# Generic and concurrent Object Pool

1.10.1

Generated by Doxygen 1.8.10

Fri Feb 26 2016 23:27:38

# Contents

# Chapter 1

# Namespace Index

## 1.1 Packages

Here are the packages with brief descriptions (if available):

# Chapter 2

# Hierarchical Index

## 2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# Chapter 3

# Class Index

## 3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 4

# File Index

## 4.1 File List

Here is a list of all files with brief descriptions:

# Chapter 5

# Namespace Documentation

## 5.1 CodeProject Namespace Reference

### Namespaces

- namespace ObjectPool

## 5.2 CodeProject.ObjectPool Namespace Reference

### Namespaces

- namespace Core

### Classes

- interface IObjectPool

  *Describes all methods available on Object Pools.*
- interface IParameterizedObjectPool

  *A parameterized version of the ObjectPool interface.*
- class ObjectPool

  *Generic object pool.*
- class **ObjectPoolConstants**

  *Constants for Object Pools.*
- class ObjectPoolDiagnostics

  *A simple class to track stats during execution. By default, this class does not record anything.*
- class ParameterizedObjectPool

  *A parameterized version of the ObjectPool class.*
- class PooledObject

  *PooledObject base class.*
- class PooledObjectWrapper

  *PooledObject wrapper, for classes which cannot inherit from that class.*

## 5.3 CodeProject.ObjectPool.Core Namespace Reference

### Classes

- class **ErrorMessages**

*Static class containing all error messages used by ObjectPool.*

# Chapter 6

# Class Documentation

## 6.1 CodeProject.ObjectPool.IObjectPool< out out T > Interface Template Reference

Describes all methods available on Object Pools.

### Public Member Functions

- T GetObject ()

  *Gets a monitored object from the pool.*

### Properties

- ObjectPoolDiagnostics Diagnostics `[get, set]`

  *Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.*
- Func< T > FactoryMethod `[get]`

  *Gets the Factory method that will be used for creating new objects.*
- int MaximumPoolSize `[get, set]`

  *Gets or sets the maximum number of objects that could be available at the same time in the pool.*
- int MinimumPoolSize `[get, set]`

  *Gets or sets the minimum number of objects in the pool.*
- int ObjectsInPoolCount `[get]`

  *Gets the count of the objects currently in the pool.*

### 6.1.1 Detailed Description

Describes all methods available on Object Pools.

**Template Parameters**

| | |
|---|---|
| *T* | The type of the objects stored in the pool. |

**Type Constraints**

***T* : *PooledObject***

Definition at line 20 of file IObjectPool.cs.

### 6.1.2 Member Function Documentation

#### 6.1.2.1 T CodeProject.ObjectPool.IObjectPool< out out T >.GetObject ( )

Gets a monitored object from the pool.

**Returns**

A monitored object from the pool.

### 6.1.3 Property Documentation

#### 6.1.3.1 ObjectPoolDiagnostics CodeProject.ObjectPool.IObjectPool< out out T >.Diagnostics [get], [set]

Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.

Definition at line 29 of file IObjectPool.cs.

#### 6.1.3.2 Func<T> CodeProject.ObjectPool.IObjectPool< out out T >.FactoryMethod [get]

Gets the Factory method that will be used for creating new objects.

Definition at line 35 of file IObjectPool.cs.

#### 6.1.3.3 int CodeProject.ObjectPool.IObjectPool< out out T >.MaximumPoolSize [get], [set]

Gets or sets the maximum number of objects that could be available at the same time in the pool.

Definition at line 42 of file IObjectPool.cs.

#### 6.1.3.4 int CodeProject.ObjectPool.IObjectPool< out out T >.MinimumPoolSize [get], [set]

Gets or sets the minimum number of objects in the pool.

Definition at line 48 of file IObjectPool.cs.

#### 6.1.3.5 int CodeProject.ObjectPool.IObjectPool< out out T >.ObjectsInPoolCount [get]

Gets the count of the objects currently in the pool.

Definition at line 54 of file IObjectPool.cs.

The documentation for this interface was generated from the following file:

- IObjectPool.cs

## 6.2 CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue > Interface Template Reference

A parameterized version of the ObjectPool interface.

### Public Member Functions

- TValue GetObject (TKey key)

  *Gets an object linked to given key.*

### Properties

- ObjectPoolDiagnostics Diagnostics `[get, set]`

  *Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.*
- Func< TKey, TValue > FactoryMethod `[get]`

  *Gets the Factory method that will be used for creating new objects.*
- int MaximumPoolSize `[get, set]`

  *Gets or sets the maximum number of objects that could be available at the same time in the pool.*
- int MinimumPoolSize `[get, set]`

  *Gets or sets the minimum number of objects in the pool.*
- int KeysInPoolCount `[get]`

  *Gets the count of the keys currently handled by the pool.*

## 6.2.1 Detailed Description

A parameterized version of the ObjectPool interface.

**Template Parameters**

| | |
|---|---|
| *TKey* | The type of the pool parameter. |
| *TValue* | The type of the objects stored in the pool. |

Definition at line 21 of file IParameterizedObjectPool.cs.

## 6.2.2 Member Function Documentation

### 6.2.2.1 TValue CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.GetObject ( TKey *key* )

Gets an object linked to given key.

**Parameters**

| | |
|---|---|
| *key* | The key linked to the object. |

**Returns**

The objects linked to given key.

## 6.2.3 Property Documentation

### 6.2.3.1 ObjectPoolDiagnostics CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.Diagnostics `[get],[set]`

Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.

Definition at line 30 of file IParameterizedObjectPool.cs.

#### 6.2.3.2 Func<TKey, TValue> CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.FactoryMethod [get]

Gets the Factory method that will be used for creating new objects.

Definition at line 36 of file IParameterizedObjectPool.cs.

#### 6.2.3.3 int CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.KeysInPoolCount [get]

Gets the count of the keys currently handled by the pool.

Definition at line 55 of file IParameterizedObjectPool.cs.

#### 6.2.3.4 int CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.MaximumPoolSize [get], [set]

Gets or sets the maximum number of objects that could be available at the same time in the pool.

Definition at line 43 of file IParameterizedObjectPool.cs.

#### 6.2.3.5 int CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >.MinimumPoolSize [get], [set]

Gets or sets the minimum number of objects in the pool.

Definition at line 49 of file IParameterizedObjectPool.cs.

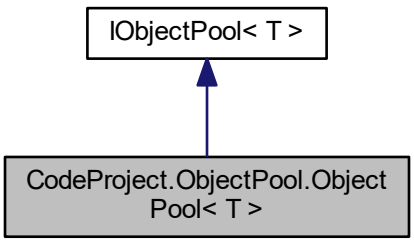
The documentation for this interface was generated from the following file:

- IParameterizedObjectPool.cs

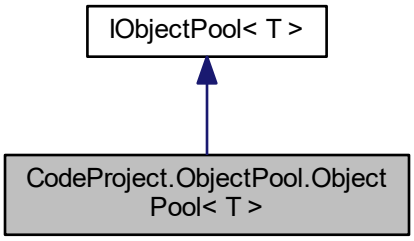
## 6.3 CodeProject.ObjectPool.ObjectPool< T > Class Template Reference

Generic object pool.

Inheritance diagram for CodeProject.ObjectPool.ObjectPool< T >:



Collaboration diagram for CodeProject.ObjectPool.ObjectPool< T >:




## Public Member Functions

- ObjectPool ()

  *Initializes a new pool with default settings.*
- ObjectPool (int minimumPoolSize, int maximumPoolSize)

  *Initializes a new pool with specified minimum pool size and maximum pool size.*
- ObjectPool (Func< T > factoryMethod)

  *Initializes a new pool with specified factory method.*
- ObjectPool (int minimumPoolSize, int maximumPoolSize, Func< T > factoryMethod)

  *Initializes a new pool with specified factory method and minimum and maximum size.*
- void Clear ()

  *Clears the pool and destroys each object stored inside it.*
- T GetObject ()

  *Gets a monitored object from the pool.*

## Public Attributes

- int ObjectsInPoolCount => _pooledObjects.Count

  *Gets the count of the objects currently in the pool.*

## Properties

- ObjectPoolDiagnostics Diagnostics `[get, set]`

  *Gets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything; you have to enable it through the ObjectPoolDiagnostics.Enabled property.*
- Func< T > FactoryMethod `[get]`

  *Gets the Factory method that will be used for creating new objects.*
- int MaximumPoolSize `[get, set]`

  *Gets or sets the maximum number of objects that could be available at the same time in the pool.*
- int MinimumPoolSize `[get, set]`

  *Gets or sets the minimum number of objects in the pool.*

### 6.3.1 Detailed Description

Generic object pool.

**Template Parameters**

| | |
|---|---|
| *T* | The type of the object that which will be managed by the pool. The pooled object have to be a sub-class of PooledObject. |

**Type Constraints**

***T* : *PooledObject***

Definition at line 23 of file ObjectPool.cs.

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 CodeProject.ObjectPool.ObjectPool< T >.ObjectPool ( )

Initializes a new pool with default settings.

Definition at line 117 of file ObjectPool.cs.

#### 6.3.2.2 CodeProject.ObjectPool.ObjectPool< T >.ObjectPool ( int *minimumPoolSize*, int *maximumPoolSize* )

Initializes a new pool with specified minimum pool size and maximum pool size.

**Parameters**

| | |
|---|---|
| *minimumPool↩ Size* | The minimum pool size limit. |
| *maximumPool↩ Size* | The maximum pool size limit |

**Exceptions**

| | |
|---|---|
| *ArgumentOutOfRange↩ Exception* | *minimumPoolSize* is less than zero, *maximumPoolSize* is less than or equal to zero, or *minimumPoolSize* is greater than *maximumPoolSize* . |

Definition at line 132 of file ObjectPool.cs.

#### 6.3.2.3 CodeProject.ObjectPool.ObjectPool< T >.ObjectPool ( Func< T > *factoryMethod* )

Initializes a new pool with specified factory method.

**Parameters**

| | |
|---|---|
| *factoryMethod* | The factory method that will be used to create new objects. |

Definition at line 141 of file ObjectPool.cs.

#### 6.3.2.4 CodeProject.ObjectPool.ObjectPool< T >.ObjectPool ( int *minimumPoolSize*, int *maximumPoolSize*, Func< T > *factoryMethod* )

Initializes a new pool with specified factory method and minimum and maximum size.

**Parameters**

| | |
|---|---|
| *minimumPool↩ Size* | The minimum pool size limit. |

| | |
|---|---|
| *maximumPoolSize* | The maximum pool size limit |
| *factoryMethod* | The factory method that will be used to create new objects. |

**Exceptions**

| | |
|---|---|
| *ArgumentOutOfRangeException* | *minimumPoolSize* is less than zero, *maximumPoolSize* is less than or equal to zero, or *minimumPoolSize* is greater than *maximumPoolSize* . |

Definition at line 157 of file ObjectPool.cs.

### 6.3.3 Member Function Documentation

#### 6.3.3.1 void CodeProject.ObjectPool.ObjectPool< T >.Clear ( )

Clears the pool and destroys each object stored inside it.

Definition at line 258 of file ObjectPool.cs.

#### 6.3.3.2 T CodeProject.ObjectPool.ObjectPool< T >.GetObject ( )

Gets a monitored object from the pool.

**Returns**

A monitored object from the pool.

Definition at line 281 of file ObjectPool.cs.

### 6.3.4 Member Data Documentation

#### 6.3.4.1 int CodeProject.ObjectPool.ObjectPool< T >.ObjectsInPoolCount => _pooledObjects.Count

Gets the count of the objects currently in the pool.

Definition at line 108 of file ObjectPool.cs.

### 6.3.5 Property Documentation

#### 6.3.5.1 ObjectPoolDiagnostics CodeProject.ObjectPool.ObjectPool< T >.Diagnostics [get], [set]

Gets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything; you have to enable it through the ObjectPoolDiagnostics.Enabled property.

Definition at line 63 of file ObjectPool.cs.

#### 6.3.5.2 Func<T> CodeProject.ObjectPool.ObjectPool< T >.FactoryMethod [get]

Gets the Factory method that will be used for creating new objects.

Definition at line 68 of file ObjectPool.cs.

#### 6.3.5.3 int CodeProject.ObjectPool.ObjectPool< T >.MaximumPoolSize [get], [set]

Gets or sets the maximum number of objects that could be available at the same time in the pool.

Definition at line 75 of file ObjectPool.cs.

#### 6.3.5.4 int CodeProject.ObjectPool.ObjectPool< T >.MinimumPoolSize [get], [set]

Gets or sets the minimum number of objects in the pool.

Definition at line 92 of file ObjectPool.cs.

The documentation for this class was generated from the following file:

- ObjectPool.cs

## 6.4 CodeProject.ObjectPool.ObjectPoolDiagnostics Class Reference

A simple class to track stats during execution. By default, this class does not record anything.

### Public Member Functions

- ObjectPoolDiagnostics ()

  *Creates a new diagnostics object, ready to record Object Pool main events.*

### Properties

- bool Enabled [get, set]

  *Gets or sets whether this object can record data about how the Pool operates.*
- long TotalLiveInstancesCount [get]

  *Gets the total count of live instances, both in the pool and in use.*
- long ObjectResetFailedCount [get]

  *Gets the count of object reset failures occured while the pool tried to re-add the object into the pool.*
- long ReturnedToPoolByResurrectionCount [get]

  *Gets the total count of object that has been picked up by the GC, and returned to pool.*
- long PoolObjectHitCount [get]

  *Gets the total count of successful accesses. The pool had a spare object to provide to the user without creating it on demand.*
- long PoolObjectMissCount [get]

  *Gets the total count of unsuccessful accesses. The pool had to create an object in order to satisfy the user request. If the number is high, consider increasing the object minimum limit.*
- long TotalInstancesCreated [get]

  *Gets the total number of pooled objected created.*
- long TotalInstancesDestroyed [get]

  *Gets the total number of objects destroyes, both in case of an pool overflow, and state corruption.*
- long PoolOverflowCount [get]

  *Gets the number of objects been destroyed because the pool was full at the time of returning the object to the pool.*
- long ReturnedToPoolCount [get]

  *Gets the total count of objects that been successfully returned to the pool.*

### 6.4.1 Detailed Description

A simple class to track stats during execution. By default, this class does not record anything.

Definition at line 18 of file ObjectPoolDiagnostics.cs.

### 6.4.2 Constructor & Destructor Documentation

#### 6.4.2.1 CodeProject.ObjectPool.ObjectPoolDiagnostics.ObjectPoolDiagnostics ( )

Creates a new diagnostics object, ready to record Object Pool main events.

Definition at line 25 of file ObjectPoolDiagnostics.cs.

### 6.4.3 Property Documentation

#### 6.4.3.1 bool CodeProject.ObjectPool.ObjectPoolDiagnostics.Enabled [get], [set]

Gets or sets whether this object can record data about how the Pool operates.

Definition at line 48 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.2 long CodeProject.ObjectPool.ObjectPoolDiagnostics.ObjectResetFailedCount [get]

Gets the count of object reset failures occured while the pool tried to re-add the object into the pool.

Definition at line 63 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.3 long CodeProject.ObjectPool.ObjectPoolDiagnostics.PoolObjectHitCount [get]

Gets the total count of successful accesses. The pool had a spare object to provide to the user without creating it on demand.

Definition at line 80 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.4 long CodeProject.ObjectPool.ObjectPoolDiagnostics.PoolObjectMissCount [get]

Gets the total count of unsuccessful accesses. The pool had to create an object in order to satisfy the user request. If the number is high, consider increasing the object minimum limit.

Definition at line 90 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.5 long CodeProject.ObjectPool.ObjectPoolDiagnostics.PoolOverflowCount [get]

Gets the number of objects been destroyed because the pool was full at the time of returning the object to the pool.

Definition at line 116 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.6 long CodeProject.ObjectPool.ObjectPoolDiagnostics.ReturnedToPoolByResurrectionCount [get]

Gets the total count of object that has been picked up by the GC, and returned to pool.

Definition at line 71 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.7 long CodeProject.ObjectPool.ObjectPoolDiagnostics.ReturnedToPoolCount [get]

Gets the total count of objects that been successfully returned to the pool.

Definition at line 124 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.8 long CodeProject.ObjectPool.ObjectPoolDiagnostics.TotalInstancesCreated [get]

Gets the total number of pooled objected created.

Definition at line 98 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.9 long CodeProject.ObjectPool.ObjectPoolDiagnostics.TotalInstancesDestroyed [get]

Gets the total number of objects destroyes, both in case of an pool overflow, and state corruption.

Definition at line 107 of file ObjectPoolDiagnostics.cs.

#### 6.4.3.10 long CodeProject.ObjectPool.ObjectPoolDiagnostics.TotalLiveInstancesCount [get]

Gets the total count of live instances, both in the pool and in use.

Definition at line 54 of file ObjectPoolDiagnostics.cs.

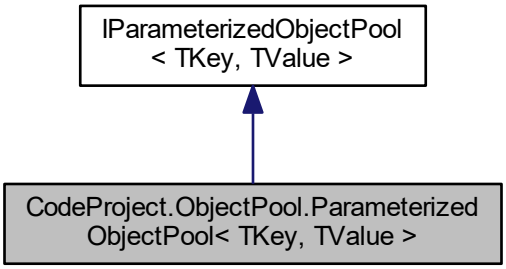
The documentation for this class was generated from the following file:

- ObjectPoolDiagnostics.cs

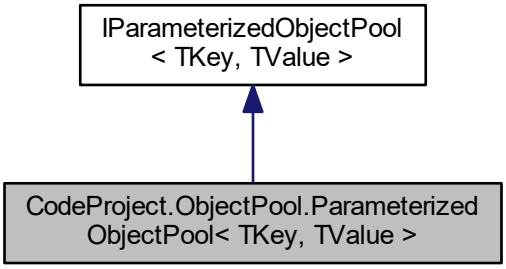
## 6.5 CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue > Class Template Reference

A parameterized version of the ObjectPool class.

Inheritance diagram for CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >:



Collaboration diagram for CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >:




## Public Member Functions

- ParameterizedObjectPool ()

  *Initializes a new pool with default settings.*
- ParameterizedObjectPool (int minimumPoolSize, int maximumPoolSize)

  *Initializes a new pool with specified minimum pool size and maximum pool size.*
- ParameterizedObjectPool (Func< TKey, TValue > factoryMethod)

  *Initializes a new pool with specified factory method.*
- ParameterizedObjectPool (int minimumPoolSize, int maximumPoolSize, Func< TKey, TValue > factory↩ Method)

  *Initializes a new pool with specified factory method and minimum and maximum size.*
- void Clear ()

  *Clears the parameterized pool and each inner pool stored inside it.*
- TValue GetObject (TKey key)

  *Gets an object linked to given key.*

## Public Attributes

- int KeysInPoolCount => _pools.Count

  *Gets the count of the keys currently handled by the pool.*

## Properties

- ObjectPoolDiagnostics Diagnostics `[get, set]`

  *Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.*
- int MaximumPoolSize `[get, set]`

  *Gets or sets the maximum number of objects that could be available at the same time in the pool.*
- int MinimumPoolSize `[get, set]`

  *Gets or sets the minimum number of objects in the pool.*
- Func< TKey, TValue > FactoryMethod `[get]`

  *Gets the Factory method that will be used for creating new objects.*

### 6.5.1 Detailed Description

A parameterized version of the ObjectPool class.

**Template Parameters**

| | |
|---|---|
| *TKey* | The type of the pool parameter. |
| *TValue* | The type of the objects stored in the pool. |

**Type Constraints**

***TValue*** **:** ***PooledObject***

Definition at line 22 of file ParameterizedObjectPool.cs.

### 6.5.2 Constructor & Destructor Documentation

#### 6.5.2.1 CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.ParameterizedObjectPool ( )

Initializes a new pool with default settings.

Definition at line 127 of file ParameterizedObjectPool.cs.

#### 6.5.2.2 CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.ParameterizedObjectPool ( int *minimumPoolSize*, int *maximumPoolSize* )

Initializes a new pool with specified minimum pool size and maximum pool size.

**Parameters**

| | |
|---|---|
| *minimumPool↩ Size* | The minimum pool size limit. |
| *maximumPool↩ Size* | The maximum pool size limit |

Definition at line 137 of file ParameterizedObjectPool.cs.

#### 6.5.2.3 CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.ParameterizedObjectPool ( Func< TKey, TValue > *factoryMethod* )

Initializes a new pool with specified factory method.

**Parameters**

| | |
|---|---|
| *factoryMethod* | The factory method that will be used to create new objects. |

Definition at line 146 of file ParameterizedObjectPool.cs.

#### 6.5.2.4 CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.ParameterizedObjectPool ( int *minimumPoolSize*, int *maximumPoolSize*, Func< TKey, TValue > *factoryMethod* )

Initializes a new pool with specified factory method and minimum and maximum size.

**Parameters**

| | |
|---|---|
| *minimumPool↩ Size* | The minimum pool size limit. |
| *maximumPool↩ Size* | The maximum pool size limit |

| | |
|---|---|
| *factoryMethod* | The factory method that will be used to create new objects. |

Definition at line 157 of file ParameterizedObjectPool.cs.

### 6.5.3 Member Function Documentation

#### 6.5.3.1 void CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.Clear ( )

Clears the parameterized pool and each inner pool stored inside it.

Definition at line 174 of file ParameterizedObjectPool.cs.

#### 6.5.3.2 TValue CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.GetObject ( TKey *key* )

Gets an object linked to given key.

**Parameters**

| | |
|---|---|
| *key* | The key linked to the object. |

**Returns**

The objects linked to given key.

Definition at line 194 of file ParameterizedObjectPool.cs.

### 6.5.4 Member Data Documentation

#### 6.5.4.1 int CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.KeysInPoolCount => _pools.Count

Gets the count of the keys currently handled by the pool.

Definition at line 118 of file ParameterizedObjectPool.cs.

### 6.5.5 Property Documentation

#### 6.5.5.1 ObjectPoolDiagnostics CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.Diagnostics [get], [set]

Gets or sets the Diagnostics class for the current Object Pool, whose goal is to record data about how the pool operates. By default, however, an object pool records anything, in order to be most efficient; in any case, you can enable it through the ObjectPoolDiagnostics.Enabled property.

Definition at line 63 of file ParameterizedObjectPool.cs.

#### 6.5.5.2 Func<TKey, TValue> CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.FactoryMethod [get]

Gets the Factory method that will be used for creating new objects.

Definition at line 113 of file ParameterizedObjectPool.cs.

#### 6.5.5.3 int CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.MaximumPoolSize [get], [set]

Gets or sets the maximum number of objects that could be available at the same time in the pool.

Definition at line 81 of file ParameterizedObjectPool.cs.

#### 6.5.5.4 int CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >.MinimumPoolSize [get], [set]

Gets or sets the minimum number of objects in the pool.

Definition at line 98 of file ParameterizedObjectPool.cs.

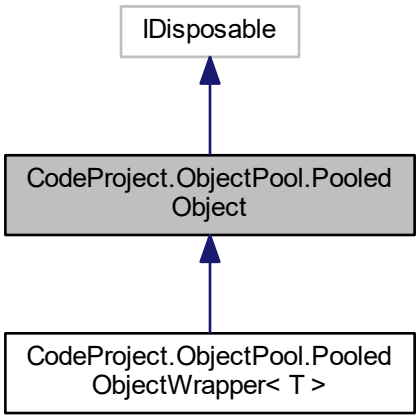
The documentation for this class was generated from the following file:

- ParameterizedObjectPool.cs

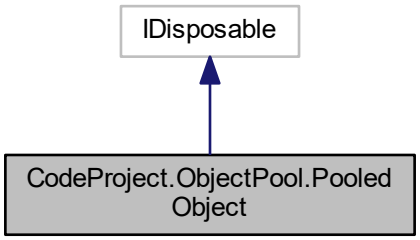
## 6.6 CodeProject.ObjectPool.PooledObject Class Reference

PooledObject base class.

Inheritance diagram for CodeProject.ObjectPool.PooledObject:




Collaboration diagram for CodeProject.ObjectPool.PooledObject:



### Public Member Functions

- void Dispose ()

  *See IDisposable docs.*

### Protected Member Functions

- virtual void OnResetState ()

  *Reset the object state to allow this object to be re-used by other parts of the application.*
- virtual void OnReleaseResources ()

  *Releases the object's resources*

#### 6.6.1 Detailed Description

PooledObject base class.

Definition at line 23 of file PooledObject.cs.

#### 6.6.2 Member Function Documentation

##### 6.6.2.1 void CodeProject.ObjectPool.PooledObject.Dispose ( )

See IDisposable docs.

Definition at line 109 of file PooledObject.cs.

##### 6.6.2.2 virtual void CodeProject.ObjectPool.PooledObject.OnReleaseResources ( ) [protected], [virtual]

Releases the object's resources

Reimplemented in CodeProject.ObjectPool.PooledObjectWrapper< T >.

Definition at line 98 of file PooledObject.cs.

##### 6.6.2.3 virtual void CodeProject.ObjectPool.PooledObject.OnResetState ( ) [protected], [virtual]

Reset the object state to allow this object to be re-used by other parts of the application.

Reimplemented in CodeProject.ObjectPool.PooledObjectWrapper< T >.

Definition at line 91 of file PooledObject.cs.

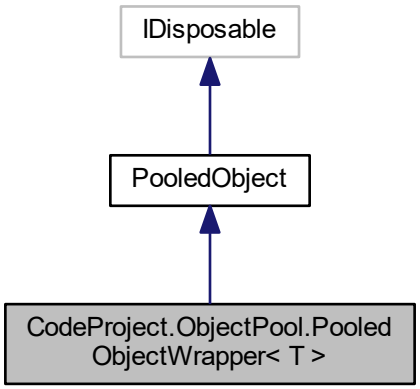
The documentation for this class was generated from the following file:

- PooledObject.cs

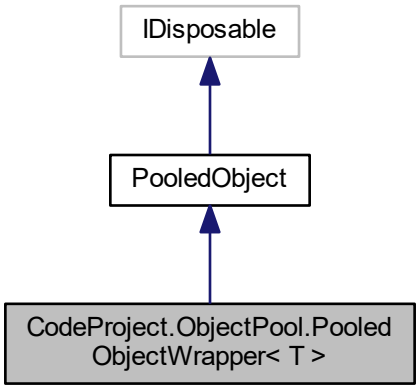
## 6.7 CodeProject.ObjectPool.PooledObjectWrapper< T > Class Template Reference

PooledObject wrapper, for classes which cannot inherit from that class.

Inheritance diagram for CodeProject.ObjectPool.PooledObjectWrapper< T >:




Collaboration diagram for CodeProject.ObjectPool.PooledObjectWrapper< T >:




## Public Member Functions

- PooledObjectWrapper (T resource)

  *Wraps a given resource so that it can be put in the pool.*

## Protected Member Functions

- override void OnReleaseResources ()

  *Triggers the WrapperReleaseResourcesAction, if any.*
- override void OnResetState ()

  *Triggers the WrapperResetStateAction, if any.*

### Properties

- Action< T > WrapperReleaseResourcesAction `[get, set]`

  *Triggered by the pool manager when there is no need for this object anymore.*

- Action< T > WrapperResetStateAction `[get, set]`

  *Triggered by the pool manager just before the object is being returned to the pool.*

- T InternalResource `[get]`

  *The resource wrapped inside this class.*

## 6.7.1 Detailed Description

PooledObject wrapper, for classes which cannot inherit from that class.

**Type Constraints**

***T* : *class***

Definition at line 151 of file PooledObject.cs.

## 6.7.2 Constructor & Destructor Documentation

### 6.7.2.1 CodeProject.ObjectPool.PooledObjectWrapper< T >.PooledObjectWrapper ( T *resource* )

Wraps a given resource so that it can be put in the pool.

**Parameters**

| | |
|---|---|
| *resource* | The resource to be wrapped. |

**Exceptions**

| | |
|---|---|
| *ArgumentNullException* | Given resource is null. |

Definition at line 158 of file PooledObject.cs.

## 6.7.3 Member Function Documentation

### 6.7.3.1 override void CodeProject.ObjectPool.PooledObjectWrapper< T >.OnReleaseResources ( ) `[protected], [virtual]`

Triggers the WrapperReleaseResourcesAction, if any.

Reimplemented from CodeProject.ObjectPool.PooledObject.

Definition at line 184 of file PooledObject.cs.

### 6.7.3.2 override void CodeProject.ObjectPool.PooledObjectWrapper< T >.OnResetState ( ) `[protected], [virtual]`

Triggers the WrapperResetStateAction, if any.

Reimplemented from CodeProject.ObjectPool.PooledObject.

Definition at line 196 of file PooledObject.cs.

### 6.7.4 Property Documentation

#### 6.7.4.1 T CodeProject.ObjectPool.PooledObjectWrapper< T >.InternalResource [get]

The resource wrapped inside this class.

Definition at line 179 of file PooledObject.cs.

#### 6.7.4.2 Action<T> CodeProject.ObjectPool.PooledObjectWrapper< T >.WrapperReleaseResourcesAction [get], [set]

Triggered by the pool manager when there is no need for this object anymore.

Definition at line 168 of file PooledObject.cs.

#### 6.7.4.3 Action<T> CodeProject.ObjectPool.PooledObjectWrapper< T >.WrapperResetStateAction [get], [set]

Triggered by the pool manager just before the object is being returned to the pool.

Definition at line 173 of file PooledObject.cs.

The documentation for this class was generated from the following file:

- PooledObject.cs

# Chapter 7

# File Documentation

## 7.1 Core/ErrorMessages.cs File Reference

### Classes

- class **CodeProject.ObjectPool.Core.ErrorMessages**

  *Static class containing all error messages used by ObjectPool.*

### Namespaces

- namespace CodeProject.ObjectPool.Core

## 7.2 ErrorMessages.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 namespace CodeProject.ObjectPool.Core
00012 {
00016     internal static class ErrorMessages
00017     {
00018         public const string NegativeMinimumPoolSize = "Minimum pool size must be greater or equals to zero.
    ";
00019         public const string NegativeOrZeroMaximumPoolSize = "Maximum pool size must be greater than zero.";
00020         public const string NullDiagnostics = "Pool diagnostics recorder cannot be null.";
00021         public const string NullResource = "Resource cannot be null.";
00022         public const string WrongCacheBounds = "Maximum pool size must be greater than the maximum pool
     size.";
00023     }
00024 }
```

## 7.3 IObjectPool.cs File Reference

### Classes

- interface CodeProject.ObjectPool.IObjectPool< out out T >

  *Describes all methods available on Object Pools.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.4 IObjectPool.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using System;
00012 using System.Diagnostics.Contracts;
00013
00014 namespace CodeProject.ObjectPool
00015 {
00020     public interface IObjectPool<out T> where T : PooledObject
00021     {
00028         [Pure]
00029         ObjectPoolDiagnostics Diagnostics { get; set; }
00030
00034         [Pure]
00035         Func<T> FactoryMethod { get; }
00036
00041         [Pure]
00042         int MaximumPoolSize { get; set; }
00043
00047         [Pure]
00048         int MinimumPoolSize { get; set; }
00049
00053         [Pure]
00054         int ObjectsInPoolCount { get; }
00055
00060         T GetObject();
00061     }
00062 }
```

## 7.5 IParameterizedObjectPool.cs File Reference

### Classes

- interface CodeProject.ObjectPool.IParameterizedObjectPool< in in TKey, out out TValue >

  *A parameterized version of the ObjectPool interface.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.6 IParameterizedObjectPool.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using System;
00012 using System.Diagnostics.Contracts;
00013
```

```
00014 namespace CodeProject.ObjectPool
00015 {
00021     public interface IParameterizedObjectPool<in TKey, out TValue>
00022     {
00029         [Pure]
00030         ObjectPoolDiagnostics Diagnostics { get; set; }
00031
00035         [Pure]
00036         Func<TKey, TValue> FactoryMethod { get; }
00037
00042         [Pure]
00043         int MaximumPoolSize { get; set; }
00044
00048         [Pure]
00049         int MinimumPoolSize { get; set; }
00050
00054         [Pure]
00055         int KeysInPoolCount { get; }
00056
00062         TValue GetObject(TKey key);
00063     }
00064 }
```

## 7.7 ObjectPool.cs File Reference

### Classes

- class CodeProject.ObjectPool.ObjectPool< T >

  *Generic object pool.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.8 ObjectPool.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using System;
00012 using System.Threading;
00013
00014 namespace CodeProject.ObjectPool
00015 {
00023     public sealed class ObjectPool<T> : IObjectPool<T> where T :
      PooledObject
00024     {
00025 #if PORTABLE
00026
00030         readonly Finsa.CodeServices.Common.Collections.Concurrent.ConcurrentQueue<T> _pooledObjects = new
      Finsa.CodeServices.Common.Collections.Concurrent.ConcurrentQueue<T>();
00031
00032 #else
00033
00037         readonly System.Collections.Concurrent.ConcurrentQueue<T> _pooledObjects = new
      System.Collections.Concurrent.ConcurrentQueue<T>();
00038
00039 #endif
00040
00046         int _adjustPoolSizeIsInProgressCasFlag; // 0 state false
00047
00051         readonly Action<PooledObject, bool> _returnToPoolAction;
00052
00053         int _maximumPoolSize;
00054         int _minimumPoolSize;
00055
```

```
00056         #region Public Properties
00057
00063         public ObjectPoolDiagnostics Diagnostics { get; set; }
00064
00068         public Func<T> FactoryMethod { get; }
00069
00074         public int MaximumPoolSize
00075         {
00076             get
00077             {
00078                 return _maximumPoolSize;
00079             }
00080             set
00081             {
00082                 ObjectPoolConstants.ValidatePoolLimits(MinimumPoolSize, value);
00083                 _maximumPoolSize = value;
00084                 AdjustPoolSizeToBounds();
00085             }
00086         }
00087
00091         public int MinimumPoolSize
00092         {
00093             get
00094             {
00095                 return _minimumPoolSize;
00096             }
00097             set
00098             {
00099                 ObjectPoolConstants.ValidatePoolLimits(value, MaximumPoolSize);
00100                 _minimumPoolSize = value;
00101                 AdjustPoolSizeToBounds();
00102             }
00103         }
00104
00108         public int ObjectsInPoolCount => _pooledObjects.Count;
00109
00110         #endregion Public Properties
00111
00112         #region C'tor and Initialization code
00113
00117         public ObjectPool()
00118             : this(ObjectPoolConstants.DefaultPoolMinimumSize, ObjectPoolConstants.DefaultPoolMaximumSize,
      null)
00119         {
00120         }
00121
00132         public ObjectPool(int minimumPoolSize, int maximumPoolSize)
00133             : this(minimumPoolSize, maximumPoolSize, null)
00134         {
00135         }
00136
00141         public ObjectPool(Func<T> factoryMethod)
00142             : this(ObjectPoolConstants.DefaultPoolMinimumSize, ObjectPoolConstants.DefaultPoolMaximumSize,
      factoryMethod)
00143         {
00144         }
00145
00157         public ObjectPool(int minimumPoolSize, int maximumPoolSize, Func<T> factoryMethod)
00158         {
00159             // Validating pool limits, exception is thrown if invalid
00160             ObjectPoolConstants.ValidatePoolLimits(minimumPoolSize, maximumPoolSize);
00161
00162             // Assigning properties
00163             FactoryMethod = factoryMethod;
00164             _maximumPoolSize = maximumPoolSize;
00165             _minimumPoolSize = minimumPoolSize;
00166
00167             // Creating a new instance for the Diagnostics class
00168             Diagnostics = new ObjectPoolDiagnostics();
00169
00170             // Setting the action for returning to the pool to be integrated in the pooled objects
00171             _returnToPoolAction = ReturnObjectToPool;
00172
00173             // Initilizing objects in pool
00174             AdjustPoolSizeToBounds();
00175         }
00176
00177         #endregion C'tor and Initialization code
00178
00179         #region Private Methods
00180
00181         internal void AdjustPoolSizeToBounds()
00182         {
00183             // If there is an Adjusting/Clear operation in progress, skip and return.
00184             if (Interlocked.CompareExchange(ref _adjustPoolSizeIsInProgressCasFlag, 1, 0) != 0)
00185             {
00186                 return;
```

```
00187             }
00188
00189             // If we reached this point, we've set the AdjustPoolSizeIsInProgressCASFlag to 1 (true)
00190             // using the above CAS function. We can now safely adjust the pool size without
00191             // interferences :)
00192
00193             // Adjusting...
00194             while (ObjectsInPoolCount < MinimumPoolSize)
00195             {
00196                 _pooledObjects.Enqueue(CreatePooledObject());
00197             }
00198
00199             while (ObjectsInPoolCount > MaximumPoolSize)
00200             {
00201                 T dequeuedObjectToDestroy;
00202                 if (_pooledObjects.TryDequeue(out dequeuedObjectToDestroy))
00203                 {
00204                     // Diagnostics update.
00205                     Diagnostics.IncrementPoolOverflowCount();
00206
00207                     DestroyPooledObject(dequeuedObjectToDestroy);
00208                 }
00209             }
00210
00211             // Finished adjusting, allowing additional callers to enter when needed.
00212             _adjustPoolSizeIsInProgressCasFlag = 0;
00213         }
00214
00215         T CreatePooledObject()
00216         {
00217             // Throws an exception if the type doesn't have default ctor - on purpose! I've could've
00218             // add a generic constraint with new (), but I didn't want to limit the user and force a
00219             // parameterless c'tor.
00220             var safeFactory = FactoryMethod;
00221             var newObject = (safeFactory != null) ? safeFactory() : Activator.CreateInstance<T>();
00222
00223             // Diagnostics update.
00224             Diagnostics.IncrementObjectsCreatedCount();
00225
00226             // Setting the 'return to pool' action in the newly created pooled object.
00227             newObject.ReturnToPool = _returnToPoolAction;
00228             return newObject;
00229         }
00230
00231         void DestroyPooledObject(PooledObject objectToDestroy)
00232         {
00233             // Making sure that the object is only disposed once (in case of application shutting
00234             // down and we don't control the order of the finalization).
00235             if (!objectToDestroy.Disposed)
00236             {
00237                 // Deterministically release object resources, nevermind the result, we are
00238                 // destroying the object.
00239                 objectToDestroy.ReleaseResources();
00240                 objectToDestroy.Disposed = true;
00241
00242                 // Diagnostics update.
00243                 Diagnostics.IncrementObjectsDestroyedCount();
00244             }
00245
00246             // The object is being destroyed, resources have been already released
00247             // deterministically, so we di no need the finalizer to fire.
00248             GC.SuppressFinalize(objectToDestroy);
00249         }
00250
00251         #endregion Private Methods
00252
00253         #region Pool Operations
00254
00258         public void Clear()
00259         {
00260             // If there is an Adjusting/Clear operation in progress, wait until it is done.
00261             while (Interlocked.CompareExchange(ref _adjustPoolSizeIsInProgressCasFlag, 1, 0) != 0)
00262             {
00263                 // Wait...
00264             }
00265
00266             // Destroy all objects.
00267             T dequeuedObjectToDestroy;
00268             while (_pooledObjects.TryDequeue(out dequeuedObjectToDestroy))
00269             {
00270                 DestroyPooledObject(dequeuedObjectToDestroy);
00271             }
00272
00273             // Finished clearing, allowing additional callers to enter when needed.
00274             _adjustPoolSizeIsInProgressCasFlag = 0;
00275         }
00276
```

```
00281         public T GetObject()
00282         {
00283             T dequeuedObject;
00284 
00285             if (_pooledObjects.TryDequeue(out dequeuedObject))
00286             {
00287                 AdjustPoolSizeToBounds();
00288 
00289                 // Diagnostics update.
00290                 Diagnostics.IncrementPoolObjectHitCount();
00291 
00292                 return dequeuedObject;
00293             }
00294 
00295             // This should not happen normally, but could be happening when there is stress on the
00296             // pool. No available objects in pool, create a new one and return it to the caller.
00297             Diagnostics.IncrementPoolObjectMissCount();
00298             return CreatePooledObject();
00299         }
00300 
00301         internal void ReturnObjectToPool(PooledObject objectToReturnToPool, bool
      reRegisterForFinalization)
00302         {
00303             var returnedObject = objectToReturnToPool as T;
00304 
00305             // Diagnostics update.
00306             if (reRegisterForFinalization)
00307             {
00308                 Diagnostics.IncrementObjectResurrectionCount();
00309             }
00310 
00311             // Checking that the pool is not full.
00312             if (ObjectsInPoolCount < MaximumPoolSize)
00313             {
00314                 // Reset the object state (if implemented) before returning it to the pool. If
00315                 // reseting the object have failed, destroy the object.
00316                 if (returnedObject != null && !returnedObject.ResetState())
00317                 {
00318                     // Diagnostics update.
00319                     Diagnostics.IncrementResetStateFailedCount();
00320 
00321                     DestroyPooledObject(returnedObject);
00322                     return;
00323                 }
00324 
00325                 // Re-registering for finalization - in case of resurrection (called from Finalize method).
00326                 if (reRegisterForFinalization)
00327                 {
00328                     GC.ReRegisterForFinalize(returnedObject);
00329                 }
00330 
00331                 // Diagnostics update.
00332                 Diagnostics.IncrementReturnedToPoolCount();
00333 
00334                 // Adding the object back to the pool.
00335                 _pooledObjects.Enqueue(returnedObject);
00336             }
00337             else
00338             {
00339                 // Diagnostics update.
00340                 Diagnostics.IncrementPoolOverflowCount();
00341 
00342                 // The Pool's upper limit has exceeded, there is no need to add this object back
00343                 // into the pool and we can destroy it.
00344                 DestroyPooledObject(returnedObject);
00345             }
00346         }
00347 
00348         #endregion Pool Operations
00349 
00350         #region Finalizer
00351 
00355         ~ObjectPool()
00356         {
00357             // The pool is going down, releasing the resources for all objects in pool.
00358             foreach (var item in _pooledObjects)
00359             {
00360                 DestroyPooledObject(item);
00361             }
00362         }
00363 
00364         #endregion Finalizer
00365     }
00366 }
```

## 7.9 ObjectPoolConstants.cs File Reference

### Classes

- class **CodeProject.ObjectPool.ObjectPoolConstants**

  *Constants for Object Pools.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.10 ObjectPoolConstants.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using CodeProject.ObjectPool.Core;
00012 using PommaLabs.Thrower;
00013 using System;
00014
00015 namespace CodeProject.ObjectPool
00016 {
00020     public static class ObjectPoolConstants
00021     {
00022         #region Constants
00023
00027         public const int DefaultPoolMinimumSize = 5;
00028
00032         public const int DefaultPoolMaximumSize = 100;
00033
00034         #endregion Constants
00035
00036         #region Validation
00037
00043         public static void ValidatePoolLimits(int minimumPoolSize, int maximumPoolSize)
00044         {
00045             Raise<ArgumentOutOfRangeException>.If(minimumPoolSize < 0, ErrorMessages.
      NegativeMinimumPoolSize);
00046             Raise<ArgumentOutOfRangeException>.If(maximumPoolSize < 1, ErrorMessages.
      NegativeOrZeroMaximumPoolSize);
00047             Raise<ArgumentOutOfRangeException>.If(minimumPoolSize > maximumPoolSize, ErrorMessages.
      WrongCacheBounds);
00048         }
00049
00050         #endregion Validation
00051     }
00052 }
```

## 7.11 ObjectPoolDiagnostics.cs File Reference

### Classes

- class CodeProject.ObjectPool.ObjectPoolDiagnostics

  *A simple class to track stats during execution. By default, this class does not record anything.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.12 ObjectPoolDiagnostics.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using System.Threading;
00012
00013 namespace CodeProject.ObjectPool
00014 {
00018     public class ObjectPoolDiagnostics
00019     {
00020         #region C'tor and Initialization code
00021
00025         public ObjectPoolDiagnostics()
00026         {
00027             // By default, diagnostics are disabled.
00028             Enabled = false;
00029         }
00030
00031         #endregion C'tor and Initialization code
00032
00033         #region Public Properties and backing fields
00034
00035         long _objectResetFailedCount;
00036         long _poolObjectHitCount;
00037         long _poolObjectMissCount;
00038         long _poolOverflowCount;
00039
00040         long _returnedToPoolByResurrectionCount;
00041         long _returnedToPoolCount;
00042         long _totalInstancesCreated;
00043         long _totalInstancesDestroyed;
00044
00048         public bool Enabled { get; set; }
00049
00053         public long TotalLiveInstancesCount
00054         {
00055             get { return _totalInstancesCreated - _totalInstancesDestroyed; }
00056         }
00057
00062         public long ObjectResetFailedCount
00063         {
00064             get { return _objectResetFailedCount; }
00065         }
00066
00070         public long ReturnedToPoolByResurrectionCount
00071         {
00072             get { return _returnedToPoolByResurrectionCount; }
00073         }
00074
00079         public long PoolObjectHitCount
00080         {
00081             get { return _poolObjectHitCount; }
00082         }
00083
00089         public long PoolObjectMissCount
00090         {
00091             get { return _poolObjectMissCount; }
00092         }
00093
00097         public long TotalInstancesCreated
00098         {
00099             get { return _totalInstancesCreated; }
00100         }
00101
00106         public long TotalInstancesDestroyed
00107         {
00108             get { return _totalInstancesDestroyed; }
00109         }
00110
00115         public long PoolOverflowCount
00116         {
00117             get { return _poolOverflowCount; }
00118         }
00119
00123         public long ReturnedToPoolCount
00124         {
00125             get { return _returnedToPoolCount; }
00126         }
```

```
00127
00128         #endregion Public Properties and backing fields
00129
00130         #region Protected Methods for incrementing the counters
00131
00135         protected internal virtual void IncrementObjectsCreatedCount()
00136         {
00137             if (Enabled)
00138             {
00139                 Interlocked.Increment(ref _totalInstancesCreated);
00140             }
00141         }
00142
00146         protected internal virtual void IncrementObjectsDestroyedCount()
00147         {
00148             if (Enabled)
00149             {
00150                 Interlocked.Increment(ref _totalInstancesDestroyed);
00151             }
00152         }
00153
00157         protected internal virtual void IncrementPoolObjectHitCount()
00158         {
00159             if (Enabled)
00160             {
00161                 Interlocked.Increment(ref _poolObjectHitCount);
00162             }
00163         }
00164
00168         protected internal virtual void IncrementPoolObjectMissCount()
00169         {
00170             if (Enabled)
00171             {
00172                 Interlocked.Increment(ref _poolObjectMissCount);
00173             }
00174         }
00175
00179         protected internal virtual void IncrementPoolOverflowCount()
00180         {
00181             if (Enabled)
00182             {
00183                 Interlocked.Increment(ref _poolOverflowCount);
00184             }
00185         }
00186
00190         protected internal virtual void IncrementResetStateFailedCount()
00191         {
00192             if (Enabled)
00193             {
00194                 Interlocked.Increment(ref _objectResetFailedCount);
00195             }
00196         }
00197
00201         protected internal virtual void IncrementObjectResurrectionCount()
00202         {
00203             if (Enabled)
00204             {
00205                 Interlocked.Increment(ref _returnedToPoolByResurrectionCount);
00206             }
00207         }
00208
00212         protected internal virtual void IncrementReturnedToPoolCount()
00213         {
00214             if (Enabled)
00215             {
00216                 Interlocked.Increment(ref _returnedToPoolCount);
00217             }
00218         }
00219
00220         #endregion Protected Methods for incrementing the counters
00221     }
00222 }
```

## 7.13 ParameterizedObjectPool.cs File Reference

### Classes

- class CodeProject.ObjectPool.ParameterizedObjectPool< TKey, TValue >

  *A parameterized version of the ObjectPool class.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.14 ParameterizedObjectPool.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010
00011 using System;
00012 using System.Diagnostics;
00013 using System.Linq;
00014
00015 namespace CodeProject.ObjectPool
00016 {
00022     public sealed class ParameterizedObjectPool<TKey, TValue> :
      IParameterizedObjectPool<TKey, TValue> where TValue :
      PooledObject
00023     {
00024 #if PORTABLE
00025
00026         readonly Finsa.CodeServices.Common.Collections.Concurrent.ConcurrentDictionary<TKey,
      ObjectPool<TValue>> _pools = new Finsa.CodeServices.Common.Collections.Concurrent.
      ConcurrentDictionary<TKey, ObjectPool<TValue>>();
00027
00028         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00029         bool TryAddToPools(TKey key, ObjectPool<TValue> value, out
      ObjectPool<TValue> foundValue)
00030         {
00031             return _pools.TryAdd(key, value, out foundValue);
00032         }
00033
00034 #else
00035
00036         readonly System.Collections.Concurrent.ConcurrentDictionary<TKey,
      ObjectPool<TValue>> _pools = new System.Collections.Concurrent.ConcurrentDictionary
      <TKey, ObjectPool<TValue>>();
00037
00038 #if (NET45 || NET46)
00039         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00040 #endif
00041         bool TryAddToPools(TKey key, ObjectPool<TValue> value, out
      ObjectPool<TValue> foundValue)
00042         {
00043             var added = false;
00044             foundValue = _pools.GetOrAdd(key, k => { added = true; return value; });
00045             return added;
00046         }
00047
00048 #endif
00049
00050         int _minimumPoolSize;
00051         int _maximumPoolSize;
00052         ObjectPoolDiagnostics _diagnostics;
00053
00054         #region Public Properties
00055
00062         public ObjectPoolDiagnostics Diagnostics
00063         {
00064             get { return _diagnostics; }
00065             set
00066             {
00067                 _diagnostics = value;
00068                 foreach (var p in _pools)
00069                 {
00070                     p.Value.Diagnostics = _diagnostics;
00071                 }
00072             }
00073         }
00074
00079         // ReSharper disable once ConvertToAutoProperty
00080         public int MaximumPoolSize
00081         {
```

```
00082             get
00083             {
00084                 return _maximumPoolSize;
00085             }
00086             set
00087             {
00088                 ObjectPoolConstants.ValidatePoolLimits(MinimumPoolSize, value);
00089                 _maximumPoolSize = value;
00090             }
00091         }
00092
00096         // ReSharper disable once ConvertToAutoProperty
00097         public int MinimumPoolSize
00098         {
00099             get
00100             {
00101                 return _minimumPoolSize;
00102             }
00103             set
00104             {
00105                 ObjectPoolConstants.ValidatePoolLimits(value, MaximumPoolSize);
00106                 _minimumPoolSize = value;
00107             }
00108         }
00109
00113         public Func<TKey, TValue> FactoryMethod { get; private set; }
00114
00118         public int KeysInPoolCount => _pools.Count;
00119
00120         #endregion Public Properties
00121
00122         #region C'tor and Initialization code
00123
00127         public ParameterizedObjectPool()
00128             : this(ObjectPoolConstants.DefaultPoolMinimumSize, ObjectPoolConstants.DefaultPoolMaximumSize,
      null)
00129         {
00130         }
00131
00137         public ParameterizedObjectPool(int minimumPoolSize, int maximumPoolSize)
00138             : this(minimumPoolSize, maximumPoolSize, null)
00139         {
00140         }
00141
00146         public ParameterizedObjectPool(Func<TKey, TValue> factoryMethod)
00147             : this(ObjectPoolConstants.DefaultPoolMinimumSize, ObjectPoolConstants.DefaultPoolMaximumSize,
      factoryMethod)
00148         {
00149         }
00150
00157         public ParameterizedObjectPool(int minimumPoolSize, int maximumPoolSize,
      Func<TKey, TValue> factoryMethod)
00158         {
00159             // Validating pool limits, exception is thrown if invalid
00160             ObjectPoolConstants.ValidatePoolLimits(minimumPoolSize, maximumPoolSize);
00161
00162             // Assigning properties
00163             Diagnostics = new ObjectPoolDiagnostics();
00164             FactoryMethod = factoryMethod;
00165             _maximumPoolSize = maximumPoolSize;
00166             _minimumPoolSize = minimumPoolSize;
00167         }
00168
00169         #endregion C'tor and Initialization code
00170
00174         public void Clear()
00175         {
00176             // Safe copy of the current pools.
00177             var innerPools = _pools.Values.ToArray();
00178
00179             // Clear the main pool.
00180             _pools.Clear();
00181
00182             // Then clear each pool, taking it from the safe copy.
00183             foreach (var innerPool in innerPools)
00184             {
00185                 innerPool.Clear();
00186             }
00187         }
00188
00194         public TValue GetObject(TKey key)
00195         {
00196             ObjectPool<TValue> pool;
00197
00198             if (!_pools.TryGetValue(key, out pool))
00199             {
00200                 // Initialize the new pool.
```

```
00201                 pool = new ObjectPool<TValue>(MinimumPoolSize, MaximumPoolSize,
      PrepareFactoryMethod(key));
00202                 ObjectPool<TValue> foundPool;
00203                 if (!TryAddToPools(key, pool, out foundPool))
00204                 {
00205                     // Someone added the pool in the meantime!
00206                     pool = foundPool;
00207                 }
00208                 else
00209                 {
00210                     // The new pool has been added, now we have to configure it.
00211                     pool.Diagnostics = _diagnostics;
00212                 }
00213             }
00214 
00215             Debug.Assert(pool != null);
00216             return pool.GetObject();
00217         }
00218 
00219         Func<TValue> PrepareFactoryMethod(TKey key)
00220         {
00221             var factory = FactoryMethod;
00222             if (factory == null)
00223             {
00224                 // Use the default parameterless constructor.
00225                 return null;
00226             }
00227             return () => factory(key);
00228         }
00229     }
00230 }
```

## 7.15 PooledObject.cs File Reference

### Classes

- class CodeProject.ObjectPool.PooledObject

  *PooledObject base class.*
- class CodeProject.ObjectPool.PooledObjectWrapper< T >

  *PooledObject wrapper, for classes which cannot inherit from that class.*

### Namespaces

- namespace CodeProject.ObjectPool

## 7.16 PooledObject.cs

```
00001 /*
00002  * Generic Object Pool Implementation
00003  *
00004  * Implemented by Ofir Makmal, 28/1/2013
00005  *
00006  * My Blog: Blogs.microsoft.co.il/blogs/OfirMakmal
00007  * Email:   Ofir.Makmal@gmail.com
00008  *
00009  */
00010 
00011 using CodeProject.ObjectPool.Core;
00012 using PommaLabs.Thrower;
00013 using System;
00014 using System.Diagnostics.Contracts;
00015 using System.Threading.Tasks;
00016 
00017 namespace CodeProject.ObjectPool
00018 {
00022     [Serializable]
00023     public abstract class PooledObject : IDisposable
00024     {
00025         #region Internal Properties
00026 
00031         internal Action<PooledObject, bool> ReturnToPool { get; set; }
00032 
```

```
00037         internal bool Disposed { get; set; }
00038
00039         #endregion Internal Properties
00040
00041         #region Internal Methods - resource and state management
00042
00048         internal bool ReleaseResources()
00049         {
00050             var successFlag = true;
00051
00052             try
00053             {
00054                 OnReleaseResources();
00055             }
00056             catch
00057             {
00058                 successFlag = false;
00059             }
00060
00061             return successFlag;
00062         }
00063
00068         internal bool ResetState()
00069         {
00070             var successFlag = true;
00071
00072             try
00073             {
00074                 OnResetState();
00075             }
00076             catch
00077             {
00078                 successFlag = false;
00079             }
00080
00081             return successFlag;
00082         }
00083
00084         #endregion Internal Methods - resource and state management
00085
00086         #region Virtual Template Methods - extending resource and state management
00087
00091         protected virtual void OnResetState()
00092         {
00093         }
00094
00098         protected virtual void OnReleaseResources()
00099         {
00100         }
00101
00102         #endregion Virtual Template Methods - extending resource and state management
00103
00104         #region Returning object to pool - Dispose and Finalizer
00105
00109         public void Dispose()
00110         {
00111             // Returning to pool
00112             HandleReAddingToPool(false);
00113         }
00114
00115         void HandleReAddingToPool(bool reRegisterForFinalization)
00116         {
00117             if (Disposed)
00118             {
00119                 return;
00120             }
00121             // If there is any case that the re-adding to the pool failes, release the resources and
00122             // set the internal Disposed flag to true
00123             try
00124             {
00125                 // Notifying the pool that this object is ready for re-adding to the pool.
00126                 ReturnToPool(this, reRegisterForFinalization);
00127             }
00128             catch
00129             {
00130                 Disposed = true;
00131                 ReleaseResources();
00132             }
00133         }
00134
00138         ~PooledObject()
00139         {
00140             // Resurrecting the object
00141             HandleReAddingToPool(true);
00142         }
00143
00144         #endregion Returning object to pool - Dispose and Finalizer
```

```
00145     }
00146
00150     [Serializable]
00151     public sealed class PooledObjectWrapper<T> : PooledObject where T :
      class
00152     {
00158         public PooledObjectWrapper(T resource)
00159         {
00160             RaiseArgumentNullException.IfIsNull(resource, nameof(resource), ErrorMessages.NullResource);
00161             // Setting the internal resource
00162             InternalResource = resource;
00163         }
00164
00168         public Action<T> WrapperReleaseResourcesAction { get; set; }
00169
00173         public Action<T> WrapperResetStateAction { get; set; }
00174
00178         [Pure]
00179         public T InternalResource { get; }
00180
00184         protected override void OnReleaseResources()
00185         {
00186             var safeAction = WrapperReleaseResourcesAction;
00187             if (safeAction != null)
00188             {
00189                 safeAction(InternalResource);
00190             }
00191         }
00192
00196         protected override void OnResetState()
00197         {
00198             var safeAction = WrapperResetStateAction;
00199             if (safeAction != null)
00200             {
00201                 safeAction(InternalResource);
00202             }
00203         }
00204     }
00205 }
```

# Index