



# **Cognitier SimpleIPC**

## **Quick Facts**

Version 1.0.0.1

April 21, 2009

Copyright © 2009 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.

## SimpleIPC Features And Benefits

SimpleIPC provides a layer of abstraction on top of certain features within the .NET Framework – the most significant being .NET Remoting over the Inter-Process Communication (IPC) channel. SimpleIPC provides your application with the ability to delegate work to another process on the same machine. Code that leaks memory or runs the risk of throwing unhandled exceptions can thus be run in a disposable container. If this delegate process encounters problems, the main application process can continue and delegate further work to a new delegate process without significant interruption.

The platform provides the following operational features and services:

***Mitigation of process crashes:*** Your long-running or user-servicing process can delegate code to be run within an IPC server. If that code produces an un-handled exception which results in a process crash, then the process that crashes will be the IPC server process, rather than your main process. Your process can detect the crash and request that the code be re-run by a different IPC server. Many IPC servers can be running simultaneously, and new IPC servers will be started based on configuration settings and the volume of client requests. As such, your application can potentially run un-interrupted amid periodic process crashes, with no need for end-users to be aware of any problems.

***Mitigation of memory leaks:*** The operating system will impose limitations on the amount of memory your long-running process can consume. If you have a piece of code which leaks memory, you may eventually exceed the process memory limitation. You can delegate memory-consumptive or memory-leaking code to be run within an IPC server. IPC servers can be configured to recycle periodically - thus releasing memory. In this way, you can reduce the amount of memory used by your long-running process and let the expendable IPC server processes perform the operations that use more memory.

***Caching, Object Pooling, and Throttling:*** The IPC servers perform the functions built into most other types of server processes. You can have multiple applications making calls into the same IPC server - for example, a web application and some VBA code in a spreadsheet. The server architecture provides the following benefits:

***Caching:*** When the server starts, or when a session starts, you can direct the IPC server to obtain certain resources (like database connections) and hold these for the life of the server or the life of the session. Caching frequently-used resources has performance advantages over acquiring and releasing resources on each request.

***Object Pooling:*** Similar to caching, you can direct the server to obtain a set of resources (one for every allowable concurrent session) on start-up, and re-use these same resources as clients obtain and release sessions throughout the life of the server. If, for example, you configure your server to allow five concurrent client sessions, your server can be directed to obtain five database connections on start-up, and re-use these same connections to service many clients.

*Throttling:* Since you can configure your application to allow a limited number of IPC servers, and you can configure your server to allow a limited number of concurrent client sessions, you can thus limit or throttle the load on a particular resource accessed within your IPC server code. If, for example, you have a third party system for which performance degrades with the volume of concurrent activity, you can designate that fewer total sessions be allowed in the application accessing that third party system, and that clients wait longer for their turn to use the application. In this way, during spikes in the number of incoming client requests, performance may slow down, but the third party system should be protected from overload.

***Dynamic Server Initiation and Resource Conservation:*** New IPC servers are started based on the volume of client requests (up to a configurable number of servers per application). There is no Windows Service that must be kept running. Likewise, if an IPC server "crashes", a replacement server will be dynamically started as clients attempt to obtain new sessions. As new client requests are received, the clients will be directed to an existing IPC Server if there is a server with the capacity to service them (based on the concurrent sessions threshold). If not, then a new IPC server will be started if the limit on the number of servers for the application has not been reached. During periods of high usage, clients will wait a configurable amount of time to obtain a session on an available server before returning a timeout error.

The platform provides the following development features:

***Scripting:*** You write the code you want your IPC servers to run in the form of "routines". A routine is really just a .NET language function that implements a specific interface. You can write these functions in any .NET language and register the assembly and class with the SimpleIPC console. The SimpleIPC console provides a rudimentary script editor which will allow you to author .NET assemblies in JScript.NET. Templates are provided to give you a head start (implementing the correct interface, etc.).

***Settings Configuration:*** The SimpleIPC console allows you to establish global variables at design time which will be accessible by your routines at runtime. Sensitive settings (like database connection strings) may be stored in encrypted format.

***Identity Configuration:*** Under some circumstances, you may want the code executed by your IPC server to be run under the same windows identity as the application which is calling into the IPC server. At other times, you may want your application to perform most of its operations under a less-privileged account, but to perform certain operations via the IPC server under an account with elevated privileges. The SimpleIPC console allows you to record a fixed account under which to run code in specified circumstances.

***Interoperability and Network Accessibility:*** Your IPC server code is written in a .NET language. However, you can invoke your IPC server routines via the .NET, COM, or Java API (Application Programming Interface). It is likewise very straightforward to make your IPC server routines accessible via web service calls from a remote machine.

***Server, Session, and Routine Events:*** At the start-up and shut-down of each server and session, and at the beginning and end of each routine invocation, you can interject additional code to be run to manipulate variables in the environment, change the parameters going into and out of routine invocations, etc. You can put

commonly-needed code into routines to be run upon these events and re-use the code among IPC server applications.

**Built-in Test Facility:** The SimpleIPC console allows you to configure and save single-routine test scenarios, and to run them under a simulated load of a configurable number of client sessions making a configurable number of calls per session.

The platform provides the following administrative features:

**Group-Based Security:** Access to various elements of the SimpleIPC infrastructure is controlled via user group membership. Different user groups may be given different levels of access.

**Application-Level Server Monitoring:** The SimpleIPC console allows you to view the status of IPC servers which are servicing a specific application. You can likewise terminate selected servers and/or start new servers.

**Instance-Level Server Monitoring:** The SimpleIPC infrastructure supports two "instances". These are two separate copies of the infrastructure for which the code executes via processes having differently-named executables. This allows you to use existing system administration tools to monitor and take action upon each separate instance. This is useful if you have two different environments (development and test, for example) running on the same machine, or if you are hosting code for two different customers on the same machine.