	Low	High	Volume Today	$09_OrderStatus			
							Quantity	Type	Limit Price	State
1	MSFT	5 Min	26.03	0.00	0.00	33,537				
2	CSCO	5 Min	18.38	0.00	0.00	128,941	100	limit	18.45	received
3	IBM	5 Min	161.51	0.00	0.00	1,700				
4	USO	5 Min	42.69	0.00	0.00	887,937	500	limit	43.07	filled
5	GLD	5 Min	140.16	0.00	0.00	880,030				
6	EURUSD	5 Min	1.39961	1.39549	1.40356	0				
7	@QM(D)	5 Min	105.775	104.250	106.950	8,051				

工作区：$09_OrderStatus

建立雷达屏窗口：

创建：任意周期

插入指标：$09_OrderStatus

组件和属性编辑器设置：

OrdersProvider1:

Symbols: *symbol*　　（在 Filters 类别下）

Updated: OrdersProvider1_Update

指标练习 #9:'$09_OrderStatus'

创建新指标并命名为：'#OrderStatus'。

点击代码编辑器左侧的工具箱选项卡，将 OrdersProvider 组件拖到代码编辑器中。默认名称 OrdersProvider1 将出现在代码编辑窗口底部的组件盘。现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 OrdersProvider1 是属性编辑器顶部的指定组件。将组件的 Name 属性改成更短的名称 OrdP。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样 Provider 将只返回当前交易代码的订单状态。将 Accounts、Status 和 Orders 属性留空，这样 Provider 会在您的所有账户中寻找任何当前交易代码的任何订单。现在，就可以用 OrdP 组件作为对象在代码中获取表格或图表中当前交易代码的订单状态了。

然后点击 ⚡ 图标切换到属性编辑器的 Event（事件）窗格。可以看到 Event 栏中列出的 Updated 值为空白。双击事件名称 Updated，在您的 EasyLanguage 文件中创建一个事件处理器，并将新方法名与属性关联。

现在，名为 OrdP_Updated 的方法已添加到文件。前面提到过，事件处理器括号中的参数是自动生成的，请勿删除或自行修改。

插入一个对 PlotValues() 的方法调用。

```
method void OrdP_Updated( elsystem.Object sender, tsdata.trading.OrderUpdatedEventA
    rgs args )
begin
    PlotValues();
end;
```

现在，创建下列代码，其中包含一个绘制所选 Order[0] 订单状态属性的方法，这代表的是集合中第一个也是最近的订单。.Count 属性表示当前交易代码订单状态条目的实际数量，但在本例中，我们只关心第一个。另外，请注意 .tostring() 方法在几个图形属性名后的用法，这会使非文本值如果插入图表会在图标状态线上显示为字符串。

```
Method void PlotValues()
begin
    If OrdP.Count>0 then
    begin
        plot1(OrdP.Order[0].EnteredQuantity,"Quantity");
        plot2(OrdP.Order[0].type.tostring(),"Type");
        plot3(OrdP.Order[0].LimitPrice,"Limit Price");
        plot4(OrdP.Order[0].state.tostring(),"State");
    end;
end;
PlotValues();
```

验证指标。

本指标针对表格分析窗口设计，例如雷达屏，并会显示针对交易代码所下的任何订单。

LastBarOnChart

LastBarOnChart 函数用于确定当前评估的 K 线是图表中的最后一根 K 线。在编写需要创建和发送订单的条件时这很有用，这样可以按实时价格而非历史价格条件下订单。

```
If LastBarOnChart and OrderCondition=true then
   Orderticket1.send();
```

在本例中，OrderCondition 条件为真时只会在当前 K 线位置发送订单。

指标 - Initialized 与 Uninitialized 事件

当指标或策略应用到图表或表格应用时，可以自动触发 Initialized 事件，该事件将在第一次运行时调用您在 EasyLanguage 文件中指定的事件处理器方法。与之类似，在指标结束时可以触发 Uninitialized 事件，例如在从图表去除或平台关闭时。如果想在指标中设置一些当指标首次运行时在指标的其他部分之前执行一次的条件，或当指标关闭时作为最后一件事执行时，这会很有用。从属性编辑器顶部的下拉列表中选择 Analysis Technique 然后点击 ⚡ 图标切换到事件窗口，可以向任意指标添加 Initialized 事件或 Uninitialized 事件。在 Event 栏可以看到一对事件，Initialized 和 Uninitialized。双击任一名称将自动在您的 EasyLanguage 中创建名为 AnalysisTechnique_Initialized 或 AnalysisTechnique_UnInitialized 的事件处理器方法，并会在属性编辑器该方法的 Value 栏插入一个引用。只要将您的 EasyLanguage 语句加入到新的事件方法，它们就会在每次指标或策略被加载或卸载时执行一次。

IntrabarPersist

IntrabarPersist 是一个变量声明关键词，用于创建可以逐笔存储并更新值的 EasyLanguage 变量或数组。当想计算 K 线内交易笔数或保存可能在一根 K 线内改变的条件的状态时，这尤其有用。默认情况下，变量值在每根 K 线收盘时保存。

```
Var:IntrabarPersist tickcount(0);
tickcount = tickcount + 1;
```

在本例中，变量 tickcount 可以对每根 K 线上的全部交易笔数计数。

OrderTicket

OrderTicket 组件为指定交易代码生成订单票号（order ticket），并将订单直接从您的 EasyLanguage 指标或策略发送到市场上。OrderTicket 对象支持大多数订单参数，这些参数在从 TradeStation 的下单栏手动下单时可用。与手动下单一样，所有通过 OrderTicket 创建的订单的状态将会和其他普通订单一起显示在 TradeManager（交易管理器）的订单选项卡下。

设置一个 OrderTicket 组件的通常步骤与之前组件的步骤相同。首先用属性编辑器填入订单值。一些属性，例如 Order Type（订单类型）和 Duration（订单期限）被设为默认值。其他属性，例如 Symbol（交易代码）、Symbol Type（交易资产）、Account（账户）、Quantity（数量）和 Market Action（交易指令，比如买卖等）必须由用户输入才能创建有效订单。当全部属性设置完成后，在您的 EasyLanguage 代码中添加一个语句来用组件对象的 Send() 方法下单，如下所示：

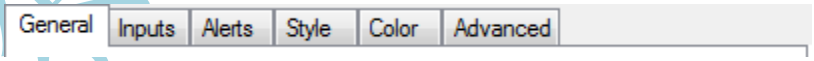
```
OrderTicket1.Send() ;
```

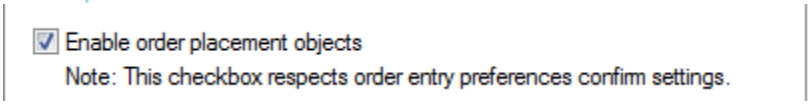
除了使用属性编辑器，如果想在发单前通过编程决定交易代码、交易数量、交易方向等，您也可以直接在 EasyLanguage 代码中设置 OrderTicket 的属性。无论用哪种方法，为指定组件调用 Send() 方法都将发送订单。

```
OrderTicket1.Symbol = "XYZ"
OrderTicket1.Quantity = 200;
OrderTicket1.Action = tsdata.trading.orderaction.Buy;
OrderTicket1.Send() ;
```

启用下单对象

为了避免从含有 OrderTicket 或其他订单组件的指标或策略中意外下单，必须专门给每个用来下单的指标或策略激活下单功能。在 EasyLanguage 可以产生订单前，通过进入每个指标或策略的 Format - General（设置-通用）选项卡并勾选 Enable order placement objects（启用下单对象）设置（选项卡中部）可以实现激活。





注意：如果没有勾选并且指标试图发出订单，则会出现运行时错误。

※ 教程示例 #10

目标：（市场订单指标）

创建一个订单票号

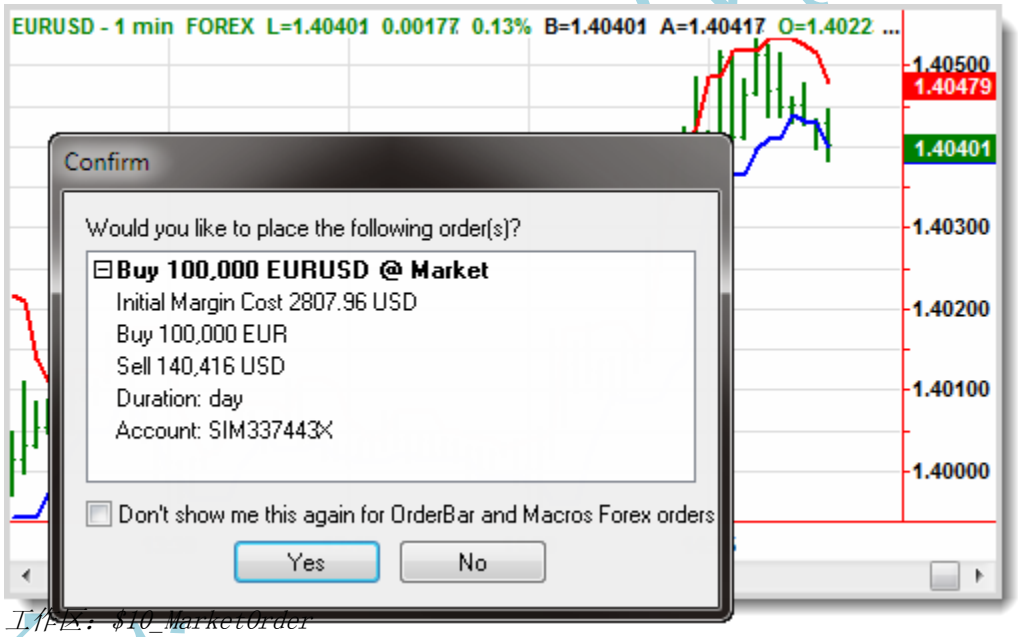
订单仅在最后一根 K 线发出一次

启用指标的下单对象设置

指标：'$10_MarketOrder'

本例使用一个 OrderTicket 组件根据简单的最低价或最高价的目标下单规则下一个 Buy（买盘）市价订单。一旦订单发出，订单有效状态标志就会被设置为 false。重新加载指标会重置指标，以发送另一个订单。

警告：本指标将产生一个市价订单——请确保未登录实盘账户。应该仅在登录到 TradeStation 模拟器时应用本指标。



工作区：$10_MarketOrder

构建图表：

创建：1 分钟图表
刻度值：与标的物数据同轴
插入指标：$10_MarketOrder

组件和属性编辑器设置：

OrderTicket1:

Symbo	*symbol*	（类型保留字）
Accoun	***iAccounts1***	（添加默认 Account 作为输入）
Quanti	***iQuantity1***	（添加所需 Quantity 作为输入）
Actio	*Buy*	
Type:	Market	

Analysis Technique:
Initialized: AnalysisTechnique_Initialized

指标练习 #10:'$10_MarketOrder'

创建新指标并命名为:'#10_MarketOrder'。

点击代码编辑器左侧的工具箱选项卡，将 OrderTicket 组件拖到代码编辑器中。默认名称 OrderTicket1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 OrderTicket 是属性编辑器顶部的指定组件。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样订单将针对当前交易代码。

与前面的示例中做过的一样，我们将把账号作为输入。在属性名称 Account 旁，输入您的模拟外汇账号。然后点击属性编辑器顶部的 图表将之设为输入。在属性编辑器中，Account 边上应该出现了 iAccount1 一词，并且您会看到在您的 EasyLanguage 代码顶部插入了下行。这个输入值将成为 OrderTicket1 组件的默认账户。在后面几个示例中，请再次确认示例中生成订单时您使用的是模拟账户。

```
Input: string iAccount1( "SIM00000" );
```

还是在属性编辑器中，给这个模拟外汇订单输入默认 Quantity（数量）100000。与之前一样，在属性编辑器顶部点击 图标来生成一个名为 iQuantity1 的输入，同时也把下行添加到代码中。

```
Input: int iQuantity1( 100000 );
```

在属性编辑器中，打开顶部的下拉列表，选择 Analysis Technique。点击 图标切换到事件窗格。可以看到 Event 栏中列出的 Initialized 值为空白。双击事件名称 Initialized，在您的 EasyLanguage 文件中创建一个事件处理器，并将新方法名与属性关联。稍后我们会在此方法中添加代码。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
    elsystem.InitializedEventArgs args )
begin
    { Insert your EasyLanguage statements below }
end;
```

现在，就可以用 OrderTicket1 组件作为对象在程序中发送表格或图表中当前交易代码的市价订单了。

在 EasyLanguage 代码中，让我们再加入一个输入，用于代表指标首次运行时订单是否有效。

```
Input:OrderActive(True);
```

接下去，我们将声明高价目标和低价目标的变量，用于在订单规则中生成订单。另外，我们将创建一个 intrabarpersist 变量，可以让我们控制下单规则的生效时间并只允许一个订单发送一次。

```
Vars: double HiTarget( 0 ), double LoTarget ( 0 ), intrabarpersist AllowTradeFlag
      ( True );
```

在 AnalysisTechnique_Initialized 方法中，我们首先用保留字 Category 将要下单的交易代码的资产类型（股票、期货、外汇）设置为当前交易代码的类型。然后我们将把 AllowTradeFlag（允许交易标志）变量的初始值设置为输入 OrderActive 的值，并在后面重置 AllowTradeFlag，从而让一个订单只能下单一次。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
     elsystem.InitializedEventArgs args )
begin
   OrderTicket1.SymbolType = Category;
   AllowTradeFlag = OrderActive;
end;
```

创建一个方法绘制指标值。注意此方法中的绘制语句仅在代码中稍后调用 PlotValues() 方法时执行。

```
Method void PlotValues()
begin
    Plot1(HiTarget,"High Target");
    Plot2(LoTarget,"Low Target");
    Plot3(AllowTradeFlag.tostring(),"Active");
end;
```

在本例中，高低价格目标基于过去三根 K 线周期内的最高或最低价格计算得出。这样我们的示例可以在应用指标后迅速得到订单条件。

```
HiTarget = HighestFC(High, 3)[1];
LoTarget = LowestFC(Low, 3)[1];
```

添加下面两个语句，当发生每笔交易时，在图表绘制值。

```
PlotValues();
```

在下一段代码中，如果当前价格达到高价或低价目标并且 AllowTradeFlag 变量的值在图表的最后 K 线上为 True 时，将执行订单条件。begin 后的语句用 OrderTicket1.send() 方法下单。下一语句将 AllowTradeFlag 变量重置为 False，从而在图表刷新前不会再下其他单。

```
If ( ( Close <= LoTarget OR Close >= HiTarget ) AND
     ( LastBarOnChart AND AllowTradeFlag ) ) then
  begin
     OrderTicket1.send();
     AllowTradeFlag = False;
  end;
```

验证指标。

打开图表并插入指标。

在图表上，打开设置#10_MarketOrder对话框，然后选择常规选项卡。选中 Enable order placement objects （启用下单对象）复选框后，从您的 EasyLanguage 代码才能生成订单。这个设置只有在指标应用到图表时保持其设置。如果删除指标后再次应用，则必须再次启用该复选框。

声明对象变量

到目前为止，我们用过的对象都是拖动一个组件到文件夹中自动生成的。如我们前面见到的，用于创建和设置组件对象的 EasyLanguage 代码已经在一个成为设计器生成代码的隐藏代码模块中加入到了您的指标中，您可以从代码编辑器的“**视图**”菜单查看这些代码。

例如，我们先看看使用 OrderTicket 组件的 $10_MarketOrder 示例中的设计器生成代码。这里是前面几行代码：

```
{ Components declaration (Designer generated code) }
var: tsdata.trading.OrderTicket OrderTicket1( NULL );
```

这里包含了一个变量的声明，该变量类型为 tsdata.trading.OrderTicket，名称为 OrderTicket1，默认值为 'NULL'。这个名称看起来很熟悉，因为这是我们在属性编辑器中建立的组件对象的名称，用于调用发送订单。每个对象变量，无论是设计器代码还是由您直接在指标代码中添加的，都看上去很相似，在对象类型后加上对象名称，初始值为 'NULL'。

在下个示例中，我们将使用一个 Order 对象跟踪一个订单的状态。下面是一个可用于追踪 Order（订单）的对象变量的变量声明：

```
Var: tsdata.trading.Order myOrder(null);
```

在本教程的后面，将会有其他对象变量的声明，可用于创建并引用您自己的非组件对象。

跟踪 Order Ticket 的订单状态

要跟踪一个订单的状态，保存发出订单实例到一个 Order 对象变量并创建一个事件处理器来自动通知指定订单的任何状态变化。下面说明了对于用 OrderTicket 发送的指定订单的状态，跟踪所需的步骤：

```
{ 声明一个名为 myOrder 的对象变量以保存一个 Order 对象的实例 }
Var: tsdata.trading.Order myOrder(null);

{ 用于处理 Order 更新事件的方法 }
Method void OrderStatusUpdate(elsystem.Object sender, tsdata.trading.
  OrderUpdatedEventArgs args)
begin
  Plot2(myOrder.State);
end;

{ 复制发送的订单到对象变量 myOrder 作为对象实例 }
myOrder = OrderTicket1.Send();

{ 将事件处理器方法与 myOrder 对象关联 }
myOrder.Updated += OrderStatusUpdate;
```

BracketOrderTicket

BracketOrderTicket 组件会产生一个包围 OCO 订单，由指定交易代码的两笔相同委托方向（买/卖）的订单组成，并且将组合 OCO 订单从 EasyLanguage 指标直接发单到市场。Order Ticket 对象支持大多数订单参数，这些参数在从 TradeStation 的下单栏使用 OCO 按钮手动下单时可用。

设置一个 BracketOrderTicket 组件的通常步骤与之前其他组件的步骤相同。首先用属性编辑器填入基本的订单值，例如 Symbol、Symbol Type（资产类型），Account、Quantity 和 Order Action（如买/卖等），这些都是创建有效订单必需的。另外，需要指定 TargetType 和 ProtectionType 的限价和止损价。

设置好所有属性后，在 EasyLanguage 代码中添加一个对 BracketOrderTicket 组件的引用，使用 Send() 方法下订单。

与手动下单一样，所有通过 BracketOrderTicket 创建的订单的状态将会和其他普通订单一起显示在 TradeManager（交易管理器）的订单选项卡下。

```
BracketOrderTicket1.Send() ;
```

除了使用属性编辑器，如果想在发单前通过编程决定交易代码、交易数量、订单操作、止损价/限价等，您也可以直接在 EasyLanguage 代码中设置 BracketOrderTicket 的属性。无论用哪种方法，为指定组件调用 Send() 方法都将发送订单。

```
BracketOrderTicket 1.Symbol = "XYZ"
BracketOrderTicket 1.Quantity = 100;
BracketOrderTicket1.LimitPrice = myOrder.AvgFilledPrice + .05);
BracketOrderTicket1.StopPrice = myOrder.AvgFilledPrice - .05;
BracketOrderTicket 1.Action = tsdata.trading.orderaction.Sell; B
racketOrderTicket 1.Send() ;
```

※ 教程示例 #11

目标：（OSO 包围订单指标）

创建一个 Bracket Order Ticket（包围订单票号）

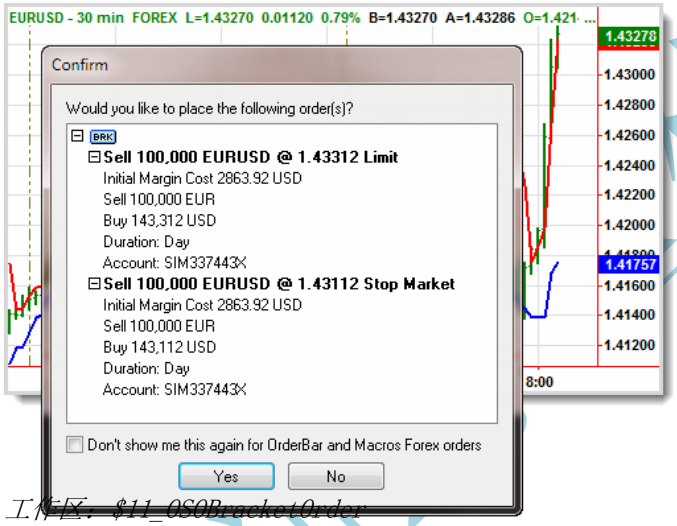
用 Order 对象关联并跟踪订单票号

创建一个事件处理器方法来跟踪 Order 状态

使用打印日志来跟踪市价订单的状态直到订单完成。

指标：'$11_OSOBracketOrder'

本指标使用一个 OrderTicket 和一个 BracketOrderTicket 组件根据指定的价格目标发送一笔市价买单，并在订单执行后发送一个 Sell（卖出）的包围 OCO 订单来设定利润目标和保护性的止损目标。订单状态显示在打印日志中。



工作区：$11_OSOBracketOrder

建立图表窗口：

创建：1 分钟周期

刻度值：与标的物数据同轴

插入指标：$11_OSOBracketOrder

组件和属性编辑器设置：

OrderTicket1:

Symbo	*symbol*	（在 Filters 类别下）
Accoun	*iAccounts1*	（作为输入添加）
Quanti	*iQuantity1*	（数量基于输入）
Actio	*Buy*	

BracketOrderTicket1:

Symbol:	*symbol*	（在 Filters 类别下）
Accounts:	*iAccounts1*	（作为输入添加）
Quantity:	*iQuantity1*	（数量基于输入）
Action:	*Sell*	
TargetTyp	*Limit*	

ProtectionType:*StopMarket*

Analysis Technique:

Initialized: AnalysisTechnique_Initialized

指标练习 #11:'$11_OSOBracketOrder'

在本例中，我们将从教程示例 #10 中创建的 #10_MarketOrder 指标开始。将之保存为 #11_OSOBracketOrder。

因为 OrderTicket1 组件已经是此前创建的指标的一部分，不用再次添加。但如果从头开始，则需要将工具箱中的 OrderTicket 组件拖到文件内，并按照上方“组件和属性编辑器设置”用属性编辑器设置组件的 Symbol、Symbol Type、Accounts、Quantity、Action 和 Type 值。

点击代码编辑器左侧的工具箱选项卡，将 BracketOrderTicket 组件拖到代码编辑器中。默认名称 BracketOrderTicket1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 BracketOrderTicket1 是属性编辑器顶部的指定组件。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样包围订单将针对当前交易代码。

点击 Account 属性旁的空文本框，然后点击右侧的下拉箭头并从下拉列表选择 iAccount1。这和我们在市价订单票号使用的是同一个模拟外汇账号。同样，对 Quantity 属性执行同样操作，点击空文本框右侧的箭头并选择 iQuantity。

还是在属性编辑器中，将 Action 属性设为 Sell。然后将 TargetType 设为 Limit，将 ProtectionType 设为 StopMarket，表明当我们下包围卖单时在市价上方和下方下单的订单类型。

回到我们的 EasyLanguage 代码，在之前声明过的输入后添加一个新的包围量输入，用于设置包围订单的上下限价。

```
Input: int iQuantity1( 100000 );
Input: string iAccount1( "SIM00000" );
Input: OrderActive(TRUE);
Input: BracketAmt(.001);
```

接下去，声明一个新的 intrabarpersist 变量，用于在包围订单发送后告诉我们，从而只生成一次包围订单。

```
Vars:    double HiTarget( 0 ), double LoTarget ( 0 ), intrabarpersist AllowTradeFlag
    ( True ) , intrabarpersist BracketSent( FALSE );
```

另外，声明一个订单对象变量，用于引用发送市价订单时创建的市价订单对象。

Var: tsdata.trading.Order MyOrder(null);在 AnalysisTechnique_Initialized 方法中，添加一个语句来将包围订单的交易代码类型设置为当前交易代码的资产类型（与前例市价订单操作相同）。记住，当您在指标的属性编辑器内双击事件名称 Initialized 时，会自动创建 AnalysisTechnique_Initialized 方法（请参阅例 10）。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
    elsystem.InitializedEventArgs args )
begin
   OrderTicket1.SymbolType = Category;
    BracketOrderTicket1.SymbolType = Category;
   AllowTradeFlag = OrderActive;
end;
```

现在，我们要创建一个新方法，让我们可以跟踪市价单和包围订单状态。在本示例的后面，我们将把这个方法关联到一个订单状态事件。在 begin 语句前声明一个名为 myStatus 的本地方法变量来保存状态消息字符串。

```
Method void OrderStatusUpdate(elsystem.Object sender, tsdata.trading.OrderUpdatedEve
    ntArgs args)
var: string myStatus;
begin
```

方法中接下去的语句将构建订单状态消息字符串并显示在打印日志中。

```
myStatus = myOrder.State.tostring();
myStatus = myStatus + "       " + myOrder.OrderID;
myStatus = myStatus + " - " + myOrder.Action.tostring();
myStatus = myStatus + "       " + myOrder.FilledQuantity.tostring();
myStatus = myStatus + "       " + symbol;
print(myStatus);
```

下列语句仅在市价订单完成后执行一次。包围价格设置在市价订单成交价的上下，然后包围订单发出。BracketSent 状态变量设为 True，这样包围订单就只会发送一次。最后，包围订单状态消息显示在打印日志中。

```
If BracketSent = False AND
 myOrder.State = tsdata.trading.orderstate.filled then
 begin
    BracketOrderTicket1.LimitPrice =myOrder.AvgFilledPrice + BracketAmt;
    BracketOrderTicket1.StopPrice = myOrder.AvgFilledPrice - BracketAmt;
    BracketOrderTicket1.Quantity = iQuantity1;
    BracketOrderTicket1.Send();
    BracketSent = True;
    print("Bracket order sent - ", BracketOrderTicket1.LimitPrice.tostring(),
        "  ", BracketOrderTicket1.StopPrice.tostring() );
end;
```

用一个 end 语句完成这个方法。

```
end;
```

这里的代码是前一示例中用来绘制指标值和发出最初市价订单的。

```
Method void PlotValues()
begin
   Plot1(HiTarget, "High Target");
   plot2(LoTarget,"Low Target");
   plot3(AllowTradeFlag.tostring(),"Active");
end;

HiTarget = HighestFC(High, 3)[1];
LoTarget = LowestFC(Low, 3)[1];

PlotValues();

If ((Close <= LoTarget OR Close >= HiTarget ) AND (LastBarOnChart AND AllowTradeFla
        g)) then
begin
```

但是，在本例中，我们将创建一个订单对象来跟踪订单的发出、接收和执行。我们将 OrderTicket1.Send() 方法返回的订单对象赋值给前面声明的 MyOrder 对象变量来实现这个目标。然后，我们将之前创建的 OrderStatusUpdate 事件处理器方法与 MyOrder 对象的 Updated 事件关联，这样每当订单状态改变时，我们可以打印订单状态并在市价单执行时发送包围订单。

```
   MyOrder = OrderTicket1.Send();
   MyOrder.Updated += OrderStatusUpdate;
   AllowTradeFlag = False;
end;
```

验证指标。

打开图表并插入指标。

打开设置#11_OSOBracketOrder 对话框，然后选择常规选项卡。选中 Enable order placement objects （启用下单对象）复选框后，从您的 EasyLanguage 代码才能生成订单。您可以一次性打开所有雷达屏行的这个功能，也可以对所选行分别启用。这个设置只有在指标应用到雷达屏窗口时保持其设置。如果删除指标后再次应用，则必须再次启用该复选框。

MarketDepthProvider

MarketDepthProvider 组件创建对象引用更新后的市场深度数据集合，其中包含指定交易代码的买入/卖出价格和买入/卖出价市场级别。这些信息与 TradeStation 的 MarketDepth（市场深度）窗口内的信息相似。

市场深度集合：

买价和卖价
买入价级别和卖出价级别
参与者（电子化交易平台[ECN]、证券公司）

Bids 和 Asks 集合属性可用于访问参与者（ECN）在每个价格级别上市的股数。BidLevel 和 AskLevel 集合属性可用于访问所有参与者在特定价格的综合数据。Participants 属性集合可用于访问特定参与者的全部数据。

General 属性可用于控制收集买卖报价的级别数以及是否在提供的数据里包含详细的 ECN 账面数据和二级市场数据。

MarketDepthProvider 可以搭配 Updated 事件使用，这样当与任何被引用市场深度数据关联的值改变时，代码就会得到通知。

在您的代码中，通常要在访问这些数据集合内的索引元素前，用 Count 属性来确定买卖报价项目的数量或买卖价格级别的深度。

MarketDepthProvider 将信息保存在五个数据集合中：

Bids & Asks（买入价/卖出价） - 按每行索引：

(marketdepthprovide1.Bids[row].Price)

Bid Levels & Ask Levels（买入价级别与卖出价级别） - 按价格级别索引：

(marketdepthprovide1.Asklevels[lvl].TotalSize)

Participants（参与者） - 按每个参与者索引：

(marketdepthprovide1.Participants[par].Bids.Count)

※ 教程示例 #12

目标：（市场深度指标）

用一个工具箱组件来创建一个对象，用于计算并显示一定买卖级别的总股数或总合约数。

按价格级别访问时长深度数据
循环总计不同级别的买单数量和卖单数量

指标：'$12_TotalBidAskSize'

这个雷达屏指标使用 MarketDepth 组件来显示指定市场深度价格级别的总买单数量和卖单数量。

	Symbol	Interval	Last	Low	High	Volume Today	$12_TotalBidAskSize	
							Bid Size	Ask Size
1	MSFT	5 Min	26.64	26.55	26.80	25,864,185	250,912	192,942
2	CSCO	5 Min	15.41	15.35	15.62	26,337,619	603,464	480,145
3	IBM	5 Min	174.77	174.61	176.15	2,561,994	2,000	1,619
4	USO	5 Min	37.18	36.94	37.63	6,258,507	194,089	225,658
5	GLD	5 Min	150.91	150.20	151.69	13,170,402	11,480	32,850
6	AAPL	5 Min	354.73	353.34	359.77	11,274,930	500	900
7	GE	5 Min	18.54	18.50	18.78	27,506,556	396,357	281,914

工作区：$12_MarketDepth

建立雷达屏窗口：

创建：5 分钟周期
插入指标：$12_TotalBidAskSize

组件和属性编辑器设置：

MDP:

- Name: *MDP*（修改组件名称）
- Symbol: *symbol*（设置为交易代码）
- MaximumLevelCount: ***iMaximumLevelCount1***（作为整型输入添加）
- Updated: MDP_Updated

指标练习 #12:'$12_TotalBidAskSize'

创建新指标并命名为:'#12_TotalBidAskSize'。

点击代码编辑器左侧的工具箱选项卡，将 OrdersProvider 组件拖到代码编辑器中。默认名称 MarketDepthProvide1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 MarketDepthProvide1 是属性编辑器顶部的指定组件。

将组件的 Name 属性改成更短的名称 MDP。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样 Provider 将只返回当前交易代码的市场深度信息。将 MaximumLevelCount 属性设置为名为 iMaximumLevelCount1 的输入。对应的输入声明会添加到 EasyLanguage 文件。

单击事件图标并双击 Updated 事件，在文件中创建事件方法。

现在，您的 EasyLanguage 文件中将有一个输入声明和一个事件处理器方法。将默认输入参数修改为 '3'。在处理器方法中添加对 PlotValues() 的调用。

```
Input: int iMaximumLevelCount1( 3 );

method void MDP_Updated( elsystem.Object sender,
  tsdata.marketdata.MarketDepthUpdatedEventArgs args )
begin
    PlotValues();
end;
```

现在，我们要创建 PlotValues 方法，其中将包含在 begin 语句前的三个本地变量。第一个变量充当统计总买单数量和卖单级别数量的循环计数器。两个本地变量会累计 MarketDepthProvider 组件请求的总买卖级别的市场深度（数量）。

```
Method void PlotValues()
var: int level, int AskDepthTot, int BidDepthTot;
begin
```

下面的语句将对组件对象的买卖价格级别的买卖单数量进行加总。首先，将买卖总数变量设为 0。然后，通过测试来确认 Provider 包含的买价级别是否足够达到在 iMaximumLevelCount1 输入中指定的最大级别数来计算出总数。如果足够，则通过循环把每个买价级别的数量加入总数。给卖价级别创建一个同样的 if 和 for 循环。

```
AskDepthTot = 0;
BidDepthTot = 0;
If MDP.bidlevels.count >= iMaximumLevelCount1 then
    For level = 0 to iMaximumLevelCount1  -  1
          begin
              BidDepthTot += MDP.bidlevels[level].totalsize;
          End;
If MDP.asklevels.count >= iMaximumLevelCount1 then
    For level = 0 to iMaximumLevelCount1 - 1
          begin
              AskDepthTot += MDP.asklevels[level].totalsize;
          end;
```

绘制总数，如果买单总数大于卖单总数，将买单数量图形的背景色设为绿色，如果卖单总数大于买单总数则将卖单数量图形的背景色设为红色。

```
 Plot1(BidDepthTot,"Bid Size");
 Plot2(AskDepthTot,"Ask Size");

 If (BidDepthTot > AskDepthTot) then
      begin
           setplotbgcolor(1,darkgreen);
           setplotbgcolor(2,black);
          end
      Else
      begin
          setplotbgcolor(2,darkred);
          setplotbgcolor(1,black);
      end;
  end;
```

验证指标。

打开雷达屏并插入指标。

QuotesProvider

QuotesProvider 组件创建的对象让您可以引用指定交易代码的报价字段（快照）值。报价字段值可以实时更新，但没有历史值可用。

在属性编辑器中，Symbol 属性必须指定想要报价的交易代码并且 Fields 属性必须包含一个由英文逗号分隔的一系列字段名供查询。两者都是必须填写的筛选属性。在提供的 QuotesProvider 对象中只提供必填报价字段（如 "AskSize, DailyLimit"）。

访问报价字段值要使用 QuotesProvider 的 Quote["Name"] 属性以及所请求字段的数据值类型（双精度值、字符串值、日期值等），例如：

```
plot1(QuotesProvider1.Quote["Ask"].doublevalue,"Ask");
```

报价字段价格通常作为 DoubleValue（双精度值），成交量、交易规模值通常作为 IntegerValue（整数值），日期和时间通常作为 DateValue（日期值），名称、描述通常为 StringValue（字符串值）。QuoteFields 类帮助主题列出了可用的报价字段名及其相关数据值类型。EL 字典特定字段名描述窗格的 Example 部分也列出了报价字段的值类型。

QuotesProvider 关联 Updated 事件可以让与任何被引用报价字段的关联值改变时，代码就会得到通知。如果您想要在指标或策略中使用某个交易代码并不处于您的图标或表格图中的报价，这个搭配就特别有用。

与其他 Provider 一样，在尝试从集合中访问索引元素前，请用 Count 属性来确定是否有 QuotesProvider 找到的任何报价。

※ 教程示例 #13

目标：（报价指标）

用 QuotesProvider 对象访问报价（快照）数据
建立一个报价字段的符号列表（Token List）供访问。
使用一个 'value' 属性读取并绘制报价字段的适当值类型

指标：'$13_Quotes'

本指标用 QuotesProvider 组件访问报价字段数据。

	Symbol	Interval	Last	Low	High	Volume Today	$13_Quotes		
							Last	Last Vol	Time
1	SPY	5 Min	132.57	132.42	133.15	39,397,680	132.57	100	10:25:12 AM
2	CSCO	5 Min	15.68	15.58	15.77	12,290,415	15.68	800	10:25:10 AM
3	IBM	5 Min	183.17	183.01	184.42	1,999,558	183.17	314	10:25:10 AM
4	USO	5 Min	38.42	38.30	38.52	1,225,917	38.42	100	10:25:10 AM
5	GLD	5 Min	154.83	154.12	154.84	5,517,164	154.83	900	10:25:12 AM
6	AAPL	5 Min	387.09	386.47	396.27	16,314,922	387.09	100	10:25:12 AM
7	GE	5 Min	18.61	18.60	18.82	13,648,549	18.61	300	10:25:12 AM

工作区：$13_Quotes

建立雷达屏窗口：

创建：5 分钟周期
插入指标：$13_Quotes

组件和属性编辑器设置：

QP:

Name:	*QP*	（修改组件名称）
Symbo	*symbol*	（在 Filters 类别下）
Field	*"last,tradevolume,tradeti*	
Update	QP_Updated	

指标练习 #13:'$13_Quotes'

创建新指标并命名为:'#13_Quotes'。

点击代码编辑器左侧的工具箱选项卡，将 QuotesProvider 组件拖到代码编辑器中。默认名称 QuotesProvider 将出现在代码编辑窗口底部的组件盘。现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 QuotesProvider 是属性编辑器顶部的指定组件。

将组件的 Name 属性改成更短的名称 QP。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样 Provider 将只返回图表或表格中当前交易代码的报价。将 Fields 属性设置为文本字符串 "last, tradevolume, tradetime"，用来指定要从交易代码获取的报价 (Quote) 字段。

在字典的 QuoteFields 类下以及 EasyLanguage 对象参考的 QuoteFields 帮助主题下提供了可用 Quote 字段名称和数据类型的列表。现在，就可以用 QP 组件作为对象在代码中获取表格或图表中当前交易代码的值了。

然后点击  图标切换到属性编辑器的 Event（事件）窗格。可以看到 Event 栏中列出的 Updated 值为空白。双击事件名称 Updated，在您的 EasyLanguage 文件中创建一个事件处理器，并将新方法名与属性关联。

现在，名为 QP_Updated 的方法已添加到文件。事件处理器括号中的参数是自动生成的，请勿删除或自行修改。

在事件处理器方法中，将 begin 语句后自动生成的注释替换为对名为 PlotValues() 方法的调用。

```
method void QA_Updated( elsystem.Object sender, tsdata.marketdat
  a.QuoteUpdatedEventArgs args ) begin PlotValues();
end;
```

现在，将下列语句添加到方法中，用于将绘制来自 Quote 集合的指定报价字段。要读取指定报价值，还需要为三个图形分别在 Quote["fieldname"] 后添加适当的属性类型（例如双精度值、整数值等）。请参考字典或 EasyLanguage 对象参考帮助中的字段名称与数据类型列表。例如，以 QuoteFields 类的 Last 字段名的字典解释为例，描述项底部的 Example 中可以看到一个代码片段，显示数据类型为 .doublevalue。

```
Method void PlotValues()
begin
    plot1(QP.Quote["Last"].doublevalue,"Last");
    plot2(QP.Quote["TradeVolume"].integervalue,"Volume");
    plot3(QP.Quote["TradeTime"].datevalue.tostring(),"Time");
end;
PlotValues();
```

验证指标。

打开雷达屏并插入指标。

FundamentalQuotesProvider

FundamentalQuotesProvider 组件创建的对象让您可以引用指定交易代码的基本面报价字段值。基本面报价字段通常不会实时更新，部分可能允许引用过往报表周期的值。

在属性编辑器中，Symbol 属性必须指定想要获取基本面报价的交易代码并且 Fields 属性必须包含一个由英文逗号分隔的一系列基本面字段名供查询。两者都是必须填写的筛选属性。在提供的 FundamentalQuotesProvider 对象中只提供必填字段（如 "sbbf,snpm,sgrp"）。

访问基本面报价字段值要使用 FundamentalQuotesProvider 的 Quote["Name"] 属性、所请求字段的数据值类型（双精度值、字符串值、日期值等）以及多少个报表周期之前（[0] 表示最近的周期），例如：

```
plot1(FundamentalQuotesProvider1.Quote["SBBF"].doublevalue[0], "EPS");
```

注意，对于基本面字段，值类型后方括号内指定几个周期前的索引值是必须的，即使特定字段并不报告历史值也是如此。

价格通常为 DoubleValue（双精度值），数量值（包括成交量和交易量）是 IntegerValue（整数值），日期和时间是 DateValue（日期值），名称/描述为 StringValue（字符串值）。Fundamental QuoteFields 类帮助主题列出了可用的基本面报价字段名及其相关数据值类型。EL 字典中特定基本面字段名描述窗格的 Example 部分也列出了报价字段的值类型。

Fundamental QuotesProvider 关联 Updated 事件可以让与任何被引用报价字段的关联值改变时，代码就会得到通知。如果您想要在指标或策略中使用某个交易代码并不处于您的图标或表格图中的报价，这个搭配就特别有用。

您也可以使用 Fundamental QuotesProvider 来访问大部分期货合同的交易商持仓报告。

与其他 Provider 一样，在尝试从集合中访问索引元素前，请用 Count 属性来确定是否有 FundamentalQuotesProvider 找到的任何报价。

※ 教程示例 #14

目标：（基本面报价指标）

使用工具箱组件创建一个对象来显示指定交易代码的基本面报价值。
访问历史基本面数据。
使用属性编辑器指定要查询的基本面字段。

指标：'$14_FundQuotes'

本指标用一个 FundamentalQuotesProvider 组件从几个 Fundamental Quote 字段获取数据，同时也计算并绘制每个代码的实时市盈率。

	Symbol	Interval	Last	$14_FundQuotes						Net Chg	Net %Chg
				EPS	Prev EPS	RT P/E	DivYield %	Px/Bk	UpDate		
1	CSCO	30 Min	14.10	0.33	0.28	42.86	1.71	1.64	6/8/2011	0.04	0.32%
2	INTC	30 Min	20.38	0.56	0.58	36.92	4.08	2.24	7/20/2011	-0.22	-1.07%
3	T	30 Min	28.60	0.61	0.58	47.67	5.96	1.50	7/21/2011	-0.25	-0.87%
4	TRV	30 Min	50.60	(0.87)	1.94		3.17	0.87	7/22/2011	-1.09	-2.11%
5	UTX	30 Min	70.38	1.48	1.13	48.49	2.68	2.86	7/26/2011	-1.19	-1.66%
6	DD	30 Min	46.30	1.31	1.54	35.72	3.51	3.69	7/29/2011	-0.41	-0.88%
7	HD	30 Min	29.54	0.51	0.36	59.37	3.32	2.65	8/2/2011	-0.61	-2.02%
8	JNJ	30 Min	61.50	1.00	1.27	62.32	3.67		8/2/2011	-0.70	-1.13%

工作区：$14_FundQuotes

建立雷达屏窗口：

创建：30 分钟周期
插入指标：$14_FundQuotes

组件和属性编辑器设置：

FQP:

- Name: *FQP*（修改组件名称）
- Symbol: *symbol*（在 Filters 类别下）
- Fields: *"sbbf,yield,price2bk,f_1stupdat"*
- Updated: FQP_Updated

#14:'$14_FundQuotes'

创建新指标并命名为：'#14_FundQuotes'。

点击代码编辑器左侧的工具箱选项卡，将 FundamentalQuotesProvider 组件拖到代码编辑器中。默认名称 FundamentalQuotesP1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 FundamentalQuotesP1 是属性编辑器顶部的指定组件。

将组件的 Name 属性改成更短的名称 FQP。

在 Filters 类别下，将 Symbol 属性设置为保留字 symbol，这样 Provider 将只返回图表或表格中当前交易代码的基本面报价。将 Fields 属性设置为文本字符串 "sbbf,yield,price2b

k,f_lstupdat"，用来指定要从交易代码获取的 Fundamental Quote（基本面报价）字段。在字典的 FundamentalQuoteFields 类下以及 EasyLanguage 对象参考的 FundamentalQuoteFields 帮助主题下提供了可用 Fundamental Quote 字段名称和数据类型的列表。

现在，就可以用 FQP 组件作为对象在代码中获取表格或图表中当前交易代码的值了。

然后点击  图标切换到属性编辑器的 Event（事件）窗格。可以看到 Event 栏中列出的 Updated 值为空白。双击事件名称 Updated，在您的 EasyLanguage 文件中创建一个事件处理器，并将新方法名与属性关联。

现在，名为 FQP_Updated 的方法已添加到文件。事件处理器括号中的参数是自动生成的，请勿删除或自行修改。

在事件处理器方法中，将 begin 语句后自动生成的注释替换为对名为 PlotValues() 方法的调用。

```
method void FQP_Updated( elsystem.Object sender, tsdata.marketdata.FundamentalQuo
          teUpdatedEventArgs args )
begin
    PlotValues();
end;
```

现在，将下列语句添加到方法中，用于将绘制来自 Quote 集合的指定基本面报价字段。这次我们将引用报价集合的每个元素，按其在字段筛选属性列出的顺序，其中 '0' 为第一个元素。如前例，需要为五个图形分别在 Quote[n] 添加适当的属性值。因为基本面报价可能包含前一报表周期的历史值，还需要使用方括号指定每个值属于'几个周期前'的周期。请参考字典或 FundamentalQuoteFields 对象参考帮助中的字段名称与数据类型列表。例如，在帮助主题中搜索 "SBBF"（基本每股盈利）显示数据类型为 **double**，意味着需要在 Quote 后添加属性 DoubleValue[0] 才能够读取最近的 "SBBF" 报价。如果没有指定 Quote 的数据类型或者“多少周期前”索引值，则很可能会出现运行时错误。

```
Method void PlotValues()
begin
   If FQP.Count > 0 then
   begin
      if FQP.HasQuoteData(0) then
         Plot1(FQP.Quote[0].DoubleValue[0],"EPS");
```

接下去我们将访问并绘制一个周期前的 EPS（每股盈利）。

```
      if FQP.HasQuoteData(0) then
         Plot2(FQP.Quote[0].DoubleValue[1],"Prev EPS");
```

然后，我们将绘制当前周期的其余字段。

```
      if FQP.HasQuoteData(1) then
         Plot4(FQP.Quote[1].DoubleValue[0],"DivYield %");
      if FQP.HasQuoteData(2) then
         Plot5(FQP.Quote[2].DoubleValue[0],"Px/Bk");
      if FQP.HasQuoteData(3) then
```

```
            Plot6(FQP.Quote[3].DateValue[0].tostring(),"UpDate");

        If Plot1 > Plot2 then
            SetPlotColor(1, Cyan)
        else
            SetPlotColor(1, Magenta);
    end;
end;
```

最后，我们将根据当前价格计算并绘制实时市盈率。然后，添加另一个 PlotValue 调用，从而以逐笔更新绘制出值。

```
if FQP.Count > 0 AND FQP.Quote[0].DoubleValue[0] > 0 then
    Plot3(Last / FQP.Quote[0].DoubleValue[0], "RT P/E" );

PlotValues():
```

验证指标。

打开雷达屏并插入指标。

Workbook (Excel)

Workbook 组件创建的对象用于在指标和 Microsoft Excel® 电子表格中的外部工作簿之间建立联系。

在属性编辑器中，FileName 筛选属性必须指定一个在电脑上存在的 Excel 电子表格文件的完整路径和文件名。其他筛选属性用来决定建立联系的其他设置，例如是否可与其他指标共享电子表格或完成后是否将数据保存到电子表格。

Workbook 组件用于指定电子表格文件内的特定工作表，然后使用所引用工作表的 Cells 属性来从指定的单元格行列读写数据。

例如，下列语句向 FileName 属性引用的电子表格中名为 "Prices" 的选项卡（工作表）的第 2 列第 4 行的单元格内写入数字 123.45：

```
Workbook1["Prices"].Cells[2,4] = 123.45;
```

类似的，下面的语句会从同一个 "Prices" 选项卡（工作表）的第 3 列第 4 行读取一个数值并赋值给 Value1：

```
Value1 = Workbook1["Prices"].CellsAsDouble[3,4];
```

我们用 CellAsDouble 属性从单元格读取,以确保读取单元格中的任何数值（或空值）。但是，如果单元格内包含一条文本信息，语句将会出错，因为单元格的数据类型不匹配。

注意，要使用 Workbook 组件，电脑上必须装有 Microsoft Excel 软件并且在 FileName 属性引用的文件夹必须已创建好电子表格文件。

写入值到 Excel

```
wkbk["sheet name"].Cells[Col, Row] = 25.50;
wkbk["AcctPL"].Cells[5,10] = 25.50;
```

从 Excel 读取值

```
Value1 = wkbk["sheet name"].CellsAsDouble[Col, Row] ;
Value1 = wkbk["AcctPL"].CellsAsDouble[5,10] ;
```

保存电子表格数据

如果用 EasyLanguage 修改了 Excel 工作簿且希望在关闭指标后保存修改，可以设置对象属性 'SaveOnClose=TRUE'。

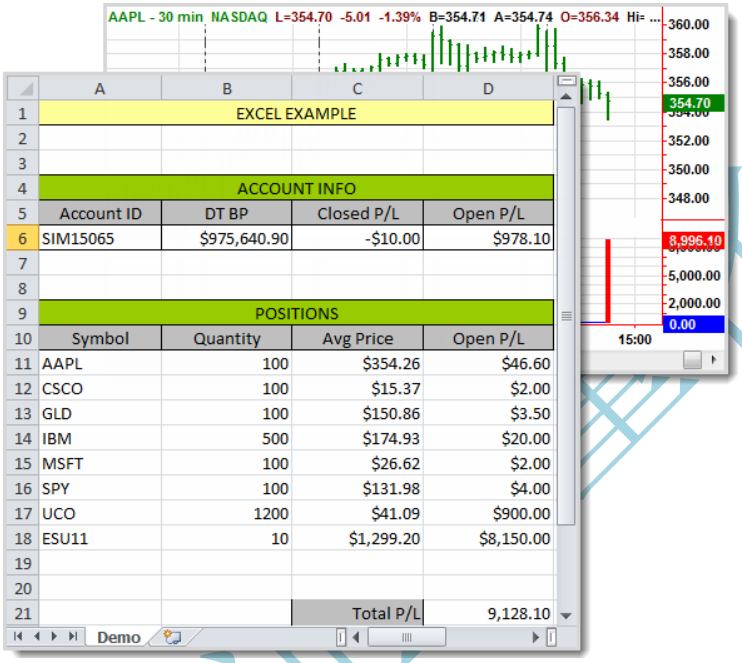
※ 教程示例 #15

目标：（Excel 指标）

引用一个预先建立的 Excel 工作簿
将更新后的值写入 Excel 电子表格
从电子表格读取计算值

指标：'$15_Excel'

本指标用一个 Workbook 组件来与雷达屏或图形分析窗口内的指定 Excel电子表格通信。注意：本指标仅在电脑上装有 Microsoft Excel 时可用。



Workspace:$15_Excel

构建图表：

创建：30 分钟周期
插入指标：$15_Excel

组件和属性编辑器设置：

WkBk:

Name:	*WkBk*	（修改组件名称）
FileName:	*"c:\exceldemo.xls"*	（位于类别 Filters 下）

AccP:

Name:	*AccP*	（修改组件名称）
Accounts:	***iAccount1***	
Updated:	AccP_Updated	

PosP:

Name:	*PosP*	（修改组件名称）
Symbols:	*symbol*	
Updated:	PosP_Updated	

指标练习 #15:'$15_Excel'

创建新指标并命名为:'#15_Excel'。

点击代码编辑器左侧的工具箱选项卡，将下列三个组件拖到代码编辑器中。Workbook、AccountsProvider 和 PositionsProvider。默认名称 Workbook1、AccountsProvider1 和 PositionsProvider1 将出现在代码编辑窗口底部的组件盘。

现在，点击代码编辑器右侧的属性选项卡打开属性面板，确保 Workbook1 是属性编辑器顶部的指定组件。将组件的 Name 属性改成更短的名称 WkBk。在 Filters 类别下，设置 FileName 属性为 c:\exceldemo.xls 从而让 Workbook 组件知道要与哪个 Excel 文件通信。现在，就可以用 WkBk 组件作为对象在代码中获取表格或图表中当前交易代码的值了。

在属性编辑器顶部选择 AccountsProvider1。将组件的 Name 属性改成更短的名称 AccP。切换到属性编辑器的事件图标并双击 Updated，在文件中创建一个 AccP 事件处理器方法。这个方法将在每次 Account 值发生变化时被调用。

选择属性编辑器顶部的 PositionsProvider1。将组件的 Name 属性改成更短的名称 PosP。切换到属性编辑器的事件图标并双击 Updated，在文件中创建一个 PosP 事件处理器方法。这个方法将在每次仓位发生变化时被调用。

在文件的顶部，创建两个变量。第一个指定您将在电子表格中引用的选项卡的名称。第二个将保存在电子表格中计算出并由您的 EasyLanguage 代码读取的总仓位损益值。

```
Var:WBTab( "Demo" ), TotalPL( 0 );
```

在 AccP 事件处理器方法中我们将添加一组语句将来自 Provider 集合的第一个账户的数个属性写入到电子表格前四列的第 6 行。与前面一样，我们首先测试确认 AccP 组件是否包含任何账户数据。注意每个 WkBk 引用都包含待写入单元格的电子表格工作表名称（来自 WBTab 变量）。

```
method void AccP_Updated( elsystem.Object sender, tsdata.trading.AccountUpdatedEven
    tArgs args )
begin
    if (AccP.Count > 0) then begin
      WkBk[WBTab].Cells[1, 6] = AccP[0].AccountID;
      WkBk[WBTab].Cells[2, 6] = AccP[0].RTDayTradingBuyingPower;
      WkBk[WBTab].Cells[3, 6] = AccP[0].RTRealizedPL;
      WkBk[WBTab].Cells[4, 6] = AccP[0].RTUnrealizedPL;
    end;
end;
```

在 PosP 事件处理器方法中我们将添加一组语句将来自 Provider 集合的每个仓位的数个属性写入到电子表格前四列的第 11 到第 20 行。

```
method void PosP_Updated( elsystem.Object sender, tsdata.tra
     ding.PositionUpdatedEventArgs args )
var: int it;
begin
     for it = 0 to PosP.Count - 1
     begin
       WkBk[WBTab].Cells[1, 11 + it] = PosP[it].Symbol;
       WkBk[WBTab].Cells[2, 11 + it] = PosP[it].Quantity;
       WkBk[WBTab].Cells[3, 11 + it] = PosP[it].AveragePrice;
       WkBk[WBTab].Cells[4, 11 + it] = PosP[it].OpenPL;
end;
```

示例假定我们的仓位数不超过 10 个。在将现有仓位数据写出后，添加一行空单元格，以确保在从集合移除一个仓位后不会显示任何前面的行。

```
     WkBk[WBTab].Cells[1, 11 + it ] = "";
     WkBk[WBTab].Cells[2, 11 + it ] = "";
     WkBk[WBTab].Cells[3, 11 + it ] = "";
     WkBk[WBTab].Cells[4, 11 + it ] = "";
```

第 4 列第 21 行的电子表格单元格包含一个公式，将第 4 列第 11 - 20 行加总。这个计算值会从电子表格读取并保存到变量 TotalPL，并实时绘制。

```
TotalPL = WkBk[WBTab].CellsAsDouble[4, 21];
If LastBarOnChart then
        plot1(TotalPL, "TotalP&L");
end;
```

最后，还要显示一个零线，作为正负损益值的基准。

```
Plot2(0);
```

末尾的附加代码让 Plot1 在收市时显示。

```
If LastBarOnChart then
begin
   TotalPL = WkBk[WBTab].CellsAsDouble[4, 21];
   plot1(TotalPL, "TotalP&L");
end;
```

验证指标。

打开图表并插入指标。

创建非组件对象

并非所有的新对象都是用工具箱组件创建的。您可以通过声明对象变量并将对象实例赋值给变量来向代码中添加对象。对象变量和新对象实例的类型（类）必须一致。创建对象的新实例并赋值给对象变量的过程也被称为实例化（instantiation）。

虽然不是必需，不过当 EasyLanguage 代码通过使用一个由指标的 Initialize 事件触发的 AnalysisTechnique_Initialized 方法（在属性编辑器内设置）第一次运行时，一般都会创建新的对象实例。

非组件对象的例子包括：集合、Windows Form 和 XML 数据库。

New

保留字 **new** 用于在赋值语句中创建指定类（类型）对象的实例（或副本）并调用该类的构造器。new 会出现在等号 (=) 右边，在类类型 (class type) 之前。新创建的对象赋值给之前声明的相同类类型的对象变量。

例如，下面的代码为名为 vectorname 的 vector 对象声明一个变量，并将一个 Vector 类型对象的新实例赋值给该变量：

```
var: elsystem.collections.Vector vectorname(null);
vectorname = New elsystem.collections.Vector;
```

注意在变量声明内使用的 Vector 类类型和新对象实例的创建包含对字典中类所在位置完整命名空间的引用。

Create()

Create() 方法也可以用于创建指定类（类型）的对象实例并调用该类的构造器。Create() 方法是保留字 new 的替代选项，当新对象要求作为 Create() 方法的一部分传递初始化参数时，经常会使用该方法。

例如，下面的代码为名为 gdname 的全局字典对象声明一个变量，并将一个 GlobalDictionary 类型对象的新实例赋值给该变量：注意这个 Create() 方法接受附加参数，在本例中，这些附加参数指定 GlobalDictionary 可以在各种分析窗口类型之间共享并且有一个独特的共享名。

```
var: elsystem.collections.GlobalDictionary gdname(null);
gdname = elsystem.collections.GlobalDictionary.Create(true,"IDstring")
```

Vector *集合*

Vector 用于将一组值保存为一个命名集合的索引元素。这与 EasyLanguage 数组类似，但是 Vector 更强大，例如各种将值插入集合的方法以及在集合内储存对象等元素的能力。

我们到目前为止接触的大部分对象都是从工具箱拖到文件中创建的。要创建一个 vector 或任何非组件对象，首先需要声明一个合适类型的变量并将同类型对象的新实例赋值给该变量。

例如，这里有一个对名为 myValues 变量的声明语句，创建为 elsystem.collection.Vector 类，初始值为 null。注意对象变量总是以初始值 null 声明。

```
var: elsystem.collections.Vector myValues(null);
```

下一步是将新 Vector 对象的实例（副本）赋值给变量。这可以在代码中任何地方进行，但通常会在代码首次运行时进行，如下所示：

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
  elsystem.InitializedEventArgs args )
begin
   myValues = New elsystem.collections.Vector;
end;
```

创建后，Vector 的长度可以随着您在任何位置添加或删减数据而相应增长或缩短。Vector 可以搭配一些核心 EL 函数来取集合内数值的平均数或总和。Vector 集合内的第一个元素的索引值为 0。

向 Vector 内添加数据元素：

Push_Back - 向 Vector 集合末端添加一个元素。

```
vectorname.Push_Back( 25.50);
```

Insert - 在 Vector 集合的指定位置添加元素，将其他元素的索引值向后推一位。

```
vectorname.Insert(Index, 25.50);
```

从 Vector 中移除数据元素：

Pop_Back - 移除 Vector 集合末端的元素。

```
vectorname.Pop_Back();
```

Erase - 移除 Vector 集合指定位置的元素，其他元素的索引值向前一位。

```
vectorname.Erase(Index);
vectorname.Erase(iStart, iEnd);
```

Clear - 删除 Vector 中的全部数据元素：

```
vectorname.Clear();
```

从 Vector 访问数据元素：

```
Value1 = vectorname[index];
```

在函数中使用 Vectors：

```
Value1 = Highest(vectorname, vectorname.count);
```

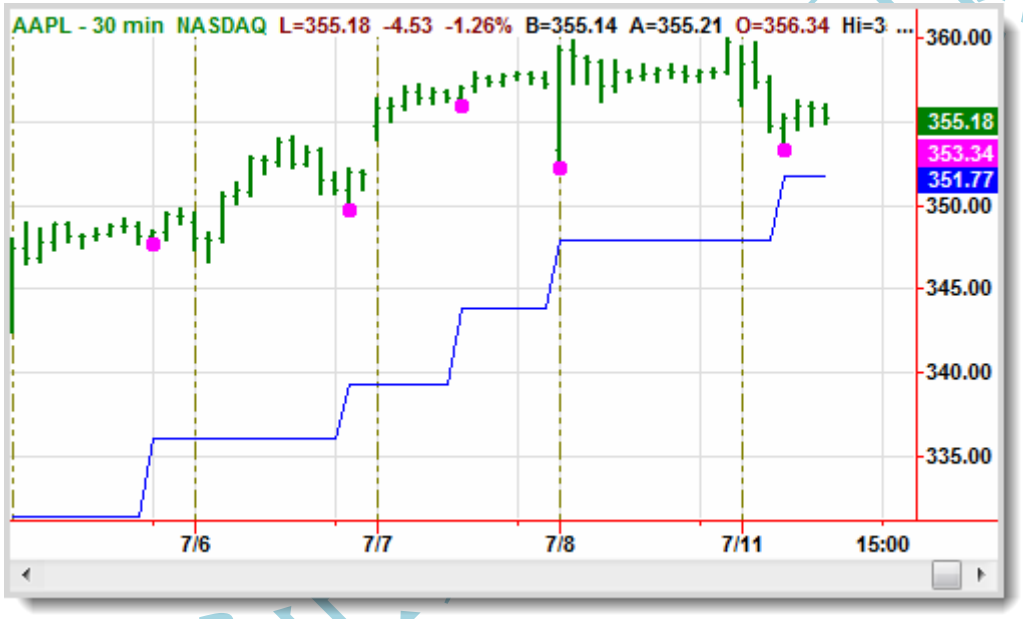
※ 教程示例 #16

目标：（Vector 指标）

直接在代码中创建一个新 Vector 对象实例
向 vector 中添加值
访问 vector 中的值
在标准的 EL 函数中使用 Vector 集合。

指标：'$16_Vector'

本指标用一个 Vector 集合为图表中指定数量的关键反转形态存储 K 线最低价格，然后将最低价平均值作为结果绘制在图表上。



工作区：$16_Vector

建立图表窗口：

创建：30 分钟周期
插入指标：$16_Vector

组件和属性编辑器设置：

无组件

Analysis Technique:
Initialized: AnalysisTechnique_Initialized

指标属性

刻度值： 与标的物数据同轴
图表样式： 第一个图形设置为 *Point* 类型

指标练习 #16:'$16_Vector'

创建新指标并命名为:'#16_Vector'。

到目前为止，我们在示例中都是用组件创建对象的。在本例中，我们将用非组件对象 Vector，它和 EasyLanguage 的数组很相似，与数组不同的是，Vector 允许向集合中任何元素添加数据且更容易管理其长度。

首先，声明一个在 Vector 中将用到的输入，用于定义元素最大数量。

```
input:VectorMax(5);
```

然后，我们要创建对象变量，用于保存 Vector 对象。对象变量用对象的类类型进行声明，在本例中用完整的类名称 elsystem.collections.Vector 来指定对象类型。对象变量的名称是 AvgLow，所有对象变量声明时的初始值均为 null。

```
var: elsystem.collections.Vector AvgLow(null);
```

第二个变量是一个用来识别是否发生关键反转条件的真假值。

```
var:KR(False);
```

接下来，在属性编辑器内点击事件图标，双击 Initialized 创建一个仅在指标第一次执行时执行一次的事件处理器方法。

在 AnalysisTechnique_Initialized 方法内将添加一个语句来创建一个 Vector 对象的 New 实例（同样使用完整类类型名）并将它赋值给对象变量 AvgLow。如果看看前面示例中组件创建对象的设计器生成代码，您会发现用过的每个组件对象都有类似的赋值语句。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
  elsystem.InitializedEventArgs args )
begin
    AvgLow = New elsystem.collections.Vector ;
end;
```

每当找到一个关键反转形态，即当前 K 线的最低价低于此前三根 K 线并且当前 K 线的收盘价高于前一根 K 线的收盘价，则下面的语句会把 KR 设为 True。如果未找到形态，则 KR 设置为 False。

```
KR = ( Low < Lowest( Low, 3 )[1] and Close > Close[1] );
```

如果找到关键反转形态，则用 Insert() 方法向 Vector 对象 AvgLow 加入新值。这种情况下，当前 K 线的最低价会作为 Vector 的第 0 个元素添加。需要特别注意的是，最新的值总是作为 Vector 的第 0 个元素添加，此前的值自动后移一位。一旦 Vector 内的项目数超过指定的最大值，则会使用 Pop_Back() 方法移除最后一项。这样，集合内的项数总是小于等于 VectorMax。Vector 被更新后，会绘制当前 K 线的最低价。需要右键单击文档，进入 Chart Style 选项卡，将"KR BAR"的 Type 值改为更大的点，这样在找到关键反转处就会出现一个粗点。

```
    if KR then
```

```
begin
    AvgLow.insert(0,Low);

    If AvgLow.Count > VectorMax then
        AvgLow.pop_back();

    Plot1(Low, "KR BAR");
end;
```

最后，如果 Vector 内有任何项，我们将把此前 Vector 内的最低价平均值绘制为一条线，这条线代表根据当前关键反转样式所下订单的建议止损出逃价格。

```
If AvgLow.Count = VectorMax then
    plot2(Average(AvgLow,VectorMax),"KR Avg Low");
```

右键单击文件，将 KR BAR 的绘图样式改为 point。
验证指标。
打开图表并插入指标。

全局字典集合

全局字典（Global Dictionary）用于保存不同窗口间指标的共享值（或对象）。GlobalDictionary 的值通过键值对（key/value）添加，其中键（key）是字典中的条目名称（例如 "mySymbol"），值（value）则是任何可以保存并能用键名重新获取的数值、字符串、布尔值或对象。

我们到目前为止接触的大部分对象都是将组件从工具箱拖到文件中创建的。要创建一个非组件对象，首先需要声明一个合适类型的变量并将同类型对象的新实例赋值给该变量。

例如，这里有一个myGD 变量 的声明语句，该变量为 elsystem.collection.GlobalDictionary 类，初始值为 null。

```
var: elsystem.collections.GlobalDictionary myGD(null);
```

接下来，要创建一个 GlobalDictionary 对象的新实例并赋值给 myGD 变量。

```
myGD = elsystem.collections.GlobalDictionary.Create();
```

不带任何参数使用 Create() 方法将创建一个在相同窗口类型（图形分析、雷达屏等）内运行的指标共享的默认（未命名）GlobalDictionary。每个读写该默认全局字典的指标都需要一个以同样的空 Create() 方法创建的 GlobalDictionary 实例赋值的对象变量。

或者，可以使用两个参数的方法 Create(shareType,shareName) 来创建一个在不同窗口类型间（例如一个图形分析和一个雷达屏）共享的 GlobalDictionary 对象并且用特定名称来避免使用默认字典可能发生的冲突。每个读写特定全局字典的指标都需要一个用 Create(shared,name) 方法创建的、具有相同真假值和共享名参数的 GlobalDictionary 对象实例。下列语句创建一个用共享名在不同窗口类型间工作的 GlobalDictionary 实例。

```
myGD = elsystem.collections.GlobalDictionary.Create(TRUE, "GD_sharedname");
```

如果键尚不存在，则向 GlobalDictionary 添加一个值。

```
If myGD.Contains("keyname") = false then myGD.Add("keyname", initialvalue);
```

键添加到字典后，与键相关联的值可以用 Items["keyname"] 属性改变或读取。注意，当读取字典值的时候 Items[] 值需要用正确的数据类型赋值给一个变量。

修改与现有 GlobalDictionary 键关联的值

```
If myGD.Contains("keyname") then myGD.Items["keyname"] = newvalue;
```

读取与现有 GlobalDictionary 键关联的值

```
If myGD.Contains("keyname") then
    value1 = myGD.Items["keyname"] astype type;
```

※ 教程示例 #17

目标：(GDWrite)

创建一个 GlobalDictionary 对象，在任何类型的分析窗口间共享值
给 GlobalDictionary 一个特定共享名
向全局字典中写入值

指标：'$17_GDWrite'

本例用一个 GlobalDictionary 对象往全局字典中写入信息，这些信息可被使用 GlobalDictionary 的其他指标访问。

工作区：$17_GDWrite

建立图表窗口：

创建：日线图
插入指标：$17_GDWrite

组件和属性编辑器设置：

无组件

```
Analysis Technique:
    Initialized:    AnalysisTechnique_Initialized
```

指标练习 #17:'$17_GDWrite'

创建新指标并命名为：'#17_GDWrite'。

在本例中，我们将使用一个名为 GlobalDictionary 集合的非组件对象并向全局字典内写入一对值，这对值可以被其他从相同全局字典读取值的指标读取。

首先，声明一个输入来定义回看时长，这个时长在计算基准代码收盘价的百分比变动时会用到。

```
Input: int LookBack( 20 );
```

然后，声明一个对象变量来引用全局字典实例，另一个变量表示基准代码表现的计算值。

```
Var: elsystem.collections.GlobalDictionary GD(null), double BenchPerf( 0 );
```

接下来，在属性编辑器里选择 Analysis Technique，点击事件图标，双击 Initialized 创建一个仅在指标第一次运行时执行一次的事件处理器方法。

在 AnalysisTechnique_Initialized 方法中添加一个语句，该语句包含一个双参数 GlobalDictionary Create 方法并且将 "myAll" 新实例赋值给对象变量 GD。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
    elsystem.InitializedEventArgs args )
begin
    GD = elsystem.collections.GlobalDictionary.Create(true,"myAll");
```

接下来的一系列语句检查字典中是否已经包含条目名 "TopSymbol"（如果另有一个 GDWrite 指标在别处运行时就有可能）。如果该条目不存在，则将其与相关 "TopValue" 条目添加进字典。记住，因为这些 EasyLanguage 语句在 AnalysisTechnique_Initialized 方法内部，因此它们只在指标加载时执行。

```
If GD.Contains("TopSymbol") = false then begin GD.Add("TopSymbol","");
    GD.Add("TopValue",BenchPerf);
end;
end;
```

一旦图表加载并到达最后一根 K 线，则会执行下面的语句。它们会根据回看时长内收盘价的百分比变化计算基准交易代码的表现，将基准表现值和交易代码名赋值给之前创建的 GlobalDictionary 项目，并绘制字典中刚刚保存的基准值。第二个图像显示一条零基准线。因为该代码是指标主体的一部分，表现值在每笔交易发生时都会计算并写入全局字典。

```
If LastBarOnChart then begin
    BenchPerf = PercentChange(Close, LookBack) * 100;
    GD.Items["TopValue"] = BenchPerf;
    GD.Items["TopSymbol"] = symbol;
    plot1(GD.Items["TopValue"] astype double,"Bench Perf");
end;

Plot2(0);
```

验证指标。

打开 30 分钟图表并插入指标。

※ 教程示例 #18

目标：（GDRead 指标）

从另一个指标引用 GlobalDictionary 对象

从指定全局字典中读取更新后的值

指标：'$18_GDRead'

本指标使用一个 GlobalDictionary 对象，每当全局字典的信息改变时，从全局字典将信息读取到一个指标。

	Symbol	Interval	Last	Net Chg	Net %Chg	$18_GDRead			Bid
						Bench Perf	SymRow Perf	Rel Perf	
1	SPY	Daily	132.06	-2.34	-1.74%	3.50	3.50	-0.01	132.06
2	AAPL	Daily	354.74	-4.97	-1.38%	3.50	8.85	5.35	354.71
3	IBM	Daily	174.90	-1.59	-0.90%	3.50	7.18	3.68	174.88
4	CSCO	Daily	15.39	-0.35	-2.22%	3.50	1.79	-1.72	15.38
5	MSFT	Daily	26.62	-0.30	-1.11%	3.50	12.32	8.82	26.62
6	EURUSD	Daily	1.40305	-0.02285	-1.60%	3.50314	-2.64507	-6.14821	1.40305
7	@QM(D)	Daily	94.625	-1.575	-1.64%	3.495	-5.090	-8.586	94.600

工作区：$18_GDRead

建立雷达屏窗口：

创建：每个交易代码的日周期图

插入指标：$18_GDRead

组件和属性编辑器设置：

无组件

Analysis Technique:

Initialized: AnalysisTechnique_Initialized

指标练习 #18:'$18_GDRead'

创建新指标并命名为：'#18_GDRead'。

在本例中，我们要对前一个练习中创建的同一个 GlobalDictionary 对象创建引用，不过这一次我们要在基准表现值每次被其他指标改变时读取。

为了少打一些字，您可以从前一个示例中复制输入和变量声明，然后加上保存另外两个表现值的变量声明以及字典内的基准交易代码名。

```
Input: int LookBack( 20 );

Var: elsystem.collections.GlobalDictionary GD(null),
      double BenchPerf(0), SymRowPerf(0), RelPerf(0),
      string BenchSymbol("");
```

与之前一样，在属性编辑器里选择 Analysis Technique，点击事件图标，双击 Initialized 创建一个仅在指标第一次运行时执行一次的事件处理器方法。

我们将在 AnalysisTechnique_Initialized 方法中添加相同的两个参数到 GlobalDictionary，然后创建一个引用 "myAll" 共享字典的方法赋值。接着，将全局字典的两个事件与一个事件处理器方法关联，该方法将在有全局字典条目添加或有值改变时调用。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,elsystem.Initiali
    zedEventArgs args )
begin
    GD = elsystem.collections.GlobalDictionary.Create(true,"myAll");
    GD.ItemAdded += Global_ItemChanged;
    GD.ItemChanged += Global_ItemChanged;
end;
```

下面的方法必须手动输入，因为非组件对象的处理器方法无法自动创建。这个方法将调用 PlotValue 方法，用于在全局字典条目有变化时显示。

```
method void Global_ItemChanged( elsystem.Object sender, elsystem.collections.itempr
    ocessedEventArgs args )
begin
    PlotValues();
end;
```

在 PlotValue()方法中，添加语句来从字典中获取基准交易代码名和表现值（如果不存在则意味着 GD_Write 指标尚未运行）。

```
Method void PlotValues()
begin
    If GD.Contains("TopValue")=true then
    begin
        BenchPerf = GD.Items["TopValue"] astype double;
        BenchSymbol = GD.Items["TopSymbol"] astype string;
    end;
```

下面的语句根据回看时长的收盘价计算雷达屏当前行内交易代码表现。后面接第二个语句来计算当前交易代码与基准交易代码表现之间的差值。

```
SymRowPerf = PercentChange(Close, LookBack) * 100;

RelPerf= SymRowPerf - BenchPerf;
```

绘制基准交易代码名称和相关表现值。

```
Plot1(BenchSymbol, "Bench Sym");
Plot2(BenchPerf, "Bench Perf");
Plot3(SymRowPerf, "SymRow Perf");
Plot4(RelPerf, "Rel Perf");

end;
```

由于表现值基于全局字典基准值的改变或者来自当前行交易代码值的更新，在代码主体中会另外出现一个对 PlotValues() 方法的调用。

```
PlotValues();
```

验证指标。

打开雷达屏并插入指标。在雷达屏中添加一组交易代码，把交易代码周期改为 30分钟以匹配图表中的基准周期。设置交易代码在图表分析窗口与雷达屏窗口联动，这样在雷达屏中点击一条交易代码就可以改变图表中的基准交易代码。

Using（保留字）

保留字 using 可以让您在 EasyLanguage 代码中引用对象类名时无需每次都输入完整的命名空间部分。

例如，当创建表单时，需要为每个控件的每个变量声明对象类型，如：

```
vars:  elsystem.windows.forms.Form form1(Null), elsystem.windows.forms.Button button1( Null ), elsystem.windows.forms.TextBox textbox1( Null );
```

保留字 "using" 可以让您说明对象类型的命名空间部分可以默认，这样在每个声明中只需要指定对象类型，如下所示：

```
using elsystem.windows.forms;

vars: Form form1(Null), Button button1( Null ), TextBox textbox1( Null );
```

窗体控件

EasyLanguage Forms 类创建的对象可以生成作为指标或策略组成部分的独立窗口。窗体对象由用于分组以及显示基本控件（如按钮、文字区域、计数器、数字转盘、组合框等）的容器控件（例如窗体、组或面板）组成。

Form 窗口在指标或策略中创建并激活。
Form 控件可以让您与 TradeStation 分析窗口建立互动。
通过 EasyLanguage 可以访问每个容器和控件对象的属性

各种窗体控件都可以通过设置大小、位置、标签以及特定功能等对象属性来在初始化方法中设置。

窗体容器

窗体 - 一个用于放置窗体相关的控件和容器的窗口。
面板 - 一个窗体内区域，用于组织一组控件的外观效果。
分组框 - 一块用于在外观上将一系列窗体控件分组的区域。

窗体控件

按钮 - 一个用于生成用户点击事件并调用处理器方法的按钮。

复选框 - 一个小方框，可以被用户勾选或取消勾选。用户单击事件会调用一个方法来检查勾选或未勾选状态。

组合框 - 在下拉列表中向用户显示一个条目列表，用户选择后触发可以调用处理器方法的事件。

标签 - 在窗体上显示的不可编辑的文字。标签可用于提供说明或标识周围的控件。

列表视图 - 以多列格式显示文本项集合。用户选择列表视图项时触发可以调用处理器方法的事件。

数字调节框 - 显示数字，用户可以使用上下箭头以预定义的幅度值对快速增减数值。每个数值变化事件都会调用一个事件方法。

单选按钮 - 通常两个或更多个成组放置，让用户可以在一组互斥选项中选择。单选按钮生成用户点击事件，可调用方法来检查按钮状态。

文本框 - 显示可由用户编辑的文字。

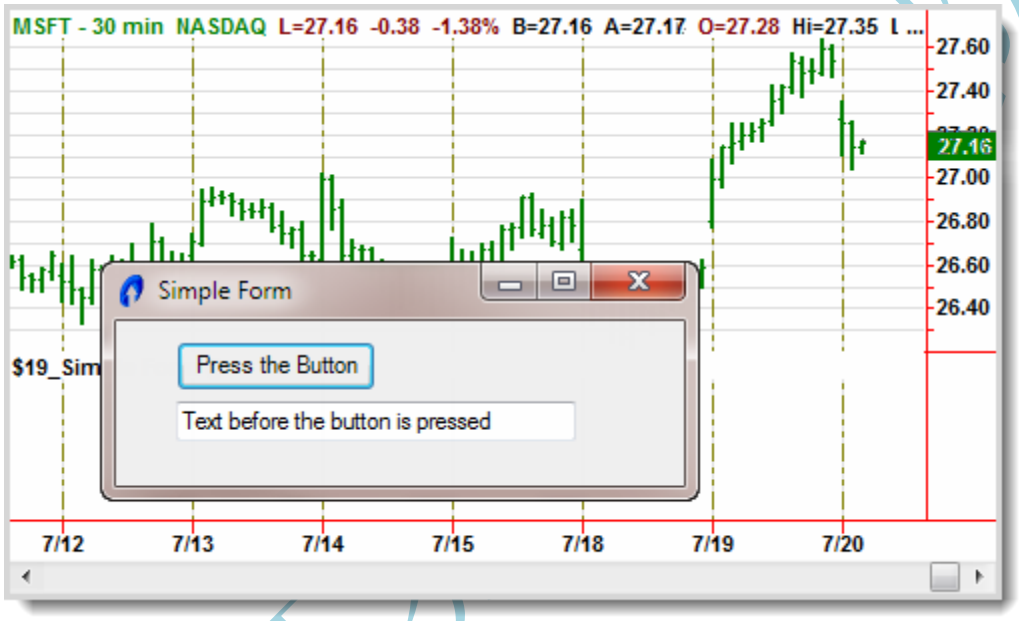
※ 教程示例 #19

目标： （Forms 指标）

创建一组在弹出窗口中显示的窗体控件对象
对特定控件指派有您的代码处理的事件。

指标：'$19_Simple_Form'

本例说明如何用按钮点击改变窗体内的文字。



工作区：$19_Simple_Form

建立图表窗口：

创建：30 分钟周期
插入指标：$19_Simple_Form

组件和属性编辑器设置：

无组件

```
Analysis Technique:
    Initialized: AnalysisTechnique_Initialized
```

指标练习 #19:'$19_Simple Form'

创建新指标并命名为:'#19_SimpleForm.'

为了避免在引用每个对象类型时需要输入完整命名空间路径，我们用一个 using 语句指明默认的命名空间。在这里，由于我们要使用 Forms 对象，所有我们要引用的对象都在 EasyLanguage 对象层级的 elsystem.window.forms 命名空间部分内。

```
using elsystem.windows.forms;
```

接下来，我们要声明三个对象变量，用来保存将会在本例中用到的三个窗体控件。这些控件包括一个含有按钮控件和文本控件的窗体容器。

```
vars: Form form1( Null ),
      Button button1( Null ),
       TextBox textbox1( Null );
```

创建一个指标初始化方法，其中含有仅在指标第一次运行时执行的语句。用属性编辑器打开指标的事件部分，双击 Initialized 事件来创建方法。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
   elsystem.InitializedEventArgs args )
begin
```

在 begin 语句后，我们要给每个控件创建一个实例并将其赋值给相应的对象变量。对大多数窗体控件来说，create 方法可用来指定控件显示的初始文本及其宽度与高度（单位为像素数）。例如，form1 控件（即用来显示其他控件的窗口）的宽度为 300 像素，高度为 120 像素，窗口标题栏显示 "Simple Form"。

```
    form1 = form.create("Simple Form", 300, 120);
    button1 = button.create("Press the Button", 100, 25 );
    textbox1 = textbox.create("Text before pressing button",200,25);
```

在指定每个控件的尺寸和初始文本后，我们还需要指定控件在窗体容器中的位置。在本例中，左上角的按钮将从窗体左侧缩进 30 像素，距窗体顶部 10 像素。文本框控件也将相应缩进 30 像素，距窗体顶部 40 像素。

```
    button1.Location( 30,10 );
    textbox1.Location( 30, 40 );
```

许多控件支持的事件可在出现特定事项（例如点击控件或控件的一个元素）时发出通知。这里，我们将名为 OnButtonClick 的事件处理器方法设置为响应 button1 点击事件。重要的是每次按下按钮都将执行关联方法中的代码。

```
    button1.Click += OnButtonClick;
```

控件创建完毕、指定位置并设置好对事件响应后，我们将它们添加进窗体容器中并显示窗体窗口，控件会处于上文指定的位置。

```
        form1.AddControl(button1);
        form1.AddControl(textbox1);
        form1.Show();
    end;
```

现在，我们要创建一个在窗体内按钮被点击时调用的事件方法。窗体控件事件处理器的方法参数通常都一样，因此可以将其当成其他窗体控件事件处理器的模板。

```
    method void OnButtonClick(elsystem.Object sender, elsystem.EventArgs args )
    begin
```

下面的语句会在按钮按下后改变按钮和文本框内显示的文字。注意我们是如何用按钮文字来判断状态，并为下一个按钮点击事件设置新按钮文字的。

```
        If button1.Text = "Press the Button" then
        begin
            textbox1.Text = "Text after the button has been Pressed";
            button1.Text = "Button Pressed";
        End
        else
          begin
            textbox1.Text = "You Pressed it Again";
            button1.Text = "Press the Button";
          end;
        end;
```

验证指标，然后打开图表并插入指标。

※ 教程示例 #20

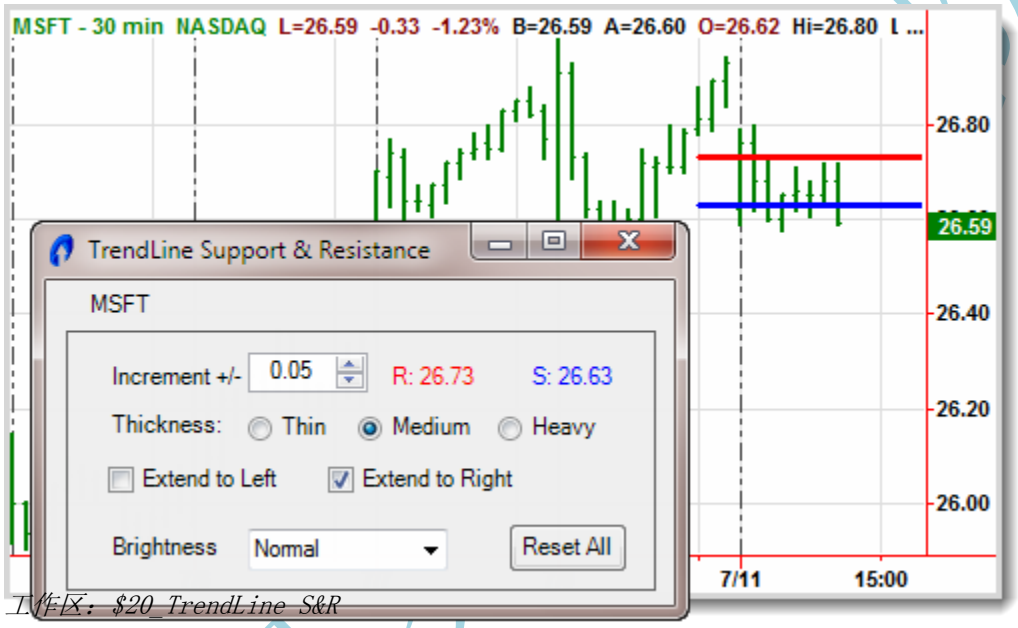
目标： （Forms 指标）

创建一组作为弹出窗口显示的窗体控件对象

为操纵一对趋势线的特定控件指派事件

指标：'$20_TrendLine S&R'

该指标在自定义弹出窗口中用窗体控件改变图表中一对趋势线的位置和样式。



工作区：$20_TrendLine S&R

建立图表窗口：

创建：30 分钟周期

插入指标：$20_TrendLine S&R

组件和属性编辑器设置：

无组件

无属性设置

```
Analysis Technique:
    Initialized:    AnalysisTechnique_Initialized
```

指标练习 #20:'$20_TrendLine S&R'

创建新指标并命名为：'#20_TrendLineS&R'。

这个例子看起来比上一个简单的按钮和文本框练习要长很多也更复杂，主要区别在于创建的窗体控件数量以及响应窗体控件事件的各种事件处理器方法。

无论有多少个控件，创建控件窗口（容器）并在其中使用控件（按钮、标签、组合框）共涉及七个步骤。这些步骤包括：1）为每个控件声明对象变量，2）给每个控件（包括初始文字和大小）创建新的实例，3）设置控件在窗体容器内的位置，4）为控件设置其他参数，5）将事件方法与控件关联，6）向容器内添加控件，7）显示窗体窗口。

像之前一样，我们为每个默认命名空间添加一个 using 语句，这样在声明对象变量或创建对象实例时可以免去大部分重复输入每个对象类型命名空间部分的工作。

```
Using elsystem.windows.forms;
Using elsystem.drawing;
```

首先，我们要声明一系列对象变量，用于引用窗体控件对象及其属性。这是创建窗体七步骤中的第一步。

```
vars:  Form form1(Null),
        Panel panel1( Null ),
        Button button1( Null ),
        NumericUpDown spinner1( Null ),
        Label label1( Null ),
        Label label2( Null ),
        Label label3( Null ),
        Label label4( Null ),
        Label label5( Null ),
        Label label6( Null ),
        RadioButton radio1 ( Null ),
        RadioButton radio2 ( Null ),
        RadioButton radio3 ( Null ),
        CheckBox checkbox1( Null ),
        CheckBox checkbox2( Null ),
        ComboBox combobox1( Null );
```

接下来，我们要声明两个数字变量来跟踪趋势线的高位和低位，我们将把趋势线添加到图表中。

```
vars:TL_High(0.0),TL_Low(0.0);
```

下面要创建一个指标初始化方法，其中包含了仅在指标第一次运行时执行的语句。窗体创建和初始化语句只需要执行一次，因此初始化方法是防止这些代码的理想选择。用属性编辑器打开指标的事件部分，双击 Initialized 事件来创建方法。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender, elsystem.Initial
    izedEventArgs args )
begin
```

在方法内的第一部分语句中，我们将创建每一个窗体控件对象的实例，包括任何初始文字和控件的大小，并将它们赋值给对应控件的对象变量。这是创建窗体七步骤的第二步。

```
form1 = Form.create("TrendLine Support & Resistance",330,200);
panel1 = Panel.create(295,130);

button1 = Button.create("Reset All", 60, 25 );
spinner1 = NumericUpDown.create(60,25);
label1 = Label.create(symbol,80,18);
label2 = Label.create("Increment +/-",80,18);
label3 = Label.create("Thickness:",60,18);
label4 = Label.create("R: 123.45",60,18);
label5 = Label.create("S:123.45",60,18);
label6 = Label.create("Brightness",60,18);
radio1 = RadioButton.create("Thin",45,25);
radio2 = RadioButton.create("Medium",65,25);
radio3 = RadioButton.create("Heavy",65,25);
checkbox1 = CheckBox.create("Extend to Left",100,18);
checkbox2 = CheckBox.create("Extend to Right",100,18);
combobox1 = ComboBox.create("",100,22);
```

然后，我们通过指定容器内控件左上角的 x、y 偏移来确定每个控件的相对位置。这是创建窗体七步骤的第三步。

```
panel1.Location(10,25);
button1.Location(220,95);
spinner1.Location(90,10);
label1.Location(20,5);
label2.Location(20,15);
label3.Location(20,39);
label4.Location(160,15);
label5.Location(230,15);
label6.Location(20,100);
radio1.Location(90,35);
radio2.Location(145,35);
radio3.Location(215,35);
checkbox1.Location(20,65);
checkbox2.Location(130,65);
combobox1.Location(90,98);
```

下面的语句设置控件的其他属性，例如初始显示值或外观。这是创建窗体七步骤的*第四步*。如需特定控件可用属性的更多信息，请参阅 EasyLanguage 字典或者帮助主题。

```
label4.ForeColor = Color.Red;
label5.ForeColor = Color.Blue;

panel1.BorderStyle = 1;

spinner1.DecimalPlaces = 2;
spinner1.TextAlign = 2;
spinner1.Increment = .05;
spinner1.Value = .05;

radio1.Checked = true;
checkbox1.Checked = false;
checkbox2.Checked = false;

combobox1.AddItem("Lighter");
combobox1.AddItem("Normal");
combobox1.AddItem("Darker");
combobox1.SelectedIndex = 1;
```

接下来，我们将把事件处理器与各种控件事件（比如点击事件或控件值改变事件）关联。这是创建窗体七步骤的第五步。本练习后半部分我们回顾事件处理器方法时，将会介绍每个事件用法的更多信息。

```
button1.Click += OnButtonClick;

spinner1.ValueChanged += SpinnerClick;

radio1.Click += Resize_TrendLine;
radio2.Click += Resize_TrendLine;
radio3.Click += Resize_TrendLine;

CheckBox1.Click += Extend_TrendLine;
CheckBox2.Click += Extend_TrendLine;

ComboBox1.SelectedIndexChanged += ChangeColor;
```

每个控件在设置好属性（大小，位置，初始值，和事件）后，就可以添加进窗体容器中了。在本例中，我们要用到一个称为“面板”的临时容器来分组放置大部分控件，然后我们将面板添加到最底层的窗体窗口容器。用二级容器（例如面板）对控件分组的好处在于可以对一组控件进行整体操作：比如在窗口内移动整组控件，或者复制整个面板及其中控件并粘贴到另一个窗体，而无需重新摆放每个控件。控件添加到面板后，窗体顶部会添加一个标签，后面跟着整个面板容器。这是创建窗体七步骤中的*第六步*。

```
panel1.AddControl(button1);
panel1.AddControl(spinner1);
panel1.AddControl(label2);
panel1.AddControl(label3);
panel1.AddControl(label4);
panel1.AddControl(label5);
```

```
        panel1.AddControl(label6);
        panel1.AddControl(radio1);
        panel1.AddControl(radio2);
        panel1.AddControl(radio3);
        panel1.AddControl(checkbox1);
        panel1.AddControl(checkbox2);
        panel1.AddControl(combobox1);

        form1.AddControl(label1);
        form1.AddControl(panel1);
```

在显示窗体窗口前，其位置被设定在距离屏幕左侧 100 像素、屏幕顶部 100 像素的位置，这样，窗口将总是出现在屏幕上的相同位置。窗体属性 TopMost 被设为真，这样窗体窗口会浮于其他窗口上方，点击窗体外区域时不会被其他窗口遮挡。这样，就完成了创建和显示自定义窗体窗口七个基本步骤的第七步。

```
        form1.Location(100,200);
        form1.TopMost = true;
        form1.show();
```

最后，当图表开始计算，在图表左侧远端要创建一对趋势线并保存每根趋势线的 ID。上趋势线（阻力线）初始颜色被设为红色，下趋势线（支撑线）初始颜色被设为绿色。另外，因为趋势线会重新定位到图表最后一根 K 线，因此会在这个初始位置看到它们。

```
        TL_High = TL_New(Date[10], Time[10], High, Date, Time, High);
        TL_Low = TL_New(Date[10], Time[10], Low, Date, Time, Low);

        TL_SetColor(TL_High,RGB(255,0,0));
        TL_SetColor(TL_Low,RGB(0,0,255));
    end;
```

接下来，我们开始创建处理每个窗体控件事件的方法。

第一个是响应点击标有 "Reset All" 按钮的方法。这个方法中有一个语句会发出一个特殊类型的异常“消息”，会导致图表重新加载、窗体控件被重置。

```
Method void OnButtonClick(elsystem.Object sender, elsystem.EventArgs args )
begin
    throw elsystem.RecalculateException.Create("");
end;
```

用默认的 sender 和 args 参数，为窗体控件事件处理器创建一个名为 Resize_Trendline 的方法（与前一个方法相同）。声明两个本地变量：一个是 RadioButton 类型的对象变量，另一个是整数型变量。

此方法用 sender 参数来获取选中的单选按钮，然后读取该按钮的文字来决定所选按钮的线条粗细样式，并向 TL_SetSize 传递新的粗细值来改变图表中上下趋势线的样式。每个单选按钮控件指定一种趋势线的粗细样式，分别为 "Thin"（细）、"Medium"（中）或 "Heavy"（粗）。

```
Method void Resize_TrendLine ( elsystem.Object sender, elsystem.EventArgs args )
var:RadioButton rButton, int TLSize;
begin
    rButton = sender astype RadioButton;
    TLSize = 0;

    Switch (rButton.Text)
    Begin
        Case "Thin":TLSize = 0;
        Case "Medium":TLSize = 2;
        Case "Heavy":TLSize = 4;
    End;

    TL_SetSize(TL_High,TLSize) ;
    TL_SetSize(TL_Low,TLSize) ;
end;
```

创建 ChangeColor 方法来处理 ComboBox1 下拉控件中的颜色选择。Case语句读取选中的 ComboBox 项目的索引值并将上、下趋势线的颜色设定为合适的 RGB（红绿蓝）值。

```
Method void ChangeColor( elsystem.Object sender, elsystem.EventArgs args )
begin
    Switch (combobox1.SelectedIndex)
    Begin
        Case 0:TL_SetColor(TL_High,RGB(255,127,127));
               TL_SetColor(TL_Low,RGB(127,127,255));
        Case 1:TL_SetColor(TL_High,RGB(255,0,0));
               TL_SetColor(TL_Low,RGB(0,0,255));
        Case 2:TL_SetColor(TL_High,RGB(127,0,0));
               TL_SetColor(TL_Low,RGB(0,0,127));
    End;
end;
```

创建 Extend_Trendline 方法来处理复选框控件的选择，并根据所选复选框对应的状

态设置两条趋势线的左右延伸值。

```
method void Extend_TrendLine( elsystem.Object sender, elsystem.EventArgs arg
      s )
begin
    TL_SetExtLeft(TL_High,checkbox1.checked);
    TL_SetExtLeft(TL_Low,checkbox1.checked);
    TL_SetExtRight(TL_High,checkbox2.checked);
    TL_SetExtRight(TL_Low,checkbox2.checked);
end;
```

创建 SpinnerClick 方法来处理数值增减（spinner1）控件的值变化。这个数值通过点击上/下微调箭头或在控件数字框内输入新值来修改。新值会传递到另一个方法用于更新趋势线。

```
method void SpinnerClick( elsystem.Object sender, elsystem.EventArgs args )
begin
    Show_TrendLine(spinner1.value);
end;
```

Show_TrendLine 方法用来在窗体上显示支撑位和阻力位，并根据 myNum 方法输入参数的值，在前一 K 线的中间价上方和下方绘制趋势线。上（阻力）趋势线和下（支撑）趋势线分别绘制在中间价上方和下方，距离为偏移量 myNum。每根趋势线在不延长时宽度均为 10 根 K 线。

```
Method void Show_TrendLine(double myNum)
var: double midprice;
begin
    MidPrice = (High[1]+Low[1]) * .5;
    Label4.Text = "R:" + NumToStr(MidPrice+MyNum,2);
    Label5.Text = "S:" + NumToStr(MidPrice-MyNum,2);
    TL_SetEnd(TL_High, Date, Time, MidPrice+myNum);
    TL_SetBegin(TL_High, Date[10], Time[10], MidPrice+myNum);
    TL_SetEnd(TL_Low, Date, Time, MidPrice-myNum);
    TL_SetBegin(TL_Low, Date[10], Time[10], MidPrice-myNum);
end;
```

我们还想根据实时价格变化更新趋势线位置，所以我们将添加一个语句，在最后一根 K 线的每笔交易价格变化时调用趋势线绘制方法。

```
If LastBarOnChart then Show_TrendLine(spinner1.value);
```

验证指标。

打开图表并插入指标。

附加练习部分

以下附加示例包含了在线教程中不包含的附加材料。这些示例将前面介绍的组件与对象与一些新元素结合起来，当您继续探索 EasyLanguage 对象的功能时，可能会发现它们很有用。

取消订单

待成交订单可通过 Order 对象的 Cancel() 方法取消。访问特定交易代码的待成交订单的一个常用方法是用 OrdersProvider 组件为当前交易代码创建一个“收到”订单的集合。得到的集合内第一个订单是最近的未成交订单。在该订单的标识符后加上 Cancel() 方法将取消该订单。

```
OrdersProvider.Order[0].Cancel();
```

要取消同一交易代码的所有未成交订单，只需遍历交易代码的所有“收到”的订单并在每个订单标识符后使用 Cancel() 方法。

要取消-替换，只需按上述步骤取消相应订单，然后以新价格发送另一个 OrderTicket。

限价订单

任何 OrderTicket 组件都支持提交限价订单。例如，通过 OrderTicket 组件，可以用属性编辑器来设置订单参数，将订单的 Type 设为 Limit 并设置指定的 LimitPrice。然后，用 OrderTicket 的 Send() 方法提交所需的限价订单。

```
OrderTicket1.Send()
```

或者，也可以在发送前直接修改代码中之前设定的 OrderTicket 订单参数，例如：

```
OrderTicket1.Type= tsdata.trading.OrderType.Limit;
OrderTicket1.LimitPrice = Close - .10;
OrderTicket1.Send();
```

附加示例 #21

目标：（限价订单 - 取消）

创建一个可以提交和取消限价订单的简单订单窗口

取消限价订单

指标：'$21_LimitCancel'

本指标用一个 OrderTicket 和 OrdersProvider 在窗体中下一个买入限价单，并允许取消待成交订单。

警告：本指标将产生一个限价订单——请确保未登录实盘账户。应该仅在登录到 TradeStation 模拟器时应用本指标。

工作区：$21_LimitCancel

建立图表窗口：

创建：使用外汇交易代码，5 分钟周期

插入指标：$21_LimitCancel

组件和属性编辑器设置：

OrderTicket1:

Symbol:	*symbol*	（当前交易代码的保留字）
Accounts:	*iAccounts1*	（添加默认 Account 作为输
Quantity:	*iQuantity1*	（添加订单 Quantity 作为输
Action:	*buy买*	
Type:	*limit*	

OrdersProvider1:

Load :	*false*	（**重要** - 启动时组件关闭）
Symbols:	*symbol*	（当前交易代码的保留字）
Accounts:	*iAccounts1*	（使用同一个 Account 输入）
Updated:	OrdersProvider1_Updated	

Analysis Technique:

- Initialized: AnalysisTechnique_Initialized

指标练习 #21:'$21_LimitCancel'

创建新指标并命名为：'#21_LimitCancel'

本例综合了之前用过的数个组件和对象。

添加一个名为 OrderTicket1 的 OrderTicket 组件对象。

添加一个名为 OrdersProvider1 的 OrdersProvider 组件对象。

使用属性编辑器按上述要求设置 OrderTicket1 的属性，包括输入初始订单数目 10000 和您的模拟外汇账号。创建输入后，下面两行将添加到代码中：

```
Input: int iQuantity1( 100000 );
Input: string iAccount1( "SIM00000" );
```

使用属性编辑器按上述要求设置 OrdersProvider1 的属性。为 Updated 事件创建一个处理器方法。注意：请确保将 Load 属性设为 false，防止订单状态事件在窗体元素创建之前访问窗体元素。

在属性编辑器中，选择 Analysis Technique 并为 Initialized 事件创建一个处理器方法。

为窗体 (forms) 和交易 (trading) 添加 using 语句，减少输入完整控件名称的工作量。

```
using elsystem.windows.forms;
using tsdata.trading;
```

为主窗体声明对象变量以及三个控件，包括一个文本框、一个标签和一个按钮。

```
vars: Form form1( Null ), TextBox OrderPx( Null ), Label label1 ( Null ), Button
      button1( Null );
```

在 AnalysisTechnique_Initialized 方法中，创建一个简单窗体对象，其中包含设置限价的控件和发单或撤单的按钮。一旦限价单生成后，"Send"（发单）按钮将重新标为 "Cancel/Replace"（撤单/改单）。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,elsystem.Initial
    izedEventArgs args )
begin

  form1 = form.create("Limit Order", 250, 120);
  form1.location(200,200);
  form1.topmost = true;

  OrderPx = TextBox.create("",50,20);
  label1 = label.create("Set Limit Price:",90,20);
  button1 = button.create("Place Buy Limit Order", 140, 25 );
```

```
OrderPx.Location( 100,10 );
label1.Location(10,12);
button1.Location( 40,40 );
```

将按钮与事件处理器方法关联，在窗体上添加控件，将初始文本框文本值设为当前买入报价，在窗体内显示图表。

```
button1.Click += OnButtonClick; OrderPx.Text = NumtoStr(InsideBid,5);

form1.AddControl(OrderPx);
form1.AddControl(label1);
form1.AddControl(button1);

form1.show();
```

将 Provider 的属性设置为 true，让订单状态事件能更新窗体，现在窗体创建完成。

```
OrdersProvider1.Load = true;
```

下面在 AnalysisTechnique_Initialized 方法和 OrdersProvider1_Updated 事件处理器方法中添加了对 ReplotButton 方法的调用。

```
    ReplotButton();
end;
```

添加 OrdersProvider1_Updated 方法。

```
method void OrdersProvider1_Updated( elsystem.Object sender, tsdata.trading.Orde
     rUpdatedEventArgs args )
begin
    ReplotButton();
end;
```

ReplotButtons 方法用于根据限价订单是否有效来设定按钮的文字。Count 属性大于 0 代表订单待成交，我们可以检查它是否“已接收”以及是否为“限价”订单。如果是，则按钮文字设置为“取消订单”。如果限价订单有效，则文本框文字设为限价订单价格。如果没有待成交订单，则按钮文字设为“提交买入限价订单”。

```
Method void ReplotButton()
begin
  if OrdersProvider1.Count > 0 AND
     OrdersProvider1.Order[0].State = OrderState.received AND
     OrdersProvider1.Order[0].Type = OrderType.limit
  then
    begin
      button1.Text = "取消订单";
      OrderPx.Text= NumtoStr(OrdersProvider1.Order[0].LimitPrice,5);
    end
  else
    begin
      button1.Text = "提交买入限价订单";
    end;
end;
```

最后，创建一个 OnButtonClick 方法来处理鼠标点击事件。检查 button1 的文字来决定发什么订单，并确保文字完全匹配。如果最近订单是有效的限价单，在点击按钮事件发生时，将发送撤单命令。如果不存在有效限价单，则以文本框中指定的价格向市场发送“买入限价”订单。

```
method void OnButtonClick( elsystem.Object sender, elsystem.EventArgs args )
begin
   If LastBarOnChart then
   begin
       if button1.Text = "取消订单" AND
           OrdersProvider1.Count > 0 AND
           OrdersProvider1.Order[0].State = OrderState.received AND
           OrdersProvider1.Order[0].Type = OrderType.limit
       then
           OrdersProvider1.Order[0].Cancel();

       If button1.Text = "提交买入限价订单" then
       begin
             orderticket1.SymbolType = Category;
             orderticket1.LimitPrice = StrtoNum(OrderPx.Text);
             orderticket1.Quantity = iQuantity1;
             orderticket1.send();
       end;
   end;
end;
```

验证指标。

打开图表并插入指标。

DateTime

DateTime 对象代表时间上的一个瞬间，用一个日期和一天中的时间表示。其取值范围是从公元 0001年1月1日 12:00:00 AM 到公元 9999年12月31日 11:59:59 PM。

除了能够读取当前日期和时间，该对象还提供了一系列属性和方法，以便您能以所选的方式方便设置和显示日期与时间值。

```
Var:DateTime myDateTime(null);

// 根据字符串值设置日期和时间并以另一种格式显示
myDateTime.Value = "12/10/2011 9:35pm"      // 设置用户指定的日期/时间
plot1(myDateTime.Format(("%m-%d-%y %H:%M")); // 显示为 12-10-11 21:35

myDateTime = elsystem.datetime.Now      // 获取电脑日期与时间
myDateTime = elsysetm.datetime.CurrentTime // 仅获取当前时间
myDateTime = elsystem.datetime.Today    // 获取当前日期，不含时间
```

TimeSpan

TimeSpan 对象代表一个时间区间，用来测算日期/时间值之间的（正负）天数、小时数、分钟数、秒数。 用一个 TimeSpan 可以保存两个日期和时间之间的差值，也可以在一个现有日期/时间上增加（或减去）一段时间来指定新的日期/时间。

例如，

```
Var:TimeSpan myTimeSpan(null), DateTime myDateTime(null);

// 在当前日期上增加 5 天
myTimeSpan.TotalDays = 5
myDateTime = elsystem.datetime.Today + myTimeSpan;

// 计算当前时间到收市前 5 分钟还有多久
myDateTime.Value = "3:55pm"
myTimeSpan = myDateTime - elsystem.DateTime.CurrentTime;
```

TokenList

TokenList 代表一个值的集合，可以从以逗号分隔的单词或数字列表创建，使用 Add 属性可以添加值。

例如，可以创建一个包含四个交易代码的 TokenList 并从得到的集合中用索引值属性访问其中的任何值。

```
Var:TokenList myList(null);

// 将值加载到 TokenList
myList = TokenList.Create("msft,csco,dell,ibm");
print( myList.Item[1] );// 打印第二个条目 "csco"
```

附加示例 #22

目标：(AlarmClock)

根据用户提供的时间（或日期加时间）创建一对闹钟

指标：'$22_AlarmClock'

本指标用 DateTime 和 TimeSpan 对象获取当前时间与目标时间（可选择附带日期）之间的时间差。它会设定一个倒计时定时器，到达目标时间后调用事件方法。

工作区：$22_AlarmClock

建立窗口：

创建：图表或雷达屏
插入指标：$22_AlarmClock

组件和属性编辑器设置：

Timer1:
- Interval : *1000* （默认值）
- AutoReset: *True* 〃〃
- Enable: *False* （启动时定时器关闭）
- Elapsed:Timer1_Elapsed

Timer2:
- Interval : *1000* （设为每秒计数一次）
- AutoReset: True 〃 〃
- Enable: True （启动定时器，带有指示器）
- Elapsed: Timer2_Elapse

Analysis Technique:
- Initialized: AnalysisTechnique_Initialized

指标练习 #21:'$22_AlarmClock'

创建新指标并命名为：*'#22_AlarmClock'*。

添加名为 *Timer1* 的 Timer 组件。再添加一个名为 *Timer2* 的 Timer 组件。

使用属性编辑器，按上述要求设置 Timer1 和 Timer2 的属性，包括为每个 Elapsed 事件添加一个处理器方法。注意 Timer1 一开始未启用。

在属性编辑器中，选择 *Analysis Technique* 并为 Initialized 事件创建一个处理器方法。

创建两个目标时间的输入。目标时间可以包括一个选填的日期，例如 "8/22/2012 9:25 am"。第一个目标应该是较早的时间（或日期），因为只有到第一个目标时间后才会评估第二个目标时间。默认情况下，目标时间设在常规开盘和收盘时间前的五分钟。

```
input: string TargetTime1("9:25am"),string TargetTime2("3:55pm");
```

声明对象变量来保存目标时间、到目标时间的初始时间差以及闹钟时间的列表。虽然本例只使用了两个闹钟，在列表对象中很方便添加其他时间。

```
var: elsystem.DateTime myAlarmTime(null),
     elsystem.TimeSpan myTimeSpan(null),
     tsdata.common.TokenList myAlarmList(null),
```

另外声明两个变量，用于闹钟时间到后播放的声音文件名以及闹钟列表中当前闹钟的索引值。

```
     string AlarmSound("alarmclockbeep.wav"),
     intrabarpersist int myAlarmIndex(0);
```

在 Initialized 方法中，将闹钟列表创建为 TokenList 对象的一个实例，并用第一个目标时间作为列表的初始值。使用加等运算符将第二个目标时间附加到闹钟列表中。注意：如果要另外添加时间，请在此处加入。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,
    elsystem.InitializedEventArgs args )
begin
    myAlarmList = tsdata.common.TokenList.create(TargetTime1);
    myAlarmList += TargetTime2;
```

使用 GetAlarmTime 来获取列表中下一个还没到的目标时间。用 GetAlarmSpan 来计算当前时间和目标时间之间的时间差。然后，用 SetAlarm 来设置并激活闹钟倒计时。

```
    myAlarmTime = GetAlarmTime(myAlarmList);
    myTimeSpan = GetAlarmSpan(myAlarmTime);
    SetAlarm(myTimeSpan);
end;
```

然后，将下列语句添加到与 Timer1 关联的事件处理器方法。当时间到后，它会在打印日志中打印出闹钟时间、播放输入中指定的声音文件并获取下一个要设置的闹钟时间段。

```
method void Timer1_Elapsed( elsystem.Object sender,
  elsystem.TimerElapsedEventArgs args )
begin
    print("ALARM sounded at  - ",elsystem.datetime.currenttime.
        tostring());
    condition1 = playsound(AlarmSound);

    myAlarmTime = GetAlarmTime(myAlarmList);
    myTimeSpan = GetAlarmSpan(myAlarmTime);
    SetAlarm(myTimeSpan);
end;
```

Timer2_Elapsed 方法每一秒执行一次。此方法会获取下一个闹钟目标的时间段并建立一个含有剩余天数、小时数、分钟数、秒数的字符串，或一条消息说明所有闹钟均已过期。

```
 method void Timer2_Elapsed(elsystem.Object sender, elsystem.TimerElapsedEventAr
      gs args)
var: string TimeLeft, elsystem.TimeSpan TS;
begin
    TS = GetAlarmSpan(myAlarmTime);

    If myTimeSpan.TotalSeconds > 0 then
        TimeLeft = TS.Days.tostring() + "d "
                   + TS.Hours.tostring() + "h "
                   + TS.Minutes.tostring() + "m "
                   + TS.Seconds.tostring() + "s "
    Else
       TimeLeft = "No alarms active";
```

第一个图形显示了剩余的时间。

```
    plot1(TimeLeft,"Count Down");
```

第二个图形显示实际目标时间 "Target"。

```
    If myAlarmTime.ELDateTimeEx>=1 then
        plot2(myAlarmTime.format("%m/%d/%y %H:%MT"+
        myAlarmIndex.tostring()),"Target")
    else
         plot2(myAlarmTime.format("%H:%MT"+ myAlarmIndex.tostring()),"Target");
end;
```

创建一个方法将时间段（剩余时间）转换为毫秒，并赋值给 Timer1 计数器 Interval 属性，并打开定时器。如果剩余时间小于 1 秒 (<1000ms)，则关闭定时器。

```
method void SetAlarm(elsystem.TimeSpan myTimeSpan)
begin
    Timer1.Interval = myTimeSpan.TotalMilliseconds;
    If Timer1.Interval < 1000 then
        Timer1.Enable = false
```

```
        Else
            Timer1.Enable = true;
    end;
```

GetAlarmTime 方法返回一个代表下一个闹钟目标时间的对象。第一个语句创建一个名为 myAlarmTime 的 DateTime 对象。For 循环会遍历列表中的闹钟时间（TokenList 中的字符串）并用 DateTime.Parse 方法将闹钟字符串转换为 DateTime 对象。

```
method elsystem.DateTime GetAlarmTime(tsdata.common.tokenlist myAlarmList)
begin
   myAlarmTime = elsystem.DateTime.Create();

   For myAlarmIndex = 0 to myAlarmList.Count-1
   begin

      myAlarmTime = elsystem.datetime.Parse(myAlarmList[myAlarmIndex]);
```

调用 GetAlarmSpan 会获取剩余时间，如果时间段是一个正数，则返回闹钟时间。如果未找到正值时间段，则返回一个空闹钟时间。

```
      myTimeSpan = GetAlarmSpan(myAlarmTime);

      If myTimeSpan.TotalSeconds>0 then
      begin
            myAlarmIndex = myAlarmIndex + 1;
            Return myAlarmTime;
       end;
    end;
    return myAlarmTime;
end;
```

GetAlarmSpan 方法返回一个代表现在和 myAlarmTime 之间剩余时间的对象。第一个语句创建一个名为 myTimeSpan 的 TimeSpan 对象。如果到下一个闹钟的天数大于等于 1，则将用完整的日期和时间来计算时间段。如果只有时间部分，则从当前时间开始计算时间段。

```
method elsystem.TimeSpan GetAlarmSpan(elsystem.DateTime myAlarmTime)
begin
   myTimeSpan = elsystem.TimeSpan.Create();

   If myAlarmTime.ELDateTimeEx >= 1 then
      myTimeSpan = myAlarmTime - elsystem.datetime.now

   Else if myAlarmTime.ELTime > 0 then
      myTimeSpan = myAlarmTime - elsystem.datetime.currenttime;

   Return myTimeSpan;
end;
```

验证指标。

打开图表或雷达屏并插入指标。使用输入值修改闹钟时间。

指标 - UnInitialized 事件

指标（Analysis Technique）的 Unitialized 事件用于调用事件处理器方法，该方法包含在指标计算结束前执行的代码。

AnalysisTechnique_*Uninitialized* 事件处理器方法可以用来在指标关闭前保存数据或执行任何可能需要运行的清理代码。当指标更新后、从分析窗口内移除后或在窗口、工作区、桌面被关闭后均可能调用此方法。但是，如果出现异常错误，有时候 uninitialized 事件处理方法可能不会被调用。

Try-Catch

Try-Catch 代码块用于在 Try 代码块内测试一系列语句的执行，从而可以在 Catch 代码块中处理任何失败、错误或异常。

例如，当访问一个组件的数据时，您可能会想测试一个预期的错误条件。如果发生特定条件，您可以设一个标志供以后使用，以防止更多计算或者向用户打印一个自定义信息。

```
try
     Value1 = FQ1.Quote[FundamentalField].DoubleValue[0] ;
     OkToCalculate = true ;
catch ( NoFundamentalDataAvailableException NFDEx )
     OkToCalculate = false ;
     Print( "NMF - No fundamental value for primary symbol.") ;
end ;
```

或者，您可能希望通过一个文件读取语句的测试成功或失败
来确定文件是否存在，如果不存在则创建该文件。

```
try   doc.Load("c:filename"); // 尝试从文件读取
catch(elsystem.io.filenotfoundexception ex)
     doc.Save("c:filename"); // 如果未找到则创建文件
end;
```

XML 对象

EasyLanguage 支持一系列对象，可以让您创建 XML 数据结构、将之保存到文件并在以后读取。EasyLanguage 中的 XML 对象基于常用的 XML 文档对象模型，其中包含节点、元素、属性和其他与 XML 数据层级关联的项目。

XML 的基本构建单位是*元素*，由放置在一对开始与结束标记间的数据构成。开始标记由带尖括号的元素名构成，例如 <symbol>，结束标记也一样，不过在元素名前还要加上斜杠，例如 </symbol>。

XML 数据结构由多个元素组成，每个元素都代表数据，这些数据可以是一条信息，例如一个单词或数字，也可以是其他有附加数据的子元素。在 EasyLanguage 中有 XML 对象可用于创建元素、在数据结构中添加或删减元素以及在元素中读写数据。

例如，下面的 XML 数据结构包含一个根（root）元素，该元素包含 'symbol'（交易代码）元素，其中又含有三个相关数据元素：交易代码名称、日期与价格。

```
<root>
   <symbol>
      <name>MSFT</name>
      <date>9/10/2011</date>
      <price>34.56</price>
   </symbol>
<root>
```

另外，开始标签可以在尖括号中包含称为*属性*的附加信息。例如，下面的代码包含两个 step 元素，第一个元素带有一个数字 1 的属性，第二个带有数字 2 的属性。

```
<root>
   <step number="1">First Item</step>
   <step number="2">Second Item</step>
</root>
```

附加示例 #23

目标：（XMLPersist 指标）

创建一个 XML 文件，用于在指标重新计算时保存和恢复数据

指标：'$23_XMLPersist'

当一个指标退出或刷新（Uninitialized）时，本指标使用 XML 对象来将当前交易代码的信息保存到一个 XML 文件中，在指标重新开始（Initialized）后可以回读。例如，这可以实现在下次对同一交易代码运行本指标时，保留当前指标实例的交易代码值。

工作区：$23_XMLPersist

建立窗口：

创建：30 分钟周期
插入指标：$23_XMLPersist

组件和属性编辑器设置：

无组件

Analysis Technique:
- Initialized: AnalysisTechnique_Initialized
- Uninitialized: AnalysisTechnique_UnInitialized

用法说明：

当第一次将指标应用到一个新的图表或代码时，指标的状态行将只会显示交易代码名称，不会有其他信息。当退出图表或按下 Ctrl+R 刷新图表后，收盘价以及当前时间和日期会保存到 XML 文件。当下一次该交易代码绘制在图表上时，在子图的状态行交易代码名后面就会显示 XML 文件中保存的价格和时间。

指标练习 #23:'$23_XMLPersist'

创建新指标并命名为：*'#23_XMLPersist'*

为 XML 命名空间添加一个 using 语句，免得需要输入 XML 对象的全名。

```
using elsystem.xml;
```

为一组 XML 对象声明对象变量，这些 XML 对象将用于在外部 XML 文件中创建和管理数据。

```
var:  XmlDocument doc(null),
      XmlElement root(null),
      XmlElement eLevel1(null),
      XmlElement eLevel2(null),
      XmlNode nNode(null);
```

声明另一个变量，用于测试 XML 数据文件是否存在。如果不存在，则需要创建该文件。

```
Var:   bool filefound(false);
```

为 XML 文件名声明一个输入。

```
input:   string xmlFileName("C:\RefSym.xml");
```

在属性编辑器中，选择 *Analysis Technique* 并为 *Initialized* 事件创建一个处理器方法。然后为 *Uninitialized* 事件再创建一个处理器方法，用于在我们退出或重新计算指标时调用。

在 Initialized 方法中，创建一个 XML 文件对象实例并将 filefound（找到文件）标志初始化为 true。

```
method void AnalysisTechnique_Initialized( elsystem.Object sender,elsystem.InitializedE
  ventArgs args )
begin
    doc = new xmldocument;
    filefound = true;
```

接下来，我们将用 Try-Catch 语句检查 XML 文件是否存在。如果 Try 语句中的 doc.Load 调用失败，则 catch 语句下的 filefound 标志设为 false。

```
    try
        doc.Load(xmlFileName);
    catch(elsystem.io.filenotfoundexception ex)
        filefound = false;
    end;
```

如果未找到文件，我们要创建该文件：首先定义一个名为 "data" 的 root 对象，将之保存到新 XML 文件。

```
    If filefound = false then
    Begin
        root = doc.CreateElement("data");
```

```
        doc.AppendChild(root);
        doc.Save(xmlFileName);
    End
```

如果文件存在，则将文件的 XML 结构读取到 root 对象。

```
        Else
        Begin
            root = doc.DocumentElement;
        end;
```

FindSymbol 方法返回包含当前交易代码数据的节点。如果此前未使用过该交易代码或文件刚刚创建，则数据为空。

```
        nNode = FindSymbol();
    end;
```

当指标关闭（如退出 TradeStation 时）或指标重新计算（如图表刷新时）时会执行 AnalysisTechnique_UnInitialized 方法。在此方法中添加一个对 SaveSymbol 的调用。

```
    method void AnalysisTechnique_UnInitialized( elsystem.Object sender, elsystem.UnInitializedEventArgs args )
    begin
        SaveSymbol();
    end;
```

在 SaveSymbol 方法中我们将更新想保存的数据（在本例中是当前的日期/时间和收盘价）并将数据写入到 XML文件中。

```
    Method void SaveSymbol()
    begin
        nNode.Item["date"].InnerText = elsystem.datetime.now.Format("%m- %d-%Y%H:%M");
        nNode.Item["price"].InnerText = Close.tostring();
        doc.Save(xmlFileName);
    end;
```

FindSymbol 方法用于查找已保存交易代码的数据，或为新交易代码创建空白数据，并返回所请求数据的 XML 节点。前两个本地变量用于查找和保存匹配当前交易代码的元素的索引值。下一个本地变量声明一个 XML 对象，用于保存可查询的 symbol 元素列表。最后的变量将用于保存方法返回的、包含交易代码数据的节点。

```
    Method xmlNode FindSymbol()
    var: int iIndex, int sNumb, XmlNodeList tList, XmlNode tNode;
    begin
```

前几个语句获取一级 symbol 元素列表并遍历列表元素，以确认是否有任何 "name" 子元素与当前代码匹配。如有，则变量 sNumb 设为该交易代码的索引值。

```
        tList = root.GetElementsByTagName("symbol");
        sNumb = -1;

        for iIndex = 0 to tList.count-1
        begin
            If tList.ItemOf[iIndex].Item["name"].innertext=symbol then sNumb = iIndex;
        end;
```

如果在列表中未找到该交易代码，则创建一个新的 symbol 元素，其名称、日期与价格作为二级子元素附加。新的 symbol 元素结构附加到 root 并保存到文件。

```
If sNumb = -1 then
begin
    eLevel1 = doc.CreateElement("symbol");

    eLevel2 = doc.CreateElement("name");
    eLevel2.innertext = symbol;
    eLevel1.AppendChild(eLevel2);

    eLevel2 = doc.CreateElement("date");
    eLevel1.AppendChild(eLevel2);

    eLevel2 = doc.CreateElement("price");
    eLevel1.AppendChild(eLevel2);

    root.AppendChild(eLevel1);

    doc.Save(xmlFileName);
```

新创建的 symbol 元素结构赋值给本地方法对象 tnode，如果交易代码之前已存在，则 tnode 设置带有索引的交易代码元素。方法的返回值随后设为节点。

```
    tNode = eLevel1;
end

Else
begin
    tNode = tList.ItemOf[sNumb];
End;

Return tNode;
end;
```

最后，创建一个方法来绘制当前节点值。对于已存在或已刷新的交易代码，日期和价格会显示为保存的值。注意对 PlotValues() 的调用是您的 EasyLanguage 代码主体中的唯一语句。

```
Method void PlotValues()
begin
    plot1(nNode.Item["name"].Innertext,” Symbol” );
    plot2(nNode.Item["date"].InnerText,” Saved Date” );
    plot3(nNode.Item["price"].InnerText,” Saved Price” );
end;

PlotValues();
```

验证指标。

打开图表并插入指标。您会发现新创建的 XML 文件已经添加到指定路径，并包含当前交易代码的初始 XML 数据。

附录 A

常用基本字段

快照字段（非历史）

基本报价字符串	说明
ATA	资产总计 (FY)
QTA	资产总计 (MRQ)
BETA_DOWN	标普500贝塔低端市场
BETA_UP	标普500贝塔高端市场
ABVPS	每股账面价值 (FY)
QBVPS	每股账面价值 (MRQ)
ACURRATIO	流动比率 (FY)
QCURRATIO	流动比率 (MRQ)
ADIVSHR	每股派息 (FY)
YIELD	股息收益率
AEPSINCLXO	每股收益（含非经常性项目）(FY)
QEPSINCLXO	每股收益（含非经常性项目）(MRQ)
IPCTHLD	机构持股比例
ACURLIAB	流动负债 (FY)
QCURLIAB	流动负债 (MRQ)
APRICE2BK	股价净值比 (FY)
PRICE2BK	股价净值比 (MRQ)
APR2REV	股票价格 (FY)
QPR2REV	股票价格 (MRQ)
AQUICKRATI	速动比率 (FY)
QQUICKRATI	速动比率 (MRQ)
ATOTD2EQ	总负债与股东权益比 (FY)
QTOTD2EQ	总负债与股东权益比 (MRQ)
F_LSTUPDAT	最近财务更新

历史字段

ATOT	总资产
STLD	总负债
LTLL	负债合计
SBBF	每股收益
SNCC	现金净值变化
RTLR	总收入
ETOE	营业费用总额
NINC	净收入
DDPS1	每股派息 - 普通股首次发行
QTCO	总流通普通股

FY = 财年, MRQ = 最近季度

附录 B

下载本教程的 EasyLanguage 代码示例

在本教程中使用的全部 EasyLanguage 示例与工作空间均可从网上下载。文件名为 ELOBJECTS.ZIP。可从下列链接下载：

www.tradestation.com/education/downloads/ELOBJECTS