NET Framework Developer's Guide

**Performance Application Pool Settings**

The performance application pool settings allow you to control the operation of the worker process. The following sections describe how to specify performance application pool settings.

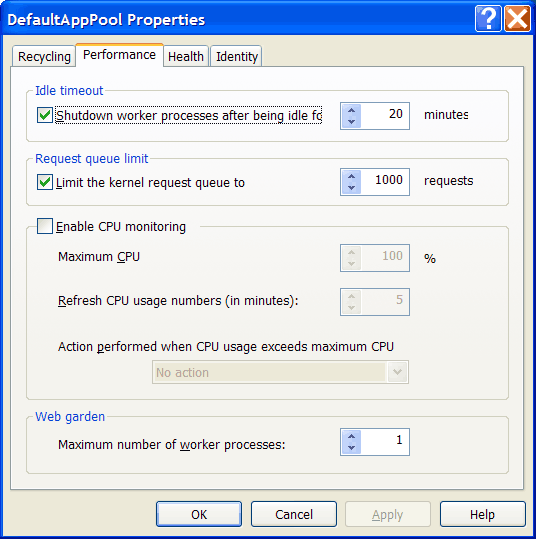
**Specifying Performance Application Pool Settings**

The performance application pool settings are specified on the **Performance** tab of an application pool's properties dialog box.

**To set the performance application pool settings**

1. Open the IIS management console and expand the local computer by clicking the plus sign.
2. Expand the Application Pools folder by clicking the plus sign.
3. Right-click the appropriate application pool and then click **Properties**. The application pool's properties dialog box appears.
4. Click the **Performance** tab, and then set the appropriate application pool settings.

**Performance tab of the application pool Properties dialog box**



**Web Garden**

**Web garden** is the equivalent application pool setting for the **webGarden** and **cpuMask** ASP.NET process model settings. To enable **Web garden**, set the **Maximum number of worker processes** to a value higher than 1. This value also determines the maximum number of processes for the application pool.

Because Web gardens enable the use of multiple processes, each process will have its own copy of application state, in-process session state, caches, and static data. Web gardens should not be used for all applications, especially if they need to maintain state. Be sure to benchmark the performance of the application before deciding whether Web garden mode is appropriate.

When using a Web garden, it is important to understand how session state and round robin works. It is also important to consider of how other application pool settings affect the application.

**Session State in a Web Garden Using Worker Process Isolation Mode**

When using session state, be aware that worker process isolation mode does not support routing requests back to a process that originated the request back. When an application is running under a Web garden with ASP session state or application state, the application needs to have its requests sent back to the originating process. When using the IIS 6.0 application pooling, make sure the application keeps a connection open so that its requests are sent back to the appropriate process. If the connection is not kept open, the request will be sent to the next available worker process servicing the Web garden.

**Round Robin in a Web Garden Using Worker Process Isolation Mode**

Round robin is a method of load balancing for the application. As each request arrives, it is automatically sent to the next sequential process. For example, if a Web garden has 4 processes, requests are sent to processes 0, 1, 2, and 3, in that order. The cycle is then repeated for additional requests.

**Application Pool Parameters in a Web Garden Using Worker Process Isolation Mode**

Some application pool settings have Web garden–specific behavior. The following table summarizes these behaviors.

|  |  |
| --- | --- |
| **Application pool setting** | **Behavior** |
| **AppPoolQueueLength** | The value of this parameter is not affected. However, requests are distributed by round robin across the worker processes servicing the Web garden. |
| **DisallowOverlappingRotation** | No behavior change in a Web garden. |
| **DisallowRotationOnConfigChange** | No behavior change in a Web garden. |
| **IdleTimeout** | Individually calculated for each process, so that each process times out independently. Depending on which routing algorithm is chosen, the number of processes will automatically configure itself to the load. |
| **LoadBalancerCapabilities** | No behavior change in a Web garden. |
| **OrphanAction** | No behavior change in a Web garden. |
| **OrphanWorkerProcess** | No behavior change in a Web garden. |
| **PeriodicRestartTime** | This parameter has a changed meaning in a Web garden. In a Web garden, this parameter specifies the time interval in which all processes are recycled. Processes take turn recycling at even intervals within the specified time amount. For example, if a Web garden has four processes and the **PeriodicRestartTime** is set to 20 hours, the first process is recycled after five hours, the second process is recycled after 10 hours, and so on.  **Note** If a process crashes, the replacement process is given a run time value of **PeriodicRestartTime**. This allows the process that crashed to recycle, along with other processes, within the specified **PeriodicRestartTime**. |
| **PeriodicRestartRequests** | This parameter has a changed meaning in a Web garden. In a Web garden, this parameter specifies that all processes are recycled after a certain number of requests. Processes take turn recycling at even intervals within the specified number of requests. For example, if a Web garden has four processed and the **PeriodicRestartRequests** is set to 40,000 requests, the first process is recycled after 10,000 request, the second process is recycled after 20,000 requests, and so on. After the initial processes are recycled, the next set of processes is assigned a **PeriodicRestartRequests** value of 40,000. |
| **PeriodicRestartSchedule** | No behavior change in a Web garden. If an administrator decides to use schedule-based recycling, all Web garden processes are recycled at the same time. |
| **PingInterval** | No behavior change in a Web garden. |
| **PingResponseTime** | No behavior change in a Web garden. |
| **PingingEnabled** | No behavior change in a Web garden. |
| **RapidFailProtection** | No behavior change in a Web garden. For example, the total failures across a Web garden are calculated and then compared over a time interval. This provides and additional level of resilience. |
| **RapidFailProtectionInterval** | No behavior change in a Web garden. For example, the total failures across a Web garden are calculated and then compared over a time interval. This provides and additional level of resilience. |
| **RapidFailProtectionMaxCrashes** | No behavior change in a Web garden. For example, the total failures across a Web garden are calculated and then compared over a time interval. This provides and additional level of resilience. |
| **SMPAffinitized** | No behavior change in a Web garden. |
| **SMPProcessorAffinityMask** | In IIS 5.0 isolation mode, you can have only as many worker processes as CPUs. In worker process isolation mode, multiple CPUs can service a single worker process. |
| **ShutdownTimeLimit** | No behavior change in a Web garden. |
| **StartupTimeLimit** | No behavior change in a Web garden. |

**Idle Timeout**

**Idle timeout** is the equivalent application pool setting for the **idleTimeout** ASP.NET process model setting. It specifies the amount of time before a worker process or application pool is shutdown due to inactivity. **Idle timeout** is enabled and set to 20 minutes by default. You can specify a different time limit by changing the value in the spin box. To disable **Idle timeout**, clear the check box.

**Request Queue Limit**

**Request queue limit** is the equivalent application pool setting for the **restartQueueLimit** ASP.NET process model setting. It specifies the maximum number of requests that are queued in the ASP.NET ISAPI while waiting for the worker process to start after an abnormal termination. **Request queue limit** is enabled and set to 1000 requests by default. You can specify a different limit to the number of requests queued by changing the value in the spin box. To disable **Request queue limit**, clear the check box.