PommaLabs.Thrower

2.2.1

Generated by Doxygen 1.8.10

Fri Feb 26 2016 11:56:17

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 PommaLabs Namespace Reference

Namespaces

- namespace Thrower

5.2 PommaLabs.Thrower Namespace Reference

Classes

- class HttpException

    Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.
- struct HttpExceptionInfo

    Additional info which will be included into HttpException.
- class Raise

    Contains methods that throw specified exception TEx if given conditions will be verified.
- class RaiseArgumentException

    Utility methods which can be used to handle bad arguments.
- class RaiseArgumentNullException

    Utility methods which can be used to handle null references.
- class RaiseArgumentOutOfRangeException

    Utility methods which can be used to handle ranges.
- class RaiseBase

    Stores items shared by various Raise<TEx> instances.
- class RaiseHttpException

    Utility methods which can be used to handle error codes through HTTP.
- class RaiseIndexOutOfRangeException

    Utility methods which can be used to handle indexes.
- class RaiseInvalidOperationException

    Utility methods which can be used to handle bad object states.
- class RaiseNotSupportedException

    Utility methods which can be used to handle unsupported operations.
- class RaiseObjectDisposedException

    Utility methods which can be used to handle bad object states.
- class ThrowerException

Exception thrown by Raise<TEx> when the type parameter passed to that class has something invalid (missing constructors, etc).

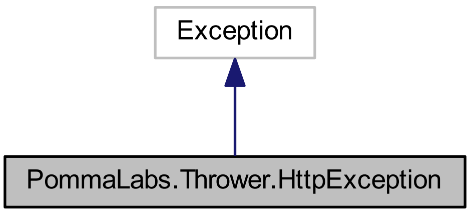
Chapter 6

Class Documentation

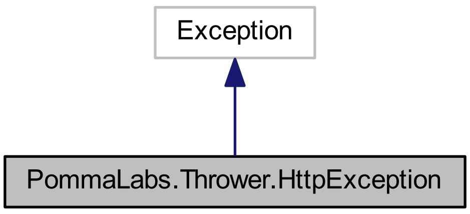
6.1 PommaLabs.Thrower.HttpException Class Reference

Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.

Inheritance diagram for PommaLabs.Thrower.HttpException:



Collaboration diagram for PommaLabs.Thrower.HttpException:



Public Member Functions

- HttpException (HttpStatusCode httpStatusCode)

  Builds the exception using given status code.
- HttpException (HttpStatusCode httpStatusCode, HttpExceptionInfo additionalInfo)

  Builds the exception using given status code.
- HttpException (HttpStatusCode httpStatusCode, string message)

  Builds the exception using given status code and message.
- HttpException (HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)

  Builds the exception using given status code, message and error code.
- HttpException (HttpStatusCode httpStatusCode, string message, Exception innerException)

  Builds the exception using given status code, message and inner exception.
- HttpException (HttpStatusCode httpStatusCode, string message, Exception innerException, HttpException↩ Info additionalInfo)

  Builds the exception using given status code, message, error code and inner exception.

Properties

- HttpStatusCode HttpStatusCode `[get]`

  The HTTP status code assigned to this exception.
- object ErrorCode `[get]`

  The application defined error code.
- static object DefaultErrorCode `[get, set]`

  The default application defined error code, used when none has been specified.
- string UserMessage = "unspecified" `[get]`

  An error message which can be shown to the user.
- static string DefaultUserMessage `[get, set]`

  The default user message.

6.1.1 Detailed Description

Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.

Definition at line 143 of file RaiseHttpException.cs.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*)

Builds the exception using given status code.

Parameters

httpStatusCode	The HTTP status code.

Definition at line 149 of file RaiseHttpException.cs.

6.1.2.2 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*, **HttpExceptionInfo** *additionalInfo*)

Builds the exception using given status code.

Parameters

httpStatusCode	The HTTP status code.
additionalInfo	Additional exception info.

Definition at line 159 of file RaiseHttpException.cs.

6.1.2.3 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*, string *message*)

Builds the exception using given status code and message.

Parameters

httpStatusCode	The HTTP status code.
message	The exception message.

Definition at line 172 of file RaiseHttpException.cs.

6.1.2.4 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*, string *message*, **HttpExceptionInfo** *additionalInfo*)

Builds the exception using given status code, message and error code.

Parameters

httpStatusCode	The HTTP status code.
message	The exception message.
additionalInfo	Additional exception info.

Definition at line 183 of file RaiseHttpException.cs.

6.1.2.5 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*, string *message*, Exception *innerException*)

Builds the exception using given status code, message and inner exception.

Parameters

httpStatusCode	The HTTP status code.
message	The exception message.
innerException	The inner exception.

Definition at line 197 of file RaiseHttpException.cs.

6.1.2.6 PommaLabs.Thrower.HttpException.HttpException (**HttpStatusCode** *httpStatusCode*, string *message*, Exception *innerException*, **HttpExceptionInfo** *additionalInfo*)

Builds the exception using given status code, message, error code and inner exception.

Parameters

httpStatusCode	The HTTP status code.
message	The exception message.
innerException	The inner exception.
additionalInfo	Additional exception info.

Definition at line 209 of file RaiseHttpException.cs.

6.1.3 Property Documentation

6.1.3.1 object PommaLabs.Thrower.HttpException.DefaultErrorCode `[static], [get], [set]`

The default application defined error code, used when none has been specified.

Definition at line 230 of file RaiseHttpException.cs.

6.1.3.2 string PommaLabs.Thrower.HttpException.DefaultUserMessage `[static], [get], [set]`

The default user message.

Definition at line 240 of file RaiseHttpException.cs.

6.1.3.3 object PommaLabs.Thrower.HttpException.ErrorCode `[get]`

The application defined error code.

Definition at line 225 of file RaiseHttpException.cs.

6.1.3.4 HttpStatusCode PommaLabs.Thrower.HttpException.HttpStatusCode `[get]`

The HTTP status code assigned to this exception.

Definition at line 220 of file RaiseHttpException.cs.

6.1.3.5 string PommaLabs.Thrower.HttpException.UserMessage = "unspecified" `[get]`

An error message which can be shown to the user.

Definition at line 235 of file RaiseHttpException.cs.

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

- RaiseHttpException.cs

6.2 PommaLabs.Thrower.HttpExceptionInfo Struct Reference

Additional info which will be included into HttpException.

Public Member Functions

- HttpExceptionInfo (object errorCode=null, string userMessage=null)

  Builds the additional exception info.

Properties

- object ErrorCode `[get, set]`

  The application defined error code.
- string UserMessage `[get, set]`

  An error message which can be shown to user.

6.2.1 Detailed Description

Additional info which will be included into HttpException.

Definition at line 113 of file RaiseHttpException.cs.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 PommaLabs.Thrower.HttpExceptionInfo.HttpExceptionInfo (object *errorCode* = `null`, string *userMessage* = `null`)

Builds the additional exception info.

Parameters

errorCode	The application defined error code.
userMessage	The user message.

Definition at line 120 of file RaiseHttpException.cs.

6.2.3 Property Documentation

6.2.3.1 object PommaLabs.Thrower.HttpExceptionInfo.ErrorCode `[get]`, `[set]`

The application defined error code.

Definition at line 130 of file RaiseHttpException.cs.

6.2.3.2 string PommaLabs.Thrower.HttpExceptionInfo.UserMessage `[get]`, `[set]`

An error message which can be shown to user.

Definition at line 136 of file RaiseHttpException.cs.

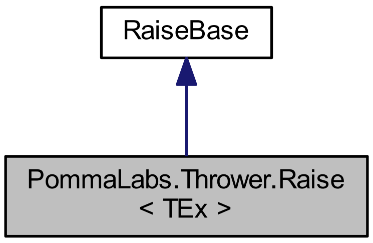
The documentation for this struct was generated from the following file:

- RaiseHttpException.cs

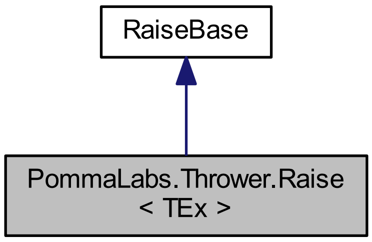
6.3 PommaLabs.Thrower.Raise< TEx > Class Template Reference

Contains methods that throw specified exception *TEx* if given conditions will be verified.

Inheritance diagram for PommaLabs.Thrower.Raise< TEx >:



Collaboration diagram for PommaLabs.Thrower.Raise< TEx >:



Static Public Member Functions

- static void If (bool cond)

  Throws an exception of type TEx if and only if specified condition is true.
- static void If (bool cond, string message)

  Throws an exception of type TEx with given message message if and only if specified condition is true.
- static void IfNot (bool cond)

  Throws an exception of type TEx if and only if specified condition is false.
- static void IfNot (bool cond, string message)

  Throws an exception of type TEx with given message message if and only if specified condition is false.
- static void IfAreEqual< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)

  Throws an exception of type TEx if and only if specified arguments are equal.
- static void IfAreEqual< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments are equal.
- static void IfAreNotEqual< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)

  Throws an exception of type TEx if and only if specified arguments are not equal.
- static void IfAreNotEqual< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments are not equal.
- static void IfAreSame< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)

  Throws an exception of type TEx if and only if specified arguments point to the same object.
- static void IfAreSame< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments point to the same object.
- static void IfAreNotSame< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)

  Throws an exception of type TEx if and only if specified arguments do not point to the same object.
- static void IfAreNotSame< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments do not point to the same object.
- static void IfIsAssignableFrom (object instance, Type type)

  Throws an exception of type TEx if and only if an instance of given type can be assigned to specified object.
- static void IfIsAssignableFrom (object instance, Type type, string message)

  Throws an exception of type TEx with given message message if and only if an instance of given type can be assigned to specified object.
- static void IfIsAssignableFrom< TType > (object instance)

  Throws an exception of type TEx if and only if an instance of given type can be assigned to specified object.

- static void IfIsAssignableFrom< TType > (object instance, string message)

  Throws an exception of type TEx with given message message if and only if an instance of given type can be assigned to specified object.
- static void IfIsNotAssignableFrom (object instance, Type type)

  Throws an exception of type TEx if and only if an instance of given type cannot be assigned to specified object.
- static void IfIsNotAssignableFrom (object instance, Type type, string message)

  Throws an exception of type TEx with given message message if and only if an instance of given type cannot be assigned to specified object.
- static void IfIsNotAssignableFrom< TType > (object instance)

  Throws an exception of type TEx if and only if an instance of given type cannot be assigned to specified object.
- static void IfIsNotAssignableFrom< TType > (object instance, string message)

  Throws an exception of type TEx with given message message if and only if an instance of given type cannot be assigned to specified object.
- static void IfIsContainedIn (object argument, System.Collections.IList collection)

  Throws an exception of type TEx if and only if specified argument is contained in given collection.
- static void IfIsContainedIn (object argument, System.Collections.IList collection, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is contained in given collection.
- static void IfIsNotContainedIn (object argument, System.Collections.IList collection)

  Throws an exception of type TEx if and only if specified argument is not contained in given collection.
- static void IfIsNotContainedIn (object argument, System.Collections.IList collection, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is not contained in given collection.
- static void IfIsContainedIn< TArg > (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)

  Throws an exception of type TEx if and only if specified argument is contained in given collection.
- static void IfIsContainedIn< TArg > (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is contained in given collection.
- static void IfIsNotContainedIn< TArg > (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)

  Throws an exception of type TEx if and only if specified argument is not contained in given collection.
- static void IfIsNotContainedIn< TArg > (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is not contained in given collection.
- static void IfIsContainedIn< TArg > (TArg arg, System.Collections.IDictionary dictionary)

  Throws an exception of type TEx if and only if specified argument is contained in given dictionary keys.
- static void IfIsContainedIn< TArg > (TArg arg, System.Collections.IDictionary dictionary, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is contained in given dictionary keys.
- static void IfIsNotContainedIn< TArg > (TArg arg, System.Collections.IDictionary dictionary)

  Throws an exception of type TEx if and only if specified argument is not contained in given dictionary keys.
- static void IfIsNotContainedIn< TArg > (TArg arg, System.Collections.IDictionary dictionary, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is not contained in given dictionary keys.
- static void IfIsContainedIn< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, System.Collections.Generic.I↩ Dictionary< TArg1, TArg2 > dictionary)

  Throws an exception of type TEx if and only if specified arguments are contained in given dictionary pairs.
- static void IfIsContainedIn< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, System.Collections.Generic.I↩ Dictionary< TArg1, TArg2 > dictionary, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments are contained in given dictionary pairs.

- static void IfIsNotContainedIn< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, System.Collections.Generic.I↩ Dictionary< TArg1, TArg2 > dictionary)

  Throws an exception of type TEx if and only if specified arguments are not contained in given dictionary pairs.
- static void IfIsNotContainedIn< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, System.Collections.Generic.I↩ Dictionary< TArg1, TArg2 > dictionary, string message)

  Throws an exception of type TEx with given message message if and only if specified arguments are not contained in given dictionary pairs.
- static void IfIsEmpty (string valueToCheck)

  Throws an exception of type TEx if and only if specified string is is null, empty, or consists only of white-space characters.
- static void IfIsEmpty (string valueToCheck, string message)

  Throws an exception of type TEx with given message message if and only if specified string is is null, empty, or consists only of white-space characters.
- static void IfIsNotEmpty (string valueToCheck)

  Throws an exception of type TEx if and only if specified string is not null, empty, or does not consist only of white-space characters.
- static void IfIsNotEmpty (string valueToCheck, string message)

  Throws an exception of type TEx with given message message if and only if specified string is not null, empty, or does not consist only of white-space characters.
- static void IfIsEmpty (System.Collections.ICollection collection)

  Throws an exception of type TEx if and only if specified collection is null or empty.
- static void IfIsEmpty (System.Collections.ICollection collection, string message)

  Throws an exception of type TEx with given message message if and only if specified collection is null or empty.
- static void IfIsNotEmpty (System.Collections.ICollection collection)

  Throws an exception of type TEx if and only if specified collection is null or not empty.
- static void IfIsNotEmpty (System.Collections.ICollection collection, string message)

  Throws an exception of type TEx with given message message if and only if specified collection is null or not empty.
- static void IfIsEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > collection)

  Throws an exception of type TEx if and only if specified collection is null or empty.
- static void IfIsEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > collection, string message)

  Throws an exception of type TEx with given message message if and only if specified collection is null or empty.
- static void IfIsNotEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > collection)

  Throws an exception of type TEx if and only if specified collection is null or not empty.
- static void IfIsNotEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > collection, string message)

  Throws an exception of type TEx with given message message if and only if specified collection is null or not empty.
- static void IfIsInstanceOf (object instance, Type type)

  Throws an exception of type TEx if and only if specified object has given type.
- static void IfIsInstanceOf (object instance, Type type, string message)

  Throws an exception of type TEx with given message message if and only if specified object has given type.
- static void IfIsInstanceOf< TType > (object instance)

  Throws an exception of type TEx if and only if specified object has given type.
- static void IfIsInstanceOf< TType > (object instance, string message)

  Throws an exception of type TEx with given message message if and only if specified object has given type.
- static void IfIsNotInstanceOf (object instance, Type type)

  Throws an exception of type TEx if and only if specified object has not given type.
- static void IfIsNotInstanceOf (object instance, Type type, string message)

  Throws an exception of type TEx with given message message if and only if specified object has not given type.
- static void IfIsNotInstanceOf< TType > (object instance)

  Throws an exception of type TEx if and only if specified object has not given type.
- static void IfIsNotInstanceOf< TType > (object instance, string message)

Throws an exception of type TEx with given message message if and only if specified object has not given type.

- static void IfIsNaN (double number)

  Throws an exception of type TEx if and only if specified double is double.NaN.
- static void IfIsNaN (double number, string message)

  Throws an exception of type TEx with given message message if and only if specified double is double.NaN.
- static void IfIsNotNaN (double number)

  Throws an exception of type TEx if and only if specified double is not double.NaN.
- static void IfIsNotNaN (double number, string message)

  Throws an exception of type TEx with given message message if and only if specified double is not double.NaN.
- static void IfIsNull< TArg > (TArg arg)

  Throws an exception of type TEx if and only if specified argument is null.
- static void IfIsNull< TArg > (TArg arg, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is null.
- static void IfIsNotNull< TArg > (TArg arg)

  Throws an exception of type TEx if and only if specified argument is not null.
- static void IfIsNotNull< TArg > (TArg arg, string message)

  Throws an exception of type TEx with given message message if and only if specified argument is not null.

Additional Inherited Members

6.3.1 Detailed Description

Contains methods that throw specified exception *TEx* if given conditions will be verified.

Template Parameters

TEx	The type of the exceptions thrown if conditions will be satisfied.

In order to achieve a good speed, the class caches an instance of the constructors found via reflection; therefore, constructors are looked for only once.

Type Constraints

TEx* : *Exception

Definition at line 70 of file Raise.cs.

6.3.2 Member Function Documentation

6.3.2.1 static void PommaLabs.Thrower.Raise< TEx >.If (bool *cond*) [static]

Throws an exception of type *TEx* if and only if specified condition is true.

Parameters

cond	The condition that determines whether an exception will be thrown.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *cond* is true, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 132 of file Raise.cs.

6.3.2.2 static void PommaLabs.Thrower.Raise< TEx >.If (bool *cond*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified condition is true.

Parameters

cond	The condition that determines whether an exception will be thrown.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *cond* is true, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 166 of file Raise.cs.

6.3.2.3 static void **PommaLabs.Thrower.Raise**< TEx >.IfAreEqual< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2*) [static]

Throws an exception of type *TEx* if and only if specified arguments are equal.

Parameters

arg1	First argument to test for equality.
arg2	Second argument to test for equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If arguments are equal, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 255 of file Raise.cs.

6.3.2.4 static void **PommaLabs.Thrower.Raise**< TEx >.IfAreEqual< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified arguments are equal.

Parameters

arg1	First argument to test for equality.
arg2	Second argument to test for equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If arguments are equal, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 290 of file Raise.cs.

6.3.2.5 static void PommaLabs.Thrower.Raise< TEx >.IfAreNotEqual< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2*) [static]

Throws an exception of type *TEx* if and only if specified arguments are not equal.

Parameters

arg1	First argument to test for equality.
arg2	Second argument to test for equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If arguments are not equal, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 318 of file Raise.cs.

6.3.2.6 static void PommaLabs.Thrower.Raise< TEx >.IfAreNotEqual< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified arguments are not equal.

Parameters

arg1	First argument to test for equality.
arg2	Second argument to test for equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If arguments are not equal, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 353 of file Raise.cs.

6.3.2.7 static void PommaLabs.Thrower.Raise< TEx >.IfAreNotSame< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2*) [static]

Throws an exception of type *TEx* if and only if specified arguments do not point to the same object.

Parameters

arg1	First argument to test for reference equality.
arg2	Second argument to test for reference equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If arguments do not point to the same object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 444 of file Raise.cs.

6.3.2.8 static void PommaLabs.Thrower.Raise< TEx >.IfAreNotSame< TArg1, TArg2 > (TArg1 *arg1*, TArg2 *arg2*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified arguments do not point to the same object.

Parameters

arg1	First argument to test for reference equality.
arg2	Second argument to test for reference equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If arguments do not point to the same object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 479 of file Raise.cs.

6.3.2.9 static void **PommaLabs.Thrower.Raise**< TEx >.IfAreSame< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2*) `[static]`

Throws an exception of type *TEx* if and only if specified arguments point to the same object.

Parameters

arg1	First argument to test for reference equality.
arg2	Second argument to test for reference equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If arguments point to the same object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 381 of file Raise.cs.

6.3.2.10 static void **PommaLabs.Thrower.Raise**< TEx >.IfAreSame< TArg1, TArg2 > (TArg1 *arg1,* TArg2 *arg2,* string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified arguments point to the same object.

Parameters

arg1	First argument to test for reference equality.
arg2	Second argument to test for reference equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If arguments point to the same object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 416 of file Raise.cs.

6.3.2.11 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsAssignableFrom (object *instance*, Type *type*) [static]

Throws an exception of type *TEx* if and only if an instance of given type can be assigned to specified object.

Parameters

instance	The object to test.
type	The type whose instance must be assigned to given object.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If an instance of given type can be assigned to specified object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 507 of file Raise.cs.

6.3.2.12 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsAssignableFrom (object *instance*, Type *type*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if an instance of given type can be assigned to specified object.

Parameters

instance	The object to test.
type	The type whose instance must be assigned to given object.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If an instance of given type can be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 543 of file Raise.cs.

6.3.2.13 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsAssignableFrom**< TType > (object *instance*) [static]

Throws an exception of type *TEx* if and only if an instance of given type can be assigned to specified object.

Template Parameters

TType	The type whose instance must be assigned to given object.

Parameters

instance	The object to test.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If an instance of given type can be assigned to specified object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 572 of file Raise.cs.

6.3.2.14 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsAssignableFrom**< TType > (object *instance,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if an instance of given type can be assigned to specified object.

Template Parameters

TType	The type whose instance must be assigned to given object.

Parameters

instance	The object to test.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If an instance of given type can be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 609 of file Raise.cs.

6.3.2.15 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsContainedIn (object *argument,* System.Collections.IList *collection*) [static]

Throws an exception of type *TEx* if and only if specified argument is contained in given collection.

Parameters

argument	The argument to check.
collection	The collection that must not contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *argument* is contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 773 of file Raise.cs.

6.3.2.16 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsContainedIn (object *argument,* System.Collections.IList *collection,* string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified argument is contained in given collection.

Parameters

argument	The argument to check.
collection	The collection that must not contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *argument* is contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 808 of file Raise.cs.

6.3.2.17 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsContainedIn**< TArg > (TArg *arg,* System.Collections.Generic.IEnumerable< TArg > *collection*) `[static]`

Throws an exception of type *TEx* if and only if specified argument is contained in given collection.

Parameters

arg	The argument to check.
collection	The collection that must not contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 899 of file Raise.cs.

6.3.2.18 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsContainedIn**< TArg > (TArg *arg,* System.Collections.Generic.IEnumerable< TArg > *collection,* string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified argument is contained in given collection.

Parameters

arg	The argument to check.
collection	The collection that must not contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 934 of file Raise.cs.

6.3.2.19 static void **PommaLabs.Thrower.Raise< TEx >.IfIsContainedIn< TArg >** (TArg *arg*, System.Collections.IDictionary *dictionary*) [static]

Throws an exception of type *TEx* if and only if specified argument is contained in given dictionary keys.

Parameters

arg	The argument to check.
dictionary	The dictionary that must not contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1025 of file Raise.cs.

6.3.2.20 static void **PommaLabs.Thrower.Raise< TEx >.IfIsContainedIn< TArg >** (TArg *arg*, System.Collections.IDictionary *dictionary*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified argument is contained in given dictionary keys.

Parameters

arg	The argument to check.
dictionary	The dictionary that must not contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1060 of file Raise.cs.

6.3.2.21 static void PommaLabs.Thrower.Raise< TEx >.IfIsContainedIn< TArg1, TArg2 > (TArg1 *arg1*, TArg2 *arg2*, System.Collections.Generic.IDictionary< TArg1, TArg2 > *dictionary*) [static]

Throws an exception of type *TEx* if and only if specified arguments are contained in given dictionary pairs.

Parameters

arg1	The key argument to check.
arg2	The value argument to check.
dictionary	The dictionary that must not contain given arguments.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg1* and *arg2* are contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1153 of file Raise.cs.

6.3.2.22 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsContainedIn**< TArg1, TArg2 > (TArg1 *arg1*, TArg2 *arg2*, System.Collections.Generic.IDictionary< TArg1, TArg2 > *dictionary*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified arguments are contained in given dictionary pairs.

Parameters

arg1	The key argument to check.
arg2	The value argument to check.
dictionary	The dictionary that must not contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg1* and *arg2* are contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1190 of file Raise.cs.

6.3.2.23 static void **PommaLabs.Thrower.Raise**< TEx >.**IfIsEmpty** (string *valueToCheck*) [static]

Throws an exception of type *TEx* if and only if specified string is is null, empty, or consists only of white-space characters.

Parameters

valueToCheck	The string to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *valueToCheck* is empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1286 of file Raise.cs.

6.3.2.24 static void PommaLabs.Thrower.Raise< TEx >.IfIsEmpty (string *valueToCheck*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified string is is null, empty, or consists only of white-space characters.

Parameters

valueToCheck	The string to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *valueToCheck* is empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1321 of file Raise.cs.

6.3.2.25 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsEmpty (System.Collections.ICollection *collection*) [static]

Throws an exception of type *TEx* if and only if specified collection is null or empty.

Parameters

collection	The collection to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *collection* is null or empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1410 of file Raise.cs.

6.3.2.26 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsEmpty (System.Collections.ICollection *collection*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified collection is null or empty.

Parameters

collection	The collection to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *collection* is null or empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1444 of file Raise.cs.

6.3.2.27 static void PommaLabs.Thrower.Raise< TEx >.IfIsEmpty< TArg > (System.Collections.Generic.I↩ Enumerable< TArg > *collection*) `[static]`

Throws an exception of type *TEx* if and only if specified collection is null or empty.

Parameters

collection	The collection to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *collection* is null or empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1532 of file Raise.cs.

6.3.2.28 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsEmpty< TArg > (System.Collections.Generic.I↩ Enumerable< TArg > *collection*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified collection is null or empty.

Parameters

collection	The collection to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *collection* is null or empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1566 of file Raise.cs.

6.3.2.29 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsInstanceOf (object *instance*, Type *type*) [static]

Throws an exception of type *TEx* if and only if specified object has given type.

Parameters

instance	The object to test.
type	The type the object must have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *instance* has given type, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1655 of file Raise.cs.

6.3.2.30 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsInstanceOf (object *instance*, Type *type*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified object has given type.

Parameters

instance	The object to test.
type	The type the object must have.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *instance* has given type, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1690 of file Raise.cs.

6.3.2.31 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsInstanceOf**< TType > (object *instance*) [static]

Throws an exception of type *TEx* if and only if specified object has given type.

Template Parameters

TType	The type the object must have.

Parameters

instance	The object to test.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *instance* has given type, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1719 of file Raise.cs.

6.3.2.32 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsInstanceOf**< TType > (object *instance,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified object has given type.

Template Parameters

TType	The type the object must have.

Parameters

instance	The object to test.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *instance* has given type, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments,

or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1755 of file Raise.cs.

6.3.2.33 static void PommaLabs.Thrower.Raise< TEx >.IfIsNaN (double *number*) [static]

Throws an exception of type *TEx* if and only if specified double is double.NaN.

Parameters

number	The double to test for double.NaN equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *number* is double.NaN, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1910 of file Raise.cs.

6.3.2.34 static void PommaLabs.Thrower.Raise< TEx >.IfIsNaN (double *number,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified double is double.NaN.

Parameters

number	The double to test for double.NaN equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *number* is double.NaN, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1944 of file Raise.cs.

6.3.2.35 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotAssignableFrom (object *instance,* Type *type*) [static]

Throws an exception of type *TEx* if and only if an instance of given type cannot be assigned to specified object.

Parameters

instance	The object to test.
type	The type whose instance must not be assigned to given object.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 637 of file Raise.cs.

6.3.2.36 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotAssignableFrom (object *instance,* Type *type,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if an instance of given type cannot be assigned to specified object.

Parameters

instance	The object to test.
type	The type whose instance must not be assigned to given object.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 674 of file Raise.cs.

6.3.2.37 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotAssignableFrom< TType > (object *instance*) [static]

Throws an exception of type *TEx* if and only if an instance of given type cannot be assigned to specified object.

Template Parameters

TType	The type whose instance must not be assigned to given object.

Parameters

instance	The object to test.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 705 of file Raise.cs.

6.3.2.38 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotAssignableFrom< TType > (object *instance,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if an instance of given type cannot be assigned to specified object.

Template Parameters

TType	The type whose instance must not be assigned to given object.

Parameters

instance	The object to test.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 745 of file Raise.cs.

6.3.2.39 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsNotContainedIn (object *argument*, System.Collections.IList *collection*) `[static]`

Throws an exception of type *TEx* if and only if specified argument is not contained in given collection.

Parameters

argument	The argument to check.
collection	The collection that must contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *argument* is not contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 836 of file Raise.cs.

6.3.2.40 static void **PommaLabs.Thrower.Raise**< TEx >.IfIsNotContainedIn (object *argument*, System.Collections.IList *collection*, string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified argument is not contained in given collection.

Parameters

argument	The argument to check.
collection	The collection that must contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *argument* is not contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 871 of file Raise.cs.

6.3.2.41 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsNotContainedIn**< TArg > (TArg *arg*, System.Collections.Generic.IEnumerable< TArg > *collection*) [static]

Throws an exception of type *TEx* if and only if specified argument is not contained in given collection.

Parameters

arg	The argument to check.
collection	The collection that must contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is not contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 962 of file Raise.cs.

6.3.2.42 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsNotContainedIn**< TArg > (TArg *arg*, System.Collections.Generic.IEnumerable< TArg > *collection*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified argument is not contained in given collection.

Parameters

arg	The argument to check.
collection	The collection that must contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is not contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 997 of file Raise.cs.

6.3.2.43 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsNotContainedIn**< TArg > (TArg *arg*, System.Collections.IDictionary *dictionary*) [static]

Throws an exception of type *TEx* if and only if specified argument is not contained in given dictionary keys.

Parameters

arg	The argument to check.
dictionary	The dictionary that must contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is not contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1088 of file Raise.cs.

6.3.2.44 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsNotContainedIn**< TArg > (TArg *arg*, System.Collections.IDictionary *dictionary*, string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified argument is not contained in given dictionary keys.

Parameters

arg	The argument to check.
dictionary	The dictionary that must contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is not contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1124 of file Raise.cs.

6.3.2.45 static void **PommaLabs.Thrower.Raise**< TEx >**.IfIsNotContainedIn**< TArg1, TArg2 > (TArg1 *arg1*, TArg2 *arg2*, System.Collections.Generic.IDictionary< TArg1, TArg2 > *dictionary*) `[static]`

Throws an exception of type *TEx* if and only if specified arguments are not contained in given dictionary pairs.

Parameters

arg1	The key argument to check.
arg2	The value argument to check.
dictionary	The dictionary that must contain given argument.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg1* and *arg2* are not contained, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1220 of file Raise.cs.

6.3.2.46 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotContainedIn< TArg1, TArg2 > (TArg1 *arg1*, TArg2 *arg2*, System.Collections.Generic.IDictionary< TArg1, TArg2 > *dictionary*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified arguments are not contained in given dictionary pairs.

Parameters

arg1	The key argument to check.
arg2	The value argument to check.
dictionary	The dictionary that must contain given argument.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg1* and *arg2* are not contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1258 of file Raise.cs.

6.3.2.47 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty (string *valueToCheck*) [static]

Throws an exception of type *TEx* if and only if specified string is not null, empty, or does not consist only of white-space characters.

Parameters

valueToCheck	The string to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *valueToCheck* is not empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1348 of file Raise.cs.

6.3.2.48 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty (string *valueToCheck,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified string is not null, empty, or does not consist only of white-space characters.

Parameters

valueToCheck	The string to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *valueToCheck* is not empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw

the exception.

Definition at line 1383 of file Raise.cs.

6.3.2.49 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty (System.Collections.ICollection *collection*) [static]

Throws an exception of type *TEx* if and only if specified collection is null or not empty.

Parameters

collection	The collection to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *collection* is null or not empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1471 of file Raise.cs.

6.3.2.50 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty (System.Collections.ICollection *collection,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified collection is null or not empty.

Parameters

collection	The collection to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *collection* is null or not empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1505 of file Raise.cs.

6.3.2.51 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > *collection*) [static]

Throws an exception of type *TEx* if and only if specified collection is null or not empty.

Parameters

collection	The collection to check for emptiness.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *collection* is null or not empty, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1593 of file Raise.cs.

6.3.2.52 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotEmpty< TArg > (System.Collections.Generic.IEnumerable< TArg > *collection*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified collection is null or not empty.

Parameters

collection	The collection to check for emptiness.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *collection* is null or not empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1627 of file Raise.cs.

6.3.2.53 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotInstanceOf (object *instance*, Type *type*) [static]

Throws an exception of type *TEx* if and only if specified object has not given type.

Parameters

instance	The object to test.
type	The type the object must not have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *instance* has not given type, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1783 of file Raise.cs.

6.3.2.54 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotInstanceOf (object *instance*, Type *type*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified object has not given type.

Parameters

instance	The object to test.

type	The type the object must not have.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *instance* has not given type, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1818 of file Raise.cs.

6.3.2.55 static void **PommaLabs.Thrower.Raise< TEx >.IfIsNotInstanceOf< TType >** (object *instance*) [static]

Throws an exception of type *TEx* if and only if specified object has not given type.

Template Parameters

TType	The type the object must not have.

Parameters

instance	The object to test.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *instance* has not given type, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1847 of file Raise.cs.

6.3.2.56 static void **PommaLabs.Thrower.Raise< TEx >.IfIsNotInstanceOf< TType >** (object *instance,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified object has not given type.

Template Parameters

TType	The type the object must not have.

Parameters

instance	The object to test.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *instance* has not given type, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 1883 of file Raise.cs.

6.3.2.57 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotNaN (double *number*) [static]

Throws an exception of type *TEx* if and only if specified double is not double.NaN.

Parameters

number	The double to test for double.NaN equality.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *number* is not double.NaN, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1971 of file Raise.cs.

6.3.2.58 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotNaN (double *number,* string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified double is not double.NaN.

Parameters

number	The double to test for double.NaN equality.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *number* is not double.NaN, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 2006 of file Raise.cs.

6.3.2.59 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotNull< TArg > (TArg *arg*) [static]

Throws an exception of type *TEx* if and only if specified argument is not null.

Parameters

arg	The argument to test for nullity.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is null, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 2094 of file Raise.cs.

6.3.2.60 static void PommaLabs.Thrower.Raise< TEx >.IfIsNotNull< TArg > (TArg *arg*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified argument is not null.

Parameters

arg	The argument to test for nullity.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is not null, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 2128 of file Raise.cs.

6.3.2.61 static void PommaLabs.Thrower.Raise< TEx >.IfIsNull< TArg > (TArg *arg*) [static]

Throws an exception of type *TEx* if and only if specified argument is null.

Parameters

arg	The argument to test for nullity.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *arg* is null, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 2033 of file Raise.cs.

6.3.2.62 static void PommaLabs.Thrower.Raise< TEx >.IfIsNull< TArg > (TArg *arg*, string *message*) [static]

Throws an exception of type *TEx* with given message *message* if and only if specified argument is null.

Parameters

arg	The argument to test for nullity.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *arg* is null, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 2067 of file Raise.cs.

6.3.2.63 static void PommaLabs.Thrower.Raise< TEx >.IfNot (bool *cond*) `[static]`

Throws an exception of type *TEx* if and only if specified condition is false.

Parameters

cond	The condition that determines whether an exception will be thrown.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor with no parameters, or *TEx* is abstract.

If *cond* is false, then an exception of type *TEx* will be thrown.
In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 193 of file Raise.cs.

6.3.2.64 static void PommaLabs.Thrower.Raise< TEx >.IfNot (bool *cond,* string *message*) `[static]`

Throws an exception of type *TEx* with given message *message* if and only if specified condition is false.

Parameters

cond	The condition that determines whether an exception will be thrown.
message	The message the thrown exception will have.

Exceptions

ThrowerException	*TEx* has not a public or internal constructor which takes, as parameters, either a string or a string and an System.Exception. The same exception is thrown when *TEx* is abstract.

If *cond* is false, then an exception of type *TEx* , with the message specified by *message* , will be thrown.
In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.
If both constructors are available, then the one which takes a string and an System.Exception will be used to throw the exception.

Definition at line 227 of file Raise.cs.

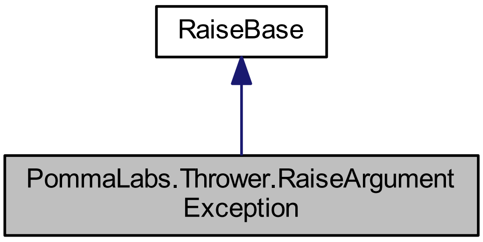
The documentation for this class was generated from the following file:

- Raise.cs

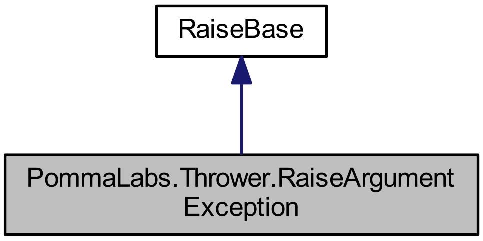
6.4 PommaLabs.Thrower.RaiseArgumentException Class Reference

Utility methods which can be used to handle bad arguments.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentException:



Static Public Member Functions

- static void If (bool condition)

  Throws ArgumentException if given condition is true.
- static void If (bool condition, string argumentName, string message=null)

  Throws ArgumentException if given condition is true.
- static void IfNot (bool condition)

  Throws ArgumentException if given condition is false.
- static void IfNot (bool condition, string argumentName, string message=null)

  Throws ArgumentException if given condition is false.
- static void IfIsNotValid< TArg > (TArg argument)

  Throws ArgumentException if given argument is not valid.
- static void IfIsNotValid< TArg > (TArg argument, string argumentName, string message=null)

  Throws ArgumentException if given argument is not valid.
- static void IfIsNotValidEmailAddress (string emailAddress)

  Throws ArgumentException if given string is not a valid email address.
- static void IfIsNotValidEmailAddress (string emailAddress, bool allowInternational)

  Throws ArgumentException if given string is not a valid email address.
- static void IfIsNotValidEmailAddress (string emailAddress, string argumentName, string message=null)

Throws ArgumentException if given string is not a valid email address.

- static void IfIsNotValidEmailAddress (string emailAddress, bool allowInternational, string argumentName, string message=null)

  Throws ArgumentException if given string is not a valid email address.
- static void IfIsNotValidPhoneNumber (string phoneNumber)

  Throws ArgumentException if given string is not a valid phone number.
- static void IfIsNotValidPhoneNumber (string phoneNumber, string argumentName, string message=null)

  Throws ArgumentException if given string is not a valid phone number.
- static void IfIsNullOrEmpty (string value)

  Throws ArgumentException if given string is null or empty.
- static void IfIsNullOrEmpty (string value, string argumentName, string message=null)

  Throws ArgumentException if given string is null or empty.
- static void IfIsNullOrWhiteSpace (string value)

  Throws ArgumentException if given string is null, empty or blank.
- static void IfIsNullOrWhiteSpace (string value, string argumentName, string message=null)

  Throws ArgumentException if given string is null, empty or blank.

Additional Inherited Members

6.4.1 Detailed Description

Utility methods which can be used to handle bad arguments.

Definition at line 33 of file RaiseArgumentException.cs.

6.4.2 Member Function Documentation

6.4.2.1 static void PommaLabs.Thrower.RaiseArgumentException.If (bool *condition*) `[static]`

Throws ArgumentException if given condition is true.

Parameters

condition	The condition.

Definition at line 47 of file RaiseArgumentException.cs.

6.4.2.2 static void PommaLabs.Thrower.RaiseArgumentException.If (bool *condition,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given condition is true.

Parameters

condition	The condition.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 68 of file RaiseArgumentException.cs.

6.4.2.3 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValid< TArg >(TArg *argument*) `[static]`

Throws ArgumentException if given argument is not valid.

Template Parameters

TArg	The type of the argument.

Parameters

argument	The argument.

Definition at line 130 of file RaiseArgumentException.cs.

6.4.2.4 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValid< TArg > (TArg *argument,* string *argumentName,* string *message* = null) [static]

Throws ArgumentException if given argument is not valid.

Template Parameters

TArg	The type of the argument.

Parameters

argument	The argument.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 153 of file RaiseArgumentException.cs.

6.4.2.5 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress*) [static]

Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.

Definition at line 176 of file RaiseArgumentException.cs.

6.4.2.6 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress,* bool *allowInternational*) [static]

Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.
allow↩ International	true if the validator should allow international characters; otherwise,

false

.

Definition at line 196 of file RaiseArgumentException.cs.

6.4.2.7 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress,* string *argumentName,* string *message* = null) [static]

Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 218 of file RaiseArgumentException.cs.

6.4.2.8 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress,* bool *allowInternational,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.
allow↩ International	true if the validator should allow international characters; otherwise,

false

.

Parameters

argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 243 of file RaiseArgumentException.cs.

6.4.2.9 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidPhoneNumber (string *phoneNumber*) `[static]`

Throws ArgumentException if given string is not a valid phone number.

Parameters

phoneNumber	A phone number.

Definition at line 266 of file RaiseArgumentException.cs.

6.4.2.10 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidPhoneNumber (string *phoneNumber,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given string is not a valid phone number.

Parameters

phoneNumber	A phone number.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 288 of file RaiseArgumentException.cs.

6.4.2.11 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrEmpty (string *value*) `[static]`

Throws ArgumentException if given string is null or empty.

Parameters

value	The string value.

Definition at line 312 of file RaiseArgumentException.cs.

6.4.2.12 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrEmpty (string *value,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given string is null or empty.

Parameters

value	The string value.
argumentName	The name of the argument.
message	The optional message.

message and *argumentName* are strictly required arguments.

Definition at line 333 of file RaiseArgumentException.cs.

6.4.2.13 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrWhiteSpace (string *value*) `[static]`

Throws ArgumentException if given string is null, empty or blank.

Parameters

value	The string value.

Definition at line 349 of file RaiseArgumentException.cs.

6.4.2.14 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrWhiteSpace (string *value,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given string is null, empty or blank.

Parameters

value	The string value.
argumentName	The name of the argument.
message	The optional message.

message and *argumentName* are strictly required arguments.

Definition at line 370 of file RaiseArgumentException.cs.

6.4.2.15 static void PommaLabs.Thrower.RaiseArgumentException.IfNot (bool *condition*) `[static]`

Throws ArgumentException if given condition is false.

Parameters

condition	The condition.

Definition at line 88 of file RaiseArgumentException.cs.

6.4.2.16 static void PommaLabs.Thrower.RaiseArgumentException.IfNot (bool *condition,* string *argumentName,* string *message* = `null`) `[static]`

Throws ArgumentException if given condition is false.

Parameters

condition	The condition.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 109 of file RaiseArgumentException.cs.

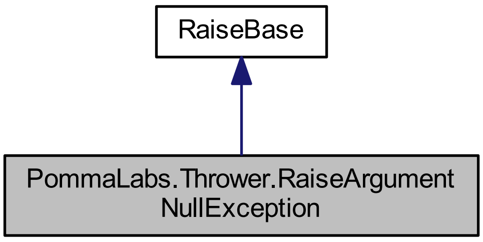
The documentation for this class was generated from the following file:

- RaiseArgumentException.cs

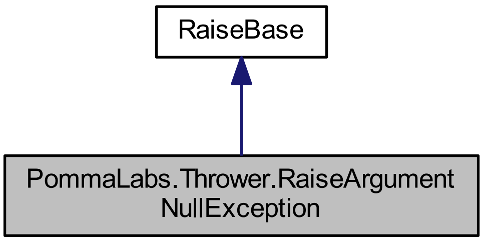
6.5 PommaLabs.Thrower.RaiseArgumentNullException Class Reference

Utility methods which can be used to handle null references.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentNullException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentNullException:



Static Public Member Functions

- static void IfIsNull< TArg > (TArg argument)

    Throws ArgumentNullException if given argument if null.

- static void IfIsNull< TArg > (TArg argument, string argumentName)

  Throws ArgumentNullException if given argument if null.
- static void IfIsNull< TArg > (TArg argument, string argumentName, string message)

  Throws ArgumentNullException if given argument if null.

Additional Inherited Members

6.5.1 Detailed Description

Utility methods which can be used to handle null references.

Definition at line 32 of file RaiseArgumentNullException.cs.

6.5.2 Member Function Documentation

6.5.2.1 static void PommaLabs.Thrower.RaiseArgumentNullException.IfIsNull< TArg > (TArg *argument*) [static]

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the argument.

Parameters

argument	The argument.

Definition at line 43 of file RaiseArgumentNullException.cs.

6.5.2.2 static void PommaLabs.Thrower.RaiseArgumentNullException.IfIsNull< TArg > (TArg *argument,* string *argumentName*) [static]

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the argument.

Parameters

argument	The argument.
argumentName	The name of the argument.

Definition at line 61 of file RaiseArgumentNullException.cs.

6.5.2.3 static void PommaLabs.Thrower.RaiseArgumentNullException.IfIsNull< TArg > (TArg *argument,* string *argumentName,* string *message*) [static]

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the argument.

Parameters

argument	The argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 80 of file RaiseArgumentNullException.cs.

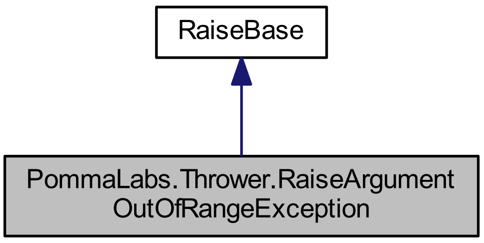
The documentation for this class was generated from the following file:

- RaiseArgumentNullException.cs

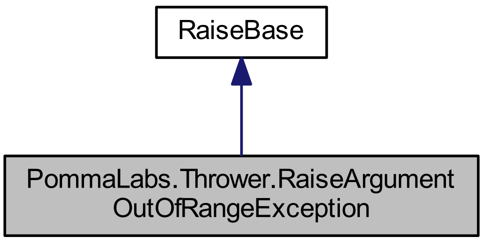
6.6 PommaLabs.Thrower.RaiseArgumentOutOfRangeException Class Reference

Utility methods which can be used to handle ranges.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentOutOfRangeException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentOutOfRangeException:



Static Public Member Functions

- static void If (bool condition, string argumentName=null)

  Throws ArgumentOutOfRangeException if given condition is true.
- static void If (bool condition, string argumentName, string message)

Throws ArgumentOutOfRangeException if given condition is true.
- static void IfNot (bool condition, string argumentName=null)

  Throws ArgumentOutOfRangeException if given condition is false.
- static void IfNot (bool condition, string argumentName, string message)

  Throws ArgumentOutOfRangeException if given condition is false.
- static void IfIsLess< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsGreater< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreaterOrEqual< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .

- static void IfIsGreaterOrEqual (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsEqual< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsNotEqual< TArg > (TArg argument1, TArg argument2)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual (IComparable argument1, IComparable argument2)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual< TArg > (TArg argument1, TArg argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual (IComparable argument1, IComparable argument2, string argumentName)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual< TArg > (TArg argument1, TArg argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual (IComparable argument1, IComparable argument2, string argumentName, string message)

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .

Additional Inherited Members

6.6.1 Detailed Description

Utility methods which can be used to handle ranges.

Definition at line 31 of file RaiseArgumentOutOfRangeException.cs.

6.6.2 Member Function Documentation

6.6.2.1 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.If (bool *condition,* string *argumentName =* `null`) `[static]`

Throws ArgumentOutOfRangeException if given condition is true.

Parameters

condition	The condition.
argumentName	The optional name of the argument.

Definition at line 44 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.2 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.If (bool *condition*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if given condition is true.

Parameters

condition	The condition.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 65 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.3 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual (IComparable *argument1*, IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 677 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.4 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 721 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.5 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 767 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.6 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual< TArg >(TArg *argument1*, TArg *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 658 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.7 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 701 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.8 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsEqual< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 746 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.9 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater (IComparable *argument1*, IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 413 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.10 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater (IComparable *argument1*, IComparable *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 457 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.11 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 503 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.12 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 394 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.13 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 437 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.14 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreater< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 482 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.15 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual (IComparable *argument1,* IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 545 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.16 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 589 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.17 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 635 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.18 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 526 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.19 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 569 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.20 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsGreaterOrEqual**< TArg > (TArg *argument1,* TArg *argument2,* string *argumentName,* string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 614 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.21 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess (IComparable *argument1,* IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 149 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.22 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess (IComparable *argument1,* IComparable *argument2,* string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 193 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.23 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 239 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.24 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 130 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.25 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 173 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.26 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLess< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 218 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.27 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 281 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.28 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 325 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.29 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 371 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.30 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual**< TArg >(TArg *argument1*, TArg *argument2*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 262 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.31 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual**< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 305 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.32 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual**< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is less than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 350 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.33 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual (IComparable *argument1*, IComparable *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 809 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.34 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*) [static]

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Definition at line 853 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.35 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*) [static]

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Definition at line 899 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.36 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 790 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.37 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual**< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 833 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.38 static void **PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsNotEqual**< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*) `[static]`

Throws ArgumentOutOfRangeException if *argument1* is not equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 878 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.39 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfNot (bool *condition,* string *argumentName =* `null`) `[static]`

Throws ArgumentOutOfRangeException if given condition is false.

Parameters

condition	The condition.
argumentName	The optional name of the argument.

Definition at line 86 of file RaiseArgumentOutOfRangeException.cs.

6.6.2.40 static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfNot (bool *condition,* string *argumentName,* string *message*) [static]

Throws ArgumentOutOfRangeException if given condition is false.

Parameters

condition	The condition.
argumentName	The name of the argument.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 107 of file RaiseArgumentOutOfRangeException.cs.

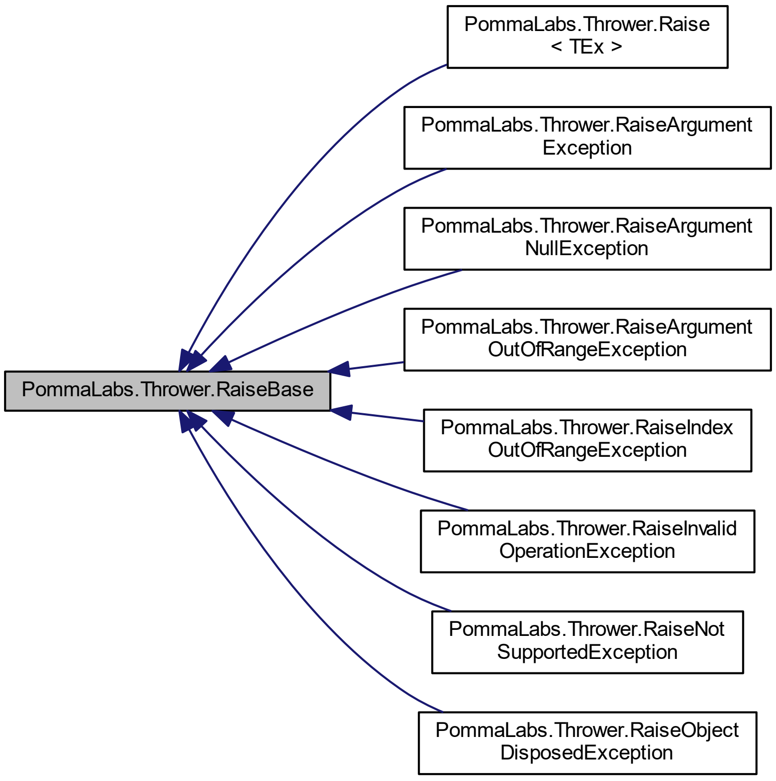
The documentation for this class was generated from the following file:

- RaiseArgumentOutOfRangeException.cs

6.7 PommaLabs.Thrower.RaiseBase Class Reference

Stores items shared by various Raise<TEx> instances.

Inheritance diagram for PommaLabs.Thrower.RaiseBase:


PommaLabs.Thrower.Raise < TEx >
PommaLabs.Thrower.RaiseArgument Exception
PommaLabs.Thrower.RaiseArgument NullException
PommaLabs.Thrower.RaiseArgument OutOfRangeException
PommaLabs.Thrower.RaiseBase
PommaLabs.Thrower.RaiseIndex OutOfRangeException
PommaLabs.Thrower.RaiseInvalid OperationException
PommaLabs.Thrower.RaiseNot SupportedException
PommaLabs.Thrower.RaiseObject DisposedException

Static Protected Attributes

- static readonly Type[ ] NoCtorTypes = new Type[0]

    Stores an empty array of System.Type used to seek constructors without parameters.
- static readonly Type[ ] StrExCtorTypes = { typeof(string), typeof(Exception) }

    Stores the types needed to seek the constructor which takes a string and an exception as parameters to instance the exception.
- static readonly Type[ ] StrCtorType = { typeof(string) }

    Stores the type needed to seek the constructor which takes a string as parameter to instance the exception.

6.7.1 Detailed Description

Stores items shared by various Raise<TEx> instances.

Definition at line 35 of file Raise.cs.

6.7.2 Member Data Documentation

6.7.2.1 readonly Type [ ] PommaLabs.Thrower.RaiseBase.NoCtorTypes = new Type[0] [static], [protected]

Stores an empty array of System.Type used to seek constructors without parameters.

Definition at line 42 of file Raise.cs.

6.7.2.2 readonly Type [ ] PommaLabs.Thrower.RaiseBase.StrCtorType = { typeof(string) } `[static]`, `[protected]`

Stores the type needed to seek the constructor which takes a string as parameter to instance the exception.

Definition at line 58 of file Raise.cs.

6.7.2.3 readonly Type [ ] PommaLabs.Thrower.RaiseBase.StrExCtorTypes = { typeof(string), typeof(Exception) } `[static]`, `[protected]`

Stores the types needed to seek the constructor which takes a string and an exception as parameters to instance the exception.

Definition at line 50 of file Raise.cs.

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

- Raise.cs

6.8 PommaLabs.Thrower.RaiseHttpException Class Reference

Utility methods which can be used to handle error codes through HTTP.

Static Public Member Functions

- static void If (bool condition, HttpStatusCode httpStatusCode, string message=null)

  Throws HttpException if given condition is true.
- static void If (bool condition, HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additional↩ Info)

  Throws HttpException if given condition is true.
- static void IfNot (bool condition, HttpStatusCode httpStatusCode, string message=null)

  Throws HttpException if given condition is false.
- static void IfNot (bool condition, HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)

  Throws HttpException if given condition is false.

6.8.1 Detailed Description

Utility methods which can be used to handle error codes through HTTP.

Definition at line 33 of file RaiseHttpException.cs.

6.8.2 Member Function Documentation

6.8.2.1 static void PommaLabs.Thrower.RaiseHttpException.If (bool *condition,* HttpStatusCode *httpStatusCode,* string *message* = `null`) `[static]`

Throws HttpException if given condition is true.

Parameters

condition	The condition.
httpStatusCode	The HTTP status code corresponding to the error.
message	The optional message.

Definition at line 45 of file RaiseHttpException.cs.

6.8.2.2 static void PommaLabs.Thrower.RaiseHttpException.If (bool *condition*, HttpStatusCode *httpStatusCode*, string *message*, HttpExceptionInfo *additionalInfo*) [static]

Throws HttpException if given condition is true.

Parameters

condition	The condition.
httpStatusCode	The HTTP status code corresponding to the error.
message	The required message.
additionalInfo	Additional exception info.

Definition at line 64 of file RaiseHttpException.cs.

6.8.2.3 static void PommaLabs.Thrower.RaiseHttpException.IfNot (bool *condition*, HttpStatusCode *httpStatusCode*, string *message* = null) [static]

Throws HttpException if given condition is false.

Parameters

condition	The condition.
httpStatusCode	The HTTP status code corresponding to the error.
message	The optional message.

Definition at line 82 of file RaiseHttpException.cs.

6.8.2.4 static void PommaLabs.Thrower.RaiseHttpException.IfNot (bool *condition*, HttpStatusCode *httpStatusCode*, string *message*, HttpExceptionInfo *additionalInfo*) [static]

Throws HttpException if given condition is false.

Parameters

condition	The condition.
httpStatusCode	The HTTP status code corresponding to the error.
message	The required message.
additionalInfo	Additional exception info.

Definition at line 101 of file RaiseHttpException.cs.

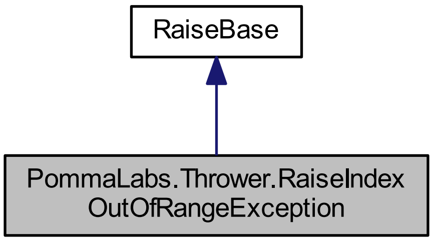
The documentation for this class was generated from the following file:

- RaiseHttpException.cs

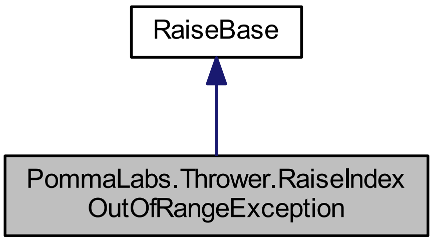
6.9 PommaLabs.Thrower.RaiseIndexOutOfRangeException Class Reference

Utility methods which can be used to handle indexes.

Inheritance diagram for PommaLabs.Thrower.RaiseIndexOutOfRangeException:



Collaboration diagram for PommaLabs.Thrower.RaiseIndexOutOfRangeException:



Static Public Member Functions

- static void IfIsLess< TArg > (TArg argument1, TArg argument2)

  Throws IndexOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLess (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is less than argument2 .
- static void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2)

  Throws IndexOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsLessOrEqual (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is less than or equal to argument2 .
- static void IfIsGreater< TArg > (TArg argument1, TArg argument2)

Throws IndexOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreater (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is greater than argument2 .
- static void IfIsGreaterOrEqual< TArg > (TArg argument1, TArg argument2)

  Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsGreaterOrEqual (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void IfIsEqual< TArg > (TArg argument1, TArg argument2)

  Throws IndexOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsEqual (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is equal to argument2 .
- static void IfIsNotEqual< TArg > (TArg argument1, TArg argument2)

  Throws IndexOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual (IComparable argument1, IComparable argument2)

  Throws IndexOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual< TArg > (TArg argument1, TArg argument2, string message)

  Throws IndexOutOfRangeException if argument1 is not equal to argument2 .
- static void IfIsNotEqual (IComparable argument1, IComparable argument2, string message)

  Throws IndexOutOfRangeException if argument1 is not equal to argument2 .

Additional Inherited Members

6.9.1 Detailed Description

Utility methods which can be used to handle indexes.

Definition at line 31 of file RaiseIndexOutOfRangeException.cs.

6.9.2 Member Function Documentation

6.9.2.1 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsEqual (IComparable *argument1,* IComparable *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 409 of file RaiseIndexOutOfRangeException.cs.

6.9.2.2 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 453 of file RaiseIndexOutOfRangeException.cs.

6.9.2.3 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 390 of file RaiseIndexOutOfRangeException.cs.

6.9.2.4 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 433 of file RaiseIndexOutOfRangeException.cs.

6.9.2.5 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreater (IComparable *argument1,* IComparable *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 237 of file RaiseIndexOutOfRangeException.cs.

6.9.2.6 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreater (IComparable *argument1,* IComparable *argument2,* string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 281 of file RaiseIndexOutOfRangeException.cs.

6.9.2.7 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreater< TArg > (TArg *argument1,* TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 218 of file RaiseIndexOutOfRangeException.cs.

6.9.2.8 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreater< TArg > (TArg *argument1,* TArg *argument2,* string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 261 of file RaiseIndexOutOfRangeException.cs.

6.9.2.9 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreaterOrEqual (IComparable *argument1,* IComparable *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 323 of file RaiseIndexOutOfRangeException.cs.

6.9.2.10 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 367 of file RaiseIndexOutOfRangeException.cs.

6.9.2.11 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 304 of file RaiseIndexOutOfRangeException.cs.

6.9.2.12 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is greater than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 347 of file RaiseIndexOutOfRangeException.cs.

6.9.2.13 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLess (IComparable *argument1,* IComparable *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is less than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 65 of file RaiseIndexOutOfRangeException.cs.

6.9.2.14 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLess (IComparable *argument1,* IComparable *argument2,* string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is less than *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 109 of file RaiseIndexOutOfRangeException.cs.

6.9.2.15 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLess< TArg > (TArg *argument1,* TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is less than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 46 of file RaiseIndexOutOfRangeException.cs.

6.9.2.16 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLess< TArg > (TArg *argument1,* TArg *argument2,* string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is less than *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 89 of file RaiseIndexOutOfRangeException.cs.

6.9.2.17 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLessOrEqual (IComparable *argument1,* IComparable *argument2*) `[static]`

Throws IndexOutOfRangeException if *argument1* is less than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 151 of file RaiseIndexOutOfRangeException.cs.

6.9.2.18 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is less than or equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 195 of file RaiseIndexOutOfRangeException.cs.

6.9.2.19 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is less than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 132 of file RaiseIndexOutOfRangeException.cs.

6.9.2.20 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is less than or equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

***TArg* : *IComparable*<*TArg*>**

Definition at line 175 of file RaiseIndexOutOfRangeException.cs.

6.9.2.21 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsNotEqual (IComparable *argument1*, IComparable *argument2*) `[static]`

Throws IndexOutOfRangeException if *argument1* is not equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.

Definition at line 495 of file RaiseIndexOutOfRangeException.cs.

6.9.2.22 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsNotEqual (IComparable *argument1*, IComparable *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is not equal to *argument2* .

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Definition at line 539 of file RaiseIndexOutOfRangeException.cs.

6.9.2.23 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsNotEqual< TArg >(TArg *argument1*, TArg *argument2*) [static]

Throws IndexOutOfRangeException if *argument1* is not equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 476 of file RaiseIndexOutOfRangeException.cs.

6.9.2.24 static void PommaLabs.Thrower.RaiseIndexOutOfRangeException.IfIsNotEqual< TArg >(TArg *argument1*, TArg *argument2*, string *message*) [static]

Throws IndexOutOfRangeException if *argument1* is not equal to *argument2* .

Template Parameters

TArg	The type of the arguments.

Parameters

argument1	The left side argument.
argument2	The right side argument.
message	The message that should be put into the exception.

Type Constraints

TArg* : *IComparable<TArg>

Definition at line 519 of file RaiseIndexOutOfRangeException.cs.

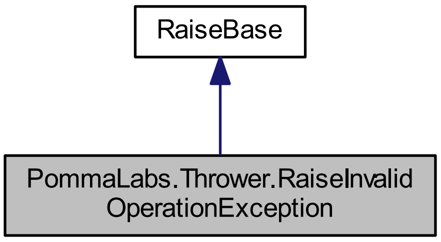
The documentation for this class was generated from the following file:

- RaiseIndexOutOfRangeException.cs

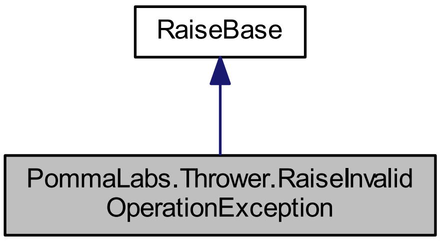
6.10 PommaLabs.Thrower.RaiseInvalidOperationException Class Reference

Utility methods which can be used to handle bad object states.

Inheritance diagram for PommaLabs.Thrower.RaiseInvalidOperationException:



Collaboration diagram for PommaLabs.Thrower.RaiseInvalidOperationException:



Static Public Member Functions

- static void If (bool condition, string message=null)
  Throws InvalidOperationException if given condition is true.
- static void IfNot (bool condition, string message=null)
  Throws InvalidOperationException if given condition is false.

Additional Inherited Members

6.10.1 Detailed Description

Utility methods which can be used to handle bad object states.

Definition at line 31 of file RaiseInvalidOperationException.cs.

6.10.2 Member Function Documentation

6.10.2.1 static void PommaLabs.Thrower.RaiseInvalidOperationException.If (bool *condition*, string *message* = `null`) `[static]`

Throws InvalidOperationException if given condition is true.

Parameters

condition	The condition.
message	The optional message.

Definition at line 42 of file RaiseInvalidOperationException.cs.

6.10.2.2 static void PommaLabs.Thrower.RaiseInvalidOperationException.IfNot (bool *condition*, string *message* = `null`) `[static]`

Throws InvalidOperationException if given condition is false.

Parameters

condition	The condition.
message	The optional message.

Definition at line 59 of file RaiseInvalidOperationException.cs.

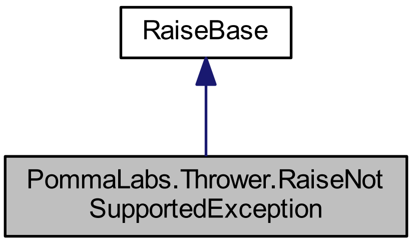
The documentation for this class was generated from the following file:

- RaiseInvalidOperationException.cs

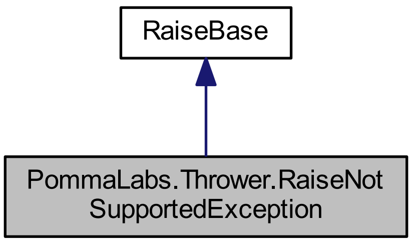
6.11 PommaLabs.Thrower.RaiseNotSupportedException Class Reference

Utility methods which can be used to handle unsupported operations.

Inheritance diagram for PommaLabs.Thrower.RaiseNotSupportedException:



Collaboration diagram for PommaLabs.Thrower.RaiseNotSupportedException:



Static Public Member Functions

- static void If (bool condition, string message=null)

  Throws NotSupportedException if given condition is true.
- static void IfNot (bool condition, string message=null)

  Throws NotSupportedException if given condition is false.

Additional Inherited Members

6.11.1 Detailed Description

Utility methods which can be used to handle unsupported operations.

Definition at line 31 of file RaiseNotSupportedException.cs.

6.11.2 Member Function Documentation

6.11.2.1 static void PommaLabs.Thrower.RaiseNotSupportedException.If (bool *condition,* string *message* = `null`) `[static]`

Throws NotSupportedException if given condition is true.

Parameters

condition	The condition.
message	The optional message.

Definition at line 42 of file RaiseNotSupportedException.cs.

6.11.2.2 static void PommaLabs.Thrower.RaiseNotSupportedException.IfNot (bool *condition,* string *message* = `null`) `[static]`

Throws NotSupportedException if given condition is false.

Parameters

condition	The condition.
message	The optional message.

Definition at line 59 of file RaiseNotSupportedException.cs.

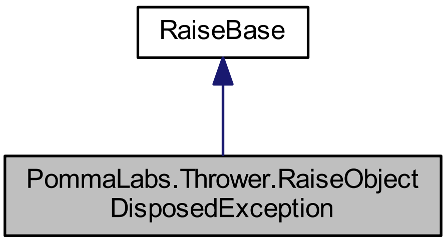
The documentation for this class was generated from the following file:

- RaiseNotSupportedException.cs

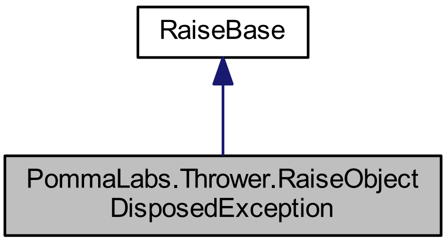
6.12 PommaLabs.Thrower.RaiseObjectDisposedException Class Reference

Utility methods which can be used to handle bad object states.

Inheritance diagram for PommaLabs.Thrower.RaiseObjectDisposedException:



Collaboration diagram for PommaLabs.Thrower.RaiseObjectDisposedException:



Static Public Member Functions

- static void If (bool disposed, string objectName, string message=null)

  Throws ObjectDisposedException if the object has been disposed.

Additional Inherited Members

6.12.1 Detailed Description

Utility methods which can be used to handle bad object states.

Definition at line 31 of file RaiseObjectDisposedException.cs.

6.12.2 Member Function Documentation

6.12.2.1 static void PommaLabs.Thrower.RaiseObjectDisposedException.If (bool *disposed,* string *objectName,* string *message* = `null`) `[static]`

Throws ObjectDisposedException if the object has been disposed.

Parameters

disposed	Whether the object has been disposed or not.
objectName	The required object name.
message	The optional message.

Definition at line 43 of file RaiseObjectDisposedException.cs.

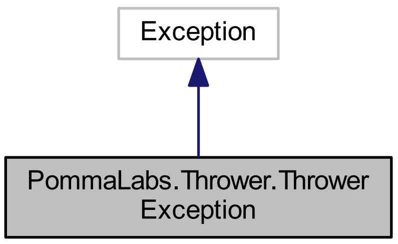
The documentation for this class was generated from the following file:

- RaiseObjectDisposedException.cs

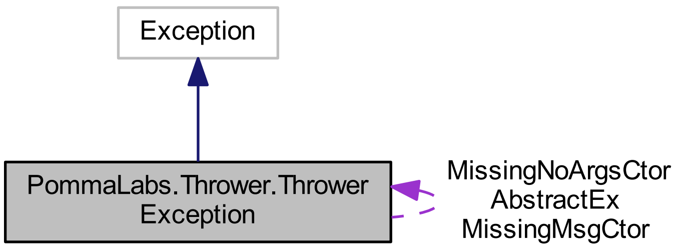
6.13 PommaLabs.Thrower.ThrowerException Class Reference

Exception thrown by Raise<TEx> when the type parameter passed to that class has something invalid (missing constructors, etc).

Inheritance diagram for PommaLabs.Thrower.ThrowerException:



Collaboration diagram for PommaLabs.Thrower.ThrowerException:



6.13.1 Detailed Description

Exception thrown by Raise<TEx> when the type parameter passed to that class has something invalid (missing constructors, etc).

Definition at line 2195 of file Raise.cs.

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

- Raise.cs

Chapter 7

File Documentation

7.1 Raise.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseBase

    Stores items shared by various Raise< TEx> instances.
- class PommaLabs.Thrower.Raise< TEx >

    Contains methods that throw specified exception TEx if given conditions will be verified.
- class PommaLabs.Thrower.ThrowerException

    Exception thrown by Raise< TEx> when the type parameter passed to that class has something invalid (missing constructors, etc).

Namespaces

- namespace PommaLabs.Thrower

7.2 Raise.cs

```
// File name: Raise.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using PommaLabs.Thrower.Reflection;
using System;
using System.Diagnostics.CodeAnalysis;
using System.Linq;
using System.Reflection;

namespace PommaLabs.Thrower
```

```
{
    public abstract class RaiseBase
    {
        [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
        [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
        protected static readonly Type[] NoCtorTypes = new Type[0];

        [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
        [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
        protected static readonly Type[] StrExCtorTypes = { typeof(string), typeof(Exception) };

        [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
        [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
        protected static readonly Type[] StrCtorType = { typeof(string) };
    }

    public sealed class Raise<TEx> : RaiseBase where TEx : Exception
    {
        private static readonly bool ExTypeIsAbstract = PortableTypeInfo.IsAbstract(typeof(TEx));

        private static readonly ConstructorInfo NoArgsCtor = GetCtor(NoCtorTypes);

        private static readonly ConstructorInfo MsgCtor = GetCtor(StrExCtorTypes) ?? GetCtor(StrCtorType);

        private static readonly int MsgArgCount = (MsgCtor == null) ? 0 : MsgCtor.GetParameters().Length;

        private Raise()
        {
            throw new InvalidOperationException("This class should not be instantiated");
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void If(bool cond)
        {
            if (cond)
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void If(bool cond, string message)
        {
            if (cond)
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfNot(bool cond)
        {
            if (!cond)
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfNot(bool cond, string message)
        {
            if (!cond)
            {
                DoThrow(message);
            }
        }

```

```
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
        {
            if (Equals(arg1, arg2))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
        {
            if (Equals(arg1, arg2))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
        {
            if (!Equals(arg1, arg2))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
        {
            if (!Equals(arg1, arg2))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
        {
            if (ReferenceEquals(arg1, arg2))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
        {
            if (ReferenceEquals(arg1, arg2))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
```

```
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
        {
            if (!ReferenceEquals(arg1, arg2))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
        {
            if (!ReferenceEquals(arg1, arg2))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsAssignableFrom(object instance, Type type)
        {
            if (PortableTypeInfo.IsAssignableFrom(instance, type))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsAssignableFrom(object instance, Type type, string message)
        {
            if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, type))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsAssignableFrom<TType>(object instance)
        {
            if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, typeof(TType)))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsAssignableFrom<TType>(object instance, string message)
        {
            if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, typeof(TType)))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotAssignableFrom(object instance, Type type)
        {
            if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, type))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotAssignableFrom(object instance, Type type, string
message)
        {
            if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, type))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsNotAssignableFrom<TType>(object instance)
        {
            if (!PortableTypeInfo.IsAssignableFrom(instance, typeof(TType)))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsNotAssignableFrom<TType>(object instance, string message)
        {
            if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, typeof(
TType)))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn(object argument,
System.Collections.IList collection)
        {
            if (ReferenceEquals(collection, null) || collection.Contains(argument))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn(object argument,
System.Collections.IList collection, string message)
        {
            if (ReferenceEquals(collection, null) || collection.Contains(argument))
            {
                DoThrow(message);
            }
        }
```

```

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn(object argument,
System.Collections.IList collection)
        {
            if (ReferenceEquals(collection, null) || !collection.Contains(argument))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn(object argument,
System.Collections.IList collection, string message)
        {
            if (ReferenceEquals(collection, null) || !collection.Contains(argument))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection)
        {
            if (ReferenceEquals(collection, null) || collection.Contains(arg))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection, string message)
        {
            if (ReferenceEquals(collection, null) || collection.Contains(arg))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection)
        {
            if (ReferenceEquals(collection, null) || !collection.Contains(arg))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection, string message)
        {
            if (ReferenceEquals(collection, null) || !collection.Contains(arg))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary)
        {
            if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary, string message)
        {
            if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary)
        {
            if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary, string message)
        {
            if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
        {
            if (ReferenceEquals(dictionary, null) || dictionary.Contains(new System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,
            string message)
        {
            if (ReferenceEquals(dictionary, null) || dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
        {
            if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,
            string message)
        {
            if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty(string valueToCheck)
        {
            if (IsNullOrWhiteSpace(valueToCheck))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty(string valueToCheck, string message)
        {
            if (IsNullOrWhiteSpace(valueToCheck))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty(string valueToCheck)
        {
            if (!IsNullOrWhiteSpace(valueToCheck))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
```

```
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty(string valueToCheck, string message)
        {
            if (!IsNullOrWhiteSpace(valueToCheck))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty(System.Collections.ICollection collection)
        {
            if (ReferenceEquals(collection, null) || collection.Count == 0)
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty(System.Collections.ICollection collection, string
message)
        {
            if (ReferenceEquals(collection, null) || collection.Count == 0)
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty(System.Collections.ICollection collection)
        {
            if (ReferenceEquals(collection, null) || collection.Count > 0)
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty(System.Collections.ICollection collection, string
 message)
        {
            if (ReferenceEquals(collection, null) || collection.Count > 0)
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection)
        {
            if (ReferenceEquals(collection, null) || !collection.Any())
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif
```

```

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection,
string message)
        {
            if (ReferenceEquals(collection, null) || !collection.Any())
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
)
        {
            if (ReferenceEquals(collection, null) || collection.Any())
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
, string message)
        {
            if (ReferenceEquals(collection, null) || collection.Any())
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsInstanceOf(object instance, Type type)
        {
            if (PortableTypeInfo.IsInstanceOf(instance, type))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsInstanceOf(object instance, Type type, string message)
        {
            if (PortableTypeInfo.IsInstanceOf(instance, type))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsInstanceOf<TType>(object instance)
        {
            if (instance is TType)
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
```

```
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsInstanceOf<TType>(object instance, string message)
        {
            if (instance is TType)
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotInstanceOf(object instance, Type type)
        {
            if (!PortableTypeInfo.IsInstanceOf(instance, type))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotInstanceOf(object instance, Type type, string message)
        {
            if (!PortableTypeInfo.IsInstanceOf(instance, type))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsNotInstanceOf<TType>(object instance)
        {
            if (!(instance is TType))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
        public static void IfIsNotInstanceOf<TType>(object instance, string message)
        {
            if (!(instance is TType))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNaN(double number)
        {
            if (double.IsNaN(number))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
```

```
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNaN(double number, string message)
        {
            if (double.IsNaN(number))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotNaN(double number)
        {
            if (!double.IsNaN(number))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotNaN(double number, string message)
        {
            if (!double.IsNaN(number))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNull<TArg>(TArg arg)
        {
            if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(arg, null))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNull<TArg>(TArg arg, string message)
        {
            if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(arg, null))
            {
                DoThrow(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotNull<TArg>(TArg arg)
        {
            if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(arg, null))
            {
                DoThrow();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
        public static void IfIsNotNull<TArg>(TArg arg, string message)
        {
            if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(arg, null))
            {
                DoThrow(message);
            }
        }

        private static ConstructorInfo GetCtor(System.Collections.Generic.IList<Type> ctorTypes)
        {
            return (from c in PortableTypeInfo.GetConstructors(typeof(TEx))
                    let args = c.GetParameters()
                    let zipArgs = args.Zip(ctorTypes, (argType, ctorType) => new { argType, ctorType })
                    where args.Length == ctorTypes.Count &&
                          (c.IsPublic || c.IsAssembly) &&
                          zipArgs.All(t => ReferenceEquals(t.argType.ParameterType, t.ctorType))
                    select c).FirstOrDefault();
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        private static void DoThrow()
        {
            // Checks whether the proper constructor exists. If not, then we produce an internal exception.
            if (ExTypeIsAbstract)
            {
                throw ThrowerException.AbstractEx;
            }
            if (NoArgsCtor == null)
            {
                throw ThrowerException.MissingNoArgsCtor;
            }
            // A proper constrctor exists: therefore, we can throw the exception.
            throw (TEx) NoArgsCtor.Invoke(new object[0]);
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        private static void DoThrow(string message)
        {
            // Checks whether the proper constructor exists. If not, then we produce an internal exception.
            if (ExTypeIsAbstract)
            {
                throw ThrowerException.AbstractEx;
            }
            if (MsgCtor == null)
            {
                throw ExTypeIsAbstract ? ThrowerException.AbstractEx : ThrowerException.MissingMsgCtor;
            }
            // A proper constrctor exists: therefore, we can throw the exception.
            var messageArgs = new object[MsgArgCount];
            messageArgs[0] = message;
            throw (TEx) MsgCtor.Invoke(messageArgs);
        }

        private static bool IsNullOrWhiteSpace(string value) => value == null || string.IsNullOrEmpty(value.Trim());
    }

    [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
    public sealed class ThrowerException : Exception
    {
        [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
        private ThrowerException(string message)
            : base(message)
        {
        }

        internal static ThrowerException AbstractEx => new ThrowerException("Given exception type is abstract");

        internal static ThrowerException MissingNoArgsCtor => new ThrowerException("Given exception type has no parameterless constructor");

        internal static ThrowerException MissingMsgCtor => new ThrowerException("Given exception type has not a valid message constructor");
    }
}
```

7.3 RaiseArgumentException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentException

  Utility methods which can be used to handle bad arguments.

Namespaces

- namespace PommaLabs.Thrower

7.4 RaiseArgumentException.cs

```
// File name: RaiseArgumentException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using PommaLabs.Thrower.Validation;
using System;
using System.Collections.Generic;

namespace PommaLabs.Thrower
{
    public sealed class RaiseArgumentException : RaiseBase
    {
        #region If

        const string DefaultIfMessage = "Argument is not valid";

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition)
        {
            if (condition)
            {
                throw new ArgumentException(DefaultIfMessage);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, string argumentName, string message = null)
        {
            if (condition)
            {
                throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
            }
        }

        #endregion If

        #region IfNot

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition)
        {
            if (!condition)
            {
                throw new ArgumentException(DefaultIfMessage);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, string argumentName, string message = null)
        {
            if (!condition)
            {
                throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
            }
        }

        #endregion IfNot

        #region IfIsNotValid

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValid<TArg>(TArg argument)
        {
            IList<ValidationError> validationErrors;
            if (!ObjectValidator.Validate(argument, out validationErrors))
            {
                throw new ArgumentException(ObjectValidator.FormatValidationErrors(validationErrors, null));
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValid<TArg>(TArg argument, string argumentName, string message = null)
        {
            IList<ValidationError> validationErrors;
            if (!ObjectValidator.Validate(argument, out validationErrors))
            {
                throw new ArgumentException(ObjectValidator.FormatValidationErrors(validationErrors, message), argumentName);
            }
        }

        #endregion IfIsNotValid

        #region IfIsNotValidEmailAddress

        const string DefaultIfIsNotValidEmailAddressMessage = "String \"{0}\" is not a valid email address";

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValidEmailAddress(string emailAddress)
        {
            if (!EmailAddressValidator.Validate(emailAddress))
            {
                var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
                throw new ArgumentException(exceptionMsg);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif
```

```

        public static void IfIsNotValidEmailAddress(string emailAddress, bool allowInternational)
        {
            if (!EmailAddressValidator.Validate(emailAddress, allowInternational))
            {
                var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
                throw new ArgumentException(exceptionMsg);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValidEmailAddress(string emailAddress, string argumentName, string message = null)
        {
            if (!EmailAddressValidator.Validate(emailAddress))
            {
                var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
                throw new ArgumentException(exceptionMsg, argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValidEmailAddress(string emailAddress, bool allowInternational, string argumentName, string message = null)
        {
            if (!EmailAddressValidator.Validate(emailAddress, allowInternational))
            {
                var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
                throw new ArgumentException(exceptionMsg, argumentName);
            }
        }

        #endregion IfIsNotValidEmailAddress

        #region IfIsNotValidPhoneNumber

        const string DefaultIfIsNotValidPhoneNumberMessage = "String \"{0}\" is not a valid phone number";

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValidPhoneNumber(string phoneNumber)
        {
            if (!PhoneNumberValidator.Validate(phoneNumber))
            {
                var exceptionMsg = string.Format(DefaultIfIsNotValidPhoneNumberMessage, phoneNumber);
                throw new ArgumentException(exceptionMsg);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotValidPhoneNumber(string phoneNumber, string argumentName, string message = null)
        {
            if (!PhoneNumberValidator.Validate(phoneNumber))
            {
                var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidPhoneNumberMessage, phoneNumber);
                throw new ArgumentException(exceptionMsg, argumentName);
            }
        }

        #endregion IfIsNotValidPhoneNumber

        #region String validation

        const string IsNullOrEmptyMessage = "Argument cannot be a null or empty string";
        const string IsNullOrWhiteSpaceMessage = "Argument cannot be a null, empty or blank string";

#if (NET45 || NET46)
```

```
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNullOrEmpty(string value)
        {
            if (value == null || value == string.Empty)
            {
                throw new ArgumentException(IsNullOrEmptyMessage);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNullOrEmpty(string value, string argumentName, string message
 = null)
        {
            if (value == null || value == string.Empty)
            {
                throw new ArgumentException(message ?? IsNullOrEmptyMessage, argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNullOrWhiteSpace(string value)
        {
            if (value == null || value.Trim() == string.Empty)
            {
                throw new ArgumentException(IsNullOrWhiteSpaceMessage);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNullOrWhiteSpace(string value, string argumentName,
string message = null)
        {
            if (value == null || value.Trim() == string.Empty)
            {
                throw new ArgumentException(message ?? IsNullOrWhiteSpaceMessage, argumentName);
            }
        }

        #endregion String validation
    }
}
```

7.5 RaiseArgumentNullException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentNullException

  Utility methods which can be used to handle null references.

Namespaces

- namespace PommaLabs.Thrower

7.6 RaiseArgumentNullException.cs

```
// File name: RaiseArgumentNullException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
```

```
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using PommaLabs.Thrower.Reflection;
using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseArgumentNullException :
RaiseBase
    {
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNull<TArg>(TArg argument)
        {
            if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
            {
                throw new ArgumentNullException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNull<TArg>(TArg argument, string argumentName)
        {
            if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
            {
                throw new ArgumentNullException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNull<TArg>(TArg argument, string argumentName, string message)
        {
            if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
            {
                throw new ArgumentNullException(argumentName, message);
            }
        }
    }
}
```

7.7 RaiseArgumentOutOfRangeException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentOutOfRangeException

  Utility methods which can be used to handle ranges.

Namespaces

- namespace PommaLabs.Thrower

7.8 RaiseArgumentOutOfRangeException.cs

```
// File name: RaiseArgumentOutOfRangeException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseArgumentOutOfRangeException : RaiseBase
    {
        #region If

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, string argumentName = null)
        {
            if (condition)
            {
                throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, string argumentName, string message)
        {
            if (condition)
            {
                throw new ArgumentOutOfRangeException(argumentName, message);
            }
        }

        #endregion If

        #region IfNot

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, string argumentName = null)
        {
            if (!condition)
            {
                throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new ArgumentOutOfRangeException(argumentName);
            }
        }
```

```

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, string argumentName, string message)
        {
            if (!condition)
            {
                throw new ArgumentOutOfRangeException(argumentName, message);
            }
        }

        #endregion IfNot

        #region Less - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion Less - Without parameter name, without message

        #region Less - With parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess(IComparable argument1, IComparable argument2, string
argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion Less - With parameter name, without message

        #region Less - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess(IComparable argument1, IComparable argument2, string argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion Less - With parameter name, with message

        #region LessEqual - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion LessEqual - Without parameter name, without message

        #region LessEqual - With parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2, string argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion LessEqual - With parameter name, without message

        #region LessEqual - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
string argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion LessEqual - With parameter name, with message

        #region Greater - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion Greater - Without parameter name, without message

        #region Greater - With parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater(IComparable argument1, IComparable argument2, string
argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion Greater - With parameter name, without message

        #region Greater - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater(IComparable argument1, IComparable argument2, string
argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion Greater - With parameter name, with message

        #region GreaterEqual - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion GreaterEqual - Without parameter name, without message

        #region GreaterEqual - With parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
argument2, string argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion GreaterEqual - With parameter name, without message

        #region GreaterEqual - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
argument2, string argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion GreaterEqual - With parameter name, with message

        #region Equal - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion Equal - Without parameter name, without message

        #region Equal - With parameter name, without message

#if (NET45 || NET46)
```

```
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual(IComparable argument1, IComparable argument2, string
argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion Equal - With parameter name, without message

        #region Equal - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual(IComparable argument1, IComparable argument2, string
argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion Equal - With parameter name, with message

        #region NotEqual - Without parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException();
            }
        }

        #endregion NotEqual - Without parameter name, without message

        #region NotEqual - With parameter name, without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
argumentName)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException(argumentName);
            }
        }

        #endregion NotEqual - With parameter name, without message

        #region NotEqual - With parameter name, with message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
argumentName, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new ArgumentOutOfRangeException(argumentName, argument1, message);
            }
        }

        #endregion NotEqual - With parameter name, with message
    }
}
```

7.9 RaiseHttpException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseHttpException

  Utility methods which can be used to handle error codes through HTTP.
- struct PommaLabs.Thrower.HttpExceptionInfo

Additional info which will be included into HttpException.

- class PommaLabs.Thrower.HttpException

  Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.

Namespaces

- namespace PommaLabs.Thrower

7.10 RaiseHttpException.cs

```
// File name: RaiseHttpException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using PommaLabs.Thrower.Validation;
using System;
using System.Net;

namespace PommaLabs.Thrower
{
    public static class RaiseHttpException
    {
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, HttpStatusCode httpStatusCode, string message = null)
        {
            if (condition)
            {
                throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
HttpException(httpStatusCode, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, HttpStatusCode httpStatusCode, string message,
HttpExceptionInfo additionalInfo)
        {
            if (condition)
            {
                throw new HttpException(httpStatusCode, message, additionalInfo);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message = null)
        {
            if (!condition)
            {
                throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new HttpException(httpStatusCode, message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)
        {
            if (!condition)
            {
                throw new HttpException(httpStatusCode, message, additionalInfo);
            }
        }
    }

    public struct HttpExceptionInfo
    {
        public HttpExceptionInfo(object errorCode = null, string userMessage = null)
        {
            ErrorCode = errorCode ?? HttpException.DefaultErrorCode;
            UserMessage = userMessage ?? HttpException.DefaultUserMessage;
        }

        [Validate(Required = false)]
        public object ErrorCode { get; set; }

        [Validate(Required = false)]
        public string UserMessage { get; set; }
    }

    public sealed class HttpException : Exception
    {
        public HttpException(HttpStatusCode httpStatusCode)
            : this(httpStatusCode, new HttpExceptionInfo())
        {
        }

        public HttpException(HttpStatusCode httpStatusCode, HttpExceptionInfo additionalInfo)
            : base()
        {
            HttpStatusCode = httpStatusCode;
            ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
            UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
        }

        public HttpException(HttpStatusCode httpStatusCode, string message)
            : this(httpStatusCode, message, new HttpExceptionInfo())
        {
        }

        public HttpException(HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)
            : base(message)
        {
            HttpStatusCode = httpStatusCode;
            ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
            UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
        }

        public HttpException(HttpStatusCode httpStatusCode, string message, Exception innerException)
            : this(httpStatusCode, message, innerException, new HttpExceptionInfo())
        {
        }

        public HttpException(HttpStatusCode httpStatusCode, string message, Exception innerException, HttpExceptionInfo additionalInfo)
            : base(message, innerException)
        {
            HttpStatusCode = httpStatusCode;
            ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
            UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
        }

        public HttpStatusCode HttpStatusCode { get; }

        public object ErrorCode { get; }

        public static object DefaultErrorCode { get; set; } = "unspecified";

        public string UserMessage { get; }

        public static string DefaultUserMessage { get; set; } = "unspecified";
    }
}
```

7.11 RaiseIndexOutOfRangeException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseIndexOutOfRangeException

  Utility methods which can be used to handle indexes.

Namespaces

- namespace PommaLabs.Thrower

7.12 RaiseIndexOutOfRangeException.cs

```
// File name: RaiseIndexOutOfRangeException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseIndexOutOfRangeException :
RaiseBase
    {
        #region Less - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion Less - Without message

        #region Less - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLess(IComparable argument1, IComparable argument2, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion Less - With message

        #region LessEqual - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion LessEqual - Without message

        #region LessEqual - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

```

```
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion LessEqual - With message

        #region Greater - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion Greater - Without message

        #region Greater - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreater(IComparable argument1, IComparable argument2, string
message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion Greater - With message

        #region GreaterEqual - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
```

```
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion GreaterEqual - Without message

        #region GreaterEqual - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion GreaterEqual - With message

        #region Equal - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion Equal - Without message

        #region Equal - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsEqual(IComparable argument1, IComparable argument2, string message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion Equal - With message

        #region NotEqual - Without message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new IndexOutOfRangeException();
            }
        }

        #endregion NotEqual - Without message

        #region NotEqual - With message

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string message)
            where TArg : IComparable<TArg>
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
message)
        {
            if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
            {
                throw new IndexOutOfRangeException(message);
            }
        }

        #endregion NotEqual - With message
    }
}
```

7.13 RaiseInvalidOperationException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseInvalidOperationException

  Utility methods which can be used to handle bad object states.

Namespaces

- namespace PommaLabs.Thrower

7.14 RaiseInvalidOperationException.cs

```
// File name: RaiseInvalidOperationException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseInvalidOperationException : RaiseBase
    {
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, string message = null)
        {
            if (condition)
            {
                throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new InvalidOperationException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, string message = null)
        {
            if (!condition)
            {
                throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new InvalidOperationException(message);
            }
        }
    }
}
```

7.15 RaiseNotSupportedException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseNotSupportedException

    Utility methods which can be used to handle unsupported operations.

Namespaces

- namespace PommaLabs.Thrower

7.16 RaiseNotSupportedException.cs

```
// File name: RaiseNotSupportedException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseNotSupportedException : RaiseBase
    {
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool condition, string message = null)
        {
            if (condition)
            {
                throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new NotSupportedException(message);
            }
        }

#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void IfNot(bool condition, string message = null)
        {
            if (!condition)
            {
                throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new NotSupportedException(message);
            }
        }
    }
}
```

7.17 RaiseObjectDisposedException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseObjectDisposedException

  Utility methods which can be used to handle bad object states.

Namespaces

- namespace PommaLabs.Thrower

7.18 RaiseObjectDisposedException.cs

```
// File name: RaiseObjectDisposedException.cs
//
// Author(s): Alessio Parma <alessio.parma@gmail.com>
//
// The MIT License (MIT)
//
// Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
//
// Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
// associated documentation files (the "Software"), to deal in the Software without restriction,
// including without limitation the rights to use, copy, modify, merge, publish, distribute,
// sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
// furnished to do so, subject to the following conditions:
//
// The above copyright notice and this permission notice shall be included in all copies or
// substantial portions of the Software.
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
// NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
// DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
// OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

using System;

namespace PommaLabs.Thrower
{
    public sealed class RaiseObjectDisposedException : RaiseBase
    {
#if (NET45 || NET46)
        [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.MethodImplOptions.AggressiveInlining)]
#endif

        public static void If(bool disposed, string objectName, string message = null)
        {
            if (disposed)
            {
                throw string.IsNullOrEmpty(message) ? new ObjectDisposedException(objectName) : new ObjectDisposedException(objectName, message);
            }
        }
    }
}
```

Index