NCache Open Source Guide

June 16, 2017

Contents

1. Introduction to NCache

Welcome to NCache version 4.6! NCache is a flexible and feature rich 100% .NET caching solution that provides high performance and scalability to handle any transaction load. NCache features and APIs are designed to cache data for applications of any size – from small to enterprise-wide global installations.

This NCache documentation contains samples, tutorials, and tools designed to help you quickly incorporate NCache into your applications. You can search for topics in the extensive class library references, overviews and step-by-step illustrations.

The audience of this guide is expected to be familiar with Java and basic database concepts. Furthermore, users are required to have NCache, a database and any Java IDE in order to successfully utilize NCache's features.

Document Conventions

The following conventions in text have been used throughout this document:

Convention	Description
bold	Specifies terms of importance for the reader.
`Monospace`	Specifies any packages and interfaces to be added into the application.
`Monospace`	Specifies inline code snippets, file, class, interface names.
`courier`	Specifies any SQL or OQL query examples within the text.
⚠	Specifies any significant step to be taken in your application.
Note:	Specifies additional and vital information for the user.

1.1 Edition Comparison

Feature	Open Source	Professional	Enterprise
Caching Topologies			
- Local Cache	✓	✓	✓
- Client Cache			✓
- Mirrored Cache			✓
- Replicated Cache	✓	✓	✓
- Partitioned Cache	✓	✓	✓
- Partition-Replica Cache (Async)			✓
- Partition-Replica Cache (Sync)			✓
- Bridge Topology			✓
Cache Clients			
- Local/Remote .NET Clients	✓	✓	✓
- Local/Remote Java Clients			✓
Big Data Processing & Analytics			
- MapReduce			✓
- Aggregator			✓
- Entry Processor			✓
Web Apps			
- ASP .NET Sessions (Basic)	✓	✓	✓
- ASP .NET Sessions (Advanced)			✓
- ASP .NET Sessions (Multi-site)			✓
- ASP .NET View State (Basic)	✓	✓	✓
- ASP .NET View State (Advanced)			✓
- ASP .NET Output Cache			✓
- Java Web Sessions			✓
Third-Party Integrations			
- Memcached Protocol Server	✓	✓	✓
- Memcached Wrapper Client (.NET)	✓	✓	✓
- Memcached Wrapper Client (Java)			✓
- NHibernate Second Level Cache (Basic)	✓	✓	✓
- NHibernate Second Level Cache (Advanced)			✓
- Entity Framework Second Level Cache Provider			✓
- NuGet Package	✓	✓	✓
- Hibernate Second Level Cache			✓
- Spring Integration			✓

Feature	Open Source	Professional	Enterprise
Cloud Platforms Supported			
- Microsoft Azure	✓	✓	✓
- Amazon Web Services	✓	✓	✓
Data Expirations			
- Absolute Expirations	✓	✓	✓
- Sliding Expirations	✓	✓	✓
- Default Expirations			✓
Cache Dependencies			
- Key Based Dependency			✓
- File Based Dependency			✓
- Custom Dependency			✓
- Multi-cache Key Dependency			✓
- Cache Sync Dependency			✓
Synchronize Cache with Database			
- SqlDependency (Events SQL Server 2005-2012)			✓
- OracleDependency (Events Oracle 10g R2 or later)			✓
- Db Dependency (Polling for OLEDB)			✓
Event Notifications			
- Item based (onUpdate/onRemove)	✓	✓	✓
- Cache level (on Add/Update/Remove/ClearCache)			✓
- Custom Events (fired by clients)			✓
- Continuous Query			✓
Object Caching Features			
- Basic operations (Get, Add, Insert, Remove)	✓	✓	✓
- Bulk operations (Get, Add, Insert, Remove)	✓	✓	✓
- Async operations (Add, Insert, Remove)			✓
- Streaming API			✓
- Lock/Unlock (exclusive locking)	✓	✓	✓
- Item Versioning (optimistic locking)			✓
- Tags			✓
- Named Tags			✓
- Groups/Subgroups			✓
- Object Query Language - OQL (Basic)	✓	✓	✓
- Object Query Language - OQL (Advanced)			✓
- LINQ (Basic)	✓	✓	✓

Feature	Open Source	Professional	Enterprise
- LINQ (Advanced)			✓
- Portable Data Types			✓
- Multiple object versions			✓
- Read-Through, Write-Through, Write-Behind			✓
- Cache Loader			✓
Evictions			
- Priority Eviction	✓	✓	✓
- Least Recently Used (LRU) Eviction			✓
- Least Frequently Used (LFU) Eviction			✓
- Do Not Evict Option			✓
Storage Options			
- Managed .NET Memory	✓	✓	✓
Cache Management			
- Visual Studio Integration			✓
- NCache Manager (GUI tool)		✓	✓
- NCache Monitor (GUI tool)			✓
- PerfMon counters	✓	✓	✓
- Command line tools	✓	✓	✓
- Cache Management .NET API			✓
- Cache Management Java API			✓
- JMX/SNMP Java Client Counters			✓
- NCache Email Alerts			✓
- Auto Restart & Join Cluster on Reboot			✓
- Multiple NIC Mapping in Cache Server & Client			✓
- Event Notifications on Cluster Changes			✓
- Cache Runs Independently		✓	✓
- LIVE Upgrade			✓
- Log Viewer			✓
General Features			
- Active Directory/LDAP Authentication			✓
- Authorization			✓
- Encryption (3DES, AES, …)			✓
- Compression			✓
- Fast Compact Serialization			✓
- Indexing on Object Attributes	✓	✓	✓
Installation Package			

Feature	Open Source	Professional	Enterprise
- Windows Installer Client & Server (.msi)	✓	✓	✓
- Java Client Installer (.msi, .tar.gz)			✓

2. Install & Configure NCache

2.1 Install NCache

The most common deployment configuration of NCache is as following:

- **Cache servers:** 2 or more dedicated cache servers (64-bit Windows 2008/2012 Server)
- **Remote clients:** Remote client is your web/app server. We recommend that you keep 4:1 or 5:1 ratio between remote clients and cache servers depending on the nature of your use. It may be higher or lower than this as well.

You should install NCache on all cache severs and remote clients.

When you install NCache by using the Windows Installer .msi file, you're asked to provide an "Installation Key". You must register with Alachisoft to obtain this key but it's free. If you don't want to register, then you must obtain the source code or binaries from https://github.com/alachisoft/ncache and use the installation scripts instead of Windows Installer to install NCache.

2.2 Cache Server Hardware Requirement

2.2.1 Windows 2008/2012 Server (64-bit)

NCache requires Windows 2008/2012 Server 64-bit platform for cache servers.

2.2.2 Adequate RAM in Cache Servers

NCache puts a minimum of 15% overhead on top of your data if you're using any indexes. However, this overhead may grow to be a lot more than 15% if you're using indexes since each index uses memory. Please keep this in mind while deciding how much memory to have in your cache servers. The total memory you need depends on how much data you plan to store in cache.

The best way to calculate your memory usage is to add some sample data in NCache once it is totally configured for your needs and then use NCache PerfMon counters to see how many items were added to the cache and CLR Memory counters to see how much memory was actually used by NCache. This should allow you to extrapolate how much you'll need.

2.2.3 1Gbit Network Interface Card in Cache Servers

Cache servers should have a dedicated 1 Gbit or faster NIC. It is recommended that you use this NIC only for NCache cluster and client communication in order to maximize its usage.

2.2.4 Dual-CPU, Quad-Core or higher

NCache is highly multi-threaded and takes full advantage of extra cores and CPUs. The most common configuration for NCache is a dual-CPU quad-core machine (meaning a total of 8 physical cores or 16 virtual cores per server).

You may need stronger processing power if you have higher transaction loads and/or larger amount of data being stored in NCache. Please note that .NET GC consumes more CPU for higher memories (e.g. 128GB or more per server).

2.2.5 Disk

NCache does not make heavy use of disk space. Hence, you don't need any extra ordinary disk space in your cache server machines.

2.3 Configure TCP Port for NCache Clients

By default, all remote clients talk to NCache servers on TCP port 9800. If your web/app servers are accessing the caching tier in a different subnet or through a firewall and you need to open specific ports to allow them to connect to cache servers, please open up this port in your firewall configuration.

If you want to change this port, then you must modify two NCache configuration files. The first file is `[InstallDirectory]\bin\service\Alachisoft.NCache.Service.exe.config` on each cache server machine and the second is `[InstallDirectory]\config\client.ncconf` on all your remote client machines.

2.3.1 Modify client.ncconf

Change the "port" below to specify a different port based on your environment preferences.

```
<configuration>
     <ncache-server connection-retries="5" retry-connection-delay="0" retry-interval="1"
                    client-request-timeout="90" connection-timeout="5" port="9800"/>
      ...
</configuration>
```

2.3.2 Modify Alachisoft.NCache.Service.exe.config

Change `NCacheServer.Port` setting in `[InstallDirectory]\bin\service\Alachisoft.NCache.Service.exe.config` files of all servers.

```
  <appSettings>
    ...
    <add key="NCacheServer.Port" value="9800" />
    ...
  </appSettings>
```

The port value in both files (in `client.ncconf` and in `Alachisoft.NCache.Service.exe.config`) must match.

NOTE: Whenever you update `Alachisoft.NCache.Service.exe.config`, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

2.4 Map Cache Server to a Network Card

When your machine has more than one network interface cards (NIC), you can specify which NIC to use for cluster-wide communication as well as client/server communication between NCache remote clients and the cache servers. Each NIC has its own ip-address. You need to specify the following in `Alachisoft.NCache.Service.exe.config`.

```
<appSettings>
    ...
    <add key="NCacheServer.BindToIP" value="20.200.20.21"/>
    ...
</appSettings>
```

NOTE: Whenever you update `Alachisoft.NCache.Service.exe.config`, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

2.5 Configure Cache Size Notification Threshold

Cache size threshold specifies the size of cache in percentage of maximum cache size at which point NCache should notify you. NCache notifies you by logging an event in Windows Event Log. You can then use third party tools to monitor Windows Event Log and be notified through a variety of mediums.

This notification helps you increase cache capacity in-time either by increasing the maximum cache size if you have more memory available on cache servers or by adding a new cache server to the cluster to add more storage capacity. Without this ability, your cache would become full and start evicting items which you may not want.

You need to make the following change in `Alachisoft.NCache.Service.exe.config`.

```
<appSettings>
    ...
    <add key=" NCacheServer.CacheSizeThreshold" value="80"/>
    ...
</appSettings>
```

NOTE: Whenever you update `Alachisoft.NCache.Service.exe.config`, you must do it on all cache servers and then you must restart NCache service afterwards. Otherwise, your changes will not take effect.

3. Cache Administration

Once NCache has been installed and you've specified all environment related settings, you're now ready to create a cache. NCache Open Source edition allows you to create a Local Cache (meaning a stand-alone cache), Partitioned Cache, and Replicated Cache.

For other caching topologies, you need to use NCache Enterprise. See more details on edition comparison between NCache Open Source and Enterprise.

3.1 Create a Cache

Create cache tool (createcache) enables the user to create a new cache on one or more server nodes using this command line utility. At the command prompt, type the following:

```
createcache [ cache-id ][ /s | /S | /T ] [ options ]
[ cache-id ] [ /s server /S cache-size ]
[ cache-id ] [ /s server /S cache-size /T path ]
[ cache-id ] [ /s server /S cache-size /t topology /c cluster-port ]
```

Parameters

Argument	Description
cache-id	Specifies the name of the cache for which cache will be registered.
/s /server	Specifies the NCache server names/IPs where Cache should be configured, separated by commas e.g. 20.200.21.11, 20.200.21.12
For Simple Case: Cache will be created by input and default configuration settings.	
/S cache-size	Specifies the size (MB) of the cache to be created.
For Advance Case: In this case all configurations related settings will be taken from specified configuration file.	
/I /inproc	Specify the isolation level for local cache.
/T path	Specifies the path of the cache source config which will be configured.
/t /topology	For topology other than local you have to give topology and cluster port. Specifies the topology in case of clustered cache. Possible values are I. local II. replicated III. partitioned
/C /cluster-port	Specifies the port of the server, at which server listens.
Option	**Description**
For Simple case:	
/y /evict-policy	Specifies the eviction policy for cache items. Cached items will be cleaned from the cache according to the specified policy if the cache reaches its limit. Possible values are i. Priority ii. LFU iii. LRU (default) NOTE: LFU and LRU are only available in Enterprise edition.
/o /ratio	Specifies the eviction ratio (Percentage) for cache items. Cached items will be cleaned from the cache according to the specified ratio if the cache reaches its limit.

	Default value is 5 (percent)
/i /interval	Specifies the time interval (seconds) after which cache cleanup is called. Default clean-interval is 15 (seconds)
/d /def-priority	Specifies the default priority in case of priority based eviction policy is selected. Possible values are i. high ii. above-normal iii. normal (default) iv. below-normal v. low
For Both cases:	
/p /port	Specifies the port on which NCache server is listening.
/G /nologo	Suppresses display of the logo banner
/?	Displays a detailed help screen

Remarks
This tool performs the following basic functions:

- Creates a local cache with minimum arguments of cache name, server and size.
- Creates a clustered cache with cache-id, size, server, topology and cluster-port.
- Creates cache with specified eviction settings.

3.1.1 Create a Local Cache

A local cache can be created through a command line tool called 'CreateCache.exe'. This tool can be found under "[InstallDirectory]\bin\tools".

The following command creates new cache named *locCache* on server 20.200.21.11 having size 512 MB and topology local for simple case. This command creates a configuration for the cache 'locCache' in "[InstallDirectory]\config\cache.config" file.

```
createcache locCache /s 20.200.20.20 /S 512 /t local
```

The following command creates a local cache named *locCache* of size 1024 MB on server 20.200.21.11, with the configuration that exists on the specified path.

```
createcache demoCache /s 20.200.20.20 /S 512 /T C:\Config.xml
```

You can always modify this config file directly if you need to change some values specified in this config. However, NCache service needs to be restarted whenever this file is manually modified. Modifying it through createcache.exe does not require you to restart NCache service.

```
<configuration>
    <cache-config config-id="0">
       <cache-settings cache-name="locCache" alias="" inproc="False" last-modified="">
         <logging enable-logs="True" trace-errors="True" trace-notices="False"
                  trace-warnings="False" trace-debug="False" log-path=""/>
```

```
        <performance-counters enable-counters="True" snmp-port="0"/>
        <cache-notifications item-remove="False" item-add="False" item-update="False"
                             cache-clear="False"/>
        <cleanup interval="15sec"/>
        <storage type="heap" cache-size="512mb"/>
        <eviction-policy enabled-evication="True" default-priority="normal" policy="priority"
                         eviction-ratio="5%"/>
        <cache-topology topology="local-cache"/>
      </cache-settings>
  </cache-config>
</configuration>
```

3.1.2 Create a Replicated Cache

Using `'creatcache.exe'`, a replicated cache named `'repCache'` can be created on one or more cache servers. The example below creates it on two cache servers (nodes) 20.200.20.20 and 20.200.20.125 of size 1024 MB using port 8701:

```
createcache repCache /s 20.200.20.20, 20.200.20.125 /S 1024 /t replicated /C 8701
```

This command creates a configuration for the cache `'repCache'` in `cache.config` file under `[InstallDir]\config` folder. You can always modify this config file if you need to change some values specified in this config. However, service needs to be restarted whenever the file is manually modified. Modifying it through `createcache.exe` does not require you to restart NCache service.

```
<configuration>
  <cache-config config-id="0">
    <cache-settings cache-name="repCache" alias="" inproc="False" last-modified="">
    <logging enable-logs="True" trace-errors="True" trace-notices="False" trace-warnings="False"
             trace-debug="False" log-path=""/>
    <performance-counters enable-counters="True" snmp-port="0"/>
    <cache-notifications item-remove="False" item-add="False" item-update="False"
                         cache-clear="False"/>
    <cleanup interval="15sec"/>
    <storage type="heap" cache-size="1024mb"/>
    <eviction-policy enabled-evication="False" default-priority="normal" policy="priority"
                     eviction-ratio="5%"/>
    <cache-topology topology="replicated">
    <cluster-settings operation-timeout="60sec" stats-repl-interval="600sec"
                      use-heart-beat="False">
      <data-replication synchronous="False"/>
      <cluster-connection-settings cluster-port="8700" port-range="1" connection-retries="2"
              connection-retry-interval="2secs" join_retry_count="24" join_retry_timeout="5"/>
    </cluster-settings>
    </cache-topology>
    </cache-settings>
    <cache-deployment>
     <servers>
      <server-node ip="20.200.20.20" active-mirror-node="False"/>
      <server-node ip="20.200.20.125" active-mirror-node="False"/>
     </servers>
    </cache-deployment>
 </cache-config>
</configuration>
```

For Replicated Cache, you should pretty much use the same default as mentioned in the above example. The only thing you'd want to change for your cache is:

> **Cluster-port:** Each replicated cache uses a unique port. And, this port needs to be the same on both cache servers that are forming this replicated cluster.

3.1.3 Create a Partitioned Cache

Similarly, we can create a partitioned cache `'partCache'` of one or more cache servers. The example below creates cache on two nodes using following command.

The following command creates new cache named demoCache on server 20.200.21.11 having size 1024 MB, topology partitioned, eviction policy priority, default priority high, eviction ratio 10%, clean interval 20, occupying cluster port 8701 for simple case.

```
createcache demoCache /s 20.200.20.20, 20.200.20.125 /S 1024 /t partitioned /y
priority /d high /o 10 /i 20 /C 8701
```

The above command produces following cache entry in the config file.

```
<configuration>
<cache-config config-id="0">
  <cache-settings cache-name="partCache" alias="" inproc="False" last-modified="">
    <logging enable-logs="True" trace-errors="True" trace-notices="False" trace-warnings="False"
             trace-debug="False" log-path=""/>
    <performance-counters enable-counters="True" snmp-port="0"/>
    <cache-notifications item-remove="False" item-add="False" item-update="False"
     cache-clear="False"/>
    <cleanup interval="20sec"/>
    <storage type="heap" cache-size="1024mb"/>
    <eviction-policy enabled-evication="False" default-priority="high" policy="priority"
                     eviction-ratio="10%"/>
    <cache-topology topology="partitioned">
      <cluster-settings operation-timeout="60sec" stats-repl-interval="60sec"
                        use-heart-beat="False">
        <data-replication synchronous="False"/>
        <cluster-connection-settings cluster-port="8701" port-range="1" connection-retries="2"
         connection-retry-interval="2secs" join_retry_count="24" join_retry_timeout="5"/>
      </cluster-settings>
    </cache-topology>
  </cache-settings>
  <cache-deployment>
    <servers>
      <server-node ip="20.200.20.20" active-mirror-node="False"/>
    <server-node ip="20.200.20.125" active-mirror-node="False"/>
    </servers>
  </cache-deployment>
</cache-config>
</configuration>
```

3.2 Start/Stop Cache

You can start and stop a cache using tools `'startcache.exe'` and `'stopcache.exe'` located under `[InstallDirectory]\bin\tools`. For example, following commands start and stop one or more named caches.

3.2.1 Start Cache

Start cache tool (startcache) enable users to start a cache using this command line utility.
At the command prompt, type the following:

```
startcache [ cache-id ] [ options ]
[ cache-id ] [ /s sever ]
[ cache-id ] [ /s server /p port ]
```

Parameters

Argument	Description
cache-id	Specifies one or more name(s) of caches separated by space registered on the server. The cache(s) with this/these name(s) is/are started on the server. Note: Space-separated cache names are to be specified in case of multiple caches.

Option	Description
/s /server-name	Specifies a server name where the NCache service is running. The default is the local machine.
/p /port	Specifies the port if the server channel is not using the default port. The default management port is 8250
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks
This tool performs the following basic functions:

- Starts the cache on current server node.
- Start the cache on the specified server.

Examples

The following command starts cache on local server.

```
startcache demoCache
```

The following command starts cache demoCache existing on server 20.200.21.11.

```
startcache demoCache /s 20.200.21.11
```

The following command uses port and starts cache demoCache existing on server 20.200.21.11.

```
startcache demoCache /s 20.200.21.11 /p 8250
```

3.2.2 Stop Cache

Stop cache tool (stopcache) enable users to stop a cache using this command line utility.

At the command prompt, type the following:

```
stopcache [ cache-id ] [ options ]
[ cache-id ] [ /s sever ]
[ cache-id ] [ /s server /g graceful-shutdown ]
```

Parameters

Argument	Description
cache-id	Specifies one or more name(s) of caches separated by space registered on the server. The cache(s) with this/these name(s) is/are stopped on the server. Note: Space-separated cache names are to be specified in case of multiple caches.

Option	Description
/s /server-name	Specifies a server name where the NCache service is running. The default is the local machine.
/p /port	Specifies the port if the server channel is not using the default port. The default management port is 8250
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Stops the cache on specified server.
- Stops a cache on specified server.
- Stops a cache gracefully.

Examples

The following command stops cache on local server.

```
stopcache demoCache
```

The following command stops cache demoCache existing on server 20.200.21.11.

```
stopcache demoCache /s 20.200.21.11
```

3.3 Add/Remove Remote Client Nodes

3.3.1 Add Remote Client Nodes

Add client node tool (addclientnode) enable users to add client nodes to the existing clustered cache(s).
At the command prompt, type the following:

```
addclientnode [ cache-id ] [ /s | /e ] [options]
[ cache-id ] [ /s server /e client-node ]
[ cache-id ] [ /s server /e client-node /p port-number ]
[ cache-id ] [ /s server /e client-node /p port-number /u /a ]
```

Parameters

Argument	Description
cache-id	Specifies one or more ids of clustered cache. Cache must exist on source server.
/s /server	Specifies a server name where the NCache service is running and a cache with the specified cache-id is registered. Client configuration is copied from this server to the client node.
/e /client-node	Specifies a client node (node to be added as client node) where the NCache service is running.

Option	Description
/p /port	Specifies a port number for communication with the NCache server. Default is 8250.
/u /update-server-config	Specifies whether to update the client-nodes sections of server node(s) of the specified cluster. The default value is true.(Useful when cluster nodes and clients are in different networks)
/a /acquire-server-mapping	Specifies whether to fetch the server mapping list from the server node(s). The default value is false. (Useful when cluster nodes and clients are in different networks).
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks
This tool performs the following basic functions:

- Adds a client node to the clustered cache existing on specified server.
- Updates the server configuration, if specified.
- Acquires server-mapping list, if specified.

Examples
The following command adds a client node 20.200.21.12 to the demoCache which exists on server 20.200.20.11.

```
addclientnode demoCache /s 20.200.21.11 /e 20.200.21.12
```

The following command adds a client node 20.200.21.12 to the demoCache which exists on server 20.200.20.11 while specifying port-number.

```
addclientnode demoCache /s 20.200.21.11 /e 20.200.21.12 /p 8250
```

The following command adds a client node 20.200.21.12 to the demoCache which exists on server 20.200.20.11 while specifying port-number and updating server config.

```
addclientnode demoCache /s 20.200.21.11 /e 20.200.21.12 /p 8250 /u
```

3.3.2 Remove Remote Client Nodes

Remove client node tool (removeclientnode) enable users to remove client nodes from the existing clustered caches.

At the command prompt, type the following:

```
removeclientnode [ cache-id ] [ /e ] [options]
[ cache-id ] [ /e client-node ]
[ cache-id ] [ /e client-node /s server-ip ]
```

Parameters

Argument	Description
cache-id	Specifies id of Clustered Cache. Cache must exist on source server.
/e /client-node	Specifies a client node where the NCache service is running.

Option	Description
/s /server	Specifies a server name where the NCache service is running and a cache with the specified cache-id is registered. Cache configuration is copied from this server to the destination server. The default is the local machine.
/p /port	Specifies a port number for communication with the NCache server.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Removes a client node from the given clustered cache.
- Removes the client node from the given clustered cache on specified server.

Examples

The following command removes client node 20.200.21.12 from the demoCache.

```
removeclientnode demoCache /e 20.200.21.12
```

The following command removes a client node 20.200.21.12 from the demoCache which exists on server 20.200.20.11.

```
removeclientnode demoCache  /e 20.200.21.12 /s 20.200.21.11
```

The following command uses port also and removes a client node 20.200.21.12 from the demoCache which exists on server 20.200.20.11.

```
removeclientnode demoCache /e 20.200.21.12 /s 20.200.21.11 /p 8250
```

3.4 Add/Remove Cache Server Nodes

You can always add a cache server to the cluster at any time using the tool `'addnode.exe'` which is located under `[InstallDirectory]\bin\tools`. You can use `'removenode.exe'` to remove a server from the cluster.

3.4.1 Add Cache Server Nodes

Add node tool (addnode) enable users to add a new server node to the existing clustered cache.

At the command prompt, type the following:

```
addnode [ cache-id ] [ /x | /N ] [ options ]
[ cache-id ] [ /x existing-server /N new-server ]
[ cache-id ] [ /x existing-server /N new-server /p port ]
```

Parameters

Argument	Description
cache-id	Specifies name of clustered cache. Cache must exist on source server.
/x /existing	Specifies a server name where the NCache service is running and a cache with the specified cache-id is registered. Cache configuration is copied from this server to the destination server.
/N /new-server	Specifies a server name where a cache with the specified cache-id needs to be registered. The cache configuration is copied from the source server name to this server.

Option	Description
/p /port	Specifies the port if the server channel is not using the default port. The default is 8251 for HTTP and 8250 for TCP channels.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Adds a new node to the existing cache on specified server.
- Enable users to have multiple server nodes in a clustered cache.

Examples

The following command adds a new node 20.200.21.12 to the cache existing on server 20.200.21.11.

```
addnode demoCache /x 20.200.21.11 /N 20.200.21.12
```

The following command adds a new node 20.200.21.12 to the cache existing on server 20.200.21.11 and port.

```
addnode demoCache /x 20.200.21.11 /N 20.200.21.12 /p 8250
```

3.4.2 Remove Cache Server Nodes

Remove node tool (removenode) enable users to remove node from existing cache.

At the command prompt, type the following:

```
removenode [cache-id ] [options]
[ cache-id ] [ /s server ]
[ cache-id ] [ /s server /g graceful-shutdown ]
```

Parameters

Argument	Description
cache-id	Specifies id of cache registered on the server. The cache with this id is unregistered on the server.

Option	Description
/s server	Specifies a server name where the NCache service is running. This server will be removed from specified cache. The default is the local machine.
/p /port	Specifies the port if the server channel is not using the default port. The default is 8251 for http and 8250 for tcp channels.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Removes node from existing cache on a server.
- Removes specified node from provided clustered cache.

Examples

The following command removes local node from demoCache.

```
removenode demoCache
```

The following command removes node 20.200.21.12 from demoCache.

```
removenode demoCache /s 20.200.21.12
```

3.5 Test the Cache Cluster

Before you should start using the cache with your application, you need to first test to make sure the cache is working properly. In order to do that you need to add some test data to the cache and then monitor to see if the data has been added or not.

Start cache on all cache servers in the cluster.

3.5.1 Verify Cluster Health

Verify the Cluster Health by using `'listcaches.exe'` command against cache server. It shows all the caches on registered on that server and for the ones that are running it shows which cache servers are currently part of the cache cluster.

```
listcaches /s 20.200.20.20 /a
```

"/s" asks you to specify name or ip-address of a cache server. "/a" displays details for all caches. For more detail on this tool, please refer to Verify Cluster Health in Cache Monitoring.

3.5.2 Add Counters to Monitor Cache

For each cache server in the cluster, add NCache counters mentioned below to Performance Monitor tool so you can watch them for activity. You can add them all to a single PerfMon tool or multiple tools. These counters show you cache activity as your application adds, updates, or fetches items in the cache. Here are the counters (see details on how to add these counters in the chapter on "Cache Monitoring"):

- **Count:** Shows how many sessions in the cache
- **Fetches/sec:** Shows you how many items are being read by your application from the cache. Remember, each Http Request results in one "Fetch" and one "Add" or "Update" call to the cache. Make sure you select your cache-id for this counter.
- **Additions/sec:** Shows you how many new items are being added per second. Make sure you select your cache-id for this counter.
- **Updates/sec:** Shows you how many existing items are being updated per second. Make sure you select your cache-id for this counter.
- **Expirations/sec:** This shows you how many items are being expired per second. Make sure you select your cache-id for this counter.

3.5.3 Method 1: Add Test Data

Add test data tool (addtestdata) enables the user to add some test data to the cache to verify if the cache is started and working properly. The items added to the cache expire after 5 minute.

At the command prompt, type the following:

```
addtestdata [ cache-id ] [ options ]
[ cache-id ] [ /c item-count ]
[ cache-id ] [ /c item-count /S item-size ]
```

Parameters

Argument	Description

cache-id	Name of the cache for which you want to use this tool.

Option	Description
/c item-count	Number of items to be added to the cache. By default 10 items are added to the cache.
/S /size	Size in bytes of each item to be added to the cache. By default, items of 1k (1024 bytes) are added to the cache.
/e /absolute-expiration	Specifies absolute expiration in seconds (default: 300, minimum: 15).
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Adds test data to the cache with expiration of 300 seconds (5 minutes).
- Adds items of specified size.

Examples

The following command adds 10 items of size 1024 bytes to demoCache with expiry of 5 minutes.

```
addtestdata demoCache
```

The following command adds 100 items of size 1024 bytes to demoCache with expiry of 5 minutes.

```
addtestdata demoCache /c 100
```

The following command adds 100 items of size 2KB to demoCache with expiry of 60 seconds.

```
addtestdata demoCache /c 100 /S 2048 /e 60
```

Using the PerfMon Counters, you can verify if the cache is working once this tool is executed. Any change in the **Count** and **Additions/sec** counters will confirm the cache cluster is working properly.

3.5.4 Method 2: Run StressTestTool

Stress test tool (stresstesttool) enables users to quickly simulate heavy transactional load on a given cache. This helps you see how NCache actually performs under stress in your own environment. Please monitor NCache performance counters in PerfMon.

At the command prompt, type the following:

```
stresstesttool[ cache-id ] [ options ]
[ cache-id ] [ /n item-count ]
[ cache-id ] [ /m item-size /t thread-count ]
[ cache-id ] [ /m item-size /t thread-count /e sliding-expiration ]
[ cache-id ] [ /m item-size /t thread-count /r reporting-interval ]
```

Parameters

Argument	Description
cache-id	Name of the cache.

Option	Description
/n item-count	How many total items you want to add. (default: infinite)
/i test-case-iterations	How many iterations within a test case (default: 20)
/d test-case-iteration-delay	How much delay (in seconds) between each test case iteration (default: 0)
/g gets-per-iteration	How many gets within one iteration of a test case (default: 1)
/u updates-per-iteration	How many updates within one iteration of a test case (default: 1)
/m item-size	Specify in bytes the size of each cache item (default: 1024)
/e sliding-expiration	Specify in seconds sliding expiration (default: 300; minimum: 15)
/t thread-count	How many client threads (default: 1; maximum: 3)
/r reporting-interval	Report after this many total iterations (default: 5000)
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Allow users to perform heavy load on cache and check performance of the cache.
- Performs add, get, update and delete operation on the cache.

Examples

The following command executes stresstesttool on demoCache.

```
stresstesttool demoCache
```

The following command executes stresstesttool on demoCache and will add items of size 2kb.

```
stresstesttool demoCache /m 2048
```

The following command executes stresstesttool on demoCache and will add items of size 2kb using two threads.

```
stresstesttool demoCache /m 2048 /t 2
```

Verify that the above mentioned counters show the right values according to the test data you added.

3.6 Clear Cache

Clear cache tool (clearcache) enables the user to clear the cache by removing all the items present in the cache. At the command prompt, type the following:

```
clearcache [cache-id ] [options]
[ cache-id ] [ /F forceclear]
```

Parameters

Argument	Description
cache-id	Name of the cache for which you want to use this tool.

Option	Description
/F /forceclear	Force the clearing of the cache. If not specified, the user is asked before clearing the

	cache.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Clears the cache, and asks user before clearing cache.
- Clears the cache forcefully, if specified.

Examples

The following command clears the demoCache.

```
clearcache demoCache
```

The following command forcefully clears the cache and does not ask the user before clearing the cache.

```
clearcache demoCache /F
```

3.7 Add/Remove Indexes for SQL Queries

NCache requires you to define indexes on all searchable attributes used in SQL queries WHERE clause. This is because, without indexing, NCache would have to traverse the entire cache in order to find items. This would be a costly operation with potential of slowing down the entire cache.

NCache provides its own indexing mechanism. First you define an index on an object attribute and this becomes part of the cache configuration. Then, when cache is started and the items are added or updated in the cache, NCache uses .NET Reflection to extract data from these items and populates the index with it. And, when items are removed from the cache, their corresponding data is also removed from the index.

Once the index is populated, then when you run SQL queries, they are executed first against the indexes to find the corresponding data and then returned to your application very quickly.

Suppose that cache contains Product object where the definition of Product is as below:

```
[Serializable]
public class Customer
{
        // Properties being defined below
        public int CustomerID { ... }
        public string FirstName { ... }
        public string LastName { ... }
        public string Address { ... }
}
```

3.7.1 Add Query Indexes

You can add query indexes on selective attributes of any type using command line tool `'addqueryindex.exe'` located under `[InstallDirectory]\bin\tools` as follows:

Add query index tool (addqueryindex) enable users to add query indexes for the objects to be added in the cache so that later on user can search items in the cache using these indexes.

At the command prompt, type the following:

```
addqueryindex [ cache-id ] [ /a | /c | /L ] [ options ]
[ cache-id ] [ /a assembly-path /c class /L attrib-list ]
[ cache-id ] [ /a assembly-path /c class /L attrib-list /s server ]
```

Parameters

Argument	Description
cache-id	Specifies the name of the cache for which query index will be configured.
/a /assembly-path	Specifies the path of the assembly which will be configured.
/c /class	Specifies the fully qualified class for query indexing.
/L /attrib-list	Specifies the attributes for query indexing ($ separated) e.g. `CustomerID$Name` ...

Option	Description
/s /server	Specifies the NCache server name/ip.
/p /port	Specifies the port on which NCache server is listening.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Indexes a class in cache.
- Enable users to perform search on indexed attributes.

Examples

The following command adds query index for the attribute CustomerID in cache.

```
addqueryindex demoCache /a C:\Data.dll /c Data.Customer /L CustomerID$FirstName
```

The following command adds query index for the attribute CustomerID in cache existing on server 20.200.21.11.

```
addqueryindex demoCache /a C:\Data.dll /c Data.Customer /L CustomerID$FirstName /s
20.200.21.11
```

The following command adds query index for the attribute CustomerID in cache existing on server 20.200.21.11 and port.

```
addqueryindex demoCache /a C:\Data.dll /c Data.Customer /L CustomerID$FirstName /s
20.200.21.11 /p 8250
```

3.7.2 Remove Query Indexes

Remove query index tool (removequeryindex) enable users to remove pre-defined query indexes for the objects to be added in the cache.

At the command prompt, type the following:

```
removequeryindex [cache-id ] [ /c ] [options]
[ cache-id ] [ /c class ]
[ cache-id ] [ /c class /L attrib-list  /s server ]
```

Parameters

Argument	Description
cache-id	Specifies the name of the cache for which query index will be removed.
/c /class	Specifies the class for removing query index.

Option	Description
/L /attrib-list	Specifies the attributes for removing from query index ($ separated) e.g. `CustomerID$Name` ...
/s /server	Specifies the NCache server name/ip.
/p /port	Specifies the port on which NCache server is listening.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks
This tool performs the following basic functions:

- Remove pre-defined query indexes for the objects.
- Remove pre-defined query indexes of the specified class on given server.

Examples
The following command removes query indexes for customer class.

```
removequeryindex democache /c Data.Customer
```

The following command removes query indexes for customer class in demoCache existing on server 20.200.21.11.

```
removequeryindex demoCache /c Data.Customer /s 20.200.21.11
```

The following command removes query indexes for some attributes of customer class in demoCache existing on server 20.200.21.11.

```
removequeryindex demoCache /c Data.Customer /L CustomerID$FirstName /s 20.200.21.11
```

You must restart the cache after defining or removing an index definition (not the data but only the index definition).

3.8 Dump Cache to Console

Dump cache tool (dumpcache) enables the user to dump keys to the console currently present in the cache.

At the command prompt, type the following:

```
dumpcache [ cache-id ] [ options ]
[ cache-id ] [ /k key-count ]
[ cache-id ] [ /F key-filter ]
[ cache-id ] [ /k key-count /F key-filter ]
```

Parameters

Argument	Description
cache-id	Specifies id of cache to be dumped.

Option	Description
/k /Key-Count	Specifies the number of keys. The default value is 1000.
/F /Key-Filter	Specifies the keys that contain this substring. By default it is empty.
/G /nologo	Suppresses display of the logo banner.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Dump the keys present in the cache to the console.
- Dump the keys filtered on the basis of key filter given.

Examples

The following command dumps the 1000 keys currently present in the cache

```
dumpcache demoCache
```

The following command dumps the 500 keys currently present in the cache

```
dumpcache demoCache  /k 500
```

The following command dumps the 500 keys currently present in the cache containing 11 as substring in keys.

```
dumpcache demoCache  /k 20 /F 11
```

3.9 Get Cache Configuration

Get cache configuration tool (getcacheconfiguration) enables the user to get the configuration for the specified cache.

At the command prompt, type the following:

```
getcacheconfiguration [ cache-id ][ /s ] [ options ]
[ cache-id ] [ /s server /T path ]
[ cache-id ] [ /s server /p port ]
```

Parameters

Argument	Description
cache-id	Specifies the name of the cache for which cache configuration will be generated.
/s /server	Specifies the NCache server name/ip.

Option	Description
/T /path	Specifies the path where config will be generated.
/p /port	Specifies the port on which NCache server is listening.
/G /nologo	Suppresses display of the logo banner
/?	Displays a detailed help screen.

Remarks
This tool performs the following basic functions:

- Generate the config file for cache registered on specified server.
- Generate the config file for cache on specified path.

Examples
The following command generates the configuration file for the demoCache.

```
getcacheconfiguration demoCache /s 20.200.21.11
```

The following command generates the configuration file for the demoCache on specified path

```
getcacheconfiguration demoCache /s 20.200.21.11 /T C:\Desktop\config.xml
```

The following command uses port also to generate the configuration file for the demoCache on specified path

```
getcacheconfiguration demoCache /s 20.200.21.11 /T C:\Desktop\config.xml /p 8250
```

3.10 Get Cache Count

Get cache count (getcachecount) enables the user to get the total count of items present in the cache.

At the command prompt, type the following:

```
getcachecount [cache-id ] [options]
[ cache-id ] [ /s server-ip ]
[ cache-id ] [ /s server-ip /p port ]
```

Parameters

Argument	Description
cache-id	Name of the cache for which you want to use this tool.

Option	Description
/s /server	Specifies a server name where the NCache service is running and a cache with the specified cache-id is registered. The default is the local machine.
/p port	Specifies the port if the server channel is not using the default port. The default port is 9800.
/G /nologo	Suppresses display of the logo banner.

/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Gets the total number of items currently stored in cache.
- Gets the count of cache registered on specified server.

Examples

The following command displays total numbers of items currently in cache.

```
getcachecount demoCache
```

The following command displays total numbers of items currently in demoCache exist on server 20.200.21.11.

```
getcachecount demoCache /s 20.200.21.11
```

The following command uses port also and displays total numbers of items currently in demoCache exist on server 20.200.21.11.

```
getcachecount demoCache /s 20.200.21.11 /p 8250
```

3.11 Verify NCache License

Verify license tool (verifylicense) enables the user to verify the NCache License. For registered version it will display the registration details. In evaluation mode it will display the remaining day if evaluation is still valid else give the expiration message.

At the command prompt, type the following:

```
verifylicense [ options ]
[ /G /nologo ]
```

Parameters

Option	Description
/G /nologo	Suppresses the startup banner and copyright message.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- Shows the registration details
- Shows evaluation period
- Gives expiry information

Examples

The following command displays registration details.

```
verifylicense
```

The following command displays registration details without displaying logo banner.

```
verifylicense /G
```

The output is similar to this:

```
Alachisoft (R) NCache Utility Verify License. Version 4.6 SP3
Copyright (C) Alachisoft 2017. All rights reserved.

This product is registered to
User    : John Smith
Email   : john@alachisoft.com
Company : Alachisoft

Edition Installed: NCache 4.6 OpenSource Edition.
```

4. Cache Monitoring

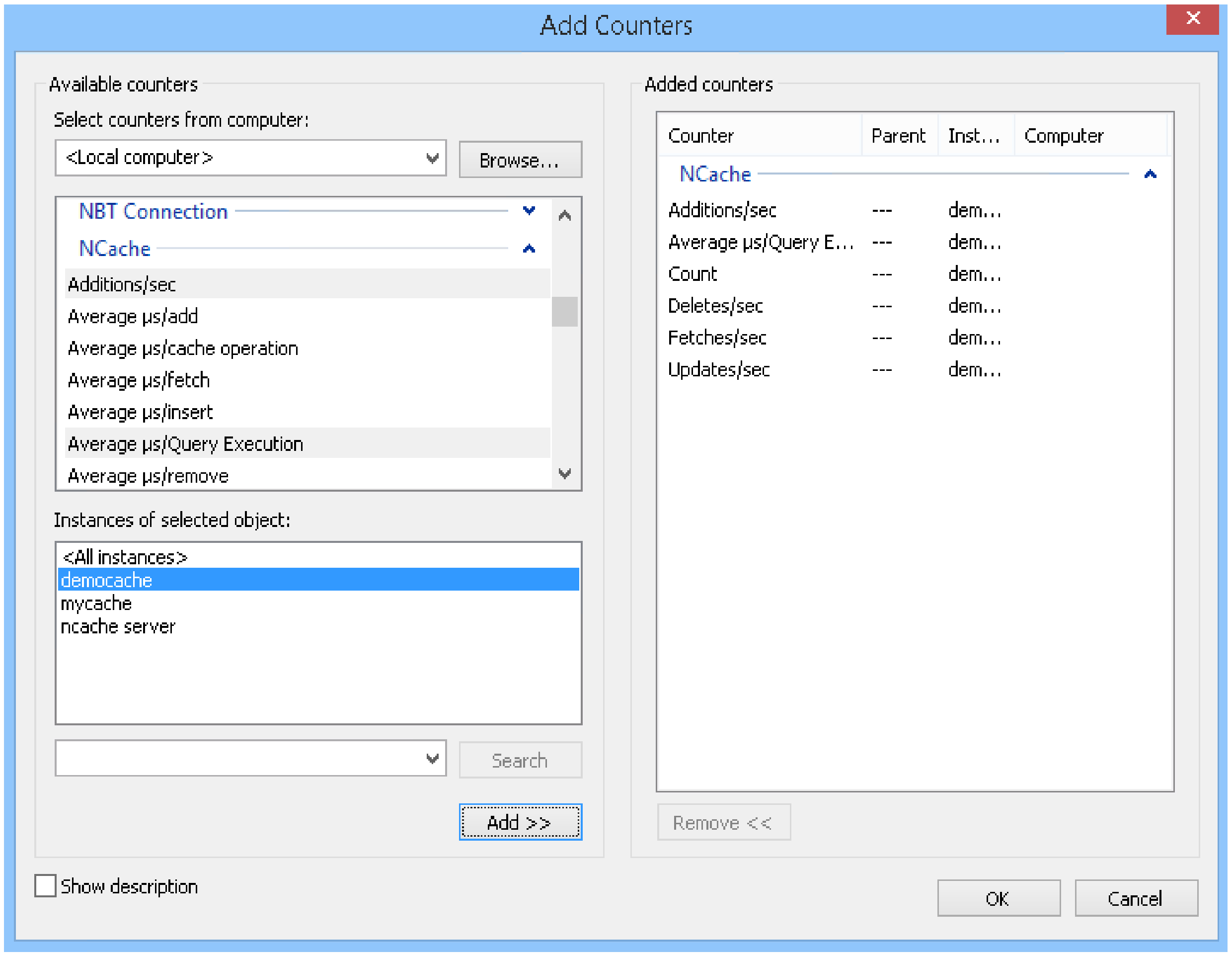
4.1 Performance Monitor

NCache publishes its performance counters through PerfMon. You can monitor NCache counters for a specific cache on any Windows machine using Windows PerfMon tool. To know more about NCache counters please read the NCache Administrator Guide.

4.1.1 Monitoring Cache Server Counters using PerfMon Tool

NCache publishes cache server counters in PerfMon under category NCache. This category has all counters related to the cache server. Follow the steps given below to monitor the NCache counters through PerfMon tool:

- Press **WINDOWS + R** keys on your keyboard **OR** click on the Windows Start menu, type **PerfMon** and press **ENTER**.
- PerfMon tool opens up; Click on the **Performance Monitor** under **Monitoring Tools**.
- Click on the cross button (in red color) to remove the default counter which is already added in it. And then click on the plus (+) button (in green color), it opens the Add Counters dialogue.
- Using the vertical slider of available counters list box, scroll upward to find **NCache** category.
- Click on the down arrow head (icon) to expand the NCache category. All of its counters listed under it. Select the required counters from this list.
- All of the current running caches (and replicas of caches) appears inside of **Instances of selected objects** list box. Select the required instance or simply click on the **<All instances>**, and click on the **Add >>** button. All of the selected counters (selected in previous step 5) for all the selected instance of caches appears in **Added Counters** list box (exist on the right side).


Add Counters

Available counters

Select counters from computer:

<Local computer>

Browse...

NBT Connection

NCache

Additions/sec
Average µs/add
Average µs/cache operation
Average µs/fetch
Average µs/insert
Average µs/Query Execution
Average µs/remove

Instances of selected object:

<All instances>
democache
mycache
ncache server

Search

Add >>

Added counters

Counter	Parent	Inst...	Computer
NCache			
Additions/sec	---	dem...	
Average µs/Query E...	---	dem...	
Count	---	dem...	
Deletes/sec	---	dem...	
Fetches/sec	---	dem...	
Updates/sec	---	dem...	

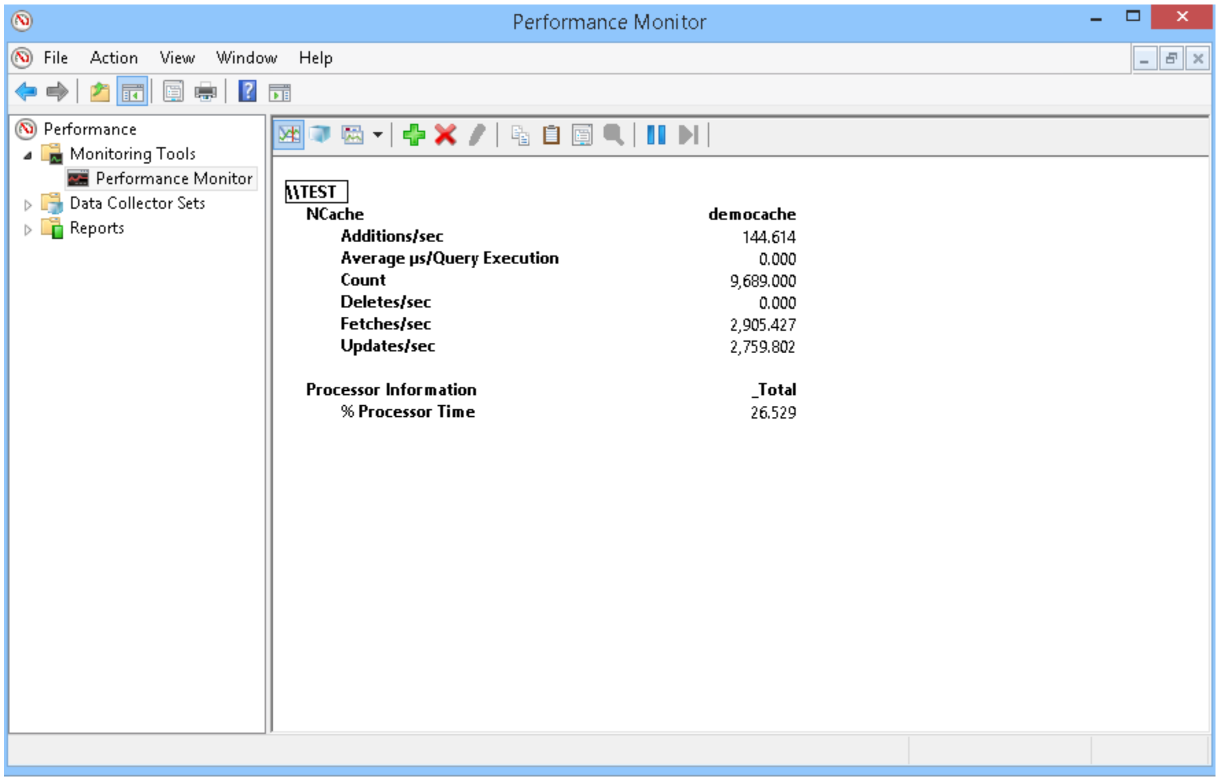
Remove <<

Show description

OK

Cancel

- Click on the OK button, available at the bottom right of this dialogue. All of the selected counters appears in PerfMon tools like this:

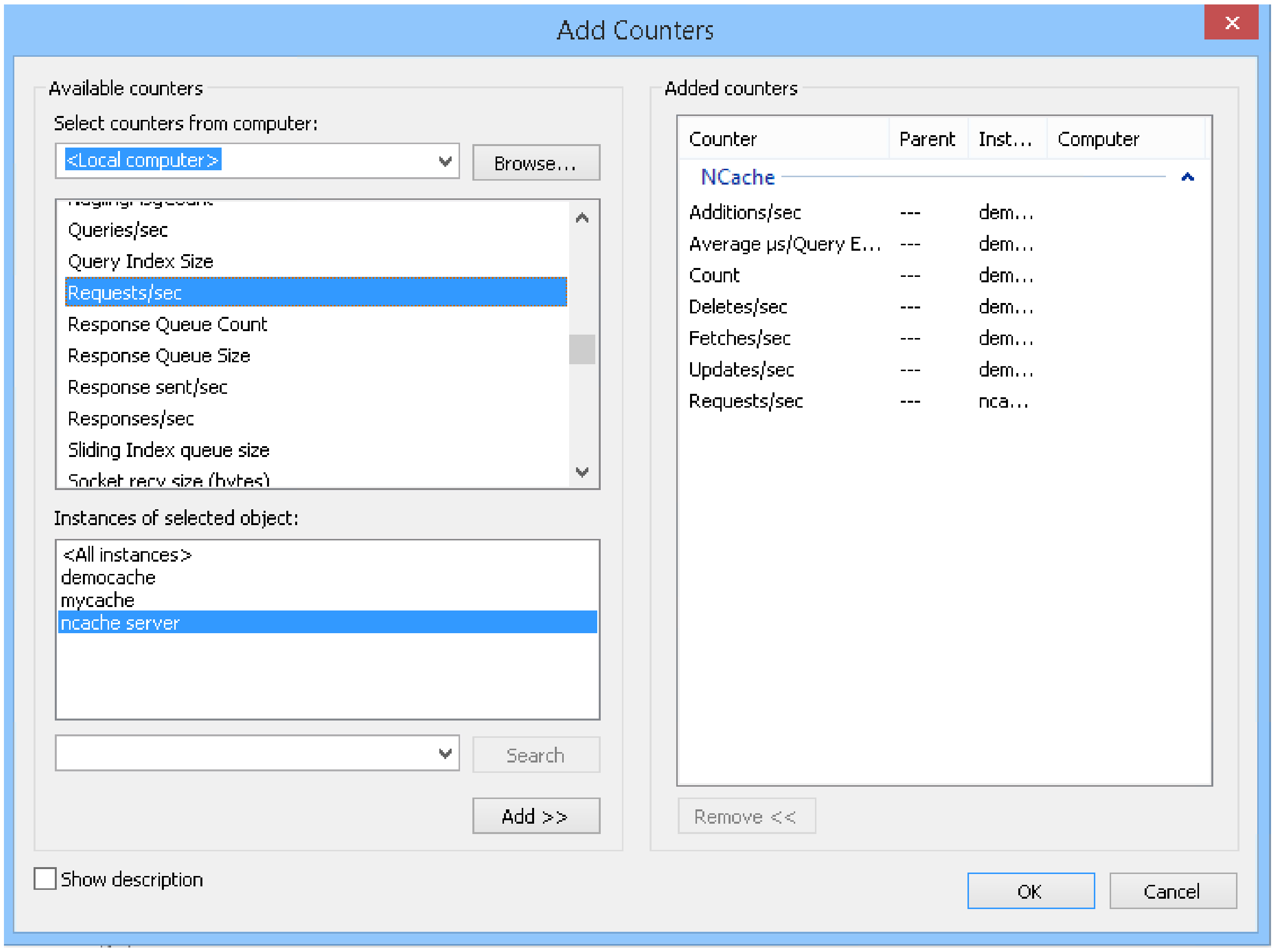


```
\\TEST
  NCache                                   democache
      Additions/sec                          144.614
      Average µs/Query Execution               0.000
      Count                                9,689.000
      Deletes/sec                              0.000
      Fetches/sec                          2,905.427
      Updates/sec                          2,759.802

  Processor Information                       _Total
      % Processor Time                        26.529
```

4.1.2 Monitoring NCache Server Counters using PerfMon Tool

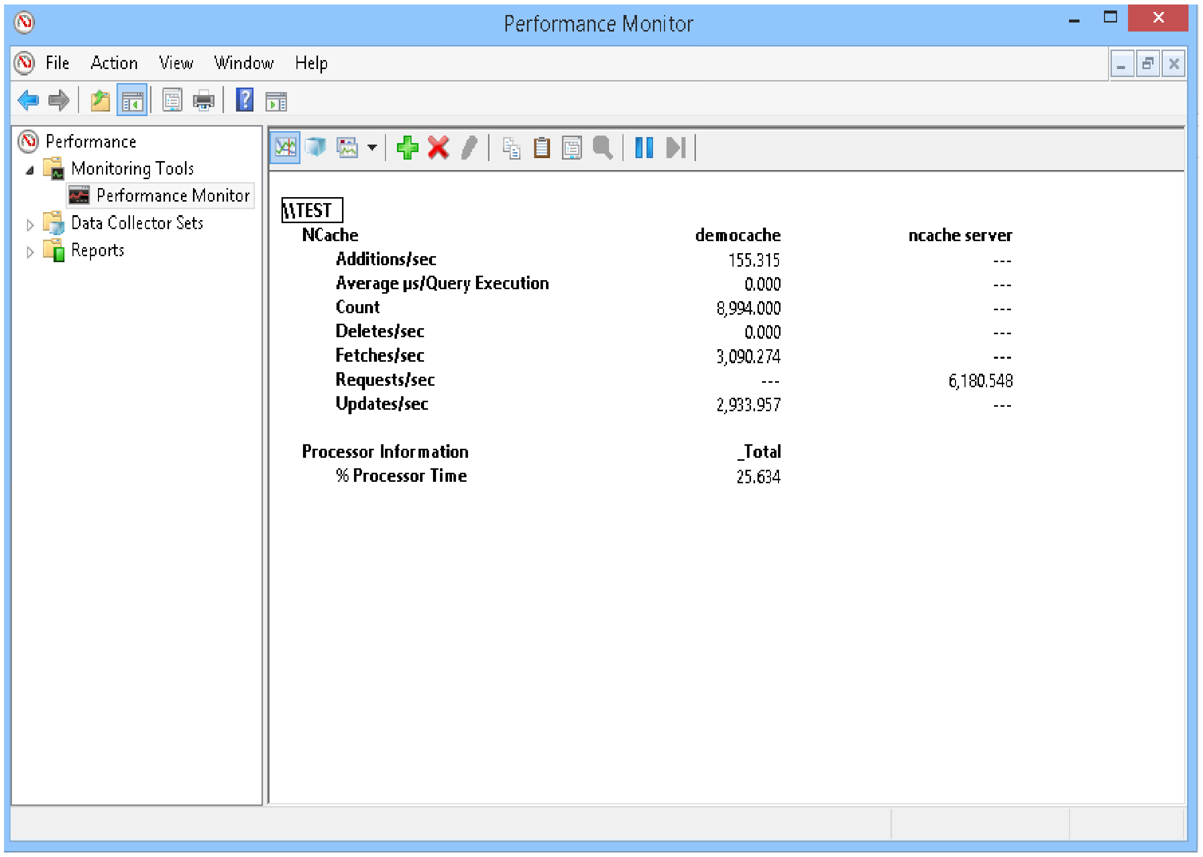
NCache publishes server counters in PerfMon under category NCache. This category has all counters related to the cache server. Follow the steps given below to monitor the NCache counters through PerfMon tool:

- Press **WINDOWS + R** keys on your keyboard OR click on the windows start menu and then type **PerfMon** and press **ENTER** key.
- PerfMon tool opens up; Click on the **Performance Monitor** under **Monitoring Tools**.
- Click on the cross button (in red color) to remove the default counter which is already added in it. And then click on the plus (+) button (in green color), it opens the Add Counters dialogue
- Using the vertical slider of available counters list box, scroll upward to find **NCache** category.
- Click on the down arrow head (icon) to expand the NCache category. All of its counters listed under it. Select the required counters from this list.
- All of the current running caches (and replicas of caches) appears inside of **Instances of selected objects** list box. Select the required instance or simply click on the **<All instances>**, and click on the **Add >>** button. All of the selected counters (selected in previous step 5) for all the selected instance of caches appears in **Added Counters** list box (exist on the right side).



Counter	Parent	Inst...	Computer
NCache			
Additions/sec	---	dem...	
Average µs/Query E...	---	dem...	
Count	---	dem...	
Deletes/sec	---	dem...	
Fetches/sec	---	dem...	
Updates/sec	---	dem...	
Requests/sec	---	nca...	

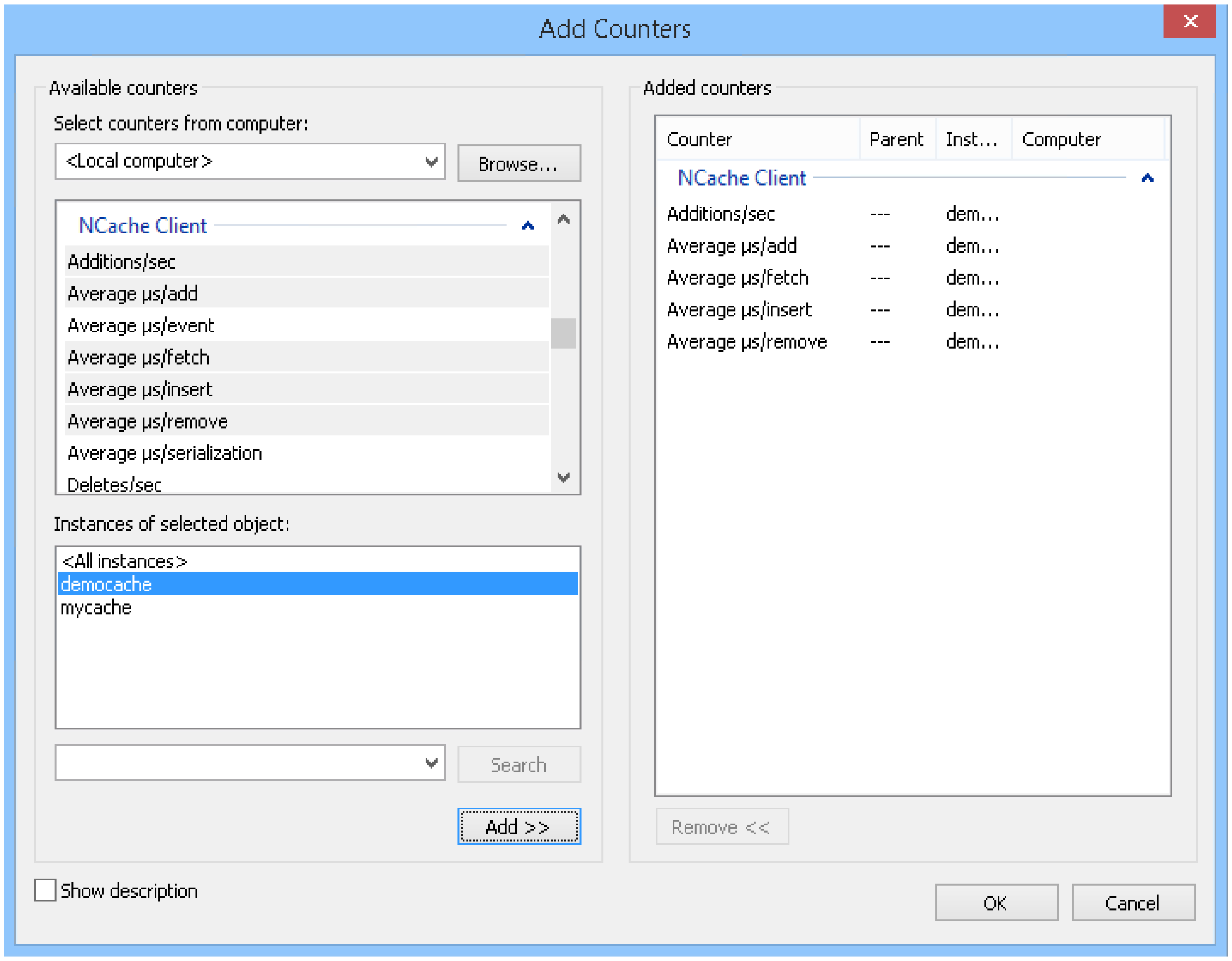
- Click on the OK button, available at the bottom right of this dialogue. All of the selected counters appears in PerfMon tools like this:



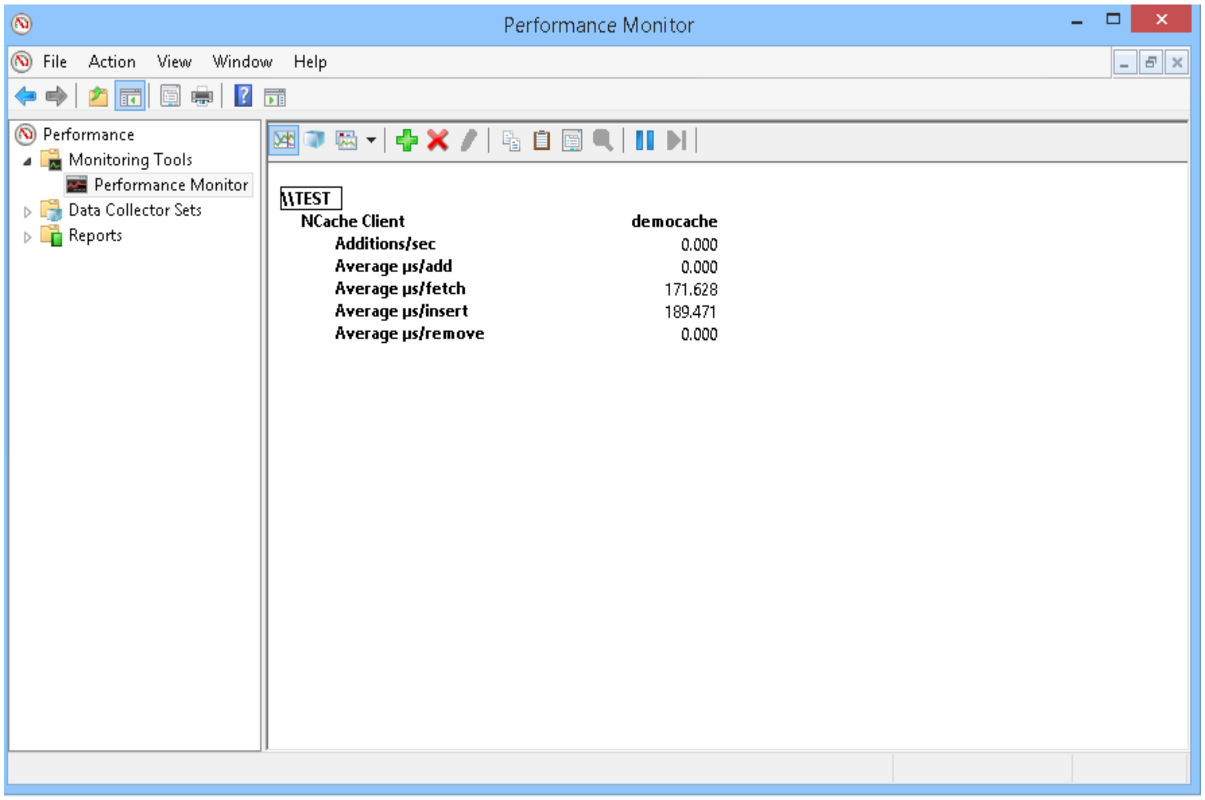
```
\\TEST
  NCache                               democache        ncache server
      Additions/sec                      155.315              ---
      Average µs/Query Execution           0.000              ---
      Count                            8,994.000              ---
      Deletes/sec                          0.000              ---
      Fetches/sec                      3,090.274              ---
      Requests/sec                           ---        6,180.548
      Updates/sec                      2,933.957              ---

  Processor Information                   _Total
      % Processor Time                    25.634
```

4.1.3 Monitoring cache client counters using PerfMon tool

- NCache publish cache client counters in PerfMon under category **NCache Client**. This category has all counters related to the cache client. Follow the below given steps to monitor the NCache client side counters through PerfMon tool:
- Press **WINDOWS + R** keys on your keyboard OR click on the windows start menu and then type **PerfMon** and press **ENTER** key.
- PerfMon tool opens up; Click on the **Performance Monitor** under **Monitoring Tools**.
- Click on the cross button (in red color) to remove the default counter which is already added in it. And then click on the plus (+) button (in green color), it opens the Add Counters dialogue
- Using the vertical slider of available counters list box, scroll upward to find **NCache client** category. Select the required counters from this list.
- Click on the down arrow head (icon) to expand the **NCache Client** category. All of its counters listed under it. Select the required counters from this list.



- All of the current running caches for which clients are running appear inside of **Instances of selected objects** list box. Select the required instance or simply click on the **<All instances>**, and click on the **Add >>** button. All of the selected counters (selected in previous step 5) for all the selected instance of caches appears in **Added Counters** list box (exist on the right side).
- Click on the **OK** button, available at the bottom right of this dialogue. All of the selected counters appears in PerfMon tools like this:



```
\\TEST
  NCache Client                         democache
      Additions/sec                         0.000
      Average µs/add                        0.000
      Average µs/fetch                    171.628
      Average µs/insert                   189.471
      Average µs/remove                     0.000
```

4.2 View Cluster Health

Cluster health can be verified using `'listcaches.exe'` tool located under `[InstallDirectory]\bin\tools`.

List caches tool (listcaches) enables the user to list the total caches registered on the server and their status. At the command prompt, type the following:

```
listcaches [ options ]
[ /a detail ]
[ /a detail /p port ]
[ /a detail /s server ]
```

Parameters

Option	Description
/a /detail	Displays detailed information about the cache(s) registered on the server.
/s /server	Specifies a server name where the NCache service is running. The default is the local machine.
/p port	Specifies the port if the server channel is not using the default port. The default port for the channel is 8250.
/G /nologo	Suppresses the startup banner and copyright message.
/?	Displays a detailed help screen.

Remarks

This tool performs the following basic functions:

- List all the caches registered on the server.
- Show the status of the caches individually if they are running or not.

- Displays detailed information about the cache(s).

Examples

The following command displays total numbers of caches currently registered on the server.

```
listcaches
```

The following command displays total numbers of caches currently registered on the server with detailed information.

```
listcaches /a
```

The following command displays total numbers of caches currently registered on the provided server with detailed information.

```
listcaches /a /s 20.200.21.11
```

This command generates an output similar to the following.

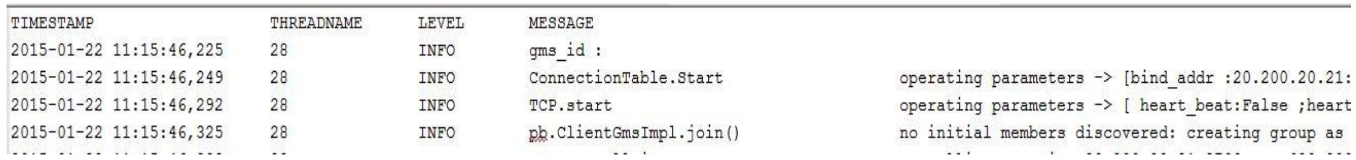
```
Listing registered caches on server 20.200.21.11:8250

Cache-ID:       repCache
Scheme:         replicated-server
Status:         Running
Cluster size:   2
                20.200.21.11:8710
                20.200.20.20:8710
UpTime:         1/13/2017 12:41:23 PM
Capacity:       1024 MB
Count:          1000000

Cache-Name:     locCache
Scheme:         Local
Status:         Running
Process-ID:     4668
UpTime:         6/14/2017 3:01:44 PM
Capacity:       512 MB
Count:          15000
```

The list of cache servers shown above indicates that these cache servers are successfully part of the cluster. Any cache server that has failed to join the cluster will not show up here. Additionally, if you have a partially connected cluster (meaning some nodes have formed one cluster and others have formed a second one), you'll only see a subset of cache servers in the above list.

So, this command is very useful in quickly seeing whether all the desired servers are part of the cluster or not.

4.3 Server Log Files

Server logs for each started cache are created under the `[InstallDirectory]\log-files` folder. These logs are very helpful in finding out the possible causes of any failures or undesired behaviors in cache. Log file name is created in following format *CacheName_timeStamp_CacheServerIP*. All Cluster level information/error will be logged in server logs.



```
TIMESTAMP                  THREADNAME    LEVEL      MESSAGE
2015-01-22 11:15:46,225    28            INFO       gms_id :
2015-01-22 11:15:46,249    28            INFO       ConnectionTable.Start                operating parameters -> [bind_addr :20.200.20.21:
2015-01-22 11:15:46,292    28            INFO       TCP.start                            operating parameters -> [ heart_beat:False ;heart
2015-01-22 11:15:46,325    28            INFO       pb.ClientGmsImpl.join()              no initial members discovered: creating group as
```

You don't have to enable server logging. It is done automatically.

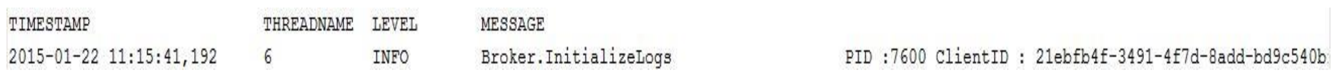
4.4 Client Log Files

Client log files are generated in `[InstallDirectory]\log-files\ClientLogs` folder. By default, client logs are not generated. You can configure to generate logs files in `client.ncconf` file located inside `[InstallDirectory]\config` folder.

You can set "enable-client-logs" flag to true in order to generate client logs. Following is an example configuration setting.

```
<cache id="mycache" load-balance="True" enable-client-logs="True" log-level="error">
  <server name="20.200.20.20"/>
</cache>
```

Client logs file name is created in following format *ClientLogs.CacheName.PID.TimeStamp_NodeAddress.* These logs are very helpful in finding out possible cause of failure.



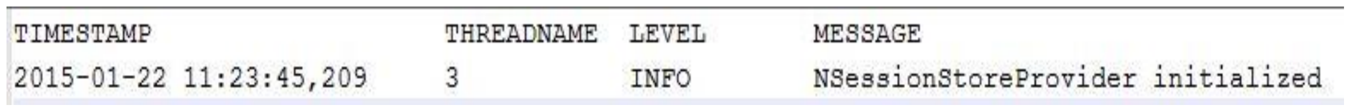
```
TIMESTAMP                THREADNAME  LEVEL     MESSAGE
2015-01-22 11:15:41,192  6           INFO      Broker.InitializeLogs               PID :7600 ClientID : 21ebfb4f-3491-4f7d-8add-bd9c540b
```

4.5 ASP.NET Session State Logs

ASP.Net log files are generated in `[InstallDirectory]\log-files\SessionStoreProvider` folder. You have to set the enableLogs attribute true in web.config to generate these log files.

```
<providers>
       <add name="NCacheSessionProvider"
       type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider" exceptionsEnabled="true"
       enableSessionLocking="true" sessionAppId="NCacheTest" useInProc="false" enableLogs="true"
       cacheName="mypartitionedcache" AsyncSession="false" />
</providers>
```

ASP.Net Session state logs file name is created in following format *CacheName.TimeStamp_NodeIPAddress.*



```
TIMESTAMP                  THREADNAME  LEVEL      MESSAGE
2015-01-22 11:23:45,209    3           INFO       NSessionStoreProvider initialized
```

4.6 Client API Usage Log

NCache provides feature of "Client Side API logging", which enables NCache client application to log all NCache API calls to file. This also logs parameters provided in API call along with the other information like size of object etc.

This logging information is very valuable when debugging a problem related to NCache because even if you didn't develop the application that is using NCache, you can quickly see which NCache calls are being made. Then, you can try to reproduce the issue by developing a separate test program that uses the same API calls in a more controlled manner.

API calls are logged in iterations of equal length, where all calls in one iteration are logged to one log file. Iterations may also have delay between them. API calls in delay interval are not logged. Total number of iterations, length of each iteration and delay between iterations can be configured.

Moreover, to avoid delay caused by logging, in memory logging is performed in each call, where as a separate logger thread writes these in memory logs to file. Interval after which logger thread will write in memory logs to file can also be configured.

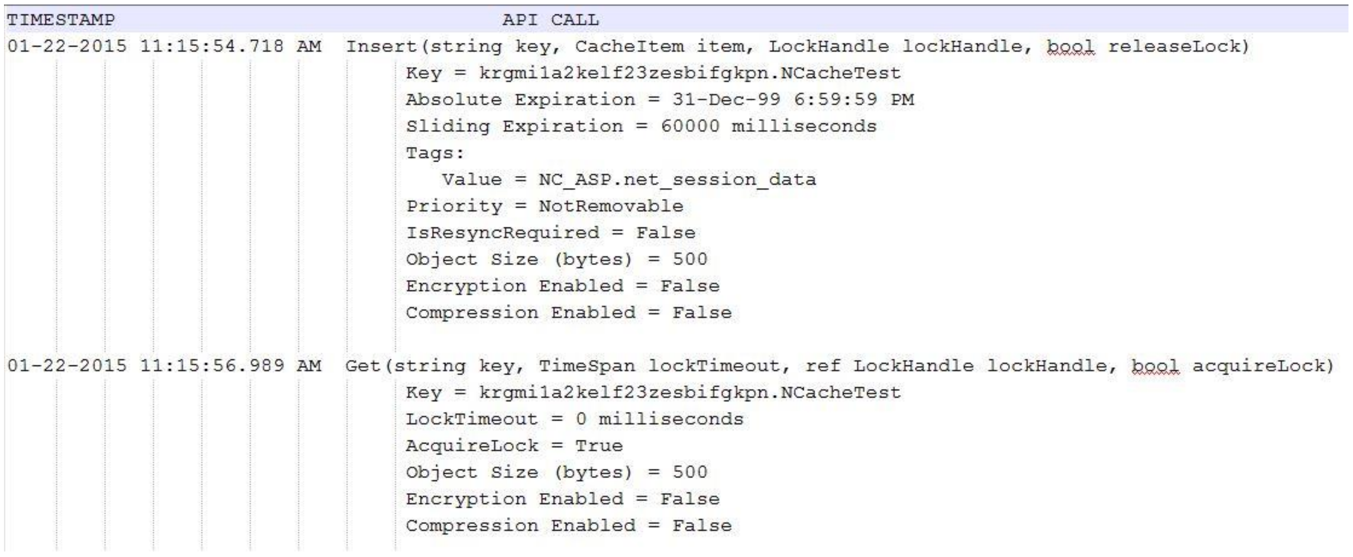
Configuring Client Side API Logging:

To enable and configure client side API logging, following properties are to be added in application's configuration file (web.config or app.config).

```
<!-- Enable/Disable Api Logging -->
<add key="CacheClient.EnableAPILogging" value="true" />

<!-- starts logging at cache initialize after specified time interval. Format hh:mm:ss-->
<add key="CacheClient.TimeBeforeLoggingStart" value="00:10:30" />

<!-- Number of logging iterations -->
<add key="CacheClient.APILogIteraions" value="4" />

<!-- Length of each iteration. values in seconds -->
<add key="CacheClient.APILogIterationLength" value="3600" />

<!-- Delay between two consecutive iterations. values in seconds -->
<add key="CacheClient.APILogDelayBetweenIteration" value="5" />

<!-- Interval in seconds after which logs will be written to file-->
<add key="CacheClient.LoggerThreadLoggingInterval" value="5" />
```

Once the configuration file is modified, application has to be restarted in order to start logging based on the above mentioned configurations. If it is an ASP.NET application, app pool of this application needs to be restarted.

Client log files are generated in `[InstallDirectory]\log-files\APIUsageLogs` folder. API Usage logs file name is created in following *format APIUsageLogs.CacheName_TimeStamp.logs*.



```
TIMESTAMP                                 API CALL
01-22-2015 11:15:54.718 AM  Insert(string key, CacheItem item, LockHandle lockHandle, bool releaseLock)
                                 Key = krgmi1a2kelf23zesbifgkpn.NCacheTest
                                 Absolute Expiration = 31-Dec-99 6:59:59 PM
                                 Sliding Expiration = 60000 milliseconds
                                 Tags:
                                    Value = NC_ASP.net_session_data
                                 Priority = NotRemovable
                                 IsResyncRequired = False
                                 Object Size (bytes) = 500
                                 Encryption Enabled = False
                                 Compression Enabled = False

01-22-2015 11:15:56.989 AM  Get(string key, TimeSpan lockTimeout, ref LockHandle lockHandle, bool acquireLock)
                                 Key = krgmi1a2kelf23zesbifgkpn.NCacheTest
                                 LockTimeout = 0 milliseconds
                                 AcquireLock = True
                                 Object Size (bytes) = 500
                                 Encryption Enabled = False
                                 Compression Enabled = False
```

5. Using NCache ASP.NET Session State

Before you can use NCache for ASP.NET sessions, you must first install and configure a cache. Please read the section on Cache Administration that explains how to do this. Once you've done that, then you're ready to follow the steps mentioned below.

5.1 Using with NuGet Package

NuGet is a popular way of simplifying the use of NCache in your .NET application. When you use NuGet to install the NCache package, it copies the library files to your Visual Studio solution and automatically updates your project (add references, change config files, etc.). If you remove a package, NuGet reverses whatever changes it made so that no clutter is left.

5.1.1 Install Package

You need to do this from inside Visual Studio when you have your project or solution opened in it. To install the NuGet package for NCache ASP.NET Session State Provider, run the following command in Package Manager Console inside Visual Studio.

```
Install-Package NC-OS-SessionStateProvider
```

5.1.2 Verify Assembly References

Once this package is installed, please verify that references of following assemblies have been added to your application.

- `Alachisoft.NCache.SessionStoreProvider.dll`
- `Alachisoft.NCache.SessionStateManagement.dll`

5.1.3 Verify web.config Changes

Please also verify that following section has been added to the web.config of ASP.NET application.

```
<sessionState cookieless="false" regenerateExpiredSessionId="true"
               mode="Custom" customProvider="NCacheSessionProvider" timeout="20">
       <providers>
              <add name="NCacheSessionProvider"
                   type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider"
                   sessionAppId="myApp"
                   cacheName="myCache"
                   writeExceptionsToEventLog="false"
                   enableLogs="false"/>
       </providers>
</sessionState>

<sessionState cookieless="false" regenerateExpiredSessionId="true" mode="Custom"
customProvider="NCacheSessionProvider" timeout="60">
      <providers>
        <add name="NCacheSessionProvider"
type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider" sessionAppId="WebF1"
cacheName="mycache" writeExceptionsToEventLog="false" asyncSession="false" enableLogs="false" />
      </providers>
    </sessionState>
```

Please note that `timeout="20"` means that your sessions will expire after 20 minutes of inactivity. You can specify whatever value that suits you here.

5.2 Manually Modify web.config

You can also add following sections in your web.config to configure NCache session state provider.

```
<assemblies>
       <add assembly="Alachisoft.NCache.SessionStoreProvider,
                       Version=4.6.0.0, Culture=neutral,
                       PublicKeyToken=1448e8d1123e9096"/>
</assemblies>

<sessionState cookieless="false" regenerateExpiredSessionId="true"
               mode="Custom" customProvider="NCacheSessionProvider" timeout="20">
       <providers>
              <add name="NCacheSessionProvider"
                   type="Alachisoft.NCache.Web.SessionState.NSessionStoreProvider"
                   sessionAppId="NCacheTest"
                   cacheName="myReplicatedCache"
                   writeExceptionsToEventLog="false"
                   enableLogs="false"/>
       </providers>
</sessionState>
```

Please note the `Version=4.6.0.0` in "`assemblies`" tag. This version must match the version of NCache you have downloaded.

If you decided to set `enableLogs="true"` then you'll have to make sure that "`NCache\log-files\SessionStoreProvider`" folder gives write permissions to "users" or to your specific ASP.NET user-id.

Similarly, if you decided to set `writeExceptionsToEventLog="true"` then you'll have to make sure that your ASP.NET user-id has the permission to write to the Event Log.

5.3 What to change in Web.Config?

With this configuration, all ASP.NET sessions of this application will be stored in 'myCache' if it is configured and running. You can modify following attributes according to your needs.

1. **cacheName:** This is name of the cache you've created.
2. **enableLogs ("true" or "false"):** This turns on or off error logging. If it is turned on, then NCache logs all errors in `NCache\log-files\SessionStoreProvider`.
3. **sessionAppId:** If you have multiple applications or app domains running on the same web farm and accessible from the same application, then you have the choice of either sharing your sessions across app domains or not. If you don't want to share sessions across app domains, then specify a unique "sessionAppId" for each app domain. This will ensure that each app domain puts its own sessionAppId to the SessionId thus making it impossible for the other app domains that might be using the same SessionId to fetch the same session.
4. **writeExceptionsToEventLog ("true" or "false"):** If this is set to true, then NCache logs errors to the Event log. However, please make sure that your ASP.NET user-id has permission to write to Event Log. Otherwise, you'll get errors when you run your application.
5. **exceptionsEnabled ("true" or "false"):** If this is set to false, then NCache ignore exceptions thrown by NCache server. If this is set to true, then NCache throws exceptions upward.

5.4 Ensure all Objects in Session are Serializable

If you were previously using the InProc mode of ASP.NET Sessions, then the objects you were putting your Session did not need to be Serializable. However, if you were using StateServer or SqlServer modes for Sessions previously, then your objects are already serializable for all of this to work.

In either case, before using NCache for your sessions, you must ensure that all your custom objects that you're putting in Session are serializable. It can be a very simple tag that you need to put on all your class definitions as shown below (in C#):

```
[Serializable]
public class Product
{
        ...
}
```

6. Using NCache ASP.NET View State

Following are the two steps that you need to follow in order to use content optimization feature of NCache:

6.1 Update/Configure App Browser File

1. Create an App browser file in your ASP.NET application. It will be created under the directory of *App_browsers*.
2. Now plug page adapters in the app browser file as following:

```
<browsers>
<!-- NCache Plug page adapters in the app browser file as following:. -->
       <browser refID="Default">
              <controlAdapters>
                     <adapter controlType="System.Web.UI.Page"
                             adapterType="Alachisoft.NCache.Adapters.PageAdapter" />
              </controlAdapters>
       </browser>
</browsers>
```

6.2 Update/Configure web.config File

1. You need to add the following assembly reference in compilation section of web.config file.

```
<assemblies>
   <add assembly="Alachisoft.NCache.Adapters, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=1448e8d1123e9096"/>
</assemblies>
```

2. You need to register a web config section which contains the settings/properties in web.config file.

```
<configSections>
<!--NCache Register config section first. -->
       <sectionGroup name="ncContentOptimization">
              <section name="settings"
              type="Alachisoft.NCache.ContentOptimization.Configurations.ContentSettings"
              allowLocation="true" allowDefinition="Everywhere"/>
       </sectionGroup>
</configSections>
```

3. Add the actual setting/configuration section as follows:

```
<!--NCache actual setting/ configuration for view state
<ncContentOptimization>
<settings enableViewstateCaching="true" enableTrace="false">
       <cacheSettings cacheName="mycache connectionRetryInterval="300">
       <expiration type="Absolute" duration="1" />
       </cacheSettings>
</settings>
</ncContentOptimization>
```

6.2.1 Configuration Members

Members	Description
`enableViewstateCaching`	Boolean value to enable /disable the View State caching.
`enableTrace`	Enable/disable traces.
`cacheName`	Name of the Cache.
`connectionRetyInterval`	Retry Connection to Cache interval. Default is 300 seconds.
`Expiration`	Sets the expiration type which can either be Sliding, Absolute or None. And also expiration interval in minutes.
`Duration`	Sets expiration interval in minutes.

7. NCache as NHibernate Second Level Cache

NHibernate is an Open Source O/R Mapping Engine for .NET. If you have an application using NHibernate, you can plug-in NCache without any code change.

7.1 Using with NuGet Package

7.1.1 Install Package

You need to do this from inside Visual Studio when you have your project or solution opened in it. To install NCache NHibernate Provider in your application run the following command in Package Manager Console inside Visual Studio.

```
Install-Package NC-NHibernateCachingProvider
```

7.1.2 Verify Assembly References

Once this package is installed, please verify that references of following assemblies have been added to your application.

- `Alachisoft.NCache.Integrations.NHibernate.Cache.dll`

7.1.3 Modify app.config

1. Please verify that following section has been added to the app.config/web.config of NHibernate application.

```
<appSettings>
  <add key="ncache.application_id" value="myapp" />
</appSettings>
```

2. Verify that following section has been added to hibernate.cfg.xml.

```
<session-factory>
  <property name="cache.provider_class">
       Alachisoft.NCache.Integrations.NHibernate.Cache.NCacheProvider,
       Alachisoft.NCache.Integrations.NHibernate.Cache</property>
  <property name="cache.use_second_level_cache">true</property>
  <property name="cache.use_query_cache">true</property>
</session-factory>
```

Following are brief descriptions of different attributes in the above given configuration.

- **cache.provider_class:** This option lets you specify NCache as second level cache provider. You need to mention two classes which implement ICacheProvider and ICache interfaces respectively. This is how NHibernate knows how to call this second level cache.

- **cache.use_second_level_cache**: Enable use of second level cache in your application by adding following property in NHibernate configuration section's **session-factory** tag:

```
<property name="cache.use_second_level_cache">true</property>
```

- **ncache.application_id**: NCache provider for NHibernate identifies each application by an application id that is later used to find appropriate configuration for that application from NCache configuration file for

NHibernate "NCacheNHibernate.xml". This application id must be specified in application's configuration file (app.config/web.config).

3. Verify that NCacheNHibernate.xml file is added to the project. Following is a sample content of this xml file.

```
<configuration>
  <application-config application-id="myapp" enable-cache-exception="true"
   default-region-name="default" key-case-sensitivity="false">
  <cache-regions>
    <region name="AbsoluteExpirationRegion" cache-name="mycache" priority="Default"
     expiration-type="sliding" expiration-period="180" />
    <region name="default" cache-name="mycache" priority="default" expiration-type="none"
            expiration-period="0"  />
  </cache-regions>

  </application-config>
</configuration>
```

Following is the brief description of each attribute in the above mentioned configuration.

Members	Description
application-id	This options let you specify the unique id for current application-config. This id is used to select appropriate configuration for a particular application.
enable-cache-exception	Identifies whether the exceptions if occurred in NCache will be propagated to NHibernate provider.
default-region-name	Allows to specify a default region for the application. If a particular region's configurations are not fount default region's configurations will be used. Specified default region's configuration must exist in cache-regions section.
key-case-sensitivity	This option allows to specify whether cache keys will be case sensitive or not. This option has to be configured according to database used. If database is case-sensitive set this option true, otherwise false.
cache-regions	This sections lets you configure multiple cache regions for NHibernate application. Each region's configuration is specified in region tag.
region	This tag contains the configurations for one particular region. Following options can be configured for any particular region. **name:** Each region is identified by its name. Name of the region should be unique. **cache-name:** This option allows to specify NCache's cache name to be used for this region. **priority:** The priority you want to use for items cached. The possible values are : • Default • Low • BelowNormal • Normal • AboveNormal • High • NotRemovable **expiration-type**: This options allows to specify type of expiration for any item in this region. Possible values are absolute/sliding/none.

	expiraion-period: Allows to specify expiration period if absolute/sliding expiration is configured. Its value should be greater than 0.

7.2 Manually Modifying App.Config

Here is what you need to mention in your app.config to use NCache:

```
<appSettings>
    <add key="ncache.application_id" value="myapp" />
</appSettings>
```

- Add following properties in session-factory of nhibernate configuration and modify the values according to your requirements. Details of each attribute is explained in previous section.

```
<nhibernate-configuration xmlns="urn:nhibernate-configuration-2.2">
    ...
    <session-factory>
      ...
      <property name="cache.provider_class">
       Alachisoft.NCache.Integrations.NHibernate.Cache.NCacheProvider,
       Alachisoft.NCache.Integrations.NHibernate.Cache</property>
      <property name="cache.use_second_level_cache">true</property>
      <property name="cache.use_query_cache">true</property>
       ...
    </session-factory>
</nhibernate-configuration>
```

- Adding NCachHibernate.xml in NHibernate Application

Create a new NCacheHibernate.xml file in application directory and modify its values according to your requirements and change xml file property 'Copy to Output Directory' to 'Copy always'. Details of each attribute is explained in previous section.

```
<configuration>
<application-config application-id="myapp" enable-cache-exception="true"
 default-region-name="default" key-case-sensitivity="false">
 <cache-regions>
   <region name="AbsoluteExpirationRegion" cache-name="mycache" priority="Default"
    expiration-type="sliding" expiration-period="180" />
   <region name="default" cache-name="mycache" priority="default" expiration-type="none"
    expiration-period="0"  />
 </cache-regions>
</application-config>
</configuration>
```

- Adding NCache NHibernate dll
  Add Alachisoft.NCache.Integations.NHibernate.Cache.dll from `[InstallDirectory]\integrations\nhibernate\assembly` to project reference.

8. NCache for Object Caching

You can cache application data by making NCache API calls directly from your application. Here are the steps to do that.

Before you can do this, you need to first make sure you've installed NCache and configured your cache. See Install NCache & Configure Cache section for more details. After that, follow these steps.

For detailed understanding on all NCache APIs, please read .NET API Reference of NCache.

⚠ To utilize the APIs, include the following namespace in your application: `Alachisoft.NCache.Web.Caching.`

8.1 Initialize Cache

After successfully configuring NCache, you can start developing the application by embedding NCache API calls. NCache client applications can communicate with the servers with an `InitializeCache` method. It establishes connection between client and server and returns a single cache handler per cache for a single application. However, an application can contain multiple separate cache instances.

8.1.1 Using Basic Initialize Method

```
Cache cache=null;

try
{
    cache = NCache.InitializeCache("mycache");
}
catch (Exception ex)
{
    //handle exception
    // Possible Exceptions:
    //1. No server available to process the request
    //2. client.ncconf does not contain current cache information
}
```

8.1.2 Initializing Multiple Caches in Single Application

For this exercise, both caches need to be configured previously and they must be in running state.

```
Cache cache1 = null;
Cache cache2 = null;
try
{
       cache1 = NCache.InitializeCache("mycache");
       cache2 = NCache.InitializeCache("myreplicatedcache");
}
catch (Exception ex)
{
       //handle exception
}
```

8.1.3 Initializing Cache using CacheInitParams

CacheInitParams allows editing values of the properties at initialization time. In this example, the values of `RetryInterval` and `ConnectionRetries` properties can be changed; for this application these values will be used instead of the ones specified in client configuration files.

```
Cache cache = null;

try
{
    // Create new InitParam instance
    CacheInitParams initParam = new CacheInitParams();
    initParam.RetryInterval = 3;
    initParam.ConnectionRetries = 2;
    cache = NCache.InitializeCache("mycache", initParam);
}
catch (Exception exp)
{
    // handle exception
}
```

8.2 Adding Items

After successful initialization of cache and gaining valid cache handle, Add operation can be performed. Add is the basic operation provided by NCache; data can be added/updated to the cache using multiple API calls.

Key-Value cache store
NCache uses a "key" and "value" structure for objects. Every object must have a unique string key associated with it. Every key has an atomic occurrence in the cache whether it is local or clustered. Cached keys are case sensitive in nature, and if you try to add another key with same value, an `OperationFailedException` is thrown by the cache.

Custom Objects
The value of the objects stored in the cache can range from being simple string types to complex objects. However, the custom object data must be serializable otherwise NCache will throw a serialization exception. You can also add data using the setValue method of the CacheItem class, which results in the data being saved internally as a CacheItem.

⚠ To utilize the APIs, include the following namespace in your application: `Alachisoft.NCache.Web.Caching.`

8.2.1 Adding Objects to Cache

In order to add data in cache it must be ensured that the object is either .NET serialized or registered with NCache Compact serialization framework. In this example, a new object of Product class is created and added to cache.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 15;

string key = "Product:" +product.ProductID ;

try
{
    cache.Add(key, product);
```

```
        }
        catch (OperationFailedException ex)
        {
            // handle exception

            // usually thrown if key already exist in cache
            // however verify the failure reason
        }
```

8.2.2 Adding Objects Using CacheItem

`CacheItem` is a custom class provided by NCache which can be used to add data to the cache. This class encapsulates data as its value property. `CacheItem` also lets you set multiple meta-info associated with an object in single operation.

```
        Product product = new Product();
        product.ProductID = 1001;
        product.ProductName = "Chai";
        product.UnitsInStock = 15;

        CacheItem cacheItem = new CacheItem(product);
        string key = "Product:" +product.ProductID ;

        try
        {
            cache.Add(key, cacheItem);
        }
        catch (OperationFailedException ex)
        {
            // handle exception
        }
```

8.2.3 Adding Items to Cache in Bulk

A collection of items can be added in cache using `AddBulk` method. This method returns a dictionary of all the keys that failed to be added along with the exception.

```
            Product product1 = new Product();
            product1.ProductID = 1001;
            product1.ProductName = "Chai";
            product1.UnitsInStock = 15;
            string key1 = "Product:" + product1.ProductID;

            Product product2 = new Product();
            product2.ProductID = 1002;
            product2.ProductName = "Chang";
            product2.UnitsInStock = 25;
            string key2 = "Product:" + product2.ProductID;

            string[] keys = { key1, key2 };

            CacheItem[] items = new CacheItem[2];
            items[0] = new CacheItem(product1);
            items[1] = new CacheItem(product2);

            try
            {
```

```
            IDictionary result = cache.AddBulk(keys, items);
        }
        catch (Exception ex)
        {
            // handle exception
        }
}
catch (Exception ex)
{
    // handle exception
}
```

8.2.4 Adding New Data with Absolute Expiration

In this example Add operation provided by NCache is used to add Absolute Expiration to item. The expiration is set by using DateTime class. Also if Absolute Expiration is used, `NoSlidingExpiration` is to be specified.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
string key = "Product:" +product.ProductID ;

try
{
    // adding absolute expiration of 5 min through Add API.
    cache.Add(key, product, System.DateTime.Now.AddMinutes(5),
                                Cache.NoSlidingExpiration, CacheItemPriority.Normal);
}
catch(OperationFailedException ex)
{
    // handle exception
}
```

8.2.5 Adding New Data with Sliding Expiration

In this example Add() operation provided by NCache is used to add Sliding Expiration to item. The expiration is set by using TimeSpan class. Also if Sliding Expiration is used, `NoAbsoluteExpiration` is to be specified.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
string key = "Product:" +product.ProductID ;

try
{
    // adding sliding expiration of 5 min through Add API.
    cache.Add(key, product, Cache.NoAbsoluteExpiration, new TimeSpan(0, 5, 0),
CacheItemPriority.Normal);

}
catch(OperationFailedException ex)
{
    // handle exception
}
```

8.3 Updating Items

Once you've initialized the cache, you can use the cache handle later in your application to insert items to the cache. An `Insert()` call adds the item if the item does not exist, and updates it if the item already exists.

8.3.1 Updating Objects in Cache

In order to update data previously added in cache, it must be ensured that the object is serialized. In this example, a new object is added with an existing key.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 5; // updated units

string key = "Product:" +product.ProductID ;

try
{
    //precondition: Cache is already initialized and item exists
    cache.Insert(key, product);
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.3.2 Updating Objects Using CacheItem

In this example, a key is updated that has already been existing in cache with object set as a property of `CacheItem`. `CacheItem` is a custom class provided by NCache which can be used to add data to the cache. This class encapsulates data as its value property. `CacheItem` also lets you set multiple meta-info associated with an object in single operation.

```
Product product = new Product();
product.ProductID = 1001;
product.ProductName = "Chai";
product.UnitsInStock = 5; // updated units
CacheItem cacheItem = new CacheItem(product);
string key = "Product:" +product.ProductID ;

try
{
    cache.Insert(key, cacheItem);
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.3.3 Updating Items in the Cache in Bulk

An existing collection of items can be updated with the help of `InsertBulk` method.

```
Product product1 = new Product();
product1.ProductID = 1001;
product1.ProductName = "Chai";
product1.UnitsInStock = 5; //updated units
```

```
string key1="Product:"+product1.ProductID;

Product product2= new Product();
product2.ProductID = 1002;
product2.ProductName = "Chang";
product2.UnitsInStock = 6; //updated units

string key2="Product:"+product2.ProductID;
string[] keys = { key1, key2 };

CacheItem[] items = new CacheItem[2];
items[0] = new CacheItem(product1);
items[1] = new CacheItem(product2);

try
{
    cache.InsertBulk(keys, items);
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.3.4 Updating Data with Absolute Expiration

In this example, `Insert` provided by NCache is used to add/update Absolute Expiration to item added to the key. The expiration is set by using DateTime class. Also as Absolute Expiration is being used, `NoSlidingExpiration` is used for Sliding Expiration.

```
try
{
    // adding absolute expiration of 5 min through Add API.
    cache.Insert(key, product, System.DateTime.Now.AddMinutes(5),
                 Cache.NoSlidingExpiration, CacheItemPriority.Normal);
}
catch(OperationFailedException ex)
{
    // handle exception
}
```

Insert() operation does not return until all the cache servers are updated based on your caching topology.

8.3.5 Updating Data with Absolute Expiration

In this example, `Insert` provided by NCache is used to add/update Sliding Expiration to item added to the key. The expiration is set by using `TimeSpan` class. Also as Sliding Expiration is being used, `NoAbsoluteExpiration` is used for Absolute Expiration.

```
try
{
       //updating sliding expiration of 5 min through Insert API.
        cache.Insert(key, product, null, Cache.NoAbsoluteExpiration, new TimeSpan(0, 5,
       0), CacheItemPriority.Default);
}
catch(OperationFailedException ex)
{
    // handle exception
```

```
        }
```

`Insert()` operation does not return until all the cache servers are updated based on your caching topology.

8.4 Fetching Items

Primarily a cache is only considered as effective as its ability to retrieve data. NCache utilizes the key-value architecture of its store to maximize the ways of data retrieval. Get method is the primary API provided to serve previously cached data; however basic indexer approach can also be used. Both of these methods are explained below.

8.4.1 Retrieving Data using Get Method

In this example, the basic Get method is used to retrieve item "Product:1001". This item has previously been added to the cache. Get method returns general object which needs to be casted accordingly. If a key does not exist in cache **null** value is returned.

```
        string key = "Product:1001";
        Product product = null;

        try
        {
                //null is returned if key does not exist in the cache.
                object result = cache.Get(key);
                if ( (result != null) && (result is Product) )
                {
                        product = (Product)result;
                }
        }
        catch (OperationFailedException ex)
        {
                // handle exception
        }
```

8.4.2 Retrieve a CacheItem

In this example, the basic `GetCacheItem` method is used to retrieve item "Product:1001". This item has been added to the cache. This method returns `CacheItem` whose value property encapsulates the data; the Object value need to be cast accordingly.

```
        string key = "Product:1001";
        Product product = null;

        try
        {
                CacheItem result = cache.GetCacheItem(key);
                if (result != null)
                    {
                     if (result.Value isProduct)
                        {
                            product = (Product)result.Value;
                        }
                    }
                }
        catch (OperationFailedException ex)
        {
                // handle exception
        }
```

8.4.3 Checking If an Item Exists in Cache

In order to verify if a key exists in the cache or not, the contains method can be used. This method determines whether cache contains the specific key and returns true if key already exists in cache and false if not.

```
    string key = "Product:1001";

    try
    {
        if (cache.Contains(key))
        {
            // do something
        }
        else
        {
            // do something
        }
    }
    catch (OperationFailedException ex)
    {
            // handle exception
    }
```

8.5 Lock/Unlock Items

A lockHandle is associated with an item to ensure that the particular item remains inaccessible throughout the cache. Two primary properties lockId and lockdate are used to ensure the appropriate behavior of locking while performing various cache operations.

NCache provides method calls exclusively for locking as well as numerous overloads that manipulate the locking mechanism.

⚠ To utilize the APIs, include the following namespace in your application: `Alachisoft.NCache.Web.Caching`.

8.5.1 Locking an Item Explicitly

Lock on an item can be acquired explicitly before performing any operation. Lock method requires a TimeSpan to lock an item for a specified time. However, if you do not want the acquired lock to expire simply specify a new TimeSpan().
The lock method used in this example associates a lockHandle. Kindly ensure that the single LockHandle is associated with a single key. Release the lock before re-using the handle; otherwise it might lead to inconsistency of behavior.

You can explicitly acquire lock on an item before performing any operation. Lock method requires a TimeSpan to lock an item for a specified time. However if you do not want the acquired lock to expire simply specify a new TimeSpan.

The Lock method used in this example associates a lockHandle with a key. Kindly ensure that the single LockHandle is associated with a single key. Release the lock before reusing the handle; otherwise it might lead to inconsistency of behavior.

```
//create a new lock Handle
LockHandle lockHandle = new LockHandle();
string key = "Product:1001";

try
{
    // Specifiying the time span of 10 sec for which the item remains locked
    bool locked = cache.Lock(key, new TimeSpan(0, 0, 10), out lockHandle);
}
catch (OperationFailedException ex)
{
    // handle exception

}
```

8.5.2 Locking an Item during Fetch Operation

An item while being retrieved. The item will be inaccessible for others unless it is released. In case of a mismatch of key, **null** value is returned.

In this example, a key and a lockHandle for the key should be provided to fetch the cached object and lock it. "`true`" should be specified if the user wants to acquire lock.

```
LockHandle lockHandle = new LockHandle();

//Specify time span of 10 sec for which the item remains locked
TimeSpan lockSpan = new TimeSpan(0,0,10);
string key="Product:1001";

try
{
    object result = cache.Get(key, lockSpan , ref lockHandle, true);
    if (result != null)
    {
        if (result is Product)
        {
            Product product = (Product)result;
        }
    }
}
catch (OperationFailedException ex)
{
     // handle exception
}
```

8.5.3 Lock Expiration

If time is specified as a value of `TimeSpan`, NCache would lock the item for that specified duration.

```
LockHandle lockHandle = new LockHandle();

//Specify time span of 10 sec for which the item remains locked
TimeSpan lockSpan = new TimeSpan(0,0,10);
string key="Product:1001";
```

```
try
{
    bool lockAcquired = cache.Lock(key, lockSpan, out lockHandle);
    //Verify that the lock is released automatically after this time period.
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.5.4 Releasing Lock with Update Operation

While updating an item, the lock can be released allowing other cache clients to use the cached data. In order to successfully release the locked item, the `lockHandle` that was initially used to lock the item must be specified. In case of an invalid or different handle, NCache would throw an `OperationFailedException`.

```
Product updatedProduct = new Product();
updatedProduct.ProductID = 1001;
updatedProduct.ProductName = "Chai";
updatedProduct.Category = 4;
string key = "Product"+ updatedProduct.ProductID;
LockHandle lockHandle = new LockHandle();

try
{
    // lock exisiting item for the time span of 30 seconds
    bool locked = cache.Lock(key, TimeSpan.FromSeconds(30), out lockHandle);

    if (locked)
    {
        cache.Insert(key, new CacheItem(updatedProduct), lockHandle, true);
    }
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.5.5 Releasing Lock Explicitly

In order to release lock forcefully on a previously locked cached item, lockHandle that was used to lock the key should be specified.

Note: NCache will ignore the locks if other overloads of Get, Insert and Remove methods are called.

```
LockHandle lockHandle = new LockHandle();
string key = "Product:1001";

try
{
    // lock an existing item and save the lockHandle for 10 seconds
    bool locked = cache.Lock(key, new TimeSpan(0, 0, 10), out lockHandle);
    if (locked)
    {
        // unlock locked item using saved LockHandle
        cache.Unlock(key, lockHandle);
    }
```

```
        }
        catch (OperationFailedException ex)
        {
            // handle exception
        }
```

8.6 Searching Items with SQL & LINQ

8.6.1 Adding and Updating Indexed Items

Before searching, items need to be indexed and added in cache. For adding indexed items, the basic APIs of add and insert as mentioned in NCache Basic Operations should be used.

```
        Product product = new Product();
        product.ProductID = 1001;
        product.ProductName = "Chai";
        string key = "Product:" +product.ProductID ;

        try
        {
            cache.Add(key, product);
        }
        catch (OperationFailedException ex)
        {
            // handle exception
        }
```

8.6.2 Cache Search Returning Keys

The following code searches the cache and returns a collection of keys. Then, it iterates over all the returned keys and individually fetches the corresponding cached items from the cache.

In case you still require to mirror the behavior of Search API, i.e., return only the keys after a query is executed, you can do so by setting the `getData` flag to `false` in `ExecuteQuery`:

```
         Hashtable values = new Hashtable();
         values.Add("ProductName", "Chai");
         values.Add("UnitsInStock", 250);
         try
         {
             // Instead of Product, specify fully qualified name of your custom class.
             string query = "SELECT Product WHERE this.ProductName =? AND this.UnitsInStock > ?";
             ICacheReader reader = cache.ExecuteReader(query, values, false);

             if (reader.FieldCount > 0)
             {
                 while (reader.Read())
                 {
                     object category = reader.GetValue(0);
                     //perform operations
                 }
             }
             else
             {
                 //no record exists
             }
             reader.Close();
         }
         catch (OperationFailedException ex)
```

```
        {
            // handle exception
        }
```

8.6.3 Cache Search Returning Items

In some situations, when the intention is to fetch all or most of the cached items associated with cache keys, it is better to fetch all the keys and items in one call. Below is an example of such a scenario. It searches the cache and returns a dictionary containing both keys and values. This way, all the data based on the search criteria is fetched in one call to NCache.

```
        string query = "SELECT this.Category, MAX(Prod.Product.ProductID) WHERE this.Category =
        ? GROUP BY this.Category";
        Hashtable values = new Hashtable();
        values.Add("Category", 4);

        try
        {
            ICacheReader reader = cache.ExecuteReader(query, values);

            //OR to get data
            ICacheReader reader = cache.ExecuteReader(query, values, true);

            //OR to get data with a specific chunk size
            ICacheReader reader = cache.ExecuteReader(query, values, true, 50);

            if (reader.FieldCount > 0)
            {
                while (reader.Read())
                {
                    //you can get value through the field name...
                    object category = reader.GetOrdinal("Category");

                    //...or through the index number
                    object result = reader.GetValue(1);

                    //perform operations
                }
            }
            else
            {
               //no record exists
            }
            reader.Close();
        }
        catch (Exception ex)
        {
            //handle exception
        }
```

8.6.4 Querying Samples for Operators

Some code samples for different queries are given below:

- **Using Equal Operator**

```
string query = "SELECT Product where this.ProductID = ?";
Hashtable values = new Hashtable();
values.Add("ProductID", 1001);
```

```
try
        {
            ICacheReader reader = cache.ExecuteReader(query, values);
            if (reader.FieldCount > 0)
            {
                while (reader.Read())
                {
                    Product result = (Product)reader.GetValue(1);
                    //perform operations
                }
            }
            else
            {
                //no record exists
            }
            reader.Close();
        }
catch(Exception ex)
{
    // handle exception
}
```

- **Using Multiple Operators**

```
        string query = "SELECT Product where this.ProductID < ? AND this.Category = ?";
        Hashtable values = new Hashtable();
        values.Add("ProductID", 1002);
        values.Add("Category", 4);
        try
        {
           ICacheReader reader = cache.ExecuteReader(query, values);
            if (reader.FieldCount > 0)
            {
                while (reader.Read())
                {
                    Product result = (Product)reader.GetValue(1);
                    //perform operations
                }
            }
            else
            {
                //no record exists
            }
            reader.Close();
        }
        catch (Exception ex)
        {
            //handle exception
        }
```

- **Using IN Operator**

```
       string query = "SELECT Product where this.ProductID IN (?,?,?)";
        ArrayList idList = new ArrayList();
        idList.Add(1001);
        idList.Add(100);
        idList.Add(500);
        Hashtable values = new Hashtable();
        values.Add("ProductID", idList);
        try
        {
            ICacheReader reader = cache.ExecuteReader(query, values);
            if (reader.FieldCount > 0)
            {
                while (reader.Read())
                {

                    Product result = (Product)reader.GetValue(1);
                    //perform operations
                }
            }
            else
            {
                //no record exists
            }

            reader.Close();

        }
        catch (Exception ex)
        {
            // handle exception
        }
```

- **Using LIKE Operator**

```
       string query = "SELECT Product where this.ProductName LIKE ?";
       ArrayList list = new ArrayList();
       list.Add("Ch*");

       Hashtable values = new Hashtable();
       values.Add("ProductName", list);
       try
        {
            ICacheReader reader = cache.ExecuteReader(query, values);
            if (reader.FieldCount > 0)
            {
                while (reader.Read())
                {

                    Product result = (Product)reader.GetValue(1);
                    //perform operations
                }
            }
            else
            {
                //no record exists
            }
            reader.Close();

        }
        catch (Exception ex)
        {
            // handle exception
        }
```

8.7 Removing Items

Deleting item(s) from the cache is also considered as one of the basic operations. NCache primarily provides two (`remove`, `delete`) methods to delete an item from cache.

8.7.1 Difference between Delete and Remove methods

DELETE	REMOVE
Returns nothing after deletion.	Deletes data from the cache and returns deleted data to the application.

8.7.2 Using Remove Method

A remove method is a basic method which removes the key from the cache and returns the removed object to the cache.

```
string key = "Product:1001";
Product product = null;
try
{
    // null is returned if key does not exist in cache
    object result = cache.Remove(key);
    if (result != null)
    {
        if (result is Product)
        {
             product = (Product)result;
        }
    }
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.7.3 Using Delete Method

A delete method is a basic method which deletes the key from the cache.

```
string key = "Product:1001";
try
{
    cache.Delete(key);
}
catch (OperationFailedException ex)
{
    // handle exception
}
```

8.8 Item Based Event Notification

NCache features event driven fetching of data whenever the data in cache is modified. This allows client applications to receive data without any explicit data fetch request to the cache server. Events work on push mechanism for whenever there is some activity of interest takes place in cache. It publishes events to its subscribed clients.

8.8.1 Types of Notifications

- **Add Notification**

  Add Notification is triggered when a new item is added to the cache. All applications that have registered this event will receive a notification when data is added to the cache.

- **Update Notification**

  Update Notification is triggered when an existing item is updated in the cache. All applications that have registered this event will receive a notification when data is updated in the cache.

- **Remove Notification**

  Remove Notification is triggered when an item is removed from the cache. All applications that have registered this event will receive a notification when data is removed from the cache.

These Notifications are **asynchronously** sent to the client so there is no overhead on client's activities.

To utilize the API, include the following namespace in your application: Alachisoft.NCache.Web.Caching.

8.8.2 Registering Event with a Particular Item

NCache provides a separate classification of event which can be registered with individual keys. To register update or remove event with a particular key(s) RegisterCacheNotification method should be used.

In this example the cache will generate notification when the appropriate operation is performed on the key.

```
static void Main(string[] args)
{
        // Register target method
        CacheDataNotificationCallback dataNotificationCallback
                = new CacheDataNotificationCallback( OnCacheDataModification);
        try
        {
                // Register notification with appropriate event type on existing key
                cache.RegisterCacheNotification(key,dataNotificationCallback,
                EventType.ItemRemoved | EventType.ItemUpdated);

                // Save the event descriptor for further usage
        }
        catch(Exception ex)
        {
                // handle exception
        }
}
// Create a target method
static void OnCacheDataModification(string key, CacheEventArg args)
 {
      switch(args.EventType)
      {
          case EventType.ItemRemoved:
              //perform appropriate operations
              break;
          case EventType.ItemUpdated:
              //perform appropriate operations
              break;
      }
```

```
        }
```

8.8.3 Registering Events with CacheItem

An event can also be registered with a particular key by registering as a property of CacheItem. The SetCacheDataNotification method in CacheItem can be used to provide all appropriate information for registering the notification.

```
static void Main(string[] args)
{
       // Register target method
       CacheDataNotificationCallback dataNotificationCallback = new CacheDataNotification
       Callback( OnCacheDataModification);
       try
       {
           Product product = new Product();
           product.ProductID = 1001;
           product.ProductName = "Chai";
           product.UnitsInStock = 15;

           CacheItem cacheItem = new CacheItem(product);

          // Registering event with cacheItem
           cacheItem.SetCacheDataNotification(dataNotificationCallback,
           EventType.ItemRemoved | EventType.ItemUpdated,
           EventDataFilter.DataWithMetadata);

           string key = "Product:" +product.ProductID ;
           // Registering
           cache.Insert(key, cacheItem);

       }
       catch(Exception ex)
       {
           // handle exception
       }
}

// Create a target method
static void OnCacheDataModification(string key, CacheEventArg args)
 {
     switch(args.EventType)
     {
         case EventType.ItemRemoved:
             //perform appropirate operations
             break;
         case EventType.ItemUpdated:
             //perform appropirate operations
             break;
     }

 }
```

8.8.4 Un-Registering Item Level Notification

You can also unregister a previously registered item level notification using the UnRegisterCacheNotification method. In this example, the appropriate key, callback as CacheDataNotificationCallback and EventType previously returned upon registering of event must be specified.

```
try
{
     cache.UnRegisterCacheNotification(key, callback, EventType);
}
catch(Exception ex)
{    // handle exception
}
```

8.9 Disconnect from the Cache

At the end of your application when you no longer need to use the cache, you should call `Dispose()` on it. That frees up the connection to the cache. If your cache was created InProc, this actually disposes your cache instance which in a Replication Cache means one of the cache nodes is leaving the cluster. Here is the code:

```
try
{
    Cache cache = NCache.Caches["myReplicatedCache"];
    cache.Dispose();
}
catch(Exception ex)
{   // handle exception
}
```

9. Configuring as Memcached Wrapper

9.1 Memcached Protocol Server

NCache can be configured to act as a Memcached Protocol Server so all Memcached applications in any language can use it without any code changes. NCache Memcached Protocol Server implements both "text" and "binary" Memcached protocols.

Architecturally, NCache Memcached Protocol Server is a Windows service process that becomes a proxy to NCache cache cluster and routes all Memcached client requests to the NCache cluster and also sends back any responses accordingly. Due to this, you can configure the Memcached Protocol Server on a different set of servers than the NCache cluster if you wish.

9.1.1 Configure NCache Memcached Gateway Service

Open "`Alachisoft.NCache.Memcached.exe.config`" file from `[InstallDirectory]\integration\Memcached Wrapper\Gateway\bin`" on cache client machine or cache server where you want to run your Gateway server.

It contains information on IP addresses and ports where it is listening to receive Memcached client requests and also target cache name, where it is going to route these requests.

You need to provide target cache name and also setup getaway IP and Port for receiving client requests. You will need to change *App.config* to talk to this service later on.

```
<?xml version="1.0"?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
  </startup>
  <appSettings>
    <add key="CacheName" value="mycache"/>
    <add key="TextProtocolIP" value="20.200.20.21"/>
    <add key="TextProtocolPort" value="11212"/>
    <add key="BinaryProtocolIP" value="20.200.20.21"/>
    <add key="BinaryProtocolPort" value="11213"/>
    <!--Maximum command length in KBs-->
    <add key="MaxCommandLength" value="200"/>
  </appSettings>
</configuration>
```

Here are details on configurations.

Members	Description
`CacheName`	Name of clustered cache on which all of your operations from Memcached end user applications are going to be performed via NCache Memcached gateway service.
`TextProtocolIP`, `TextProtocolPort`	IP and port where gateway service will run and listen all client requests for applications using Text protocol of Memcached. Here you can specify either remote client address or cache server address depending upon client gateway and server gateway deployments respectively.
`BinaryProtocolIP ,`	IP and port where gateway service will run and listen all client requests for

BinaryProtocolPort	applications using Binary protocol of Memcached.
MaxCommandLength	Specifies the maximum length of command in KBs for gateway server Text protocol.

9.1.2 Start Memcached Gateway Server

Start Memcached Wrapper for NCache service from windows services. Memcached Wrapper for NCache Service is not set to automatic by default. You must start the service manually at first and can change startup option to "`Automatic`" later on.

9.1.3 Specify Gateway Server in Memcached Client Applications

Here you need to specify NCache Memcached Gateway Server (IP address and port) in App.config instead of regular Memcached server in client applications. Here is a sample configuration for a popular Memcached client (enyim) which is using default Memcached server settings.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <sectionGroup name="enyim.com">
      <section name="Memcached" type="Enyim.Caching.Configuration.MemcachedClientSection,
       Enyim.Caching" />
    </sectionGroup>
  </configSections>
  <enyim.com>
    <Memcached protocol="Text">
      <servers>
        <add address="20.200.20.21" port="11211" />
      </servers>
      <socketPool minPoolSize="10" maxPoolSize="20" connectionTimeout="00:00:10"
      deadTimeout="00:00:10" />
    </Memcached>
  </enyim.com>
</configuration>
```

Highlighted part is changed as follows replacing only NCache gateway server settings for Text and binary protocol.

```
<Memcached protocol="Text">
  <servers>
     <add address="20.200.20.21" port="11212" />
  </servers>
  <socketPool minPoolSize="10" maxPoolSize="20" connectionTimeout="00:00:10"
   deadTimeout="00:00:10" />
</Memcached>
```

If you are specifying IP address and port in the code instead of configuration files, then it needs to be changed their accordingly.

9.1.4 Run your application to use NCache Memcached Gateway

Run your application and verify that operations are being performed through your applications on the specified cluster by viewing NCache statistics for cluster in NCache Manager.

9.2 Memcached Wrapper for .NET

NCache provides a highly optimized implementation of the popular open source Memcached .NET client libraries including Eniym, BeIT Memcached and .NET Memcached client library. This implementation by NCache fully implements the public API of these libraries so it can plug-in without any code changes or even recompilation into existing Memcached .NET applications.

Internally, NCache implements NCache specific calls in these libraries so they can directly to an NCache cluster instead of going through a Memcached Protocol Server. This not only improves performance but also ensures high availability that NCache provides over Memcached. These modified Memcached .NET client libraries are provided along with their source code.

So, if you have a .NET application using Memcached, we recommend that you take the Memcached .NET Client option instead of the NCache Memcached Protocol Server option.

Here are details on NCache client plug-in assemblies for a few popular .NET clients with their specific path. Source code of these plug-ins is also shipped along with these assemblies.

Clients	Remove DLL	Re-Reference DLL
.Net Memcached Client Library	Memcached.ClientLibrary.dll	`NCache\integration\Memcached\Clients\.NET Memcached Client Library\bin\Memcached.ClientLibrary.dll`
Enyim	Enyim.Caching.dll	`NCache\integration\Memcached \Clients\Enyim\bin\Enyim.Caching.dll`
BeIT	BeITMemcached.dll	`NCache\integration\Memcached \Clients\BeITMemcached\bin\BeITMemcached.dll`

9.2.1 Replace Memcached Client Assemblies

If you are using one of the above client implementation of Memcached, you need to replace the referred assembly with the assembly shipped with NCache that has the same name.

9.2.2 Add Cache Name in App Settings

NCache Plug-in client needs to know which cache cluster you want to use to perform operations. You need to specify cache name in application configuration files. Add the following in App.config file of your application.

```
<appSettings>
  ...
  <add key="NCache.CacheName" value="democache" />
</appSettings>
```

9.2.3 Run your Client application

Now you have Memcached plug-in client ready to interact with NCache server. Run your application and your client will automatically connect to the NCache Server. Verify that operations are being performed on specified cluster by viewing statistics of cluster in performance monitor.