GetCache Robots User Guide

version 1.1.0

Introduction

GetCache is an in-memory key-value store developed with .Net 4.5 very simple to use and manage.

GetCache is a distributed in-memory cache that supports data sharding on multiple nodes and data replication. Nodes can be added and removed to the cluster without causing service interruptions.

GetCache stores any type of data, objects can be serialized using JSON or XML, row object like images or videos can be stored as byte arrays. GetCache supports automatic data expiration.

GetCache Robots are server components developed by users, that can be used to load in an automatic way the data in the storage.

Developing Robots

The first step to develop your own robots is to download the GetCache.Robot library from Nuget, alternatively the library can be downloaded from www.getcache.net. Then you can proceed to define your own Robot class.

Let us make example of a robot that loads the list of zip code in the cache. Now we proceed to define a class and assigning an arbitrary name, in this example we use *ZipCodeListRobot*.
We import the namespace of the GetCache.Robot library

```
using GetCache.Server.Framework.Robot;
using GetCache.Server.Framework.Robot.Helper;
```

and we add the attribute Robot to the class ZipCodeListRobot

```
[Robot("ZipCodeListRobot")]
public class ZipCodeListRobot
{
	public ZipCodeListRobot()
	{
	}
}
```

the parameter passed to the attribute *Robot* is the name of the component used by the server. The *ZipCodeListRobot* class and any GetCache Robots components class must have a public constructor with no input parameters.

We proceed with add a business method that retrieves a list of data and returns it.

```
[LoadData("zipcodes", RefreshTime=20, Persistence=StoringMethod.SINGLE)]
public Object GetZipCodeList()
{
	// simulate a query from a data source
	List<ZipCode> zipcodes = new List<ZipCode>();
	// populate the list retrieving data from a data source

	//...

	// return the serialized object
	return SerializationHelper.SerializeDataJson(zipcodes);
}
```

The business method is annotated with the *LoadData* attribute that requires the mandatory parameter *key* (“zipcodes” in the example) and the optional parameters *RefreshTime* that is 60 seconds by default and *Persistence* that is Single by default.

A valid business method must be public and must has not input parameters and must return a serialized object.

This method will be executed by the GetCache server after the component is deployed on the server. The server puts the object returned by the method in the cache using the key defined by the *LoadData* attribute and the after object can be read by GetCache clients.
The server refresh the object value calling the business method periodically every *RefreshTime* seconds.

The utility class *SerializationHelper* can be used to serialize the object in JSON or XML.

Robot module

A module is a set of DLL that contains the compiled classes necessary to the robots. All the DLL must be included in a directory that has the name of the module.

A module can contain more than one robot and a robot can have more than one business method.

Robot deployment

The component must be installed on at least one node of the GetCache cluster.

To deploy the Robot, compile the component and create a subdirectory under the *deploy* directory of the node. The name of the subdirectory is the name that you have decided for the module. For example create the subdirectory “ZipCode” under the directory *deploy*.

Copy all the DLL referenced by the Robot class in the “ZipCode” directory. It’s not necessary to copy the DLL *GetCache.Server.Framework.Robot.dll* because these classes are inherited by the server classes AppDomain.

Then go to the server console and type the command *deploy* followed by the module name:

```
>deploy ZipCode
```

The GetCache server installs the robot and starts it.

Alternatively, you can restart the server after copying the DLL. Each time you restart the server, GetCache loads the robots and starts them.

Robot undeployment

By carefully remove a previously installed robot on a GetCache node, you can use the *undeploy* console command.

For example

```
>undeploy ZipCode
```

remove the “ZipCode” and all the robots defined in the module.

Remember to delete the module directory and its DLL from the *deploy* directory or the server will deploy again the robot on the next restart.

Check deployed Robots

You can check the list of deployed robots on a GetCache node using the console command *show*, typing

```
>show modules
```

What can I do with Robots?

Robots classes can access any data-source, like database, web services, REST API or external resources like files.
Robots can aggregate datas and put in in the storage of GetCache.
Clients should only read the datas that is automatically and periodically refreshed by Robots.

The ideal use of Robots in recovering data that do not change or that change little over time, such as the list of the states, the list of provinces, list of postal codes, but also data that changes little during the day as the list of customer, the list of users, or statistical data, etc.
All this type of datas can be preloaded in the GetCache storage to be always available to client applications.

Example code

This is the full code of the example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using GetCache.Server.Framework.Robot;
using GetCache.Server.Framework.Robot.Helper;

[Robot("ZipCodeListRobot")]
public class ZipCodeListRobot
{
	public ZipCodeListRobot()
	{
		Console.WriteLine("ZipCodeListRobot created.");
	}

	[LoadData("zipcodes", RefreshTime=20, Persistence=StoringMethod.SINGLE)]
	public Object GetZipCodeList()
	{
		Console.WriteLine("GetZipCodeList called");
		// simulate a query from a data source
		List<ZipCode> zipcodes = new List<ZipCode>();
		zipcodes.Add(new ZipCode(){
		Code = "26100",
		Name = "Cremona"
		});
		zipcodes.Add(new ZipCode()
		{
		Code = "00010",
		Name = "Roma"
		});
		zipcodes.Add(new ZipCode()
		{
		Code = "10010",
```

```
                Name = "Torino"
                });
                zipcodes.Add(new ZipCode()
                {
                Code = "20010",
                Name = "Milano"
                });
                zipcodes.Add(new ZipCode()
                {
                Code = "14010",
                Name = "Asti"
                });
                zipcodes.Add(new ZipCode()
                {
                Code = "01010",
                Name = "Viterbo"
                });
                zipcodes.Add(new ZipCode()
                {
                Code = "19010",
                Name = "La Spezia"
                });
                zipcodes.Add(new ZipCode()
                {
                Code = "97010",
                Name = "Ragusa"
                });
                return SerializationHelper.SerializeDataJson(zipcodes);
        }
}
```

This is a data class used in the example

```
public class ZipCode
{
        public String Code { get; set; }
        public String Name { get; set; }
}
```