openspan

where people and technology meet®

OpenSpan Best Practice

Threading

01 September 2013

This document provides both general advice applicable to any programming project and specific practices recommended for using threading within OpenSpan Studio.

This document includes these topics:

OpenSpan, Inc.
Suite 200
11175 Cicero Drive
Alpharetta, Georgia 30022

Phone (International) +1 (678) 527-5400
Phone (US and Canada) (877) 733-1136
Email: sales@openspan.com

Contents

UNDERSTANDING THREADING

There are four different types of threading you use in OpenSpan:

- User interface threads
- OpenSpan automations and user interface thread
- Adapter threads
- Automation threads

The User Interface Thread

When OpenSpan Runtime starts a project, the first thread it creates is the user interface thread. The user interface thread is responsible for displaying any Windows forms used in the project. Typically, the user interface thread is either painting the user interface to the screen, or waiting for user input. When the user interacts with a Windows form or its child controls, Windows sends messages describing the user activity to the user interface thread. The user interface processes hundreds of messages every second. In response to these messages the user interface thread raises Windows forms events such as *Click*, *KeyPressed* or *TextChanged*. All Windows forms events are raised *synchronously* and execute on the user interface thread.

OpenSpan Automations and User Interface Thread

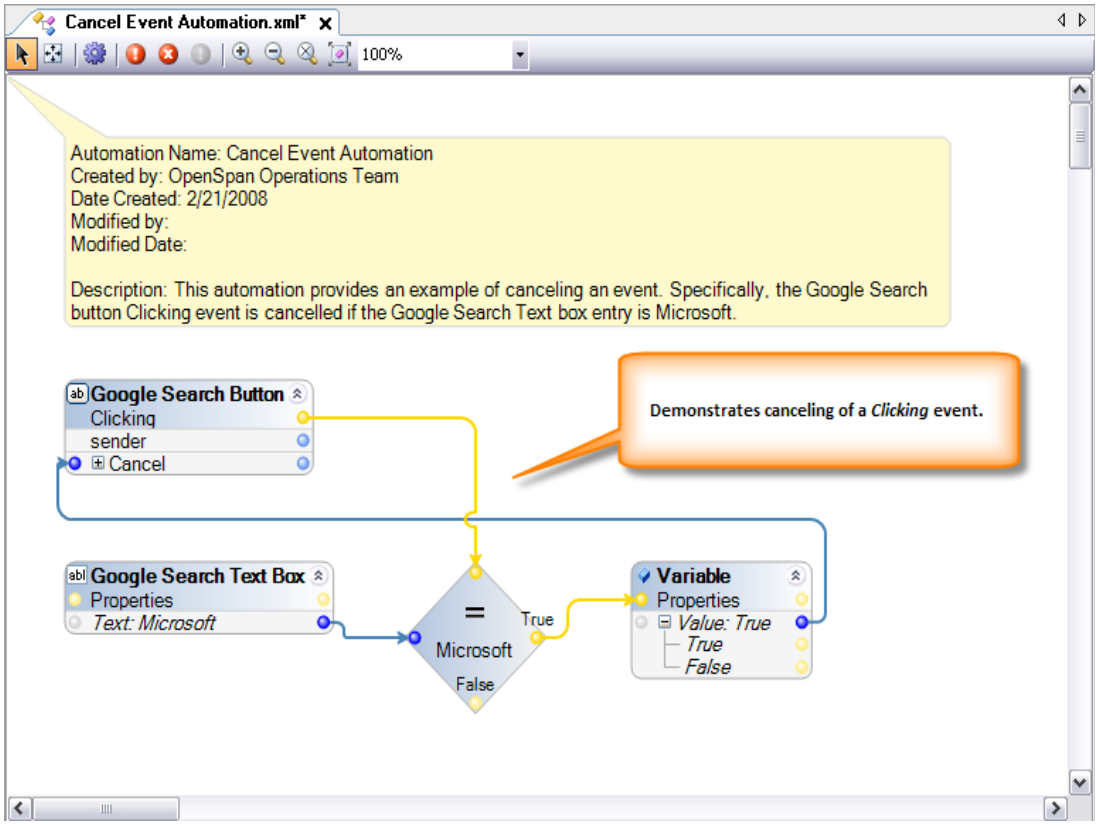
Since the user interface thread is also responsible for painting to the screen, any long-running activity that occurs on the user interface thread can block both painting and message processing. This gives the user the impression that the user interface is not responding. To avoid this, any automation that is triggered by a Windows form event should execute an *asynchronous* link to release the user interface thread before it interacts with any adapter controls or non-windows forms components, such as web services and data access.

Adapter Threads

When OpenSpan Runtime starts an adapter, the adapter creates a thread to monitor the target application. OpenSpan injects code within the target application to forward messages from the target application to the monitor thread. In response to some messages, the monitor thread initiates matching to identify adapter controls. And in response to other messages, notably user input, the monitor thread raises adapter control events such as *Click*, *KeyPressed* or *TextChanged*.

The monitor thread may raise adapter control events synchronously or asynchronously.

Synchronous events are blocking (prevents a thread from executing). When an automation is triggered by a synchronous event, the target application blocks until the automation completes or an asynchronous link executes. Most synchronous events are cancelable. Within an automation, the developer can choose to cancel the event in which case the event is not raised within the target application. Synchronous events are always suffixed with *ing,* such as *Clicking* or *DoubleClicking*.

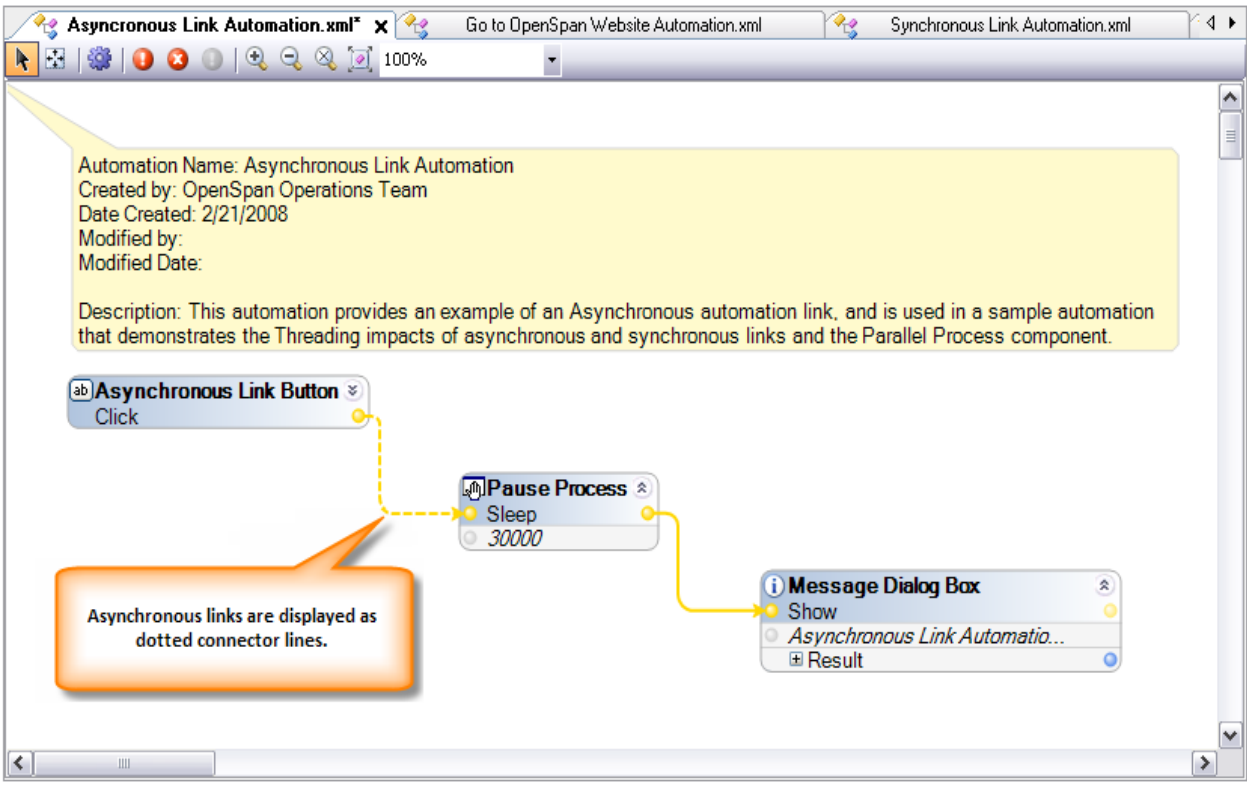


Asynchronous events are not blocking. When an automation is triggered by an asynchronous event, the target application will not block and executes normally. OpenSpan creates a new thread for each asynchronous event it raises. In most scenarios, asynchronous events are preferable to synchronous events because they are imperceptible to the target application user. Synchronous events should only be used if the automation must perform an activity *before* the event occurs such as canceling, validating, or retrieving data. Asynchronous events are always raised *after* the event occurs in the target application.

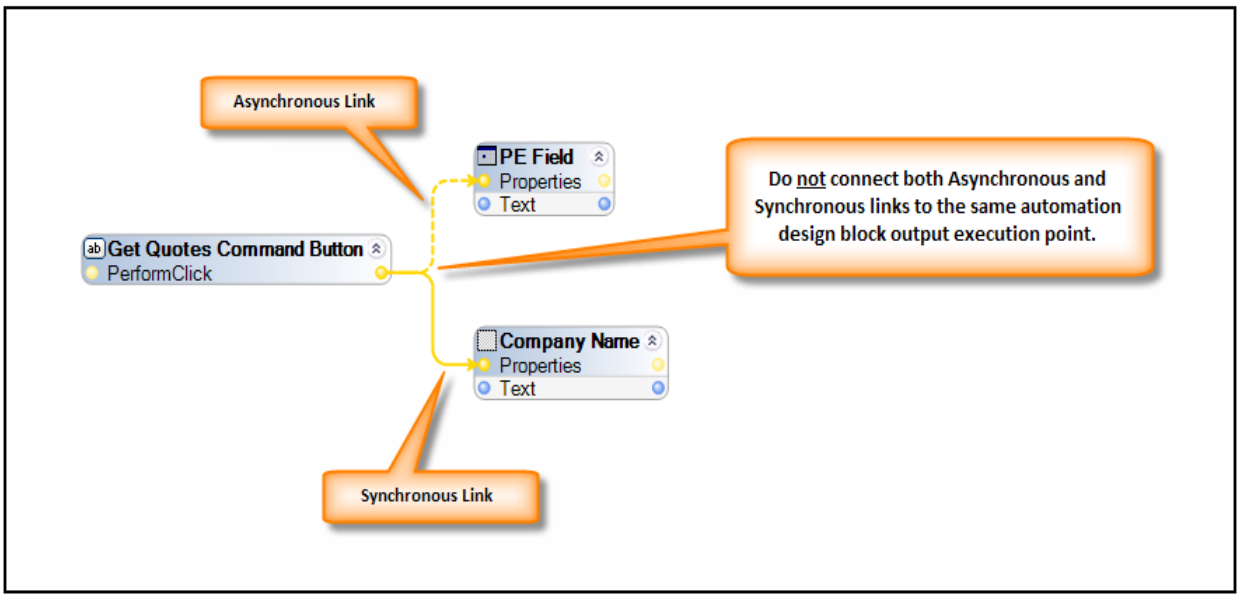
Automation Threads

Automations run on the same thread as the event that triggered them. This allows automations to respond to both synchronous and asynchronous events. OpenSpan automations can control threading behavior using asynchronous links and the Parallel Process component.

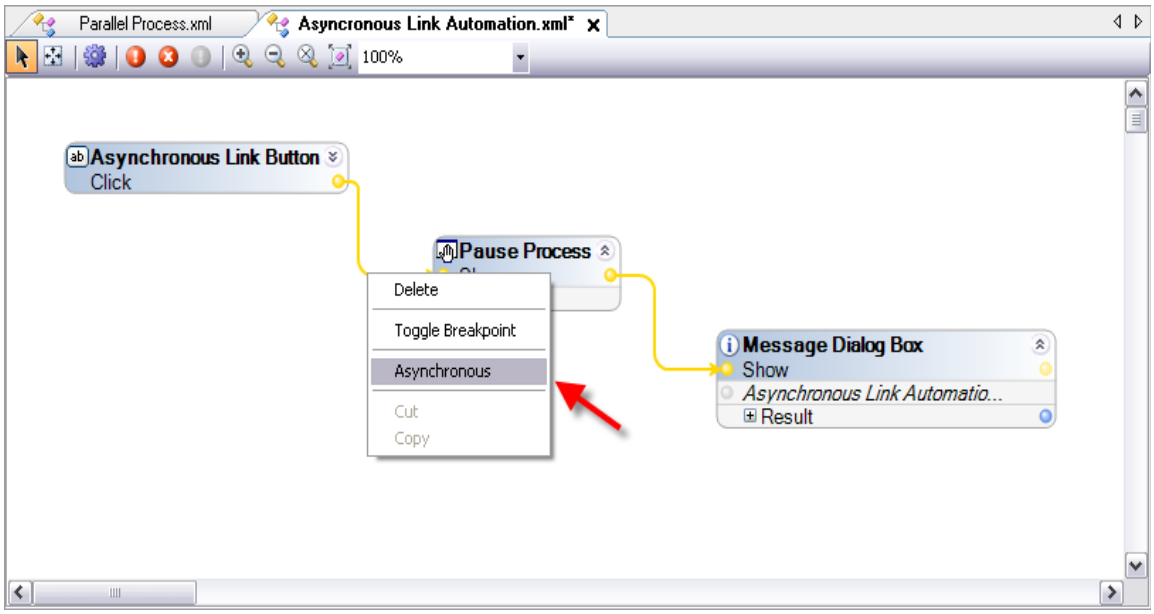
Asynchronous links are an easy way to launch a new thread within an automation. Asynchronous links are represented in automations as dashed lines. When an asynchronous link is encountered during the execution of an automation, OpenSpan creates a new thread and executes the automation design blocks after the asynchronous link on the new thread. The existing thread finishes after this point and releases any resources.



You should not connect both asynchronous and synchronous links to the same automation design block output execution point, as it may result in unpredictable behavior.



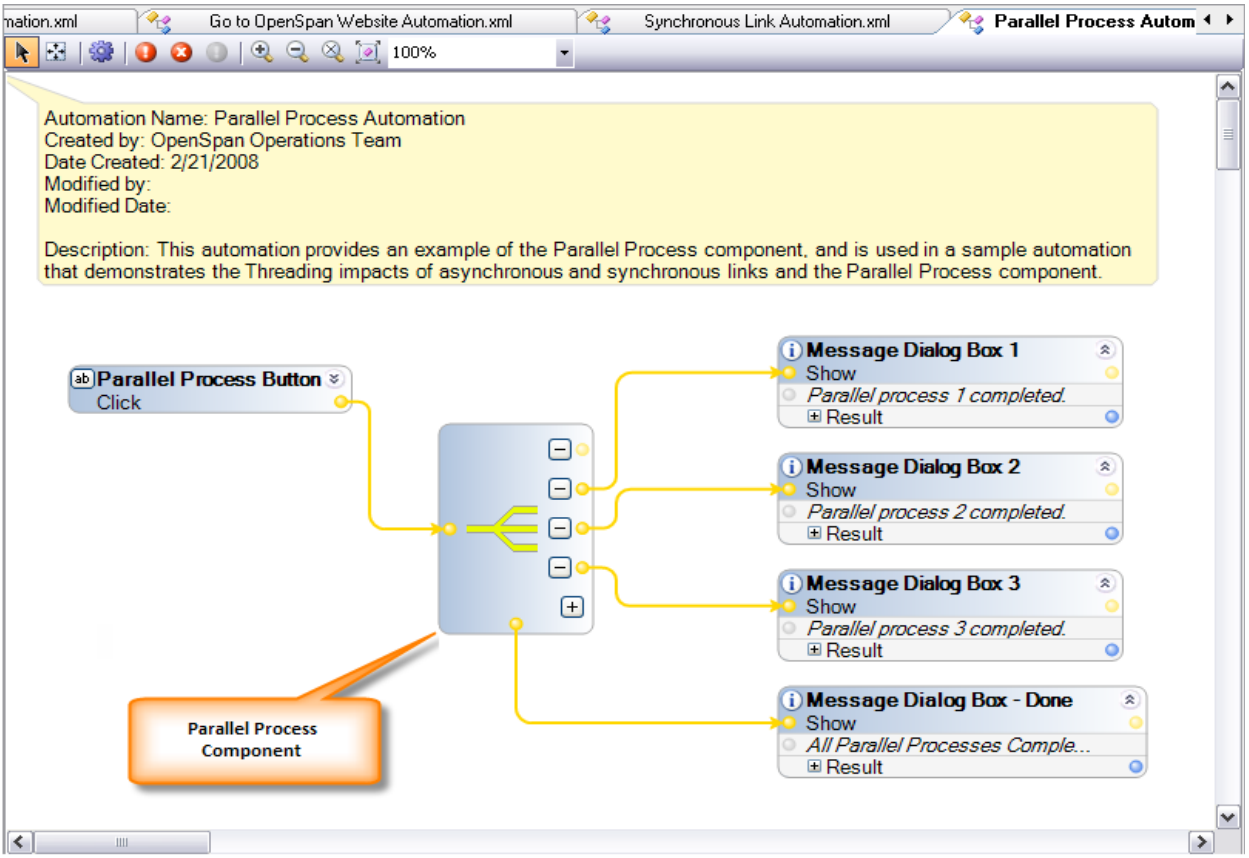
Links default as Synchronous. To toggle a link from Synchronous to Asynchronous, right-click the link and select the Asynchronous option from the context menu. The link's appearance changes from a solid to a dashed line.



Asynchronous links are particularly useful in the following situations:

- Releasing Windows forms event threads
- Releasing synchronous adapter event threads
- Launching simultaneous activities

When a parallel process is executed, all output events occur on different threads.


mation.xml
Go to OpenSpan Website Automation.xml
Synchronous Link Automation.xml
Parallel Process Autom
100%
Automation Name: Parallel Process Automation
Created by: OpenSpan Operations Team
Date Created: 2/21/2008
Modified by:
Modified Date:
Description: This automation provides an example of the Parallel Process component, and is used in a sample automation that demonstrates the Threading impacts of asynchronous and synchronous links and the Parallel Process component.
Parallel Process Button
Click
Message Dialog Box 1
Show
Parallel process 1 completed.
Result
Message Dialog Box 2
Show
Parallel process 2 completed.
Result
Message Dialog Box 3
Show
Parallel process 3 completed.
Result
Message Dialog Box - Done
Show
All Parallel Processes Comple...
Result
Parallel Process Component

Threading Best Practices

Why Consider Threads

Threads are a very powerful tool for developers but they also must be used with great care. Threads are a resource within the operating system and only a limited number are available at a time. Using too many threads simultaneously can cause a computer's CPU to *thrash.*

Thrashing occurs when there are too many threads competing for CPU time and slows the entire operating system. Client operating systems such as Windows were primarily designed to handle user input. Typically, a user only interacts with one application, a single thread at a time. OpenSpan allows developers to automate several applications simultaneously. While, this is powerful, it can also degrade performance if the interaction amongst applications is not properly orchestrated.

How Best to Use Threads

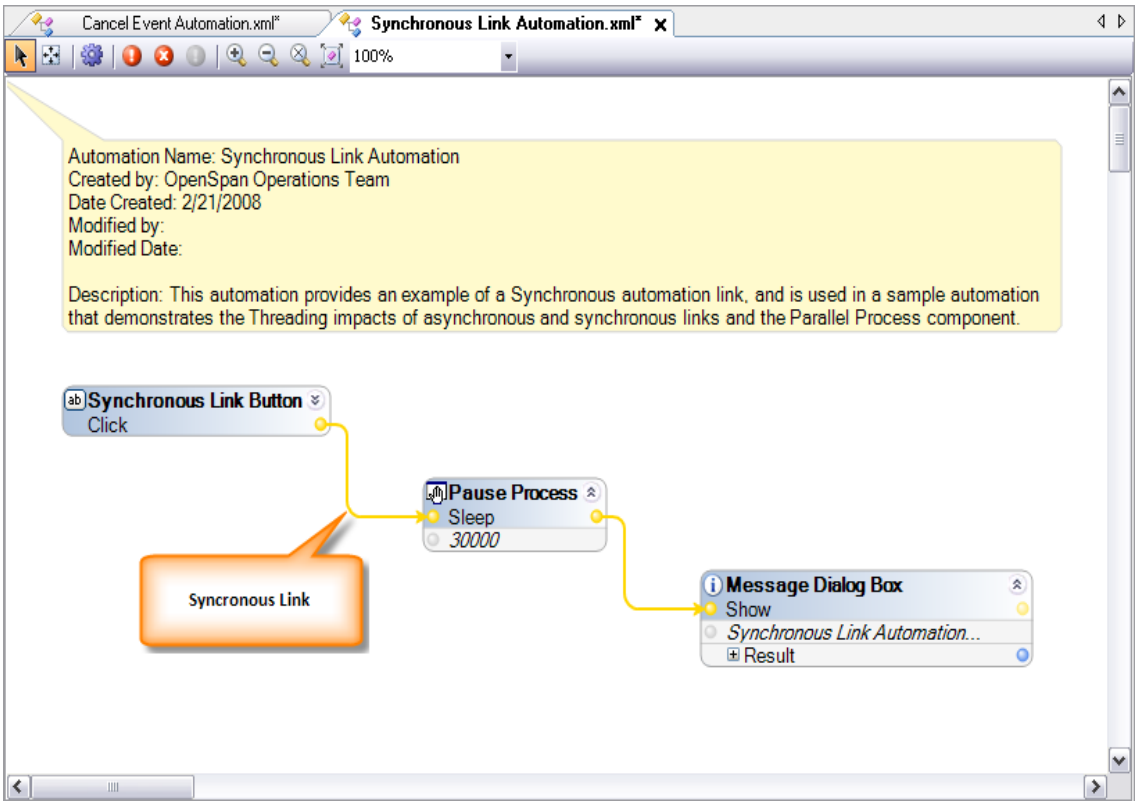
- Do not use threads to improve performance.

  One common misconception is that using threads will improve the performance of an automation. For example, if a particular script runs slowly, why not invoke the script on five different threads? Unfortunately, now that same slow script is running slowly on five different threads rather than just one, potentially degrading the performance of the entire system. If a script or component is performing slowly it is usually because of the way it was written rather than the thread it runs on.

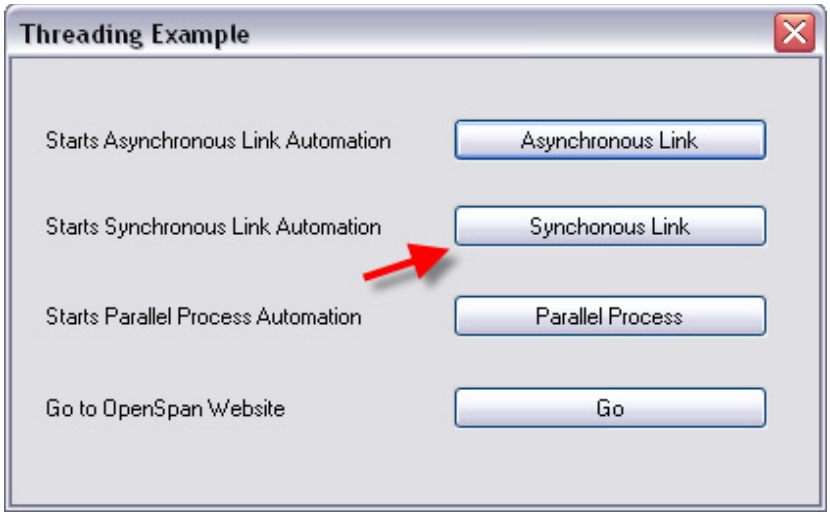
- Avoid running automations on the user interface thread.

  A blank or non-responsive Windows form that appears when running an automation is usually due to using synchronous links when responding to Windows forms events. If a synchronous event is used, the user interface is only repainted after the entire automation finishes.

  In the following example, a synchronous link connects a Windows form Click event to a Pause Process component set to halt execution for 30,000 milliseconds (30 seconds).



The Synchronous Link Button, shown in the above automation, resides on the Threading Example Windows form.



Clicking the Synchronous Link button executes the Synchronous Link Automation. Because the button is connected to the Pause Process via a synchronous link in the automation, all controls on the Threading Example window are disabled during the 30-second pause. Alternatively, if the Synchronous Link button is connected to the Pause Process via an asynchronous link, then the Threading Example window will remain enabled during the 30-second pause and the user is free to interact with other window controls.

You can test these scenarios yourself:

1. Open the Threading.zip file and unzip the OpenSpan deployment file for the Threading Example Solution.
2. Run the Threading Example Solution.
3. In the Threading Example window, click the Synchronous Link button.
4. Immediately try to click the Go button to access the OpenSpan web site. Nothing happens because the Threading Example window is disabled due to the synchronous link between the button and the Pause Process. After the 30-second pause period, an informational message displays and the Threading Example window becomes available.
5. In the Threading Example window, click the Asynchronous Link button.
6. Prior to the 30-second pause period expiring, click the Go button on the Threading Example window. In this case, clicking the Go button triggers the Go to OpenSpan Website Automation, which opens and directs Internet Explorer to the OpenSpan Website.

When using threading, it is important to remember...

- Use synchronous adapter events only when necessary and release them as soon as possible

  A target application not responding during an automation is usually due to the developer accidentally using a synchronous event and not releasing it when it is finished. There's a big difference between Clicking and Click and using the wrong event can severely impact performance. Synchronous events should only be used when the automation must perform an activity before the event occurs. Synchronous events are often used to validate or collect data before allowing a click or other action that destroys the controls to occur. Immediately after the necessary actions are performed (e.g., collecting data) an asynchronous link should be added to the automation to release the adapter thread.
- Launch applications on demand rather than all at once

  Excessive time to launch applications when starting OpenSpan is usually due to the number of applications being started. If the applications are not going to be used immediately, it is preferable to start applications when they are required by the user rather than all at once. Starting too many applications at the same time can cause a computer's CPU to *thrash* because each application competes for CPU time while starting.
- Do not use multiple threads when automating a single application

  When automating windows, web and host applications it is easy to forget that most applications were designed for user interaction rather than automation. One common mistake is to do things in an automation that a user cannot possibly do, such as interacting with two windows simultaneously within the same application. In general, developers should attempt to simulate user behavior.

More Resources

For more information about Threading in general and in Microsoft Windows operating systems in particular, see the many articles in the Microsoft® MSDN® developer program at www.msdn.microsoft.com.

OpenSpan Studio Help and Sample Solution

To find threading information while working in OpenSpan Studio, choose Help | Search, and search for the keyword *Threading.*

The OpenSpan deployment file containing the sample solution used in this document is located in the Threading.zip file. Download this file and unzip the OpenSpan deployment file for the Example Threading Solution.

Note This solution works only with OpenSpan Studio versions 4.5 and higher.