



Cognitier SimpleIPC
Out-Of-Process COM-Visible Components in
.NET
White Paper

Version 1.0.0.1

July 12, 2010

Copyright © 2010 Cognitier, Inc. All rights reserved.

Cognitier and the Cognitier logo are trademarks of Cognitier, Inc. All other company and product names referred to may be trademarks of their respective owners. This documentation is copyrighted material and is intended exclusively for use by licensed users of Cognitier software. The information in this document is subject to change without notice.

Introduction

The .NET Framework has replaced the Component Object Model (COM) as the preferred approach for writing reusable components for the Windows operating system. It is still possible, however, to author classes in .NET which are consumable from a COM client such as VBScript. The caveat to this is that the .NET component must run in-process with the COM client. .NET does not support the authoring of an out-of-process COM server.

Advantages To Running Out-Of-Process

A component running in-process with the client application will generally provide better performance. However, there are several other factors which have led developers to create a multitude of out-of-process COM servers over the years:

- **Application stability:** If an application loads an in-process COM component and that COM component generates a serious error, this can have the effect of terminating the entire application process. If a serious error occurs in an out-of-process COM component, only the component process will terminate. The application can create another instance of out-of-process COM component later, if appropriate.
- **Control of memory consumption and leaks:** If there is a memory leak in a component running in-process with an application, the memory consumption of the application may exceed the limit imposed by the operating system. If the component is running out-of-process, then memory consumed by the component does not count against the application's limit on memory consumption. Also, out-of-process components can be more easily recycled to free up memory.
- **Identity configuration:** The "dcomcnfg" application can be used to easily configure the identity under which an out-of-process COM component will run. This is useful because sometimes an application requires certain functions to be performed under an account with a different set of privileges than the main application account.
- **Limited access to shared resources:** There is typically just one running instance of the executable hosting the out-of-process COM component. Multiple client applications might interact with the COM component via that same host process. Access to shared resources on the machine can be controlled within the component host process, and thus access to shared resources may be limited across multiple client applications.

The .NET Alternative Approach - .NET Remoting

The .NET Framework provides a facility for realizing all of the benefits of out-of-process COM components. That facility is .NET Remoting. .NET Remoting offers different "channels", or communication paths, by which client

applications may communicate with .NET Remoting servers. The channel which is most similar to the basic implementation of an out-of-process COM component is the Inter-Process Communication (IPC) channel. The IPC channel allows a .NET client application to communicate with a .NET component hosted in a separate process on the same machine. .NET Remoting provides all of the advantages of out-of-process COM components, and arguably makes it easier to control the behavior of the process hosting the .NET Remoting component.

A COM client, such as VBScript, cannot directly access a .NET Remoting server. If a developer wants to write code modules in .NET to be consumed by a COM client, then he would most likely build a sort of bridging component which acts as a .NET Remoting client, but is visible as a COM component and runs in-process with the client application. He would thus distribute both the out-of-process .NET Remoting component and the in-process, COM-visible Remoting client.

There is a small learning curve to implementing .NET Remoting clients and servers. Developers must be conscious of the name of the channel over which the client and the server are communicating and must likewise designate the user groups authorized to use the channel. In short, a .NET Remoting server is not quite as readily consumable as an out-of-process COM component.

.NET Remoting With SimpleIPC

The SimpleIPC product is designed to simplify the implementation of .NET Remoting over the IPC channel. SimpleIPC provides the “plumbing”, so to speak, for the implementation such that the developer can concentrate on writing the code to be executed within the out-of-process component. All of the process communication and remote process life-cycle management tasks are accomplished via configuration, rather than coding. The advantages to using SimpleIPC to implement out-of-process components include the following:

- Install SimpleIPC once, and use it to implement any number of out-of-process components. The actual code to be executed out-of-process can be written in .NET and “plugged into” the SimpleIPC infrastructure. That code can even be scripted in JScript.NET.
- The developer has the ability to run custom code during all of the events in the remote process life-cycle (start-up, shut-down, etc.).
- Several types of clients can access the remote component using the same basic API. Clients can be .NET, COM or even Java.
- Most remote component management tasks are handled via configuration. The number of remote servers, the identity under which the servers run, and the recycle rate may be configured. SimpleIPC may be easily configured to disallow concurrent access to certain shared resources, even when multiple concurrent applications are acting as SimpleIPC clients.

SimpleIPC provides the infrastructure and the “container” for the remote component code. There is no commitment to a proprietary technology required to use SimpleIPC. The code modules plugged into SimpleIPC are written like any other .NET code module. As such, that code can be shared among other non-SimpleIPC applications and/or easily migrated at a later time.

Beyond The Local Machine

The .NET Remoting facility offers additional channels such that communication with the hosted component can occur over a network. SimpleIPC likewise offers a web services interface to allow communication over a network. In addition, there is a companion product to SimpleIPC called the Messaging Peer which allows communication through firewalls. Using the Messaging Peer, developers can host code modules in SimpleIPC on a private computer (such as a home computer) and expose those code modules to the public internet, if desired.

More Information, Samples, and Tutorials

Detailed information on SimpleIPC and the Messaging Peer are available in the documentation library (http://www.cognitier.com/All-Docs_ep_66.html). The documents available include product tutorials with step-by-step instructions and sample code.