

Z2 Computer Solutions
Wayne L. Atchison
303-999-0701
10347 Adams Place
Thornton, Colorado 80229
www.z2cs.com

The Snippet Engine

September 5, 2011

During the years 1985 through 1992 the C++ compiler and the C++ object oriented concepts began to capture the computer software industry. It was at this time that I realized that this popular movement would ultimately take software development into the wrong direction.

It is not that the concept of class-objects is a wrong direction. I use C++ classes all of the time. Rather it is the simple fact that the C++ class-object inherently limits the software design in a number of very significant ways. The primary limitations are that the C++ class-object does not allow the software design to automatically take advantage of CPU-core threads, nor does it allow for external objects to be included as part of itself. It was clear to me back then that the popular movement would ultimately mean that software development would become overly complicated and very expensive, as multi-core networking computers became the norm. Today I no longer need to defend this vision, as today the number of failed multi-million dollar software projects are too numerous to itemize.

The Snippet Engine is designed specifically for modern networking computers having many CPU-cores. The Snippet Engine wraps the classic C++ class-object into a new kind of object. The Snippet Engine's object is a very high level object which automatically takes full advantage of massive parallel processing through the multiple CPU-cores, and which allows both local and external Snippet Engine objects to be subsets of each other.

The result is a software design which is composed of numerous asynchronous, autonomous, and self-aware "Knowledge Centers". The resulting Snippet Engine object always executes asynchronously, by whichever CPU-core is available next, and essentially has no scope limitations as each may interact directly with any other Snippet Engine object running anywhere in the world. Thus each Snippet Engine object is its own self-aware "Knowledge Center", each object knows only how to do its designed smaller piece of the whole application.

The result is also a software project which can be managed. Because each Snippet Engine object is autonomous, it may be unit-tested independently, and that testing can be done from anywhere

in the world. This means that adding more programmers to the project will actually help the project get done faster.

Explanation A:

The Snippet Engine is normal C++ code that acts like a software-wrapper around a developer's application C++ classes. The Snippet Engine wrapper elevates the developer's C++ classes into very high level objects. Each high level object is capable of asynchronous, autonomous, and self-aware execution as a CPU-core thread. The CPU-thread is the foundational level that software is executed by a computer. The more CPU-cores a computer has, the more Snippet Engine objects will be executed simultaneously in mass parallel processing.

Explanation B:

The Snippet Engine is a software-wrapper around a developer's application components, creating Snippet Engine Nodes. Each Snippet Engine Node is a very high level object, but yet it executes as a core-thread. Although executing at the lowest-level of the computer, each Node is self scheduling, has its own persistent command queue, its own persistent data store, and can directly communicate asynchronously with any other Node located on any other computer any where in the world. This means that each Node can perform any small piece of the whole application, as if it were itself its own application.

Explanation C:

The Snippet Engine allows the application developer to create asynchronous, autonomous, and self-aware Nodes of execution. Each Snippet Engine Node is a designed component, which is responsible for doing one piece of the whole application. This allows the developer to split the application into many C++ class components, which by definition will always be executed as independent core-threads. The whole task is accomplished as each Node interacts with the other asynchronous Nodes to perform its own designed duty. Since any Node may communicate with any other Node any where in the world, there is no inherent limit to a Node's designed duties, nor any limit to the application's aggregate scope.

This means that very large applications are accomplished by the mass parallel processing of its many asynchronous and autonomous objects, each doing its own small piece of the task, and all of the objects interacting to accumulate their results into the whole solution.

Explanation D:

The Snippet Engine allows the application developer to design an application into objects which know how to asynchronously and autonomously interact with the other Nodes to perform their designed duties. Nodes are created, deleted, and saved as needed.

For example: there may be a Node which knows how to interact with a Browser-User looking at a restaurant menu. So one of these Nodes are created each time a Browser-User begins to look at the restaurant's menu. If there are 100 Browser-Users simultaneously looking at the restaurant's menu, then there would be 100 of these Nodes created, each executing as asynchronous, autonomous, and self-aware core-threads. As Browser-Users "go away", their Nodes are simply deleted.

Explanation E:

The Snippet Engine allows the application developer to design an application into objects which are self-scheduling, have their own persistent data store and persistent command queue. This means that each Node is in essence self-aware. Nodes may therefore anticipate, and thereby "think ahead".

For example: there is a Node designed which knows how to interact with a Browser-User looking at a restaurant menu. There is another Node designed to know how to interact with an outside service. This outside service's external server is tracking its own delivery cars, which this restaurant has a contract for scheduling take-out deliveries. The first Node may now "think ahead". As the user picks items to look at in detail, the first Node automatically interacts with this second Node's command queue in order to calculate estimated delivery times, in anticipation of that item actually being picked for delivery.

Nodes do not have to wait for a Browser-User to click something. Nodes are C++ executions, and therefore are "always alive". Countless tasks may be performed ahead of time, in full anticipation of what the Browser-User may do next.