# DemolitionQueue

## Overview

This project contains all the code for the job queue. The job queue is responsible for the threaded execution of lengthy processes – typically processes defined by [Workers](#_Workers).

## Program.cs

This is a very simple class that contains the main method that runs the queue. It was separated out to facilitate the transition of the queue from a console application to a Windows Service. The structure was kept when reverting back to a console application.

## QueueRunner.cs

This is the meat of the queue, straight forward as it is. In an infinite loop, it checks to see if any jobs have been entered into the database. If so, it deserializes the payload, which tells the queue *which* worker object has created the job. It then starts the worker’s work on a ThreadPool and removes the ob entry from the database, going along its merry way.

Note that by default, ThreadPool restricts the amount of threads by 25. This can be changed by altering the contents of the mscoree.h file as described [here](http://msdn.microsoft.com/en-us/library/system.threading.threadpool(VS.71).aspx).

In order for the Demolition project to work, the queue must be started before any of the [workers](#_Workers) are called in the application.

# Workers

## Worker

Serializable and abstract, this mandates that all Workers can queue themselves onto the DemolitionQueue, and has a Work() method. It is serializable so that the queue does not have to check for each of the types of the workers, it can merely deserializes the xml entry in the ‘Job’ table and create the class on the fly.

## CreateInstanceWorker

This worker is responsible for: 1) Starting an Amazon EC2 instance, 2) Waiting for the instance to become available, 3) SSH all the desired applications onto the instance, and 4) Starting the applications on IIS.

## DataWorker

## ShutDownDemoWorker