PommaLabs.Thrower

3.0.0

Generated by Doxygen 1.8.11

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

Namespaces

- namespace ExceptionHandlers
- namespace Reflection
- namespace Validation

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

  New exception handling mechanism, which is more fluent than the old ones.
- 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).

5.3 PommaLabs.Thrower.ExceptionHandlers Namespace Reference

Classes

- class ArgumentExceptionHandler

  Handler for ArgumentException
- class ArgumentNullExceptionHandler

  Handler for ArgumentNullException.
- class ArgumentOutOfRangeExceptionHandler

  Handler for System.ArgumentOutOfRangeException
- class GenericExceptionHandler

  Generic handler used for common exceptions like NotSupportedException.
- class HttpExceptionHandler

  Handler for HttpException
- class IndexOutOfRangeExceptionHandler

  Handler for System.IndexOutOfRangeException
- class InvalidOperationExceptionHandler

  Handler for InvalidOperationException.
- class NotSupportedExceptionHandler

  Handler for NotSupportedException.
- class ObjectDisposedExceptionHandler

  Handler for ObjectDisposedException.

5.4 PommaLabs.Thrower.Reflection Namespace Reference

Namespaces

- namespace FastMember

Classes

- class PortableTypeInfo

  Portable version of some useful reflection methods.

5.5 PommaLabs.Thrower.Reflection.FastMember Namespace Reference

Classes

- class **CallSiteCache**
- class Member

  Represents an abstracted view of an individual member defined for a type.
- class MemberSet

  Represents an abstracted view of the members defined for a type.
- class ObjectAccessor

  Represents an individual object, allowing access to members by-name.
- class **ObjectReader**

  Provides a means of reading a sequence of objects as a data-reader, for example for use with SqlBulkCopy or other data-base oriented code
- class TypeAccessor

  Provides by-name member-access to objects of a given type.

5.6 PommaLabs.Thrower.Validation Namespace Reference

Classes

- class EmailAddressValidator

  An email address validator.
- class ObjectValidator

  Validates an object public properties that have been decorated with the ValidateAttribute custom attribute.
- class PhoneNumberValidator

  A phone number validator.
- class ValidateAttribute

  Indicates that the property should be validated.
- struct ValidationError

  Represents an error found while validating an object.

Chapter 6

Class Documentation

6.1 PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler Class Reference

Handler for ArgumentException

Public Member Functions

- void If (bool condition)

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

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

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

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

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

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

  Throws ArgumentException if given string is not a valid email address.
- void IfIsNotValidEmailAddress (string emailAddress, EmailAddressValidator.Options validatorOptions)

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

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

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

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

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

  Throws ArgumentException if given string is null or empty.

- void IfIsNullOrEmpty (string value, string argumentName, string message=null)

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

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

  Throws ArgumentException if given string is null, empty or blank.
- void IfIsNullOrEmpty< TItem > (ICollection< TItem > value)

  Throws ArgumentException if given collection is null or empty.
- void IfIsNullOrEmpty< TItem > (ICollection< TItem > value, string argumentName, string message=null)

  Throws ArgumentException if given collection is null or empty.

6.1.1 Detailed Description

Handler for ArgumentException

Definition at line 35 of file ArgumentExceptionHandler.cs.

6.1.2 Member Function Documentation

6.1.2.1 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.If (bool *condition*)

Throws ArgumentException if given condition is true.

Parameters

condition	The condition.

Exceptions

ArgumentException	If given condition is true.

Definition at line 46 of file ArgumentExceptionHandler.cs.

6.1.2.2 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.If (bool *condition,* string *argumentName,* string *message* = `null`)

Throws ArgumentException if given condition is true.

Parameters

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

Exceptions

ArgumentException	If given condition is true.

message and *argumentName* are strictly required arguments.

Definition at line 64 of file ArgumentExceptionHandler.cs.

6.1.2.3 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValid< TArg > (TArg *argument*)

Throws ArgumentException if given argument is not valid.

Template Parameters

TArg	The type of the argument.

Parameters

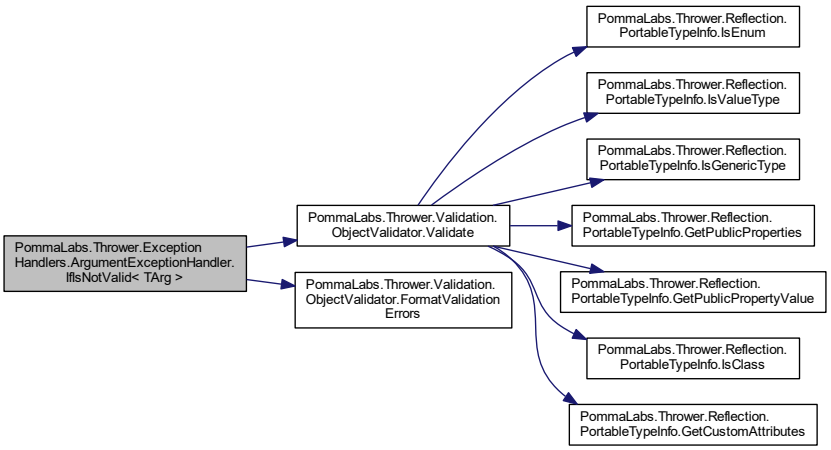
argument	The argument.

Exceptions

ArgumentException	If given argument is not valid.

Definition at line 117 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.4 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValid< TArg > (TArg *argument*, string *argumentName*, string *message* = `null`)

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.

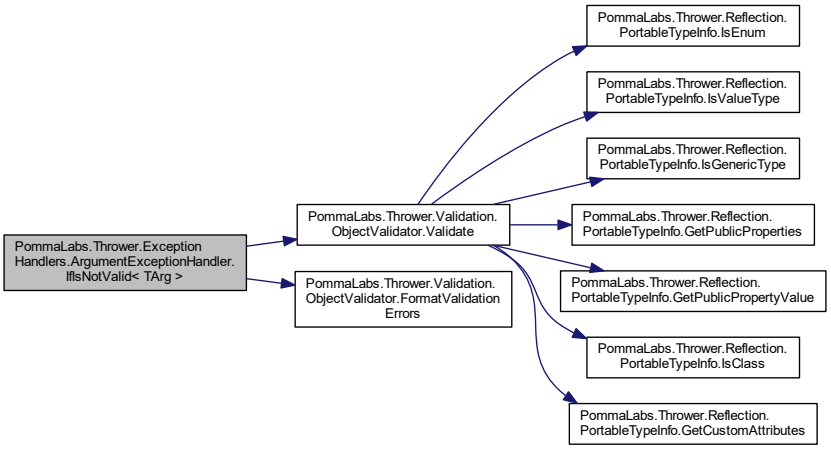
Exceptions

ArgumentException	If given argument is not valid.

message and *argumentName* are strictly required arguments.

Definition at line 137 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.5 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidEmailAddress (string *emailAddress*)

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

Parameters

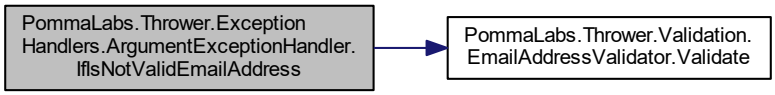
emailAddress	An email address.

Exceptions

ArgumentException	If given string is not a valid email address.

Definition at line 157 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.6 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidEmailAddress (string *emailAddress*, EmailAddressValidator.Options *validatorOptions*)

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

Parameters

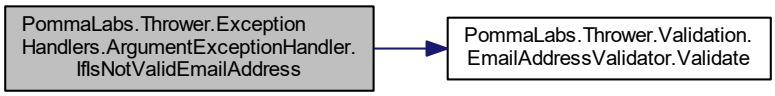
emailAddress	An email address.
validatorOptions	Customizations for the validation process.

Exceptions

ArgumentException	If given string is not a valid email address.

Definition at line 172 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.7 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidEmailAddress (string *emailAddress*, string *argumentName*, string *message* = `null`)

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.

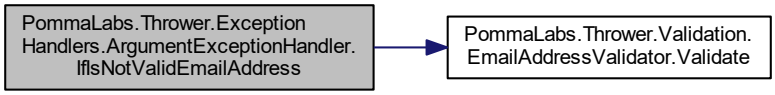
Exceptions

ArgumentException	If given string is not a valid email address.

message and *argumentName* are strictly required arguments.

Definition at line 191 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.8 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidEmailAddress (string *emailAddress*, string *argumentName*, EmailAddressValidator.Options *validatorOptions*, string *message* = `null`)

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

Parameters

emailAddress	An email address.
argumentName	The name of the argument.
validatorOptions	Customizations for the validation process.
message	The message.

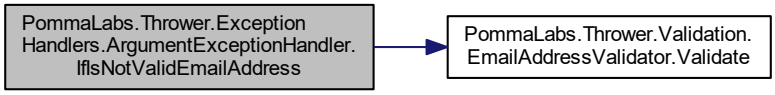
Exceptions

ArgumentException	If given string is not a valid email address.

message and *argumentName* are strictly required arguments.

Definition at line 211 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.9 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidPhoneNumber (string *phoneNumber*)

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

Parameters

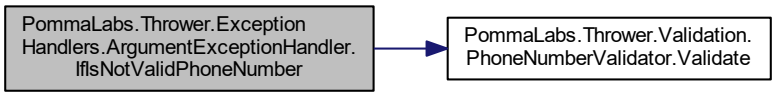
phoneNumber	A phone number.

Exceptions

ArgumentException	If given string is not a valid phone number.

Definition at line 231 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.10 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNotValidPhoneNumber (string *phoneNumber*, string *argumentName*, string *message* = `null`)

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.

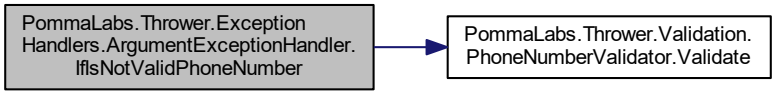
Exceptions

ArgumentException	If given string is not a valid phone number.

message and *argumentName* are strictly required arguments.

Definition at line 250 of file ArgumentExceptionHandler.cs.

Here is the call graph for this function:



6.1.2.11 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrEmpty (string *value*)

Throws ArgumentException if given string is null or empty.

Parameters

value	The string value.

Exceptions

ArgumentException	If given string is null or empty.

Definition at line 271 of file ArgumentExceptionHandler.cs.

6.1.2.12 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrEmpty (string *value*, string *argumentName*, string *message* = `null`)

Throws ArgumentException if given string is null or empty.

Parameters

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

Exceptions

ArgumentException	If given string is null or empty.

message and *argumentName* are strictly required arguments.

Definition at line 289 of file ArgumentExceptionHandler.cs.

6.1.2.13 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrEmpty< TItem > (ICollection< TItem > *value*)

Throws ArgumentException if given collection is null or empty.

Template Parameters

TItem	The type of the items contained in the collection.

Parameters

value	The collection.

Exceptions

ArgumentException	If given collection is null or empty.

Definition at line 340 of file ArgumentExceptionHandler.cs.

6.1.2.14 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrEmpty< TItem > (ICollection< TItem > *value*, string *argumentName*, string *message* = `null`)

Throws ArgumentException if given collection is null or empty.

Template Parameters

TItem	The type of the items contained in the collection.

Parameters

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

Exceptions

ArgumentException	If given collection is null or empty.

message and *argumentName* are strictly required arguments.

Definition at line 359 of file ArgumentExceptionHandler.cs.

6.1.2.15 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrWhiteSpace (string *value*)

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

Parameters

value	The string value.

Exceptions

ArgumentException	If given string is null, empty or blank.

Definition at line 302 of file ArgumentExceptionHandler.cs.

6.1.2.16 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfIsNullOrWhiteSpace (string *value*, string *argumentName*, string *message* = `null`)

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.

Exceptions

ArgumentException	If given string is null, empty or blank.

message and *argumentName* are strictly required arguments.

Definition at line 320 of file ArgumentExceptionHandler.cs.

6.1.2.17 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfNot (bool *condition*)

Throws ArgumentException if given condition is false.

Parameters

condition	The condition.

Exceptions

ArgumentException	If given condition is false.

Definition at line 81 of file ArgumentExceptionHandler.cs.

6.1.2.18 void PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler.IfNot (bool *condition*, string *argumentName*, string *message* = `null`)

Throws ArgumentException if given condition is false.

Parameters

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

Exceptions

ArgumentException	If given condition is false.

message and *argumentName* are strictly required arguments.

Definition at line 99 of file ArgumentExceptionHandler.cs.

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

- ExceptionHandlers/ArgumentExceptionHandler.cs

6.2 PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler Class Reference

Handler for ArgumentNullException.

Public Member Functions

- void If (bool condition)

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

  Throws ArgumentException if given condition is true.
- void IfIsNull< TArg > (TArg argument)

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

  Throws ArgumentNullException if given argument if null.
- void IfIsNull< TArg > (TArg?argument)

  Throws ArgumentNullException if given argument if null.
- void IfIsNull< TArg > (ref TArg?argument)

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

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

  Throws ArgumentNullException if given argument if null.

6.2.1 Detailed Description

Handler for ArgumentNullException.

Definition at line 34 of file ArgumentNullExceptionHandler.cs.

6.2.2 Member Function Documentation

6.2.2.1 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.If (bool *condition*)

Throws ArgumentNullException if given condition is true.

Parameters

condition	The condition.

Exceptions

ArgumentNullException	If given condition is true.

Definition at line 45 of file ArgumentNullExceptionHandler.cs.

6.2.2.2 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.If (bool *condition*, string *argumentName*, string *message* = `null`)

Throws ArgumentException if given condition is true.

Parameters

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

Exceptions

ArgumentNullException	If given condition is true.

message and *argumentName* are strictly required arguments.

Definition at line 63 of file ArgumentNullExceptionHandler.cs.

6.2.2.3 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (TArg *argument*)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the argument.

Parameters

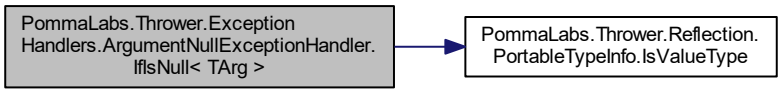
argument	The argument.

Exceptions

ArgumentNullException	If given argument is null.

Definition at line 81 of file ArgumentNullExceptionHandler.cs.

Here is the call graph for this function:



6.2.2.4 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (TArg *argument*, string *argumentName*, string *message* = `null`)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the argument.

Parameters

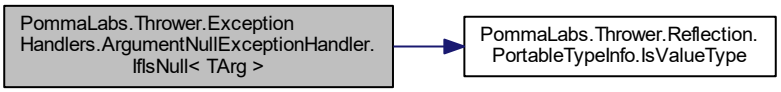
argument	The argument.
argumentName	The name of the argument.
message	The message that should be put into the exception.

Exceptions

ArgumentNullException	If given argument is null.

Definition at line 97 of file ArgumentNullExceptionHandler.cs.

Here is the call graph for this function:



6.2.2.5 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (TArg? *argument*)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the nullable argument.

Parameters

argument	The argument.

Exceptions

ArgumentNullException	If given argument has no value.

Type Constraints

TArg* : *struct

Definition at line 115 of file ArgumentNullExceptionHandler.cs.

6.2.2.6 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (ref TArg? *argument*)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the nullable argument.

Parameters

argument	The argument, by reference.

Exceptions

ArgumentNullException	If given argument has no value.

Type Constraints

TArg* : *struct

Definition at line 130 of file ArgumentNullExceptionHandler.cs.

6.2.2.7 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (TArg? *argument*, string *argumentName*, string *message* = `null`)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the nullable argument.

Parameters

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

Exceptions

ArgumentNullException	If given argument has no value.

Type Constraints

TArg* : *struct

Definition at line 147 of file ArgumentNullExceptionHandler.cs.

6.2.2.8 void PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler.IfIsNull< TArg > (ref TArg? *argument*, string *argumentName*, string *message* = `null`)

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the nullable argument.

Parameters

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

Exceptions

ArgumentNullException	If given argument has no value.

Type Constraints

TArg **:** ***struct***

Definition at line 164 of file ArgumentNullExceptionHandler.cs.

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

- ExceptionHandlers/ArgumentNullExceptionHandler.cs

6.3 PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler Class Reference

Handler for System.ArgumentOutOfRangeException

Public Member Functions

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

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

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

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

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

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

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

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

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

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

Throws ArgumentOutOfRangeException if argument1 is less than argument2 .

- void IfIsLessOrEqual< TArg > (TArg argument1, TArg argument2)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .

- void IfIsNotEqual< TArg > (TArg argument1, TArg argument2)

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

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

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

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

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

  Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .

6.3.1 Detailed Description

Handler for System.ArgumentOutOfRangeException

Definition at line 33 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2 Member Function Documentation

6.3.2.1 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.If (bool *condition,* string *argumentName* = `null`)

Throws ArgumentOutOfRangeException if given condition is true.

Parameters

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

Definition at line 42 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.2 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.If (bool *condition,* string *argumentName,* string *message*)

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 59 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.3 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 559 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.4 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*)

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 595 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.5 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*)

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 633 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.6 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 544 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.7 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*)

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 579 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.8 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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.

Parameters

message	The message that should be put into the exception.

Type Constraints

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

Definition at line 616 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.9 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 343 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.10 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater (IComparable *argument1*, IComparable *argument2*, string *argumentName*)

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 379 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.11 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*)

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 417 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.12 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater< TArg >(TArg *argument1*, TArg *argument2*)

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 328 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.13 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater< TArg >(TArg *argument1*, TArg *argument2*, string *argumentName*)

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 363 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.14 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreater< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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 400 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.15 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*)

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 451 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.16 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*)

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 487 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.17 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*)

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 525 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.18 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 436 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.19 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*)

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.

Parameters

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

Type Constraints

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

Definition at line 471 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.20 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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 508 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.21 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 127 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.22 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess (IComparable *argument1*, IComparable *argument2*, string *argumentName*)

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 163 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.23 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*)

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 201 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.24 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*)

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 112 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.25 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*)

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 147 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.26 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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 184 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.27 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*)

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 235 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.28 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*)

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 271 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.29 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *argumentName*, string *message*)

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 309 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.30 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*)

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.

Parameters

argument2	The right side argument.

Type Constraints

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

Definition at line 220 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.31 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*)

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 255 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.32 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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.

Parameters

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

Type Constraints

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

Definition at line 292 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.33 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual (IComparable *argument1,* IComparable *argument2*)

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

Parameters

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

Definition at line 667 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.34 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual (IComparable *argument1,* IComparable *argument2,* string *argumentName*)

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 703 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.35 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual (IComparable *argument1,* IComparable *argument2,* string *argumentName,* string *message*)

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 741 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.36 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 652 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.37 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*)

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 687 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.38 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfIsNotEqual< TArg > (TArg *argument1*, TArg *argument2*, string *argumentName*, string *message*)

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 724 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.39 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfNot (bool *condition*, string *argumentName* = `null`)

Throws ArgumentOutOfRangeException if given condition is false.

Parameters

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

Definition at line 76 of file ArgumentOutOfRangeExceptionHandler.cs.

6.3.2.40 void PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler.IfNot (bool *condition*, string *argumentName*, string *message*)

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 93 of file ArgumentOutOfRangeExceptionHandler.cs.

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

- ExceptionHandlers/ArgumentOutOfRangeExceptionHandler.cs

6.4 PommaLabs.Thrower.Validation.EmailAddressValidator Class Reference

An email address validator.

Public Types

- enum Options { Options.None = 0, Options.AllowInternational = 1, Options.AllowTopLevelDomains = 2 }
  Options used by validation process.

Static Public Member Functions

- static bool Validate (string emailAddress, Options options=Options.None)
  Validates the specified email address.

6.4.1 Detailed Description

An email address validator.

An email address validator.

Definition at line 30 of file EmailAddressValidator.cs.

6.4.2 Member Enumeration Documentation

6.4.2.1 enum PommaLabs.Thrower.Validation.EmailAddressValidator.Options [strong]

Options used by validation process.

Enumerator

None No option specified.

AllowInternational Whether the validator should allow international characters or not.

AllowTopLevelDomains Whether the validator should allow addresses at top-level domains or not.

Definition at line 337 of file EmailAddressValidator.cs.

6.4.3 Member Function Documentation

6.4.3.1 static bool PommaLabs.Thrower.Validation.EmailAddressValidator.Validate (string *emailAddress*, Options *options* = Options.None) [static]

Validates the specified email address.

Validates the syntax of an email address.

If *options* contains Options.AllowInternational, then the validator will use the newer International Email standards for validating the email address.

Returns

`true` if the email address is valid; otherwise `false`.

Parameters

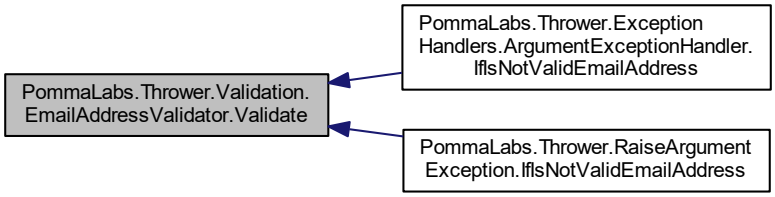
emailAddress	An email address.
options	Customizations for the validation process.

Exceptions

System.ArgumentNullException	*emailAddress* is `null`.

Definition at line 265 of file EmailAddressValidator.cs.

Here is the caller graph for this function:



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

- Validation/EmailAddressValidator.cs

6.5 PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< TException > Class Template Reference

Generic handler used for common exceptions like NotSupportedException.

Public Member Functions

- void If (bool condition, string message=null)
  Throws TException if given condition is true.
- void IfNot (bool condition, string message=null)
  Throws TException if given condition is false.

Protected Member Functions

- abstract TException NewWithMessage (string message)
  Creates an exception with given message.

6.5.1 Detailed Description

Generic handler used for common exceptions like NotSupportedException.

Template Parameters

TException	The type of the handled exception.

Type Constraints

TException* : *Exception

TException* : *new()

Definition at line 32 of file GenericExceptionHandler.cs.

6.5.2 Member Function Documentation

6.5.2.1 void PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< TException >.If (bool *condition*, string *message* = `null`)

Throws *TException* if given condition is true.

Parameters

condition	The condition.
message	The optional message.

Exceptions

Exception	If given condition is true, an exception of type *TException* is thrown.

Definition at line 56 of file GenericExceptionHandler.cs.

6.5.2.2 void PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< TException >.IfNot (bool *condition*, string *message* = `null`)

Throws *TException* if given condition is false.

Parameters

condition	The condition.
message	The optional message.

Exceptions

Exception	If given condition is true, an exception of type *TException* is thrown.

Definition at line 72 of file GenericExceptionHandler.cs.

6.5.2.3 abstract TException PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< TException >.NewWithMessage (string *message*) [protected], [pure virtual]

Creates an exception with given message.

Parameters

message	The message used by the exception.

Returns

An exception with given message.

Implemented in PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler, and PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler.

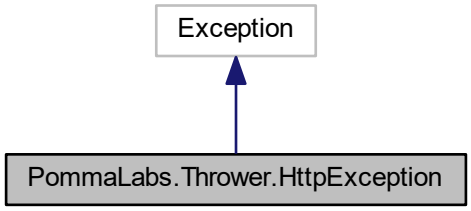
The documentation for this class was generated from the following file:

- ExceptionHandlers/GenericExceptionHandler.cs

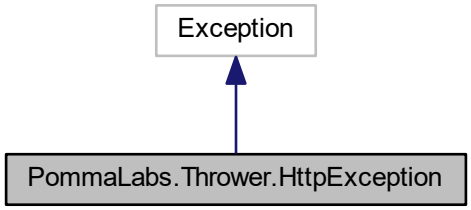
6.6 PommaLabs.Thrower.HttpException Class Reference

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.

6.6.1 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 70 of file HttpException.cs.

6.6.2 Constructor & Destructor Documentation

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

Builds the exception using given status code.

Parameters

httpStatusCode	The HTTP status code.

Definition at line 76 of file HttpException.cs.

6.6.2.2 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 86 of file HttpException.cs.

6.6.2.3 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 100 of file HttpException.cs.

6.6.2.4 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 111 of file HttpException.cs.

6.6.2.5 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 127 of file HttpException.cs.

6.6.2.6 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 139 of file HttpException.cs.

6.6.3 Property Documentation

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

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

Definition at line 162 of file HttpException.cs.

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

The default user message.

Definition at line 172 of file HttpException.cs.

6.6.3.3 object PommaLabs.Thrower.HttpException.ErrorCode [get]

The application defined error code.

Definition at line 157 of file HttpException.cs.

6.6.3.4 HttpStatusCode PommaLabs.Thrower.HttpException.HttpStatusCode [get]

The HTTP status code assigned to this exception.

Definition at line 152 of file HttpException.cs.

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

An error message which can be shown to the user.

Definition at line 167 of file HttpException.cs.

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

- HttpException.cs

6.7 PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler Class Reference

Handler for HttpException

Public Member Functions

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

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

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

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

  Throws HttpException if given condition is false.

6.7.1 Detailed Description

Handler for HttpException

Definition at line 33 of file HttpExceptionHandler.cs.

6.7.2 Member Function Documentation

6.7.2.1 void PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler.If (bool *condition*, HttpStatusCode *httpStatusCode*, string *message* = null)

Throws HttpException if given condition is true.

Parameters

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

Exceptions

HttpException	If given condition is true.

Definition at line 42 of file HttpExceptionHandler.cs.

6.7.2.2 void PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler.If (bool *condition*, HttpStatusCode *httpStatusCode*, string *message*, HttpExceptionInfo *additionalInfo*)

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.

Exceptions

HttpException	If given condition is true.

Definition at line 58 of file HttpExceptionHandler.cs.

6.7.2.3 void PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler.IfNot (bool *condition*, HttpStatusCode *httpStatusCode*, string *message* = `null`)

Throws HttpException if given condition is false.

Parameters

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

Exceptions

HttpException	If given condition is false.

Definition at line 73 of file HttpExceptionHandler.cs.

6.7.2.4 void PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler.IfNot (bool *condition*, HttpStatusCode *httpStatusCode*, string *message*, HttpExceptionInfo *additionalInfo*)

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.

Exceptions

HttpException	If given condition is false.

Definition at line 89 of file HttpExceptionHandler.cs.

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

- ExceptionHandlers/HttpExceptionHandler.cs

6.8 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.

6.8.1 Detailed Description

Additional info which will be included into HttpException.

Definition at line 38 of file HttpException.cs.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 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 45 of file HttpException.cs.

6.8.3 Property Documentation

6.8.3.1 object PommaLabs.Thrower.HttpExceptionInfo.ErrorCode [get], [set]

The application defined error code.

Definition at line 55 of file HttpException.cs.

6.8.3.2 string PommaLabs.Thrower.HttpExceptionInfo.UserMessage [get], [set]

An error message which can be shown to user.

Definition at line 61 of file HttpException.cs.

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

- HttpException.cs

6.9 PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler Class Reference

Handler for System.IndexOutOfRangeException

Public Member Functions

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

- void IfIsLessOrEqual (IComparable argument1, IComparable argument2, string message)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  Throws IndexOutOfRangeException if argument1 is not equal to argument2 .

6.9.1 Detailed Description

Handler for System.IndexOutOfRangeException

Definition at line 33 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2 Member Function Documentation

6.9.2.1 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsEqual (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 339 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.2 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsEqual (IComparable *argument1*, IComparable *argument2*, string *message*)

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 375 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.3 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 324 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.4 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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 359 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.5 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreater (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 199 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.6 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreater (IComparable *argument1*, IComparable *argument2*, string *message*)

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 235 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.7 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreater< TArg > (TArg *argument1*, TArg *argument2*)

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 184 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.8 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreater< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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 219 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.9 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*)

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 269 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.10 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreaterOrEqual (IComparable *argument1*, IComparable *argument2*, string *message*)

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 305 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.11 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 254 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.12 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsGreaterOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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 289 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.13 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLess (IComparable *argument1*, IComparable *argument2*)

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

Parameters

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

Definition at line 59 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.14 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLess (IComparable *argument1*, IComparable *argument2*, string *message*)

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 95 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.15 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*)

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 44 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.16 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLess< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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 79 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.17 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*)

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 129 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.18 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLessOrEqual (IComparable *argument1*, IComparable *argument2*, string *message*)

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 165 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.19 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*)

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 114 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.20 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsLessOrEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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.

Parameters

message	The message that should be put into the exception.

Type Constraints

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

Definition at line 149 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.21 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsNotEqual (IComparable *argument1,* IComparable *argument2*)

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

Parameters

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

Definition at line 409 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.22 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsNotEqual (IComparable *argument1,* IComparable *argument2,* string *message*)

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 445 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.23 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsNotEqual< TArg >(TArg *argument1,* TArg *argument2*)

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 394 of file IndexOutOfRangeExceptionHandler.cs.

6.9.2.24 void PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler.IfIsNotEqual< TArg > (TArg *argument1*, TArg *argument2*, string *message*)

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 429 of file IndexOutOfRangeExceptionHandler.cs.

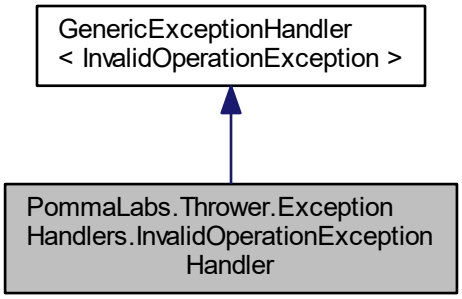
The documentation for this class was generated from the following file:

- ExceptionHandlers/IndexOutOfRangeExceptionHandler.cs

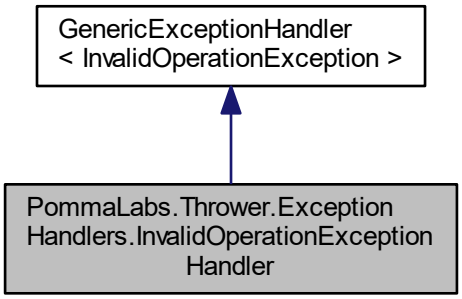
6.10 PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler Class Reference

Handler for InvalidOperationException.

Inheritance diagram for PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler:



Collaboration diagram for PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler:



Protected Member Functions

- override InvalidOperationException NewWithMessage (string message)

  Creates an exception with given message.

Additional Inherited Members

6.10.1 Detailed Description

Handler for InvalidOperationException.

Definition at line 33 of file InvalidOperationExceptionHandler.cs.

6.10.2 Member Function Documentation

6.10.2.1 override InvalidOperationException PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler.NewWithMessage (string *message*) [protected], [virtual]

Creates an exception with given message.

Parameters

message	The message used by the exception.

Returns

An exception with given message.

Implements PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< InvalidOperationException >.

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

- ExceptionHandlers/InvalidOperationExceptionHandler.cs

6.11 PommaLabs.Thrower.Reflection.FastMember.Member Class Reference

Represents an abstracted view of an individual member defined for a type.

Public Member Functions

- bool IsDefined (Type attributeType)
  Is the attribute specified defined on this type.

Public Attributes

- string Name => member.Name
  The name of this member.

Properties

- Type Type [get]
  The type of value stored in this member.

6.11.1 Detailed Description

Represents an abstracted view of an individual member defined for a type.

Definition at line 105 of file MemberSet.cs.

6.11.2 Member Function Documentation

6.11.2.1 bool PommaLabs.Thrower.Reflection.FastMember.Member.IsDefined (Type *attributeType*)

Is the attribute specified defined on this type.

Parameters

attributeType	The attribute type.

Definition at line 139 of file MemberSet.cs.

6.11.3 Member Data Documentation

6.11.3.1 string PommaLabs.Thrower.Reflection.FastMember.Member.Name => member.Name

The name of this member.

Definition at line 117 of file MemberSet.cs.

6.11.4 Property Documentation

6.11.4.1 Type PommaLabs.Thrower.Reflection.FastMember.Member.Type [get]

The type of value stored in this member.

Definition at line 123 of file MemberSet.cs.

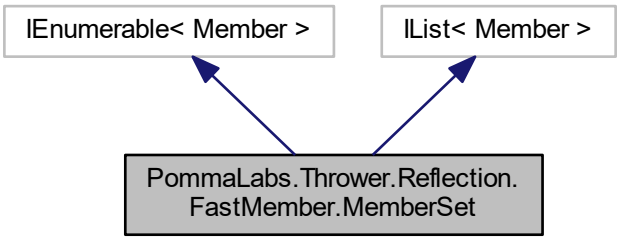
The documentation for this class was generated from the following file:

- Reflection/FastMember/MemberSet.cs

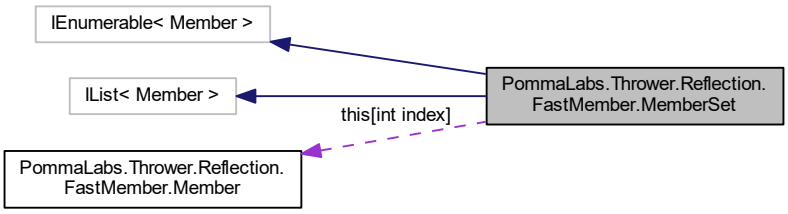
6.12 PommaLabs.Thrower.Reflection.FastMember.MemberSet Class Reference

Represents an abstracted view of the members defined for a type.

Inheritance diagram for PommaLabs.Thrower.Reflection.FastMember.MemberSet:



Collaboration diagram for PommaLabs.Thrower.Reflection.FastMember.MemberSet:



Public Member Functions

- IEnumerator< Member > GetEnumerator ()

  Return a sequence of all defined members.

Public Attributes

- Member this[int index] => members[index]

  Get a member by index
- int Count => members.Length

  The number of members defined for this type.

6.12.1 Detailed Description

Represents an abstracted view of the members defined for a type.

Definition at line 25 of file MemberSet.cs.

6.12.2 Member Function Documentation

6.12.2.1 IEnumerator<Member> PommaLabs.Thrower.Reflection.FastMember.MemberSet.GetEnumerator ( )

Return a sequence of all defined members.

Definition at line 42 of file MemberSet.cs.

6.12.3 Member Data Documentation

6.12.3.1 int PommaLabs.Thrower.Reflection.FastMember.MemberSet.Count => members.Length

The number of members defined for this type.

Definition at line 55 of file MemberSet.cs.

6.12.3.2 Member PommaLabs.Thrower.Reflection.FastMember.MemberSet.this[int index] => members[index]

Get a member by index

Definition at line 50 of file MemberSet.cs.

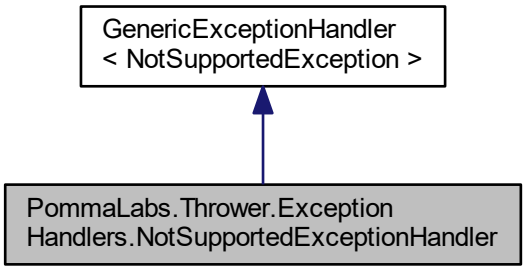
The documentation for this class was generated from the following file:

- Reflection/FastMember/MemberSet.cs

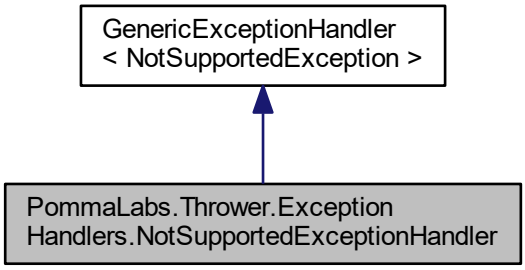
6.13 PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler Class Reference

Handler for NotSupportedException.

Inheritance diagram for PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler:



Collaboration diagram for PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler:



Protected Member Functions

- override NotSupportedException NewWithMessage (string message)

  Creates an exception with given message.

Additional Inherited Members

6.13.1 Detailed Description

Handler for NotSupportedException.

Definition at line 33 of file NotSupportedExceptionHandler.cs.

6.13.2 Member Function Documentation

6.13.2.1 override NotSupportedException PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler.NewWithMessage (string *message*) [protected], [virtual]

Creates an exception with given message.

Parameters

message	The message used by the exception.

Returns

An exception with given message.

Implements PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< NotSupportedException >.

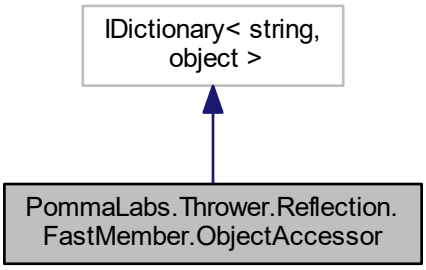
The documentation for this class was generated from the following file:

- ExceptionHandlers/NotSupportedExceptionHandler.cs

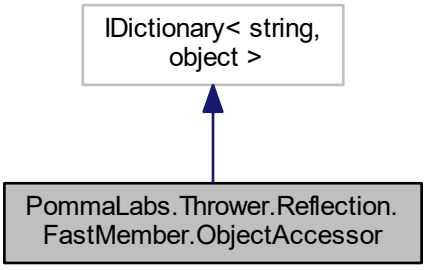
6.14 PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor Class Reference

Represents an individual object, allowing access to members by-name.

Inheritance diagram for PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor:



Collaboration diagram for PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor:



Public Member Functions

- override bool Equals (object obj)

  Use the target types definition of equality.
- override int GetHashCode ()

  Obtain the hash of the target object.
- override string ToString ()

  Use the target's definition of a string representation.
- abstract bool ContainsKey (string key)

  Determines whether the T:System.Collections.Generic.IDictionary‘2 contains an element with the specified key.
- void Add (string key, object value)

  Adds an element with the provided key and value to the T:System.Collections.Generic.IDictionary‘2.
- bool Remove (string key)

  Removes the element with the specified key from the T:System.Collections.Generic.IDictionary‘2.
- abstract bool TryGetValue (string key, out object value)

  Gets the value associated with the specified key.

- void Add (KeyValuePair< string, object > item)

  Adds an item to the T:System.Collections.Generic.ICollection‘1.
- void Clear ()

  Removes all items from the T:System.Collections.Generic.ICollection‘1.
- abstract bool Contains (KeyValuePair< string, object > item)

  Determines whether the T:System.Collections.Generic.ICollection‘1 contains a specific value.
- void CopyTo (KeyValuePair< string, object >[ ] array, int arrayIndex)

  Copies the elements of the T:System.Collections.Generic.ICollection‘1 to an T:System.Array, starting at a particular T:System.Array index.
- bool Remove (KeyValuePair< string, object > item)

  Removes the first occurrence of a specific object from the T:System.Collections.Generic.ICollection‘1.
- abstract IEnumerator< KeyValuePair< string, object > > GetEnumerator ()

  Returns an enumerator that iterates through the collection.

Static Public Member Functions

- static ObjectAccessor Create (object target)

  Wraps an individual object, allowing by-name access to that instance.
- static ObjectAccessor Create (object target, bool allowNonPublicAccessors)

  Wraps an individual object, allowing by-name access to that instance

Public Attributes

- bool IsReadOnly => true

  Gets a value indicating whether the T:System.Collections.Generic.ICollection‘1 is read-only.

Properties

- abstract object this[string name] `[get, set]`

  Get or Set the value of a named member for the underlying object.
- abstract object Target `[get]`

  The object represented by this instance.
- abstract ICollection< string > Keys `[get]`

  Gets an T:System.Collections.Generic.ICollection‘1 containing the keys of the T:System.Collections.Generic.IDictionary‘2.
- abstract ICollection< object > Values `[get]`

  Gets an T:System.Collections.Generic.ICollection‘1 containing the values in the T:System.Collections.Generic.IDictionary‘2.
- abstract int Count `[get]`

  Gets the number of elements contained in the T:System.Collections.Generic.ICollection‘1.

6.14.1 Detailed Description

Represents an individual object, allowing access to members by-name.

Definition at line 29 of file ObjectAccessor.cs.

6.14.2 Member Function Documentation

6.14.2.1 void PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Add (string *key*, object *value*)

Adds an element with the provided key and value to the T:System.Collections.Generic.IDictionary‘2.

Parameters

key	The object to use as the key of the element to add.
value	The object to use as the value of the element to add.

Exceptions

T:System.ArgumentNullException	*key* is null.
T:System.ArgumentException	An element with the same key already exists in the T:System.Collections.Generic.IDictionary‘2.
T:System.NotSupportedException	The T:System.Collections.Generic.IDictionary‘2 is read-only.

Definition at line 140 of file ObjectAccessor.cs.

6.14.2.2 void PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Add (KeyValuePair< string, object > *item*)

Adds an item to the T:System.Collections.Generic.ICollection‘1.

Parameters

item	The object to add to the T:System.Collections.Generic.ICollection‘1.

Exceptions

T:System.NotSupportedException	The T:System.Collections.Generic.ICollection‘1 is read-only.

Definition at line 186 of file ObjectAccessor.cs.

6.14.2.3 void PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Clear ( )

Removes all items from the T:System.Collections.Generic.ICollection‘1.

Exceptions

T:System.NotSupportedException	The T:System.Collections.Generic.ICollection‘1 is read-only.

Definition at line 197 of file ObjectAccessor.cs.

6.14.2.4 abstract bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Contains (KeyValuePair< string, object > *item*) [pure virtual]

Determines whether the T:System.Collections.Generic.ICollection‘1 contains a specific value.

Returns

true if *item* is found in the T:System.Collections.Generic.ICollection‘1; otherwise, false.

Parameters

item	The object to locate in the T:System.Collections.Generic.ICollection‘1.

6.14.2.5 abstract bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.ContainsKey (string *key*) [pure virtual]

Determines whether the T:System.Collections.Generic.IDictionary‘2 contains an element with the specified key.

Returns

true if the T:System.Collections.Generic.IDictionary‘2 contains an element with the key; otherwise, false.

Parameters

key	The key to locate in the T:System.Collections.Generic.IDictionary‘2.

Exceptions

T:System.ArgumentNullException	*key* is null.

6.14.2.6 void PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.CopyTo (KeyValuePair< string, object >[] *array*, int *arrayIndex*)

Copies the elements of the T:System.Collections.Generic.ICollection‘1 to an T:System.Array, starting at a particular T:System.Array index.

Parameters

array	The one-dimensional T:System.Array that is the destination of the elements copied from T:System.Collections.Generic.ICollection‘1. The T:System.Array must have zero-based indexing.
arrayIndex	The zero-based index in *array* at which copying begins.

Exceptions

T:System.ArgumentNullException	*array* is null.
T:System.ArgumentOutOfRangeException	*arrayIndex* is less than 0.
T:System.ArgumentException	The number of elements in the source T:System.Collections.Generic.ICollection‘1 is greater than the available space from *arrayIndex* to the end of the destination *array* .

Definition at line 236 of file ObjectAccessor.cs.

6.14.2.7 static ObjectAccessor PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Create (object *target*) [static]

Wraps an individual object, allowing by-name access to that instance.

Parameters

target	The target object.

6.14.2.8 static ObjectAccessor PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Create (object *target,* bool *allowNonPublicAccessors*) [static]

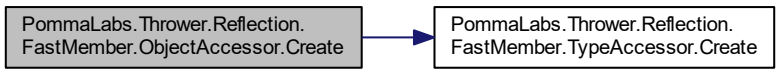
Wraps an individual object, allowing by-name access to that instance

Parameters

target	The target object.
allowNonPublicAccessors	Allow usage of non public accessors.

Definition at line 68 of file ObjectAccessor.cs.

Here is the call graph for this function:



6.14.2.9 override bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Equals (object *obj*)

Use the target types definition of equality.

Parameters

obj	The object.

6.14.2.10 abstract IEnumerator<KeyValuePair<string, object> > PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.GetEnumerator ( ) [pure virtual]

Returns an enumerator that iterates through the collection.

Returns

An enumerator that can be used to iterate through the collection.

<filterpriority>1</filterpriority>

6.14.2.11 override int PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.GetHashCode ( )

Obtain the hash of the target object.

6.14.2.12 bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Remove (string *key*)

Removes the element with the specified key from the T:System.Collections.Generic.IDictionary‘2.

Returns

true if the element is successfully removed; otherwise, false. This method also returns false if *key* was not found in the original T:System.Collections.Generic.IDictionary‘2.

Parameters

key	The key of the element to remove.

Exceptions

T:System.ArgumentNullException	*key* is null.
T:System.NotSupportedException	The T:System.Collections.Generic.IDictionary‘2 is read-only.

Definition at line 157 of file ObjectAccessor.cs.

6.14.2.13 bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Remove (KeyValuePair< string, object > *item*)

Removes the first occurrence of a specific object from the T:System.Collections.Generic.ICollection‘1.

Returns

true if *item* was successfully removed from the T:System.Collections.Generic.ICollection‘1; otherwise, false. This method also returns false if *item* is not found in the original T:System.Collections.Generic.ICollection‘1.

Parameters

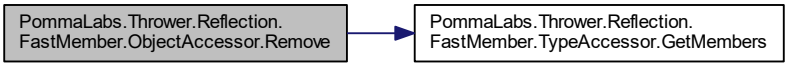
item	The object to remove from the T:System.Collections.Generic.ICollection‘1.

Exceptions

T:System.NotSupportedException	The T:System.Collections.Generic.ICollection‘1 is read-only.

Definition at line 256 of file ObjectAccessor.cs.

Here is the call graph for this function:



6.14.2.14 override string PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.ToString ( )

Use the target's definition of a string representation.

6.14.2.15 abstract bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.TryGetValue (string *key,* out object *value*) [pure virtual]

Gets the value associated with the specified key.

Returns

true if the object that implements T:System.Collections.Generic.IDictionary‘2 contains an element with the specified key; otherwise, false.

Parameters

key	The key whose value to get.
value	When this method returns, the value associated with the specified key, if the key is found; otherwise, the default value for the type of the *value* parameter. This parameter is passed uninitialized.

Exceptions

T:System.ArgumentNullException	*key* is null.

6.14.3 Member Data Documentation

6.14.3.1 bool PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.IsReadOnly => true

Gets a value indicating whether the T:System.Collections.Generic.ICollection‘1 is read-only.

Returns

true if the T:System.Collections.Generic.ICollection‘1 is read-only; otherwise, false.

Definition at line 114 of file ObjectAccessor.cs.

6.14.4 Property Documentation

6.14.4.1 abstract int PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Count [get]

Gets the number of elements contained in the T:System.Collections.Generic.ICollection‘1.

Returns

The number of elements contained in the T:System.Collections.Generic.ICollection‘1.

Definition at line 104 of file ObjectAccessor.cs.

6.14.4.2 abstract ICollection<string> PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Keys [get]

Gets an T:System.Collections.Generic.ICollection‘1 containing the keys of the T:System.Collections.Generic.I↩ Dictionary‘2.

Returns

An T:System.Collections.Generic.ICollection‘1 containing the keys of the object that implements T:System.↩ Collections.Generic.IDictionary‘2.

Definition at line 88 of file ObjectAccessor.cs.

6.14.4.3 abstract object PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Target [get]

The object represented by this instance.

Definition at line 39 of file ObjectAccessor.cs.

6.14.4.4 abstract object PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.this[string name] [get], [set]

Get or Set the value of a named member for the underlying object.

Definition at line 34 of file ObjectAccessor.cs.

6.14.4.5 abstract ICollection<object> PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor.Values [get]

Gets an T:System.Collections.Generic.ICollection‘1 containing the values in the T:System.Collections.Generic.I↩ Dictionary‘2.

Returns

An T:System.Collections.Generic.ICollection‘1 containing the values in the object that implements T:↩ System.Collections.Generic.IDictionary‘2.

Definition at line 98 of file ObjectAccessor.cs.

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

- Reflection/FastMember/ObjectAccessor.cs

6.15 PommaLabs.Thrower.ExceptionHandlers.ObjectDisposedExceptionHandler Class Reference

Handler for ObjectDisposedException.

Public Member Functions

- void If (bool disposed, string objectName, string message=null)
  Throws ObjectDisposedException if the object has been disposed.

6.15.1 Detailed Description

Handler for ObjectDisposedException.

Definition at line 33 of file ObjectDisposedExceptionHandler.cs.

6.15.2 Member Function Documentation

6.15.2.1 void PommaLabs.Thrower.ExceptionHandlers.ObjectDisposedExceptionHandler.If (bool *disposed*, string *objectName*, string *message* = `null`)

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.

Exceptions

ObjectDisposedException	If the object has been disposed.

Definition at line 42 of file ObjectDisposedExceptionHandler.cs.

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

- ExceptionHandlers/ObjectDisposedExceptionHandler.cs

6.16 PommaLabs.Thrower.Validation.ObjectValidator Class Reference

Validates an object public properties that have been decorated with the ValidateAttribute custom attribute.

Static Public Member Functions

- static string FormatValidationErrors (IEnumerable< ValidationError > validationErrors, string startMessage=null)

  Prepares a readable messages containing all validation errors.
- static bool Validate (object obj, out IList< ValidationError > validationErrors)

  Validates given object using information contained in the ValidateAttribute custom attribute.

Public Attributes

- const string RootPlaceholder = "$"

  The placeholder used to indicate the starting object.

6.16.1 Detailed Description

Validates an object public properties that have been decorated with the ValidateAttribute custom attribute.

Definition at line 37 of file ObjectValidator.cs.

6.16.2 Member Function Documentation

6.16.2.1 static string PommaLabs.Thrower.Validation.ObjectValidator.FormatValidationErrors (IEnumerable< ValidationError > *validationErrors*, string *startMessage* = `null`) [static]

Prepares a readable messages containing all validation errors.

Parameters

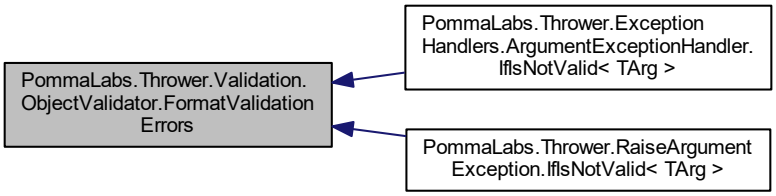
validationErrors	The validation errors.
startMessage	An optional prefix.

Returns

A readable messages containing all validation errors.

Definition at line 69 of file ObjectValidator.cs.

Here is the caller graph for this function:



6.16.2.2 static bool PommaLabs.Thrower.Validation.ObjectValidator.Validate (object *obj*, out IList< ValidationError > *validationErrors*) [static]

Validates given object using information contained in the ValidateAttribute custom attribute.

Parameters

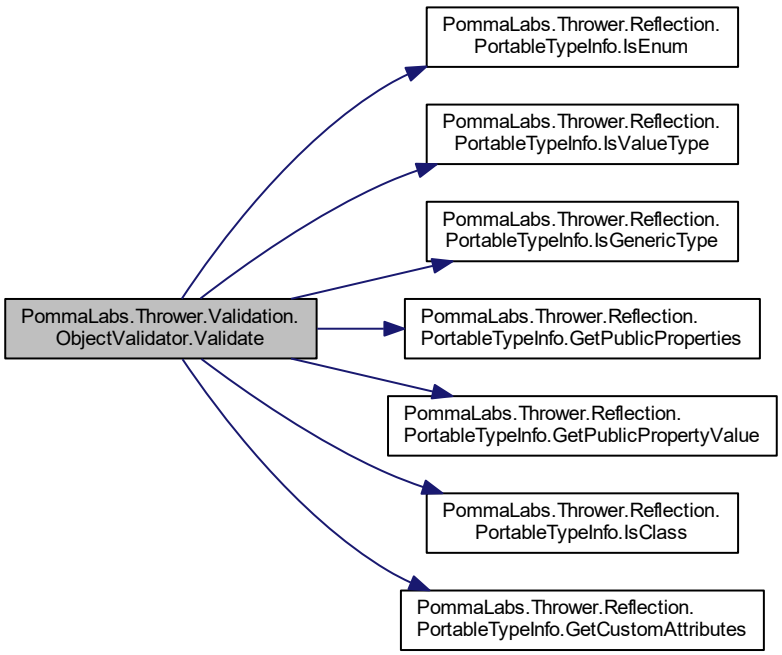
obj	The object to be validated.
validationErrors	All validation errors found.

Returns

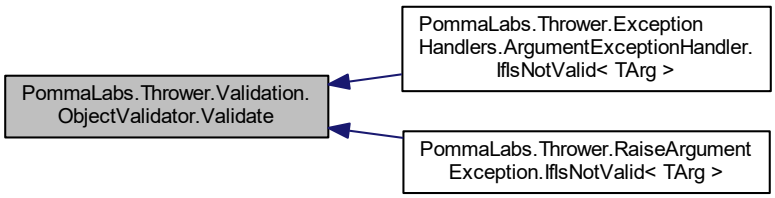
True if object is valid, false otherwise.

Definition at line 92 of file ObjectValidator.cs.

Here is the call graph for this function:



Here is the caller graph for this function:



6.16.3 Member Data Documentation

6.16.3.1 const string PommaLabs.Thrower.Validation.ObjectValidator.RootPlaceholder = "$"

The placeholder used to indicate the starting object.

Definition at line 42 of file ObjectValidator.cs.

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

- Validation/ObjectValidator.cs

6.17 PommaLabs.Thrower.Validation.PhoneNumberValidator Class Reference

A phone number validator.

Static Public Member Functions

- static bool Validate (string phoneNumber)
  Validates the specified phone number.

6.17.1 Detailed Description

A phone number validator.

A phone number validator.

Definition at line 11 of file PhoneNumberValidator.cs.

6.17.2 Member Function Documentation

6.17.2.1 static bool PommaLabs.Thrower.Validation.PhoneNumberValidator.Validate (string *phoneNumber*) [static]

Validates the specified phone number.

Parameters

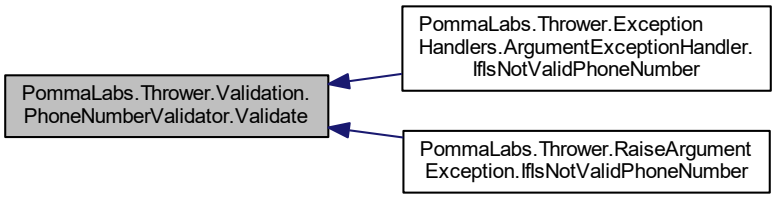
phoneNumber	A phone number.

Returns

`true` if the phone number is valid; otherwise `false`.

Definition at line 20 of file PhoneNumberValidator.cs.

Here is the caller graph for this function:



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

- Validation/PhoneNumberValidator.cs

6.18 PommaLabs.Thrower.Reflection.PortableTypeInfo Class Reference

Portable version of some useful reflection methods.

Static Public Member Functions

- static IList< Attribute > GetCustomAttributes (MemberInfo memberInfo, bool inherit)

  Gets the custom attributes for given member.
- static IList< ConstructorInfo > GetConstructors (Type type)

  Gets the constructors for given type.
- static IList< ConstructorInfo > GetConstructors< T > ()

  Gets the constructors for given type.
- static Type GetBaseType (Type type)

  Gets the base type of given type.
- static Type GetGenericTypeDefinition (Type type)

  Gets the generic type definition of given type.
- static IList< Type > GetGenericTypeArguments (Type type)

  Gets the generic type arguments of given type.
- static IList< Type > GetInterfaces (Type type)

  Gets the interfaces for given type.

- static IList< PropertyInfo > GetPublicProperties (Type type)

  Gets all the public instance properties for given type.
- static IList< PropertyInfo > GetPublicProperties< T > ()

  Gets all the instance properties for given type.
- static object GetPublicPropertyValue (object instance, PropertyInfo propertyInfo)

  Gets the value of given property on given instance.
- static object GetPublicPropertyValue (TypeAccessor typeAccessor, object instance, PropertyInfo property↩ Info)

  Gets the value of given property on given instance.
- static bool IsAbstract (Type type)

  Determines whether the specified type is abstract.
- static bool IsAbstract< T > ()

  Determines whether the specified type is abstract.
- static bool IsClass (Type type)

  Determines whether the specified type is a class.
- static bool IsClass< T > ()

  Determines whether the specified type is a class.
- static bool IsAssignableFrom (object obj, Type type)

  Determines whether an instance of the current T:System.Type can be assigned from an instance of the specified Type.
- static bool IsEnum (Type type)

  Determines whether the specified type is an enumeration.
- static bool IsEnum< T > ()

  Determines whether the specified type is an enumeration.
- static bool IsGenericType (Type type)

  Determines whether the specified type is a generic type.
- static bool IsGenericType< T > ()

  Determines whether the specified type is a generic type.
- static bool IsGenericTypeDefinition (Type type)

  Determines whether the specified type is a generic type definition.
- static bool IsGenericTypeDefinition< T > ()

  Determines whether the specified type is a generic type definition.
- static bool IsInstanceOf (object obj, Type type)

  Determines whether the specified object is an instance of the current T:System.Type.
- static bool IsInterface (Type type)

  Determines whether the specified type is an interface.
- static bool IsInterface< T > ()

  Determines whether the specified type is an interface.
- static bool IsPrimitive (Type type)

  Determines whether the specified type is primitive.
- static bool IsPrimitive< T > ()

  Determines whether the specified type is primitive.
- static bool IsValueType (Type type)

  Determines whether the specified type is a value type.
- static bool IsValueType< T > ()

  Determines whether the specified type is a value type.

6.18.1 Detailed Description

Portable version of some useful reflection methods.

Definition at line 40 of file PortableTypeInfo.cs.

6.18.2 Member Function Documentation

6.18.2.1 static Type PommaLabs.Thrower.Reflection.PortableTypeInfo.GetBaseType (Type *type*) [static]

Gets the base type of given type.

Parameters

type	The type.

Returns

The base type of given type.

Definition at line 108 of file PortableTypeInfo.cs.

6.18.2.2 static IList<ConstructorInfo> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetConstructors (Type *type*) [static]

Gets the constructors for given type.

Parameters

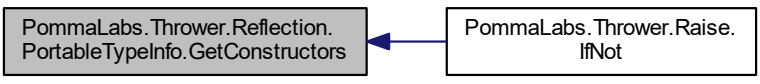
type	The type.

Returns

The constructors for given type.

Definition at line 79 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.3 static IList<ConstructorInfo> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetConstructors< T > ( ) [static]

Gets the constructors for given type.

Template Parameters

T	The type.

Returns

The constructors for given type.

6.18.2.4 static IList<Attribute> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetCustomAttributes (MemberInfo *memberInfo,* bool *inherit*) [static]

Gets the custom attributes for given member.

Parameters

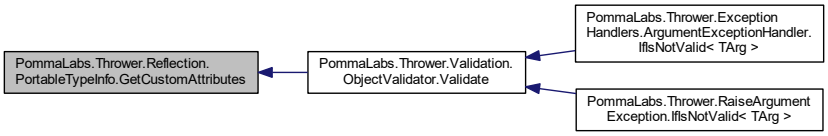
memberInfo	The member.
inherit	True to search this member's inheritance chain to find the attributes; otherwise, false.

Returns

The custom attributes for given member.

Definition at line 61 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.5 static IList<Type> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetGenericTypeArguments (Type *type*) [static]

Gets the generic type arguments of given type.

Parameters

type	The type.

Returns

The generic type arguments of given type.

Definition at line 144 of file PortableTypeInfo.cs.

6.18.2.6 static Type PommaLabs.Thrower.Reflection.PortableTypeInfo.GetGenericTypeDefinition (Type *type*) [static]

Gets the generic type definition of given type.

Parameters

type	The type.

Returns

The generic type definition of given type.

Definition at line 126 of file PortableTypeInfo.cs.

6.18.2.7 static IList<Type> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetInterfaces (Type *type*) [static]

Gets the interfaces for given type.

Parameters

type	The type.

Returns

The interfaces for given type.

Definition at line 162 of file PortableTypeInfo.cs.

6.18.2.8 static IList<PropertyInfo> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetPublicProperties (Type *type*) [static]

Gets all the public instance properties for given type.

Parameters

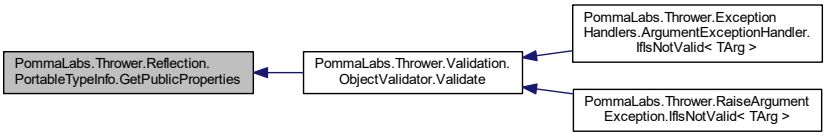
type	The type.

Returns

The public instance properties for given type.

Definition at line 180 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.9 static IList<PropertyInfo> PommaLabs.Thrower.Reflection.PortableTypeInfo.GetPublicProperties< T >( ) [static]

Gets all the instance properties for given type.

Template Parameters

T	The type.

Returns

The instance properties for given type.

6.18.2.10 static object PommaLabs.Thrower.Reflection.PortableTypeInfo.GetPublicPropertyValue (object *instance*, PropertyInfo *propertyInfo*) [static]

Gets the value of given property on given instance.

Parameters

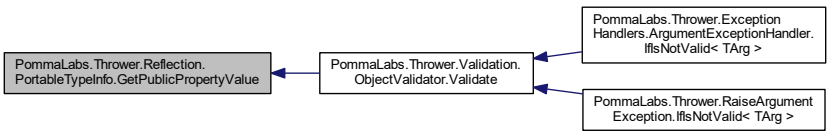
instance	The instance.
propertyInfo	The property info.

Returns

The value of given property on given instance.

Definition at line 219 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.11 static object PommaLabs.Thrower.Reflection.PortableTypeInfo.GetPublicPropertyValue (TypeAccessor *typeAccessor,* object *instance,* PropertyInfo *propertyInfo*) [static]

Gets the value of given property on given instance.

Parameters

typeAccessor	The type accessor.
instance	The instance.
propertyInfo	The property info.

Returns

The value of given property on given instance.

Definition at line 238 of file PortableTypeInfo.cs.

6.18.2.12 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsAbstract (Type *type*) [static]

Determines whether the specified type is abstract.

Parameters

type	The type.

Returns

Whether the specified type is abstract.

Definition at line 260 of file PortableTypeInfo.cs.

6.18.2.13 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsAbstract< T >( ) [static]

Determines whether the specified type is abstract.

Template Parameters

T	The type.

Returns

Whether the specified type is abstract.

6.18.2.14 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsAssignableFrom (object *obj,* Type *type*) [static]

Determines whether an instance of the current T:System.Type can be assigned from an instance of the specified Type.

Parameters

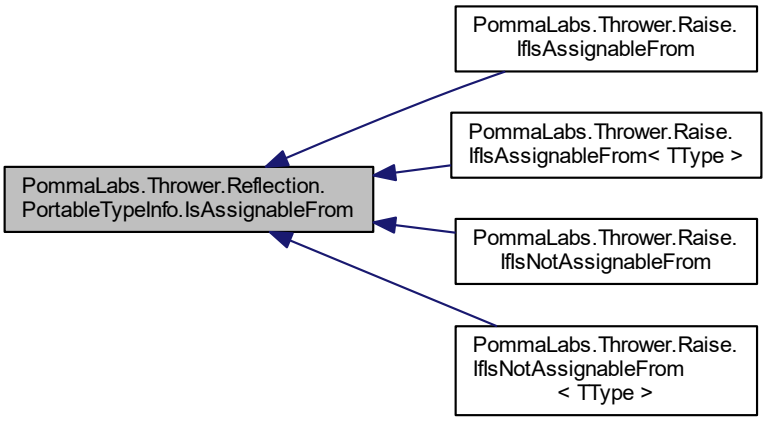
obj	The object.
type	The type.

Returns

Whether an instance of the current T:System.Type can be assigned from an instance of the specified Type.

Definition at line 329 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.15 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsClass (Type *type*) [static]

Determines whether the specified type is a class.

Parameters

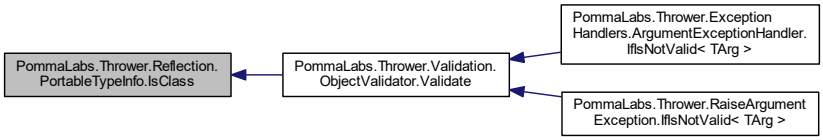
type	The type.

Returns

Whether the specified type is a class.

Definition at line 293 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.16 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsClass< T > ( ) [static]

Determines whether the specified type is a class.

Template Parameters

T	The type.

Returns

Whether the specified type is a class.

6.18.2.17 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsEnum (Type *type*) [static]

Determines whether the specified type is an enumeration.

Parameters

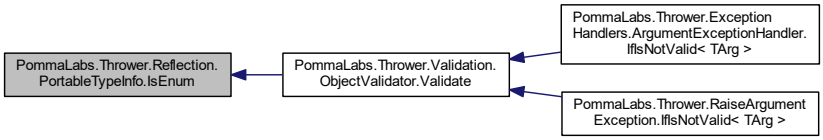
type	The type.

Returns

Whether the specified type is an enumeration.

Definition at line 354 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.18 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsEnum< T >( ) [static]

Determines whether the specified type is an enumeration.

Template Parameters

T	The type.

Returns

Whether the specified type is an enumeration.

6.18.2.19 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsGenericType (Type *type*) [static]

Determines whether the specified type is a generic type.

Parameters

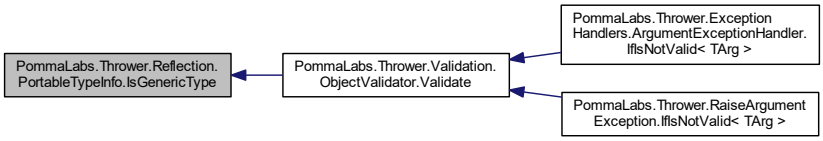
type	The type.

Returns

Whether the specified type is a generic type.

Definition at line 387 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.20 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsGenericType< T >( ) [static]

Determines whether the specified type is a generic type.

Template Parameters

T	The type.

Returns

Whether the specified type is a generic type.

6.18.2.21 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsGenericTypeDefinition (Type *type*) [static]

Determines whether the specified type is a generic type definition.

Parameters

type	The type.

Returns

Whether the specified type is a generic type definition.

Definition at line 420 of file PortableTypeInfo.cs.

6.18.2.22 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsGenericTypeDefinition< T > ( ) [static]

Determines whether the specified type is a generic type definition.

Template Parameters

T	The type.

Returns

Whether the specified type is a generic type definition.

6.18.2.23 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsInstanceOf (object *obj*, Type *type*) [static]

Determines whether the specified object is an instance of the current T:System.Type.

Parameters

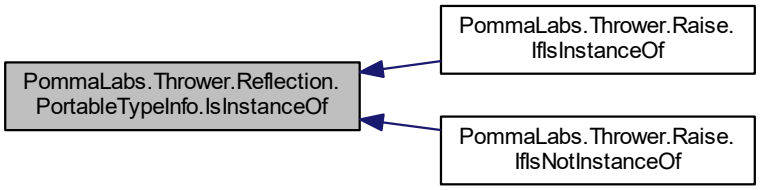
obj	The object.
type	The type.

Returns

Whether the specified object is an instance of the current T:System.Type.

Definition at line 452 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.24 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsInterface (Type *type*) [static]

Determines whether the specified type is an interface.

Parameters

type	The type.

Returns

Whether the specified type is an interface.

Definition at line 477 of file PortableTypeInfo.cs.

6.18.2.25 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsInterface< T >( ) [static]

Determines whether the specified type is an interface.

Template Parameters

T	The type.

Returns

Whether the specified type is an interface.

6.18.2.26 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsPrimitive (Type *type*) [static]

Determines whether the specified type is primitive.

Parameters

type	The type.

Returns

Whether the specified type is primitive.

Definition at line 510 of file PortableTypeInfo.cs.

6.18.2.27 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsPrimitive< T >( ) [static]

Determines whether the specified type is primitive.

Template Parameters

T	The type.

Returns

Whether the specified type is primitive.

6.18.2.28 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsValueType (Type *type*) [static]

Determines whether the specified type is a value type.

Parameters

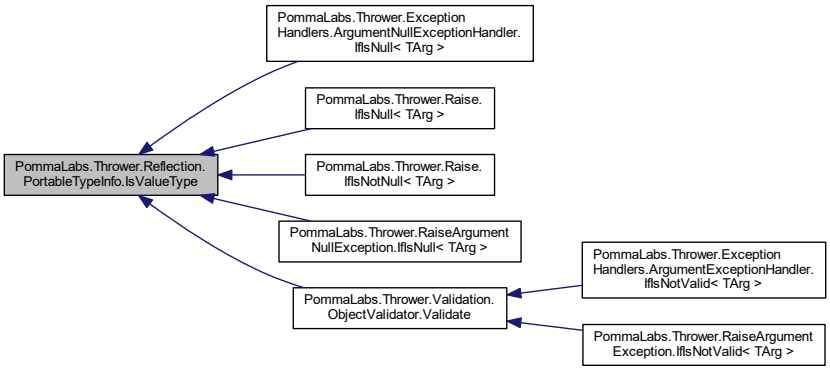
type	The type.

Returns

Whether the specified type is a value type.

Definition at line 543 of file PortableTypeInfo.cs.

Here is the caller graph for this function:



6.18.2.29 static bool PommaLabs.Thrower.Reflection.PortableTypeInfo.IsValueType< T >( ) [static]

Determines whether the specified type is a value type.

Template Parameters

T	The type.

Returns

Whether the specified type is a value type.

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

- Reflection/PortableTypeInfo.cs

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

New exception handling mechanism, which is more fluent than the old ones.

6.19.1 Detailed Description

New exception handling mechanism, which is more fluent than the old ones.

Definition at line 33 of file Raise.cs.

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

- Obsolete/Raise.cs

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

New exception handling mechanism, which is more fluent than the old ones.

6.20.1 Detailed Description

New exception handling mechanism, which is more fluent than the old ones.

Definition at line 33 of file Raise.cs.

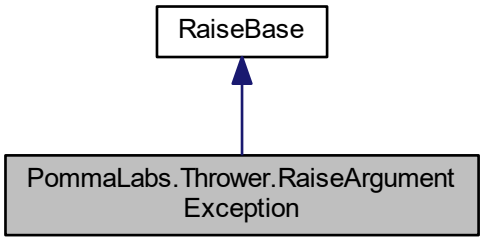
The documentation for this class was generated from the following file:

- Obsolete/Raise.cs

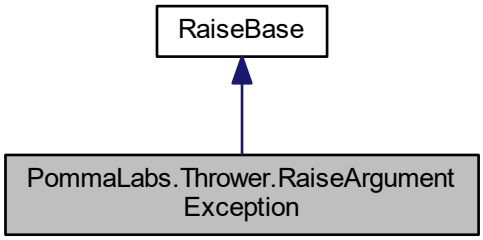
6.21 PommaLabs.Thrower.RaiseArgumentException Class Reference

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, EmailAddressValidator.Options validatorOptions)

  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, string argumentName, EmailAddressValidator.↩ Options validatorOptions, 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.
- static void IfIsNullOrEmpty< TItem > (ICollection< TItem > value)

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

  Throws ArgumentException if given collection is null or empty.

Additional Inherited Members

6.21.1 Detailed Description

Utility methods which can be used to handle bad arguments.

This class is no longer maintained.

Definition at line 37 of file RaiseArgumentException.cs.

6.21.2 Member Function Documentation

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

Throws ArgumentException if given condition is true.

Parameters

condition	The condition.

Definition at line 51 of file RaiseArgumentException.cs.

6.21.2.2 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 72 of file RaiseArgumentException.cs.

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

Throws ArgumentException if given argument is not valid.

Template Parameters

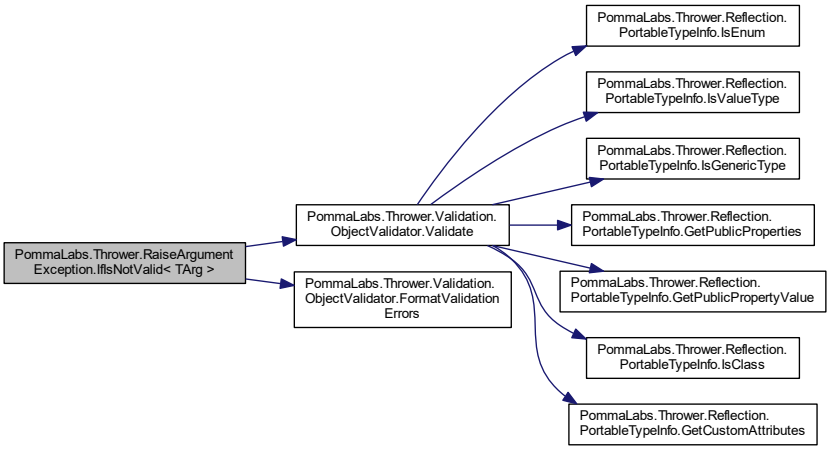
TArg	The type of the argument.

Parameters

argument	The argument.

Definition at line 134 of file RaiseArgumentException.cs.

Here is the call graph for this function:



6.21.2.4 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.

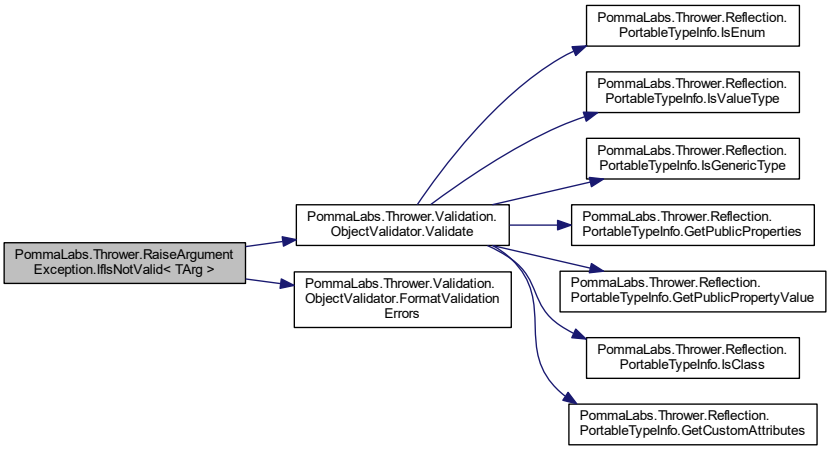
Parameters

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

message and *argumentName* are strictly required arguments.

Definition at line 157 of file RaiseArgumentException.cs.

Here is the call graph for this function:



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

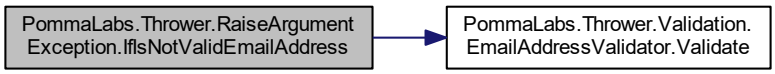
Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.

Definition at line 180 of file RaiseArgumentException.cs.

Here is the call graph for this function:



6.21.2.6 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress*, EmailAddressValidator.Options *validatorOptions*) [static]

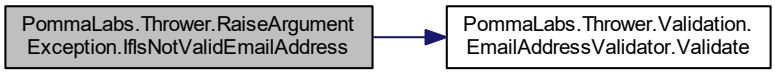
Throws ArgumentException if given string is not a valid email address.

Parameters

emailAddress	An email address.
validatorOptions	Customizations for the validation process.

Definition at line 198 of file RaiseArgumentException.cs.

Here is the call graph for this function:



6.21.2.7 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.

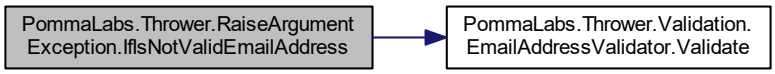
Parameters

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

message and *argumentName* are strictly required arguments.

Definition at line 220 of file RaiseArgumentException.cs.

Here is the call graph for this function:



6.21.2.8 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNotValidEmailAddress (string *emailAddress*, string *argumentName*, **EmailAddressValidator.Options** *validatorOptions*, string *message* = `null`) [static]

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

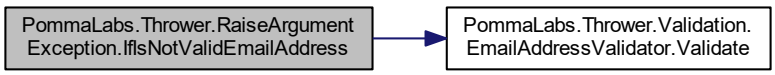
Parameters

emailAddress	An email address.
argumentName	The name of the argument.
validatorOptions	Customizations for the validation process.
message	The message.

message and *argumentName* are strictly required arguments.

Definition at line 243 of file RaiseArgumentException.cs.

Here is the call graph for this function:



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

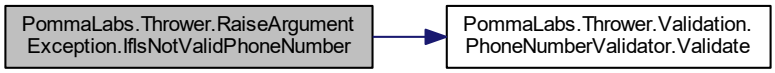
Throws ArgumentException if given string is not a valid phone number.

Parameters

phoneNumber	A phone number.

Definition at line 266 of file RaiseArgumentException.cs.

Here is the call graph for this function:



6.21.2.10 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.

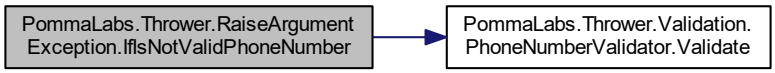
Parameters

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.

Here is the call graph for this function:



6.21.2.11 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.

6.21.2.12 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.

6.21.2.13 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrEmpty< TItem > (ICollection< TItem > *value*) [static]

Throws ArgumentException if given collection is null or empty.

Template Parameters

TItem	The type of the items contained in the collection.

Parameters

value	The collection.

Definition at line 393 of file RaiseArgumentException.cs.

6.21.2.14 static void PommaLabs.Thrower.RaiseArgumentException.IfIsNullOrEmpty< TItem > (ICollection< TItem > *value*, string *argumentName*, string *message* = `null`) [static]

Throws ArgumentException if given collection is null or empty.

Template Parameters

TItem	The type of the items contained in the collection.

Parameters

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

message and *argumentName* are strictly required arguments.

Definition at line 415 of file RaiseArgumentException.cs.

6.21.2.15 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.

6.21.2.16 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.

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

Throws ArgumentException if given condition is false.

Parameters

condition	The condition.

Definition at line 92 of file RaiseArgumentException.cs.

6.21.2.18 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 113 of file RaiseArgumentException.cs.

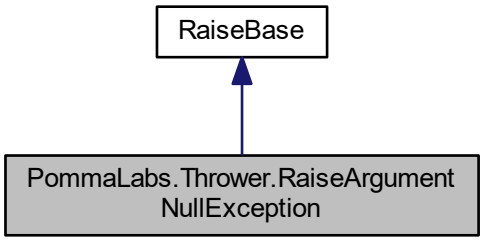
The documentation for this class was generated from the following file:

- Obsolete/RaiseArgumentException.cs

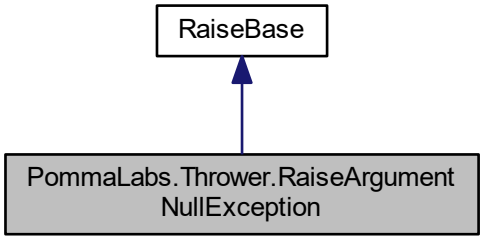
6.22 PommaLabs.Thrower.RaiseArgumentNullException Class Reference

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.
- 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

6.22.1 Detailed Description

Utility methods which can be used to handle null references.

This class is no longer maintained.

Definition at line 36 of file RaiseArgumentNullException.cs.

6.22.2 Member Function Documentation

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

Throws ArgumentNullException if given argument if null.

Template Parameters

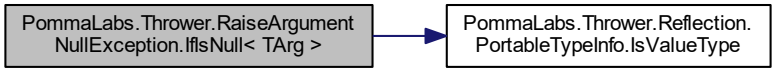
TArg	The type of the argument.

Parameters

argument	The argument.

Definition at line 49 of file RaiseArgumentNullException.cs.

Here is the call graph for this function:



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

Throws ArgumentNullException if given argument if null.

Template Parameters

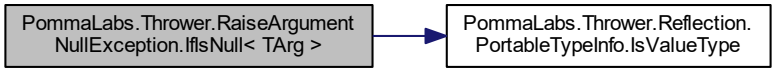
TArg	The type of the argument.

Parameters

argument	The argument.
argumentName	The name of the argument.

Definition at line 67 of file RaiseArgumentNullException.cs.

Here is the call graph for this function:



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

Throws ArgumentNullException if given argument if null.

Template Parameters

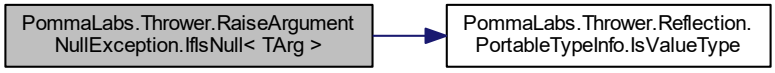
TArg	The type of the argument.

Parameters

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

Definition at line 86 of file RaiseArgumentNullException.cs.

Here is the call graph for this function:



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

Throws ArgumentNullException if given argument if null.

Template Parameters

TArg	The type of the nullable argument.

Parameters

argument	The argument.

Type Constraints

TArg* : *struct

Definition at line 107 of file RaiseArgumentNullException.cs.

6.22.2.5 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 nullable argument.

Parameters

argument	The argument.
argumentName	The name of the argument.

Type Constraints

TArg* : *struct

Definition at line 126 of file RaiseArgumentNullException.cs.

6.22.2.6 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 nullable argument.

Parameters

argument	The argument.

Parameters

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

Type Constraints

TArg **:** ***struct***

Definition at line 146 of file RaiseArgumentNullException.cs.

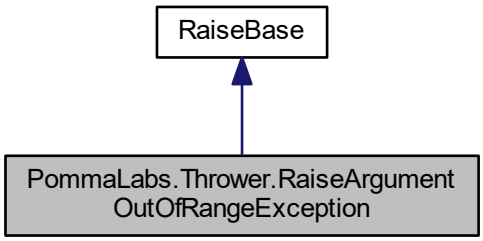
The documentation for this class was generated from the following file:

- Obsolete/RaiseArgumentNullException.cs

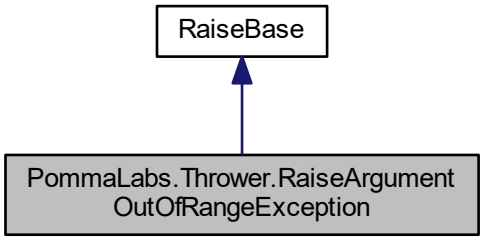
6.23 PommaLabs.Thrower.RaiseArgumentOutOfRangeException Class Reference

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

6.23.1 Detailed Description

Utility methods which can be used to handle ranges.

This class is no longer maintained.

Definition at line 35 of file RaiseArgumentOutOfRangeException.cs.

6.23.2 Member Function Documentation

6.23.2.1 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 48 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.2 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 69 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.3 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 681 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.4 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 725 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.5 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 771 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.6 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 662 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.7 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 705 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.8 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 750 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.9 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 417 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.10 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 461 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.11 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 507 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.12 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 398 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.13 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 441 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.14 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 486 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.15 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 549 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.16 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 593 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.17 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 639 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.18 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 530 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.19 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 573 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.20 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.

Parameters

message	The message that should be put into the exception.

Type Constraints

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

Definition at line 618 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.21 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 153 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.22 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 197 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.23 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 243 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.24 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 134 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.25 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 177 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.26 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 222 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.27 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 285 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.28 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 329 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.29 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 375 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.30 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 266 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.31 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 309 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.32 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 354 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.33 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 813 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.34 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 857 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.35 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 903 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.36 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 794 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.37 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 837 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.38 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 882 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.39 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 90 of file RaiseArgumentOutOfRangeException.cs.

6.23.2.40 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 111 of file RaiseArgumentOutOfRangeException.cs.

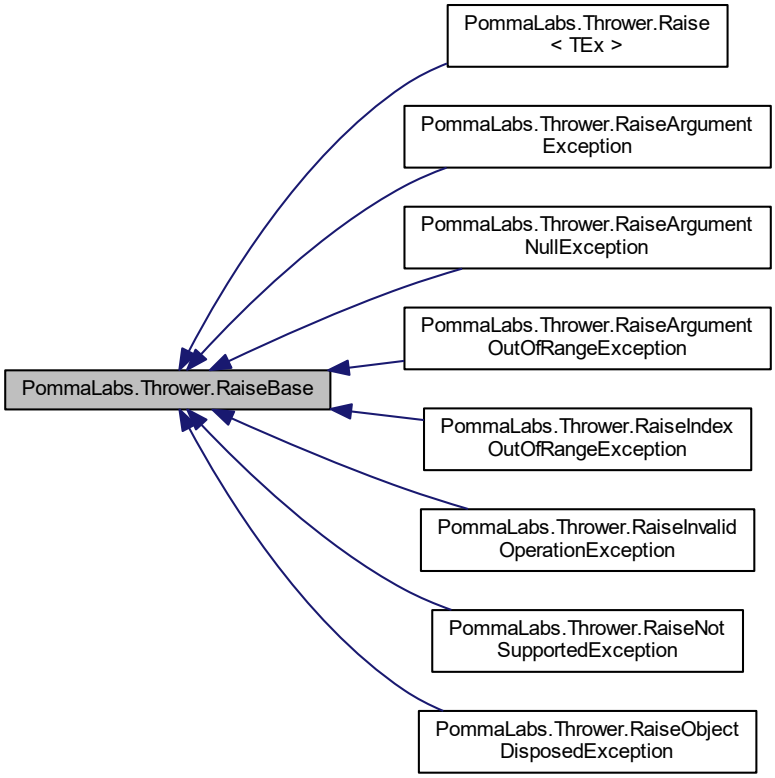
The documentation for this class was generated from the following file:

- Obsolete/RaiseArgumentOutOfRangeException.cs

6.24 PommaLabs.Thrower.RaiseBase Class Reference

Stores items shared by various Raise<TEx> instances.

Inheritance diagram for PommaLabs.Thrower.RaiseBase:



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.

6.24.1 Detailed Description

Stores items shared by various Raise<TEx> instances.

Definition at line 36 of file RaiseGeneric.cs.

6.24.2 Member Data Documentation

6.24.2.1 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 43 of file RaiseGeneric.cs.

6.24.2.2 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 59 of file RaiseGeneric.cs.

6.24.2.3 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 51 of file RaiseGeneric.cs.

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

- RaiseGeneric.cs

6.25 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 additionalInfo)

  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.

6.25.1 Detailed Description

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

This class is no longer maintained.

Definition at line 34 of file RaiseHttpException.cs.

6.25.2 Member Function Documentation

6.25.2.1 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 47 of file RaiseHttpException.cs.

6.25.2.2 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 67 of file RaiseHttpException.cs.

6.25.2.3 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 86 of file RaiseHttpException.cs.

6.25.2.4 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 106 of file RaiseHttpException.cs.

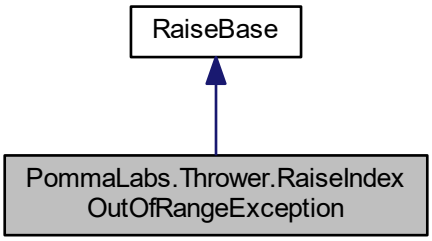
The documentation for this class was generated from the following file:

- Obsolete/RaiseHttpException.cs

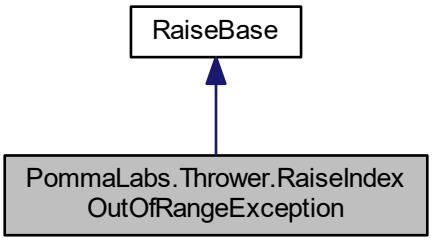
6.26 PommaLabs.Thrower.RaiseIndexOutOfRangeException Class Reference

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

6.26.1 Detailed Description

Utility methods which can be used to handle indexes.

This class is no longer maintained.

Definition at line 35 of file RaiseIndexOutOfRangeException.cs.

6.26.2 Member Function Documentation

6.26.2.1 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 413 of file RaiseIndexOutOfRangeException.cs.

6.26.2.2 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 457 of file RaiseIndexOutOfRangeException.cs.

6.26.2.3 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 394 of file RaiseIndexOutOfRangeException.cs.

6.26.2.4 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 437 of file RaiseIndexOutOfRangeException.cs.

6.26.2.5 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 241 of file RaiseIndexOutOfRangeException.cs.

6.26.2.6 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 285 of file RaiseIndexOutOfRangeException.cs.

6.26.2.7 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 222 of file RaiseIndexOutOfRangeException.cs.

6.26.2.8 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 265 of file RaiseIndexOutOfRangeException.cs.

6.26.2.9 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 327 of file RaiseIndexOutOfRangeException.cs.

6.26.2.10 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 371 of file RaiseIndexOutOfRangeException.cs.

6.26.2.11 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 308 of file RaiseIndexOutOfRangeException.cs.

6.26.2.12 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 351 of file RaiseIndexOutOfRangeException.cs.

6.26.2.13 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 69 of file RaiseIndexOutOfRangeException.cs.

6.26.2.14 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 113 of file RaiseIndexOutOfRangeException.cs.

6.26.2.15 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 50 of file RaiseIndexOutOfRangeException.cs.

6.26.2.16 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 93 of file RaiseIndexOutOfRangeException.cs.

6.26.2.17 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 155 of file RaiseIndexOutOfRangeException.cs.

6.26.2.18 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 199 of file RaiseIndexOutOfRangeException.cs.

6.26.2.19 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 136 of file RaiseIndexOutOfRangeException.cs.

6.26.2.20 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 179 of file RaiseIndexOutOfRangeException.cs.

6.26.2.21 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 499 of file RaiseIndexOutOfRangeException.cs.

6.26.2.22 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 543 of file RaiseIndexOutOfRangeException.cs.

6.26.2.23 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 480 of file RaiseIndexOutOfRangeException.cs.

6.26.2.24 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 523 of file RaiseIndexOutOfRangeException.cs.

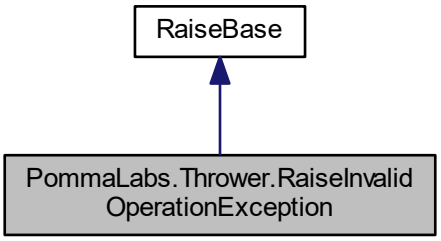
The documentation for this class was generated from the following file:

- Obsolete/RaiseIndexOutOfRangeException.cs

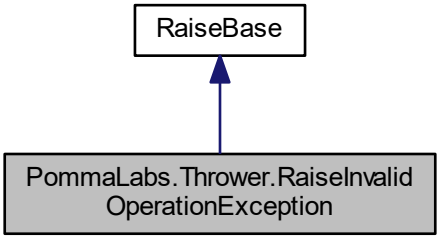
6.27 PommaLabs.Thrower.RaiseInvalidOperationException Class Reference

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

6.27.1 Detailed Description

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

This class is no longer maintained.

Definition at line 35 of file RaiseInvalidOperationException.cs.

6.27.2 Member Function Documentation

6.27.2.1 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 46 of file RaiseInvalidOperationException.cs.

6.27.2.2 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 63 of file RaiseInvalidOperationException.cs.

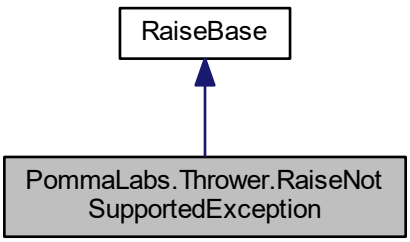
The documentation for this class was generated from the following file:

- Obsolete/RaiseInvalidOperationException.cs

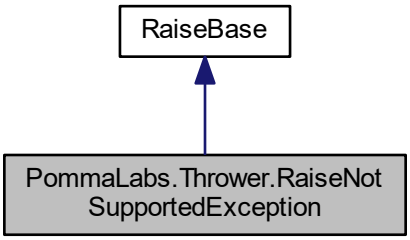
6.28 PommaLabs.Thrower.RaiseNotSupportedException Class Reference

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

6.28.1 Detailed Description

Utility methods which can be used to handle unsupported operations.

This class is no longer maintained.

Definition at line 35 of file RaiseNotSupportedException.cs.

6.28.2 Member Function Documentation

6.28.2.1 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 46 of file RaiseNotSupportedException.cs.

6.28.2.2 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 63 of file RaiseNotSupportedException.cs.

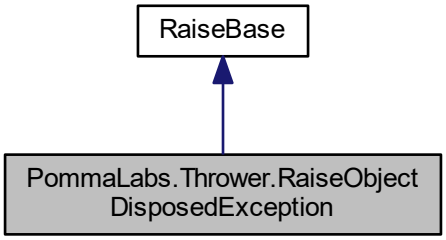
The documentation for this class was generated from the following file:

- Obsolete/RaiseNotSupportedException.cs

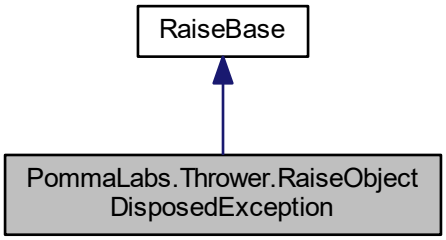
6.29 PommaLabs.Thrower.RaiseObjectDisposedException Class Reference

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

6.29.1 Detailed Description

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

This class is no longer maintained.

Definition at line 35 of file RaiseObjectDisposedException.cs.

6.29.2 Member Function Documentation

6.29.2.1 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 47 of file RaiseObjectDisposedException.cs.

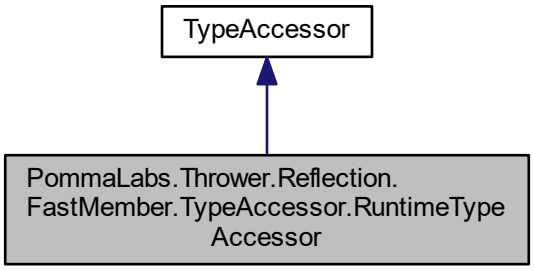
The documentation for this class was generated from the following file:

- Obsolete/RaiseObjectDisposedException.cs

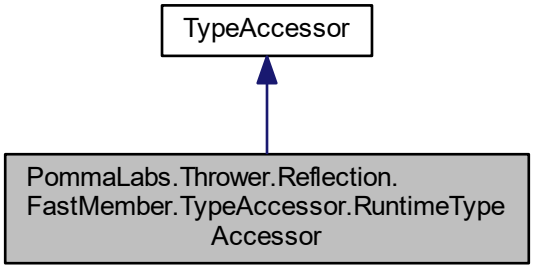
6.30 PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor Class Reference

A TypeAccessor based on a Type implementation, with available member metadata

Inheritance diagram for PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor:



Collaboration diagram for PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor:



Public Member Functions

- override MemberSet GetMembers ()

  Query the members available for this type

Public Attributes

- override bool GetMembersSupported => true

  Can this type be queried for member availability?

Properties

- abstract Type Type `[get]`

  Returns the Type represented by this accessor

Additional Inherited Members

6.30.1 Detailed Description

A TypeAccessor based on a Type implementation, with available member metadata

Definition at line 215 of file TypeAccessor.cs.

6.30.2 Member Function Documentation

6.30.2.1 override MemberSet PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor.GetMembers ( ) [virtual]

Query the members available for this type

Reimplemented from PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.

6.30.3 Member Data Documentation

6.30.3.1 override bool PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor.GetMembers↩ Supported => true

Can this type be queried for member availability?

Definition at line 225 of file TypeAccessor.cs.

6.30.4 Property Documentation

6.30.4.1 abstract Type PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor.Type [get], [protected]

Returns the Type represented by this accessor

Definition at line 220 of file TypeAccessor.cs.

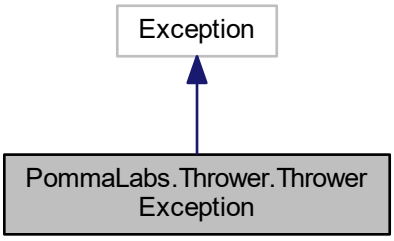
The documentation for this class was generated from the following file:

- Reflection/FastMember/TypeAccessor.cs

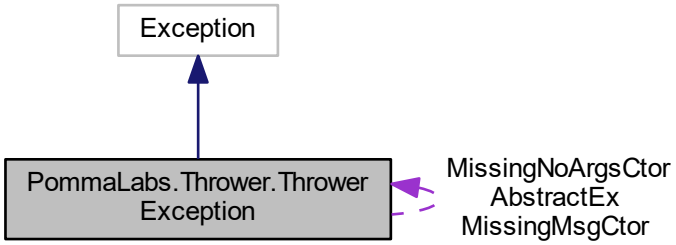
6.31 PommaLabs.Thrower.ThrowerException Class Reference

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:



6.31.1 Detailed Description

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

Definition at line 35 of file ThrowerException.cs.

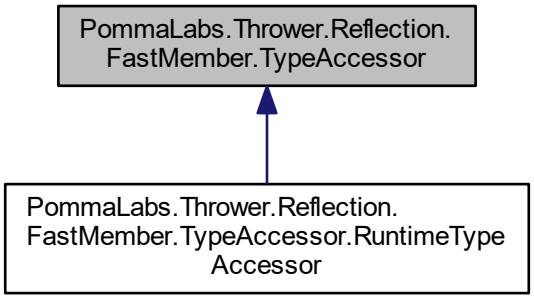
The documentation for this class was generated from the following file:

- ThrowerException.cs

6.32 PommaLabs.Thrower.Reflection.FastMember.TypeAccessor Class Reference

Provides by-name member-access to objects of a given type.

Inheritance diagram for PommaLabs.Thrower.Reflection.FastMember.TypeAccessor:



Classes

- class RuntimeTypeAccessor

  A TypeAccessor based on a Type implementation, with available member metadata

Public Member Functions

- virtual object CreateNew ()

  Create a new instance of this type.
- virtual MemberSet GetMembers ()

  Query the members available for this type.

Static Public Member Functions

- static TypeAccessor Create (Type type)

  Provides a type-specific accessor, allowing by-name access for all objects of that type.
- static TypeAccessor Create< T > ()

  Provides a type-specific accessor, allowing by-name access for all objects of that type.
- static TypeAccessor Create (Type type, bool allowNonPublicAccessors)

  Provides a type-specific accessor, allowing by-name access for all objects of that type.
- static TypeAccessor Create< T > (bool allowNonPublicAccessors)

  Provides a type-specific accessor, allowing by-name access for all objects of that type.

Public Attributes

- virtual bool CreateNewSupported => false

  Does this type support new instances via a parameterless constructor?
- virtual bool GetMembersSupported => false

  Can this type be queried for member availability?

Properties

- abstract object this[object target, string name] `[get, set]`

  Get or set the value of a named member on the target instance

6.32.1 Detailed Description

Provides by-name member-access to objects of a given type.

Definition at line 31 of file TypeAccessor.cs.

6.32.2 Member Function Documentation

6.32.2.1 static TypeAccessor PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.Create (Type *type*) `[static]`

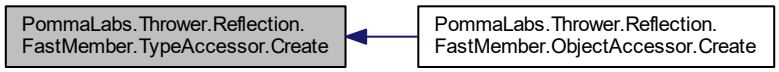
Provides a type-specific accessor, allowing by-name access for all objects of that type.

Parameters

type	The type.

The accessor is cached internally; a pre-existing accessor may be returned.

Here is the caller graph for this function:



6.32.2.2 static TypeAccessor PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.Create (Type *type,* bool *allowNonPublicAccessors*) `[static]`

Provides a type-specific accessor, allowing by-name access for all objects of that type.

Parameters

type	The type.
allowNonPublicAccessors	Allow usage of non public accessors.

The accessor is cached internally; a pre-existing accessor may be returned

Definition at line 75 of file TypeAccessor.cs.

6.32.2.3 static TypeAccessor PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.Create< T > ( ) [static]

Provides a type-specific accessor, allowing by-name access for all objects of that type.

The accessor is cached internally; a pre-existing accessor may be returned.

6.32.2.4 static TypeAccessor PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.Create< T > (bool *allowNonPublicAccessors*) [static]

Provides a type-specific accessor, allowing by-name access for all objects of that type.

Parameters

allowNonPublicAccessors	Allow usage of non public accessors.

The accessor is cached internally; a pre-existing accessor may be returned.

6.32.2.5 virtual object PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.CreateNew ( ) [virtual]

Create a new instance of this type.

Definition at line 44 of file TypeAccessor.cs.

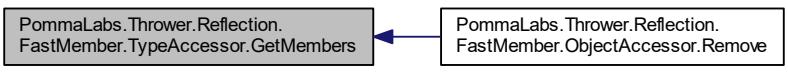
6.32.2.6 virtual MemberSet PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.GetMembers ( ) [virtual]

Query the members available for this type.

Reimplemented in PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor.

Definition at line 54 of file TypeAccessor.cs.

Here is the caller graph for this function:



6.32.3 Member Data Documentation

6.32.3.1 virtual bool PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.CreateNewSupported => false

Does this type support new instances via a parameterless constructor?

Definition at line 39 of file TypeAccessor.cs.

6.32.3.2 virtual bool PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.GetMembersSupported => false

Can this type be queried for member availability?

Definition at line 49 of file TypeAccessor.cs.

6.32.4 Property Documentation

6.32.4.1 abstract object PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.this[object target, string name] [get], [set]

Get or set the value of a named member on the target instance

Definition at line 429 of file TypeAccessor.cs.

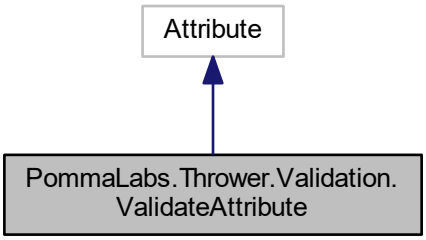
The documentation for this class was generated from the following file:

- Reflection/FastMember/TypeAccessor.cs

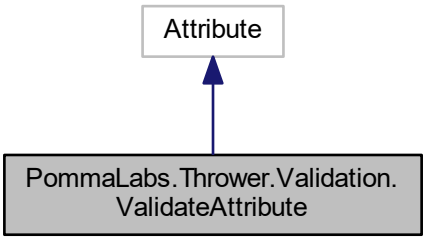
6.33 PommaLabs.Thrower.Validation.ValidateAttribute Class Reference

Indicates that the property should be validated.

Inheritance diagram for PommaLabs.Thrower.Validation.ValidateAttribute:



Collaboration diagram for PommaLabs.Thrower.Validation.ValidateAttribute:



Properties

- bool Required `[get, set]`

  Indicates that the property is required, that is, it will be checked against null.
- bool Enumerable = false `[get, set]`

  If the property is an IEnumerable, then this flag controls whether it should enumerated or not.
- bool EnumerableItemsRequired = true `[get, set]`

  If the property is an IEnumerable, then this flag controls whether its items are required or not.
- long CollectionItemsMinCount = false `[get, set]`

  If the property is an ICollection, then this flag controls the minimum value for ICollection.Count.
- long CollectionItemsMaxCount = 0L `[get, set]`

  If the property is an ICollection, then this flag controls the maximum value for ICollection.Count.

6.33.1 Detailed Description

Indicates that the property should be validated.

Definition at line 33 of file ValidateAttribute.cs.

6.33.2 Property Documentation

6.33.2.1 long PommaLabs.Thrower.Validation.ValidateAttribute.CollectionItemsMaxCount = 0L `[get]`, `[set]`

If the property is an ICollection, then this flag controls the maximum value for ICollection.Count.

Default value is long.MaxValue.

Definition at line 76 of file ValidateAttribute.cs.

6.33.2.2 long PommaLabs.Thrower.Validation.ValidateAttribute.CollectionItemsMinCount = false `[get]`, `[set]`

If the property is an ICollection, then this flag controls the minimum value for ICollection.Count.

Default value is `0L`

Definition at line 68 of file ValidateAttribute.cs.

6.33.2.3 bool PommaLabs.Thrower.Validation.ValidateAttribute.Enumerable = false `[get]`, `[set]`

If the property is an IEnumerable, then this flag controls whether it should enumerated or not.

Default value is true.

Definition at line 48 of file ValidateAttribute.cs.

6.33.2.4 bool PommaLabs.Thrower.Validation.ValidateAttribute.EnumerableItemsRequired = true [get], [set]

If the property is an IEnumerable, then this flag controls whether its items are required or not.

Default value is false.

Definition at line 56 of file ValidateAttribute.cs.

6.33.2.5 bool PommaLabs.Thrower.Validation.ValidateAttribute.Required [get], [set]

Indicates that the property is required, that is, it will be checked against null.

Default value is false.

Definition at line 40 of file ValidateAttribute.cs.

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

- Validation/ValidateAttribute.cs

6.34 PommaLabs.Thrower.Validation.ValidationError Struct Reference

Represents an error found while validating an object.

Properties

- string Path [get, set]
  The path to the wrong property.
- string Reason [get, set]
  What caused the error.

6.34.1 Detailed Description

Represents an error found while validating an object.

Definition at line 32 of file ValidationError.cs.

6.34.2 Property Documentation

6.34.2.1 string PommaLabs.Thrower.Validation.ValidationError.Path [get], [set]

The path to the wrong property.

Definition at line 37 of file ValidationError.cs.

6.34.2.2 string PommaLabs.Thrower.Validation.ValidationError.Reason [get], [set]

What caused the error.

Definition at line 42 of file ValidationError.cs.

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

- Validation/ValidationError.cs

Chapter 7

File Documentation

7.1 ExceptionHandlers/ArgumentExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.ArgumentExceptionHandler

  Handler for ArgumentException

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.2 ArgumentExceptionHandler.cs

```
00001 // File name: ArgumentExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Validation;
00025 using System;
00026 using System.Collections.Generic;
00027
00028 #pragma warning disable CC0091 // Use static method
00029
00030 namespace PommaLabs.Thrower.ExceptionHandlers
00031 {
00035     public sealed class ArgumentExceptionHandler
00036     {
```

```
00037         #region If
00038
00039         private const string DefaultIfMessage = "Argument is not valid";
00040
00046         public void If(bool condition)
00047         {
00048             if (condition)
00049             {
00050                 throw new ArgumentException(DefaultIfMessage);
00051             }
00052         }
00053
00064         public void If(bool condition, string argumentName, string message = null)
00065         {
00066             if (condition)
00067             {
00068                 throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00069             }
00070         }
00071
00072         #endregion If
00073
00074         #region IfNot
00075
00081         public void IfNot(bool condition)
00082         {
00083             if (!condition)
00084             {
00085                 throw new ArgumentException(DefaultIfMessage);
00086             }
00087         }
00088
00099         public void IfNot(bool condition, string argumentName, string message = null)
00100         {
00101             if (!condition)
00102             {
00103                 throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00104             }
00105         }
00106
00107         #endregion IfNot
00108
00109         #region IfIsNotValid
00110
00117         public void IfIsNotValid<TArg>(TArg argument)
00118         {
00119             IList<ValidationError> validationErrors;
00120             if (!ObjectValidator.Validate(argument, out validationErrors))
00121             {
00122                 throw new ArgumentException(ObjectValidator.
      FormatValidationErrors(validationErrors, null));
00123             }
00124         }
00125
00137         public void IfIsNotValid<TArg>(TArg argument, string argumentName, string message = null)
00138         {
00139             IList<ValidationError> validationErrors;
00140             if (!ObjectValidator.Validate(argument, out validationErrors))
00141             {
00142                 throw new ArgumentException(ObjectValidator.
      FormatValidationErrors(validationErrors, message), argumentName);
00143             }
00144         }
00145
00146         #endregion IfIsNotValid
00147
00148         #region IfIsNotValidEmailAddress
00149
00150         private const string DefaultIfIsNotValidEmailAddressMessage = "String \"{0}\" is not a valid email
       address";
00151
00157         public void IfIsNotValidEmailAddress(string emailAddress)
00158         {
00159             if (!EmailAddressValidator.Validate(emailAddress))
00160             {
00161                 var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
00162                 throw new ArgumentException(exceptionMsg);
00163             }
00164         }
00165
00172         public void IfIsNotValidEmailAddress(string emailAddress,
      EmailAddressValidator.Options validatorOptions)
00173         {
00174             if (!EmailAddressValidator.Validate(emailAddress, validatorOptions
      ))
00175             {
00176                 var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
```

```
00177                 throw new ArgumentException(exceptionMsg);
00178             }
00179         }
00180 
00191         public void IfIsNotValidEmailAddress(string emailAddress, string
      argumentName, string message = null)
00192         {
00193             if (!EmailAddressValidator.Validate(emailAddress))
00194             {
00195                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage,
      emailAddress);
00196                 throw new ArgumentException(exceptionMsg, argumentName);
00197             }
00198         }
00199 
00211         public void IfIsNotValidEmailAddress(string emailAddress, string
      argumentName, EmailAddressValidator.Options validatorOptions, string message = null)
00212         {
00213             if (!EmailAddressValidator.Validate(emailAddress, validatorOptions
      ))
00214             {
00215                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage,
      emailAddress);
00216                 throw new ArgumentException(exceptionMsg, argumentName);
00217             }
00218         }
00219 
00220         #endregion IfIsNotValidEmailAddress
00221 
00222         #region IfIsNotValidPhoneNumber
00223 
00224         private const string DefaultIfIsNotValidPhoneNumberMessage = "String \"{0}\" is not a valid phone
       number";
00225 
00231         public void IfIsNotValidPhoneNumber(string phoneNumber)
00232         {
00233             if (!PhoneNumberValidator.Validate(phoneNumber))
00234             {
00235                 var exceptionMsg = string.Format(DefaultIfIsNotValidPhoneNumberMessage, phoneNumber);
00236                 throw new ArgumentException(exceptionMsg);
00237             }
00238         }
00239 
00250         public void IfIsNotValidPhoneNumber(string phoneNumber, string argumentName,
       string message = null)
00251         {
00252             if (!PhoneNumberValidator.Validate(phoneNumber))
00253             {
00254                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidPhoneNumberMessage,
      phoneNumber);
00255                 throw new ArgumentException(exceptionMsg, argumentName);
00256             }
00257         }
00258 
00259         #endregion IfIsNotValidPhoneNumber
00260 
00261         #region String validation
00262 
00263         private const string StringIsNullOrEmptyMessage = "Argument cannot be a null or empty string";
00264         private const string StringIsNullOrWhiteSpaceMessage = "Argument cannot be a null, empty or blank
       string";
00265 
00271         public void IfIsNullOrEmpty(string value)
00272         {
00273             if (ReferenceEquals(value, null) || string.Empty.Equals(value))
00274             {
00275                 throw new ArgumentException(StringIsNullOrEmptyMessage);
00276             }
00277         }
00278 
00289         public void IfIsNullOrEmpty(string value, string argumentName, string message = null
      )
00290         {
00291             if (ReferenceEquals(value, null) || string.Empty.Equals(value))
00292             {
00293                 throw new ArgumentException(message ?? StringIsNullOrEmptyMessage, argumentName);
00294             }
00295         }
00296 
00302         public void IfIsNullOrWhiteSpace(string value)
00303         {
00304             if (ReferenceEquals(value, null) || string.Empty.Equals(value.Trim()))
00305             {
00306                 throw new ArgumentException(StringIsNullOrWhiteSpaceMessage);
00307             }
00308         }
00309 
```

```
00320         public void IfIsNullOrWhiteSpace(string value, string argumentName, string
      message = null)
00321         {
00322             if (ReferenceEquals(value, null) || string.Empty.Equals(value.Trim()))
00323             {
00324                 throw new ArgumentException(message ?? StringIsNullOrWhiteSpaceMessage, argumentName);
00325             }
00326         }
00327 
00328         #endregion String validation
00329 
00330         #region Collection validation
00331 
00332         internal const string CollectionIsNullOrEmptyMessage = "Argument cannot be a null or empty
       collection";
00333 
00340         public void IfIsNullOrEmpty<TItem>(ICollection<TItem> value)
00341         {
00342             if (ReferenceEquals(value, null) || value.Count == 0)
00343             {
00344                 throw new ArgumentException(CollectionIsNullOrEmptyMessage);
00345             }
00346         }
00347 
00359         public void IfIsNullOrEmpty<TItem>(ICollection<TItem> value, string argumentName, string message =
      null)
00360         {
00361             if (ReferenceEquals(value, null) || value.Count == 0)
00362             {
00363                 throw new ArgumentException(message ?? CollectionIsNullOrEmptyMessage, argumentName);
00364             }
00365         }
00366 
00367         #endregion Collection validation
00368     }
00369 }
00370 
00371 #pragma warning restore CC0091 // Use static method
```

7.3 ExceptionHandlers/ArgumentNullExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.ArgumentNullExceptionHandler

  Handler for ArgumentNullException.

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.4 ArgumentNullExceptionHandler.cs

```
00001 // File name: ArgumentNullExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
```

```
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Reflection;
00025 using System;
00026
00027 #pragma warning disable CC0091 // Use static method
00028
00029 namespace PommaLabs.Thrower.ExceptionHandlers
00030 {
00034     public sealed class ArgumentNullExceptionHandler
00035     {
00036         private const string DefaultMessage = "Argument, or a nested object, is null";
00037
00038         #region If
00039
00045         public void If(bool condition)
00046         {
00047             if (condition)
00048             {
00049                 throw new ArgumentNullException();
00050             }
00051         }
00052
00063         public void If(bool condition, string argumentName, string message = null)
00064         {
00065             if (condition)
00066             {
00067                 throw new ArgumentNullException(argumentName, message ?? DefaultMessage);
00068             }
00069         }
00070
00071         #endregion If
00072
00073         #region Classes
00074
00081         public void IfIsNull<TArg>(TArg argument)
00082         {
00083             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      argument, null))
00084             {
00085                 throw new ArgumentNullException();
00086             }
00087         }
00088
00097         public void IfIsNull<TArg>(TArg argument, string argumentName, string message = null)
00098         {
00099             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      argument, null))
00100             {
00101                 throw new ArgumentNullException(argumentName, message ?? DefaultMessage);
00102             }
00103         }
00104
00105         #endregion Classes
00106
00107         #region Nullable structs
00108
00115         public void IfIsNull<TArg>(TArg? argument)
00116             where TArg : struct
00117         {
00118             if (!argument.HasValue)
00119             {
00120                 throw new ArgumentNullException();
00121             }
00122         }
00123
00130         public void IfIsNull<TArg>(ref TArg? argument)
00131             where TArg : struct
00132         {
00133             if (!argument.HasValue)
00134             {
00135                 throw new ArgumentNullException();
00136             }
00137         }
00138
00147         public void IfIsNull<TArg>(TArg? argument, string argumentName, string message = null)
00148             where TArg : struct
00149         {
00150             if (!argument.HasValue)
00151             {
00152                 throw new ArgumentNullException(argumentName, message ?? DefaultMessage);
00153             }
00154         }
00155
```

```
00164         public void IfIsNull<TArg>(ref TArg? argument, string argumentName, string message = null)
00165             where TArg : struct
00166         {
00167             if (!argument.HasValue)
00168             {
00169                 throw new ArgumentNullException(argumentName, message ?? DefaultMessage);
00170             }
00171         }
00172 
00173         #endregion Nullable structs
00174     }
00175 }
00176 
00177 #pragma warning restore CC0091 // Use static method
```

7.5 ExceptionHandlers/ArgumentOutOfRangeExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.ArgumentOutOfRangeExceptionHandler

  Handler for System.ArgumentOutOfRangeException

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.6 ArgumentOutOfRangeExceptionHandler.cs

```
00001 // File name: ArgumentOutOfRangeExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023 
00024 using System;
00025 
00026 #pragma warning disable CC0091 // Use static method
00027 
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class ArgumentOutOfRangeExceptionHandler
00034     {
00035         #region If
00036 
00042         public void If(bool condition, string argumentName = null)
00043         {
00044             if (condition)
00045             {
00046                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
    ArgumentOutOfRangeException(argumentName);
00047             }
00048         }
00049 
```

```
00059         public void If(bool condition, string argumentName, string message)
00060         {
00061             if (condition)
00062             {
00063                 throw new ArgumentOutOfRangeException(argumentName, message);
00064             }
00065         }
00066 
00067         #endregion If
00068 
00069         #region IfNot
00070 
00076         public void IfNot(bool condition, string argumentName = null)
00077         {
00078             if (!condition)
00079             {
00080                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
      ArgumentOutOfRangeException(argumentName);
00081             }
00082         }
00083 
00093         public void IfNot(bool condition, string argumentName, string message)
00094         {
00095             if (!condition)
00096             {
00097                 throw new ArgumentOutOfRangeException(argumentName, message);
00098             }
00099         }
00100 
00101         #endregion IfNot
00102 
00103         #region Less - Without parameter name, without message
00104 
00112         public void IfIsLess<TArg>(TArg argument1, TArg argument2)
00113             where TArg : IComparable<TArg>
00114         {
00115             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00116             {
00117                 throw new ArgumentOutOfRangeException();
00118             }
00119         }
00120 
00127         public void IfIsLess(IComparable argument1, IComparable argument2)
00128         {
00129             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00130             {
00131                 throw new ArgumentOutOfRangeException();
00132             }
00133         }
00134 
00135         #endregion Less - Without parameter name, without message
00136 
00137         #region Less - With parameter name, without message
00138 
00147         public void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName)
00148             where TArg : IComparable<TArg>
00149         {
00150             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00151             {
00152                 throw new ArgumentOutOfRangeException(argumentName);
00153             }
00154         }
00155 
00163         public void IfIsLess(IComparable argument1, IComparable argument2, string argumentName)
00164         {
00165             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00166             {
00167                 throw new ArgumentOutOfRangeException(argumentName);
00168             }
00169         }
00170 
00171         #endregion Less - With parameter name, without message
00172 
00173         #region Less - With parameter name, with message
00174 
00184         public void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName, string message)
00185             where TArg : IComparable<TArg>
00186         {
00187             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00188             {
00189                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00190             }
00191         }
00192 
00201         public void IfIsLess(IComparable argument1, IComparable argument2, string argumentName,
      string message)
00202         {
```

```
00203             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00204             {
00205                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00206             }
00207         }
00208 
00209         #endregion Less - With parameter name, with message
00210 
00211         #region LessEqual - Without parameter name, without message
00212 
00220         public void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00221             where TArg : IComparable<TArg>
00222         {
00223             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00224             {
00225                 throw new ArgumentOutOfRangeException();
00226             }
00227         }
00228 
00235         public void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00236         {
00237             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00238             {
00239                 throw new ArgumentOutOfRangeException();
00240             }
00241         }
00242 
00243         #endregion LessEqual - Without parameter name, without message
00244 
00245         #region LessEqual - With parameter name, without message
00246 
00255         public void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00256             where TArg : IComparable<TArg>
00257         {
00258             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00259             {
00260                 throw new ArgumentOutOfRangeException(argumentName);
00261             }
00262         }
00263 
00271         public void IfIsLessOrEqual(IComparable argument1, IComparable argument2, string
    argumentName)
00272         {
00273             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00274             {
00275                 throw new ArgumentOutOfRangeException(argumentName);
00276             }
00277         }
00278 
00279         #endregion LessEqual - With parameter name, without message
00280 
00281         #region LessEqual - With parameter name, with message
00282 
00292         public void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
    message)
00293             where TArg : IComparable<TArg>
00294         {
00295             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00296             {
00297                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00298             }
00299         }
00300 
00309         public void IfIsLessOrEqual(IComparable argument1, IComparable argument2, string
    argumentName, string message)
00310         {
00311             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00312             {
00313                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00314             }
00315         }
00316 
00317         #endregion LessEqual - With parameter name, with message
00318 
00319         #region Greater - Without parameter name, without message
00320 
00328         public void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00329             where TArg : IComparable<TArg>
00330         {
00331             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00332             {
00333                 throw new ArgumentOutOfRangeException();
00334             }
00335         }
00336 
00343         public void IfIsGreater(IComparable argument1, IComparable argument2)
00344         {
```

```
00345             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00346             {
00347                 throw new ArgumentOutOfRangeException();
00348             }
00349         }
00350 
00351         #endregion Greater - Without parameter name, without message
00352 
00353         #region Greater - With parameter name, without message
00354 
00363         public void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName)
00364             where TArg : IComparable<TArg>
00365         {
00366             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00367             {
00368                 throw new ArgumentOutOfRangeException(argumentName);
00369             }
00370         }
00371 
00379         public void IfIsGreater(IComparable argument1, IComparable argument2, string
      argumentName)
00380         {
00381             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00382             {
00383                 throw new ArgumentOutOfRangeException(argumentName);
00384             }
00385         }
00386 
00387         #endregion Greater - With parameter name, without message
00388 
00389         #region Greater - With parameter name, with message
00390 
00400         public void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName, string message)
00401             where TArg : IComparable<TArg>
00402         {
00403             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00404             {
00405                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00406             }
00407         }
00408 
00417         public void IfIsGreater(IComparable argument1, IComparable argument2, string
      argumentName, string message)
00418         {
00419             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00420             {
00421                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00422             }
00423         }
00424 
00425         #endregion Greater - With parameter name, with message
00426 
00427         #region GreaterEqual - Without parameter name, without message
00428 
00436         public void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
00437             where TArg : IComparable<TArg>
00438         {
00439             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00440             {
00441                 throw new ArgumentOutOfRangeException();
00442             }
00443         }
00444 
00451         public void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2)
00452         {
00453             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00454             {
00455                 throw new ArgumentOutOfRangeException();
00456             }
00457         }
00458 
00459         #endregion GreaterEqual - Without parameter name, without message
00460 
00461         #region GreaterEqual - With parameter name, without message
00462 
00471         public void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00472             where TArg : IComparable<TArg>
00473         {
00474             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00475             {
00476                 throw new ArgumentOutOfRangeException(argumentName);
00477             }
00478         }
00479 
00487         public void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2,
      string argumentName)
00488         {
```

```
00489             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00490             {
00491                 throw new ArgumentOutOfRangeException(argumentName);
00492             }
00493         }
00494 
00495         #endregion GreaterEqual - With parameter name, without message
00496 
00497         #region GreaterEqual - With parameter name, with message
00498 
00508         public void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
      message)
00509             where TArg : IComparable<TArg>
00510         {
00511             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00512             {
00513                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00514             }
00515         }
00516 
00525         public void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2,
     string argumentName, string message)
00526         {
00527             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00528             {
00529                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00530             }
00531         }
00532 
00533         #endregion GreaterEqual - With parameter name, with message
00534 
00535         #region Equal - Without parameter name, without message
00536 
00544         public void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00545             where TArg : IComparable<TArg>
00546         {
00547             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00548             {
00549                 throw new ArgumentOutOfRangeException();
00550             }
00551         }
00552 
00559         public void IfIsEqual(IComparable argument1, IComparable argument2)
00560         {
00561             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00562             {
00563                 throw new ArgumentOutOfRangeException();
00564             }
00565         }
00566 
00567         #endregion Equal - Without parameter name, without message
00568 
00569         #region Equal - With parameter name, without message
00570 
00579         public void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00580             where TArg : IComparable<TArg>
00581         {
00582             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00583             {
00584                 throw new ArgumentOutOfRangeException(argumentName);
00585             }
00586         }
00587 
00595         public void IfIsEqual(IComparable argument1, IComparable argument2, string argumentName)
00596         {
00597             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00598             {
00599                 throw new ArgumentOutOfRangeException(argumentName);
00600             }
00601         }
00602 
00603         #endregion Equal - With parameter name, without message
00604 
00605         #region Equal - With parameter name, with message
00606 
00616         public void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string message)
00617             where TArg : IComparable<TArg>
00618         {
00619             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00620             {
00621                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00622             }
00623         }
00624 
00633         public void IfIsEqual(IComparable argument1, IComparable argument2, string argumentName,
     string message)
00634         {
```

```
00635             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00636             {
00637                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00638             }
00639         }
00640 
00641         #endregion Equal - With parameter name, with message
00642 
00643         #region NotEqual - Without parameter name, without message
00644 
00652         public void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00653             where TArg : IComparable<TArg>
00654         {
00655             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00656             {
00657                 throw new ArgumentOutOfRangeException();
00658             }
00659         }
00660 
00667         public void IfIsNotEqual(IComparable argument1, IComparable argument2)
00668         {
00669             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00670             {
00671                 throw new ArgumentOutOfRangeException();
00672             }
00673         }
00674 
00675         #endregion NotEqual - Without parameter name, without message
00676 
00677         #region NotEqual - With parameter name, without message
00678 
00687         public void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00688             where TArg : IComparable<TArg>
00689         {
00690             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00691             {
00692                 throw new ArgumentOutOfRangeException(argumentName);
00693             }
00694         }
00695 
00703         public void IfIsNotEqual(IComparable argument1, IComparable argument2, string
      argumentName)
00704         {
00705             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00706             {
00707                 throw new ArgumentOutOfRangeException(argumentName);
00708             }
00709         }
00710 
00711         #endregion NotEqual - With parameter name, without message
00712 
00713         #region NotEqual - With parameter name, with message
00714 
00724         public void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string message)
00725             where TArg : IComparable<TArg>
00726         {
00727             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00728             {
00729                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00730             }
00731         }
00732 
00741         public void IfIsNotEqual(IComparable argument1, IComparable argument2, string
      argumentName, string message)
00742         {
00743             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00744             {
00745                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00746             }
00747         }
00748 
00749         #endregion NotEqual - With parameter name, with message
00750     }
00751 }
00752 
00753 #pragma warning restore CC0091 // Use static method
```

7.7 ExceptionHandlers/GenericExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.GenericExceptionHandler< TException >

Generic handler used for common exceptions like NotSupportedException.

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.8 GenericExceptionHandler.cs

```
00001 // File name: GenericExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower.ExceptionHandlers
00027 {
00032     public abstract class GenericExceptionHandler<TException>
00033         where TException : Exception, new()
00034     {
00035         #region Abstract members
00036
00042         protected abstract TException NewWithMessage(string message);
00043
00044         #endregion Abstract members
00045
00046         #region Public members
00047
00056         public void If(bool condition, string message = null)
00057         {
00058             if (condition)
00059             {
00060                 throw string.IsNullOrEmpty(message) ? new TException() : NewWithMessage(message);
00061             }
00062         }
00063
00072         public void IfNot(bool condition, string message = null)
00073         {
00074             if (!condition)
00075             {
00076                 throw string.IsNullOrEmpty(message) ? new TException() : NewWithMessage(message);
00077             }
00078         }
00079
00080         #endregion Public members
00081     }
00082 }
```

7.9 ExceptionHandlers/HttpExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.HttpExceptionHandler

  Handler for HttpException

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.10 HttpExceptionHandler.cs

```
00001 // File name: HttpExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System.Net;
00025
00026 #pragma warning disable CC0091 // Use static method
00027
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class HttpExceptionHandler
00034     {
00042         public void If(bool condition, HttpStatusCode httpStatusCode, string message = null)
00043         {
00044             if (condition)
00045             {
00046                 throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
      HttpException(httpStatusCode, message);
00047             }
00048         }
00049
00058         public void If(bool condition, HttpStatusCode httpStatusCode, string message,
      HttpExceptionInfo additionalInfo)
00059         {
00060             if (condition)
00061             {
00062                 throw new HttpException(httpStatusCode, message, additionalInfo);
00063             }
00064         }
00065
00073         public void IfNot(bool condition, HttpStatusCode httpStatusCode, string message = null)
00074         {
00075             if (!condition)
00076             {
00077                 throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
      HttpException(httpStatusCode, message);
00078             }
00079         }
00080
00089         public void IfNot(bool condition, HttpStatusCode httpStatusCode, string message,
      HttpExceptionInfo additionalInfo)
00090         {
00091             if (!condition)
00092             {
00093                 throw new HttpException(httpStatusCode, message, additionalInfo);
00094             }
00095         }
00096     }
00097 }
00098
00099 #pragma warning restore CC0091 // Use static method
```

7.11 ExceptionHandlers/IndexOutOfRangeExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.IndexOutOfRangeExceptionHandler

  Handler for System.IndexOutOfRangeException

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.12 IndexOutOfRangeExceptionHandler.cs

```
00001 // File name: IndexOutOfRangeExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 #pragma warning disable CC0091 // Use static method
00027
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class IndexOutOfRangeExceptionHandler
00034     {
00035         #region Less - Without message
00036
00044         public void IfIsLess<TArg>(TArg argument1, TArg argument2)
00045             where TArg : IComparable<TArg>
00046         {
00047             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00048             {
00049                 throw new IndexOutOfRangeException();
00050             }
00051         }
00052
00059         public void IfIsLess(IComparable argument1, IComparable argument2)
00060         {
00061             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00062             {
00063                 throw new IndexOutOfRangeException();
00064             }
00065         }
00066
00067         #endregion Less - Without message
00068
00069         #region Less - With message
00070
00079         public void IfIsLess<TArg>(TArg argument1, TArg argument2, string message)
00080             where TArg : IComparable<TArg>
00081         {
00082             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00083             {
00084                 throw new IndexOutOfRangeException(message);
```

```
00085             }
00086         }
00087
00095         public void IfIsLess(IComparable argument1, IComparable argument2, string message)
00096         {
00097             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00098             {
00099                 throw new IndexOutOfRangeException(message);
00100             }
00101         }
00102
00103         #endregion Less - With message
00104
00105         #region LessEqual - Without message
00106
00114         public void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00115             where TArg : IComparable<TArg>
00116         {
00117             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00118             {
00119                 throw new IndexOutOfRangeException();
00120             }
00121         }
00122
00129         public void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00130         {
00131             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00132             {
00133                 throw new IndexOutOfRangeException();
00134             }
00135         }
00136
00137         #endregion LessEqual - Without message
00138
00139         #region LessEqual - With message
00140
00149         public void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00150             where TArg : IComparable<TArg>
00151         {
00152             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00153             {
00154                 throw new IndexOutOfRangeException(message);
00155             }
00156         }
00157
00165         public void IfIsLessOrEqual(IComparable argument1, IComparable argument2, string
      message)
00166         {
00167             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00168             {
00169                 throw new IndexOutOfRangeException(message);
00170             }
00171         }
00172
00173         #endregion LessEqual - With message
00174
00175         #region Greater - Without message
00176
00184         public void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00185             where TArg : IComparable<TArg>
00186         {
00187             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00188             {
00189                 throw new IndexOutOfRangeException();
00190             }
00191         }
00192
00199         public void IfIsGreater(IComparable argument1, IComparable argument2)
00200         {
00201             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00202             {
00203                 throw new IndexOutOfRangeException();
00204             }
00205         }
00206
00207         #endregion Greater - Without message
00208
00209         #region Greater - With message
00210
00219         public void IfIsGreater<TArg>(TArg argument1, TArg argument2, string message)
00220             where TArg : IComparable<TArg>
00221         {
00222             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00223             {
00224                 throw new IndexOutOfRangeException(message);
00225             }
00226         }
```

```
00227
00235         public void IfIsGreater(IComparable argument1, IComparable argument2, string message)
00236         {
00237             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00238             {
00239                 throw new IndexOutOfRangeException(message);
00240             }
00241         }
00242
00243         #endregion Greater - With message
00244
00245         #region GreaterEqual - Without message
00246
00254         public void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
00255             where TArg : IComparable<TArg>
00256         {
00257             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00258             {
00259                 throw new IndexOutOfRangeException();
00260             }
00261         }
00262
00269         public void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2)
00270         {
00271             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00272             {
00273                 throw new IndexOutOfRangeException();
00274             }
00275         }
00276
00277         #endregion GreaterEqual - Without message
00278
00279         #region GreaterEqual - With message
00280
00289         public void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00290             where TArg : IComparable<TArg>
00291         {
00292             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00293             {
00294                 throw new IndexOutOfRangeException(message);
00295             }
00296         }
00297
00305         public void IfIsGreaterOrEqual(IComparable argument1, IComparable argument2,
      string message)
00306         {
00307             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00308             {
00309                 throw new IndexOutOfRangeException(message);
00310             }
00311         }
00312
00313         #endregion GreaterEqual - With message
00314
00315         #region Equal - Without message
00316
00324         public void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00325             where TArg : IComparable<TArg>
00326         {
00327             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00328             {
00329                 throw new IndexOutOfRangeException();
00330             }
00331         }
00332
00339         public void IfIsEqual(IComparable argument1, IComparable argument2)
00340         {
00341             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00342             {
00343                 throw new IndexOutOfRangeException();
00344             }
00345         }
00346
00347         #endregion Equal - Without message
00348
00349         #region Equal - With message
00350
00359         public void IfIsEqual<TArg>(TArg argument1, TArg argument2, string message)
00360             where TArg : IComparable<TArg>
00361         {
00362             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00363             {
00364                 throw new IndexOutOfRangeException(message);
00365             }
00366         }
00367
00375         public void IfIsEqual(IComparable argument1, IComparable argument2, string message)
```

```
00376         {
00377             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00378             {
00379                 throw new IndexOutOfRangeException(message);
00380             }
00381         }
00382 
00383         #endregion Equal - With message
00384 
00385         #region NotEqual - Without message
00386 
00394         public void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00395             where TArg : IComparable<TArg>
00396         {
00397             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00398             {
00399                 throw new IndexOutOfRangeException();
00400             }
00401         }
00402 
00409         public void IfIsNotEqual(IComparable argument1, IComparable argument2)
00410         {
00411             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00412             {
00413                 throw new IndexOutOfRangeException();
00414             }
00415         }
00416 
00417         #endregion NotEqual - Without message
00418 
00419         #region NotEqual - With message
00420 
00429         public void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string message)
00430             where TArg : IComparable<TArg>
00431         {
00432             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00433             {
00434                 throw new IndexOutOfRangeException(message);
00435             }
00436         }
00437 
00445         public void IfIsNotEqual(IComparable argument1, IComparable argument2, string message)
00446         {
00447             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00448             {
00449                 throw new IndexOutOfRangeException(message);
00450             }
00451         }
00452 
00453         #endregion NotEqual - With message
00454     }
00455 }
00456 
00457 #pragma warning restore CC0091 // Use static method
```

7.13 ExceptionHandlers/InvalidOperationExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.InvalidOperationExceptionHandler

  Handler for InvalidOperationException.

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.14 InvalidOperationExceptionHandler.cs

```
00001 // File name: InvalidOperationException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 #pragma warning disable CC0091 // Use static method
00027
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class InvalidOperationExceptionHandler :
      GenericExceptionHandler<InvalidOperationException>
00034     {
00040         protected override InvalidOperationException NewWithMessage(string message) => new
      InvalidOperationException(message);
00041     }
00042 }
00043
00044 #pragma warning restore CC0091 // Use static method
```

7.15 ExceptionHandlers/NotSupportedExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.NotSupportedExceptionHandler

  Handler for NotSupportedException.

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.16 NotSupportedExceptionHandler.cs

```
00001 // File name: NotSupportedExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
```

```
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 #pragma warning disable CC0091 // Use static method
00027
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class NotSupportedExceptionHandler :
      GenericExceptionHandler<NotSupportedException>
00034     {
00040         protected override NotSupportedException NewWithMessage(string message) => new
      NotSupportedException(message);
00041     }
00042 }
00043
00044 #pragma warning restore CC0091 // Use static method
```

7.17 ExceptionHandlers/ObjectDisposedExceptionHandler.cs File Reference

Classes

- class PommaLabs.Thrower.ExceptionHandlers.ObjectDisposedExceptionHandler

  Handler for ObjectDisposedException.

Namespaces

- namespace PommaLabs.Thrower.ExceptionHandlers

7.18 ObjectDisposedExceptionHandler.cs

```
00001 // File name: ObjectDisposedExceptionHandler.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 #pragma warning disable CC0091 // Use static method
00027
00028 namespace PommaLabs.Thrower.ExceptionHandlers
00029 {
00033     public sealed class ObjectDisposedExceptionHandler
00034     {
00042         public void If(bool disposed, string objectName, string message = null)
```

```
00043         {
00044             if (disposed)
00045             {
00046                 throw string.IsNullOrEmpty(message) ? new ObjectDisposedException(objectName) : new
    ObjectDisposedException(objectName, message);
00047             }
00048         }
00049     }
00050 }
00051
00052 #pragma warning restore CC0091 // Use static method
```

7.19 HttpException.cs File Reference

Classes

- 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

7.20 HttpException.cs

```
00001 // File name: HttpException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Validation;
00025 using System;
00026 using System.Collections;
00027 using System.Collections.Generic;
00028 using System.Diagnostics.CodeAnalysis;
00029 using System.Net;
00030 using System.Runtime.Serialization;
00031
00032 namespace PommaLabs.Thrower
00033 {
00037     [Serializable]
00038     public struct HttpExceptionInfo
00039     {
00045         public HttpExceptionInfo(object errorCode = null, string userMessage = null)
00046         {
00047             ErrorCode = errorCode ?? HttpException.DefaultErrorCode;
00048             UserMessage = userMessage ?? HttpException.
```

```
      DefaultUserMessage;
00049         }
00050
00054         [Validate(Required = false)]
00055         public object ErrorCode { get; set; }
00056
00060         [Validate(Required = false)]
00061         public string UserMessage { get; set; }
00062     }
00063
00068     [Serializable]
00069     [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
00070     public sealed class HttpException : Exception
00071     {
00076         public HttpException(HttpStatusCode httpStatusCode)
00077             : this(httpStatusCode, new HttpExceptionInfo())
00078         {
00079         }
00080
00086         public HttpException(HttpStatusCode httpStatusCode,
      HttpExceptionInfo additionalInfo)
00087         {
00088             HttpStatusCode = httpStatusCode;
00089             ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00090             UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00091
00092             CustomizeException();
00093         }
00094
00100         public HttpException(HttpStatusCode httpStatusCode, string message)
00101             : this(httpStatusCode, message, new HttpExceptionInfo())
00102         {
00103         }
00104
00111         public HttpException(HttpStatusCode httpStatusCode, string message,
      HttpExceptionInfo additionalInfo)
00112             : base(message)
00113         {
00114             HttpStatusCode = httpStatusCode;
00115             ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00116             UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00117
00118             CustomizeException();
00119         }
00120
00127         public HttpException(HttpStatusCode httpStatusCode, string message, Exception
      innerException)
00128             : this(httpStatusCode, message, innerException, new
      HttpExceptionInfo())
00129         {
00130         }
00131
00139         public HttpException(HttpStatusCode httpStatusCode, string message, Exception
      innerException, HttpExceptionInfo additionalInfo)
00140             : base(message, innerException)
00141         {
00142             HttpStatusCode = httpStatusCode;
00143             ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00144             UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00145
00146             CustomizeException();
00147         }
00148
00152         public HttpStatusCode HttpStatusCode { get; }
00153
00157         public object ErrorCode { get; }
00158
00162         public static object DefaultErrorCode { get; set; } = "unspecified";
00163
00167         public string UserMessage { get; }
00168
00172         public static string DefaultUserMessage { get; set; } = "unspecified";
00173
00174         private void CustomizeException()
00175         {
00176             HResult = (int) HttpStatusCode;
00177
00178             Data.Add(nameof(HttpStatusCode), HttpStatusCode.ToString());
00179             Data.Add(nameof(ErrorCode), ErrorCode?.ToString());
00180             Data.Add(nameof(UserMessage), UserMessage);
00181         }
00182     }
00183 }
```

7.21 Obsolete/Raise.cs File Reference

Classes

- class PommaLabs.Thrower.Raise< TEx >

  New exception handling mechanism, which is more fluent than the old ones.

Namespaces

- namespace PommaLabs.Thrower

7.22 Raise.cs

```
00001 // File name: Raise.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Reflection;
00025 using System;
00026 using System.Diagnostics.CodeAnalysis;
00027 using System.Linq;
00028
00029 namespace PommaLabs.Thrower
00030 {
00031 #pragma warning disable RECS0096 // Type parameter is never used
00032
00033     public partial class Raise<TEx>
00034 #pragma warning restore RECS0096 // Type parameter is never used
00035     {
00051 #if (NET45 || NET46 || PORTABLE)
00052
00053         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00054 #endif
00055
00056         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00057         public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00058         {
00059             if (Equals(arg1, arg2))
00060             {
00061                 DoThrow();
00062             }
00063         }
00064
00087 #if (NET45 || NET46 || PORTABLE)
00088
00089         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00090 #endif
00091
00092         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00093         public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00094         {
00095             if (Equals(arg1, arg2))
```

```
00096             {
00097                 DoThrow(message);
00098             }
00099         }
00100 
00116 #if (NET45 || NET46 || PORTABLE)
00117 
00118         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00119 #endif
00120 
00121         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00122         public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00123         {
00124             if (!Equals(arg1, arg2))
00125             {
00126                 DoThrow();
00127             }
00128         }
00129 
00152 #if (NET45 || NET46 || PORTABLE)
00153 
00154         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00155 #endif
00156 
00157         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00158         public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00159         {
00160             if (!Equals(arg1, arg2))
00161             {
00162                 DoThrow(message);
00163             }
00164         }
00165 
00181 #if (NET45 || NET46 || PORTABLE)
00182 
00183         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00184 #endif
00185 
00186         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00187         public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00188         {
00189             if (ReferenceEquals(arg1, arg2))
00190             {
00191                 DoThrow();
00192             }
00193         }
00194 
00217 #if (NET45 || NET46 || PORTABLE)
00218 
00219         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00220 #endif
00221 
00222         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00223         public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00224         {
00225             if (ReferenceEquals(arg1, arg2))
00226             {
00227                 DoThrow(message);
00228             }
00229         }
00230 
00246 #if (NET45 || NET46 || PORTABLE)
00247 
00248         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00249 #endif
00250 
00251         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00252         public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00253         {
00254             if (!ReferenceEquals(arg1, arg2))
00255             {
00256                 DoThrow();
00257             }
00258         }
00259 
00282 #if (NET45 || NET46 || PORTABLE)
00283 
00284         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00285 #endif
00286 
00287         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
```

```
00288         public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00289         {
00290             if (!ReferenceEquals(arg1, arg2))
00291             {
00292                 DoThrow(message);
00293             }
00294         }
00295
00311 #if (NET45 || NET46 || PORTABLE)
00312
00313         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00314 #endif
00315
00316         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00317         public static void IfIsAssignableFrom(object instance, Type type)
00318         {
00319             if (PortableTypeInfo.IsAssignableFrom(instance, type))
00320             {
00321                 DoThrow();
00322             }
00323         }
00324
00349 #if (NET45 || NET46 || PORTABLE)
00350
00351         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00352 #endif
00353
00354         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00355         public static void IfIsAssignableFrom(object instance, Type type, string message)
00356         {
00357             if (ReferenceEquals(instance, null) || PortableTypeInfo.
      IsAssignableFrom(instance, type))
00358             {
00359                 DoThrow(message);
00360             }
00361         }
00362
00378 #if (NET45 || NET46 || PORTABLE)
00379
00380         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00381 #endif
00382
00383         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00384         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00385         public static void IfIsAssignableFrom<TType>(object instance)
00386         {
00387             if (ReferenceEquals(instance, null) || PortableTypeInfo.
      IsAssignableFrom(instance, typeof(TType)))
00388             {
00389                 DoThrow();
00390             }
00391         }
00392
00417 #if (NET45 || NET46 || PORTABLE)
00418
00419         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00420 #endif
00421
00422         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00423         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00424         public static void IfIsAssignableFrom<TType>(object instance, string message)
00425         {
00426             if (ReferenceEquals(instance, null) || PortableTypeInfo.
      IsAssignableFrom(instance, typeof(TType)))
00427             {
00428                 DoThrow(message);
00429             }
00430         }
00431
00447 #if (NET45 || NET46 || PORTABLE)
00448
00449         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00450 #endif
00451
00452         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00453         public static void IfIsNotAssignableFrom(object instance, Type type)
00454         {
00455             if (ReferenceEquals(instance, null) || !PortableTypeInfo.
      IsAssignableFrom(instance, type))
00456             {
00457                 DoThrow();
00458             }
```

```
00459         }
00460
00485 #if (NET45 || NET46 || PORTABLE)
00486
00487         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00488 #endif
00489
00490         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00491         public static void IfIsNotAssignableFrom(object instance, Type type, string
      message)
00492         {
00493             if (ReferenceEquals(instance, null) || !PortableTypeInfo.
      IsAssignableFrom(instance, type))
00494             {
00495                 DoThrow(message);
00496             }
00497         }
00498
00516 #if (NET45 || NET46 || PORTABLE)
00517
00518         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00519 #endif
00520
00521         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00522         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00523         public static void IfIsNotAssignableFrom<TType>(object instance)
00524         {
00525             if (!PortableTypeInfo.IsAssignableFrom(instance, typeof(TType))
      )
00526             {
00527                 DoThrow();
00528             }
00529         }
00530
00557 #if (NET45 || NET46 || PORTABLE)
00558
00559         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00560 #endif
00561
00562         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00563         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00564         public static void IfIsNotAssignableFrom<TType>(object instance, string message)
00565         {
00566             if (ReferenceEquals(instance, null) || !PortableTypeInfo.
      IsAssignableFrom(instance, typeof(TType)))
00567             {
00568                 DoThrow(message);
00569             }
00570         }
00571
00587 #if (NET45 || NET46 || PORTABLE)
00588
00589         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00590 #endif
00591
00592         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00593         public static void IfIsContainedIn(object argument,
      System.Collections.IList collection)
00594         {
00595             if (ReferenceEquals(collection, null) || collection.Contains(argument))
00596             {
00597                 DoThrow();
00598             }
00599         }
00600
00623 #if (NET45 || NET46 || PORTABLE)
00624
00625         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00626 #endif
00627
00628         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00629         public static void IfIsContainedIn(object argument,
      System.Collections.IList collection, string message)
00630         {
00631             if (ReferenceEquals(collection, null) || collection.Contains(argument))
00632             {
00633                 DoThrow(message);
00634             }
00635         }
00636
00652 #if (NET45 || NET46 || PORTABLE)
00653
```

```
00654         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00655 #endif
00656
00657         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00658         public static void IfIsNotContainedIn(object argument,
      System.Collections.IList collection)
00659         {
00660             if (ReferenceEquals(collection, null) || !collection.Contains(argument))
00661             {
00662                 DoThrow();
00663             }
00664         }
00665
00688 #if (NET45 || NET46 || PORTABLE)
00689
00690         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00691 #endif
00692
00693         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00694         public static void IfIsNotContainedIn(object argument,
      System.Collections.IList collection, string message)
00695         {
00696             if (ReferenceEquals(collection, null) || !collection.Contains(argument))
00697             {
00698                 DoThrow(message);
00699             }
00700         }
00701
00717 #if (NET45 || NET46 || PORTABLE)
00718
00719         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00720 #endif
00721
00722         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00723         public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
      TArg> collection)
00724         {
00725             if (ReferenceEquals(collection, null) || collection.Contains(arg))
00726             {
00727                 DoThrow();
00728             }
00729         }
00730
00753 #if (NET45 || NET46 || PORTABLE)
00754
00755         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00756 #endif
00757
00758         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00759         public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
      TArg> collection, string message)
00760         {
00761             if (ReferenceEquals(collection, null) || collection.Contains(arg))
00762             {
00763                 DoThrow(message);
00764             }
00765         }
00766
00782 #if (NET45 || NET46 || PORTABLE)
00783
00784         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00785 #endif
00786
00787         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00788         public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
      TArg> collection)
00789         {
00790             if (ReferenceEquals(collection, null) || !collection.Contains(arg))
00791             {
00792                 DoThrow();
00793             }
00794         }
00795
00818 #if (NET45 || NET46 || PORTABLE)
00819
00820         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00821 #endif
00822
00823         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00824         public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
      TArg> collection, string message)
```

```
00825         {
00826             if (ReferenceEquals(collection, null) || !collection.Contains(arg))
00827             {
00828                 DoThrow(message);
00829             }
00830         }
00831 
00847 #if (NET45 || NET46 || PORTABLE)
00848 
00849         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00850 #endif
00851 
00852         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00853         public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary)
00854         {
00855             if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
00856             {
00857                 DoThrow();
00858             }
00859         }
00860 
00884 #if (NET45 || NET46 || PORTABLE)
00885 
00886         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00887 #endif
00888 
00889         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00890         public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary,
       string message)
00891         {
00892             if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
00893             {
00894                 DoThrow(message);
00895             }
00896         }
00897 
00913 #if (NET45 || NET46 || PORTABLE)
00914 
00915         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00916 #endif
00917 
00918         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00919         public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary
      dictionary)
00920         {
00921             if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
00922             {
00923                 DoThrow();
00924             }
00925         }
00926 
00950 #if (NET45 || NET46 || PORTABLE)
00951 
00952         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00953 #endif
00954 
00955         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00956         public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary
      dictionary, string message)
00957         {
00958             if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
00959             {
00960                 DoThrow(message);
00961             }
00962         }
00963 
00980 #if (NET45 || NET46 || PORTABLE)
00981 
00982         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00983 #endif
00984 
00985         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00986         public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
      System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
00987         {
00988             if (ReferenceEquals(dictionary, null) || dictionary.Contains(new
      System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
00989             {
00990                 DoThrow();
00991             }
00992         }
00993 
```

```
01019 #if (NET45 || NET46 || PORTABLE)
01020
01021         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01022 #endif
01023
01024         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01025         public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
      System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,
01026             string message)
01027         {
01028             if (ReferenceEquals(dictionary, null) || dictionary.Contains(new
      System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01029             {
01030                 DoThrow(message);
01031             }
01032         }
01033
01050 #if (NET45 || NET46 || PORTABLE)
01051
01052         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01053 #endif
01054
01055         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01056         public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
      System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
01057         {
01058             if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
      System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01059             {
01060                 DoThrow();
01061             }
01062         }
01063
01089 #if (NET45 || NET46 || PORTABLE)
01090
01091         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01092 #endif
01093
01094         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01095         public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
      System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,
01096             string message)
01097         {
01098             if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
      System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01099             {
01100                 DoThrow(message);
01101             }
01102         }
01103
01118 #if (NET45 || NET46 || PORTABLE)
01119
01120         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01121 #endif
01122
01123         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01124         public static void IfIsEmpty(string valueToCheck)
01125         {
01126             if (IsNullOrWhiteSpace(valueToCheck))
01127             {
01128                 DoThrow();
01129             }
01130         }
01131
01154 #if (NET45 || NET46 || PORTABLE)
01155
01156         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01157 #endif
01158
01159         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01160         public static void IfIsEmpty(string valueToCheck, string message)
01161         {
01162             if (IsNullOrWhiteSpace(valueToCheck))
01163             {
01164                 DoThrow(message);
01165             }
01166         }
01167
01182 #if (NET45 || NET46 || PORTABLE)
01183
01184         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
```

```
01185 #endif
01186
01187         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01188         public static void IfIsNotEmpty(string valueToCheck)
01189         {
01190             if (!IsNullOrWhiteSpace(valueToCheck))
01191             {
01192                 DoThrow();
01193             }
01194         }
01195
01218 #if (NET45 || NET46 || PORTABLE)
01219
01220         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01221 #endif
01222
01223         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01224         public static void IfIsNotEmpty(string valueToCheck, string message)
01225         {
01226             if (!IsNullOrWhiteSpace(valueToCheck))
01227             {
01228                 DoThrow(message);
01229             }
01230         }
01231
01246 #if (NET45 || NET46 || PORTABLE)
01247
01248         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01249 #endif
01250
01251         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01252         public static void IfIsEmpty(System.Collections.ICollection collection)
01253         {
01254             if (ReferenceEquals(collection, null) || collection.Count == 0)
01255             {
01256                 DoThrow();
01257             }
01258         }
01259
01281 #if (NET45 || NET46 || PORTABLE)
01282
01283         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01284 #endif
01285
01286         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01287         public static void IfIsEmpty(System.Collections.ICollection collection, string
      message)
01288         {
01289             if (ReferenceEquals(collection, null) || collection.Count == 0)
01290             {
01291                 DoThrow(message);
01292             }
01293         }
01294
01309 #if (NET45 || NET46 || PORTABLE)
01310
01311         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01312 #endif
01313
01314         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01315         public static void IfIsNotEmpty(System.Collections.ICollection collection)
01316         {
01317             if (ReferenceEquals(collection, null) || collection.Count > 0)
01318             {
01319                 DoThrow();
01320             }
01321         }
01322
01344 #if (NET45 || NET46 || PORTABLE)
01345
01346         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01347 #endif
01348
01349         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01350         public static void IfIsNotEmpty(System.Collections.ICollection collection, string
       message)
01351         {
01352             if (ReferenceEquals(collection, null) || collection.Count > 0)
01353             {
01354                 DoThrow(message);
01355             }
01356         }
```

```
01357
01372 #if (NET45 || NET46 || PORTABLE)
01373
01374         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01375 #endif
01376
01377         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01378         public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection)
01379         {
01380             if (ReferenceEquals(collection, null) || !collection.Any())
01381             {
01382                 DoThrow();
01383             }
01384         }
01385
01407 #if (NET45 || NET46 || PORTABLE)
01408
01409         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01410 #endif
01411
01412         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01413         public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection,
      string message)
01414         {
01415             if (ReferenceEquals(collection, null) || !collection.Any())
01416             {
01417                 DoThrow(message);
01418             }
01419         }
01420
01435 #if (NET45 || NET46 || PORTABLE)
01436
01437         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01438 #endif
01439
01440         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01441         public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
      )
01442         {
01443             if (ReferenceEquals(collection, null) || collection.Any())
01444             {
01445                 DoThrow();
01446             }
01447         }
01448
01470 #if (NET45 || NET46 || PORTABLE)
01471
01472         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01473 #endif
01474
01475         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01476         public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
      , string message)
01477         {
01478             if (ReferenceEquals(collection, null) || collection.Any())
01479             {
01480                 DoThrow(message);
01481             }
01482         }
01483
01499 #if (NET45 || NET46 || PORTABLE)
01500
01501         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01502 #endif
01503
01504         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01505         public static void IfIsInstanceOf(object instance, Type type)
01506         {
01507             if (PortableTypeInfo.IsInstanceOf(instance, type))
01508             {
01509                 DoThrow();
01510             }
01511         }
01512
01535 #if (NET45 || NET46 || PORTABLE)
01536
01537         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01538 #endif
01539
01540         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01541         public static void IfIsInstanceOf(object instance, Type type, string message)
```

```
01542         {
01543             if (PortableTypeInfo.IsInstanceOf(instance, type))
01544             {
01545                 DoThrow(message);
01546             }
01547         }
01548 
01564 #if (NET45 || NET46 || PORTABLE)
01565 
01566         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01567 #endif
01568 
01569         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01570         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01571         public static void IfIsInstanceOf<TType>(object instance)
01572         {
01573             if (instance is TType)
01574             {
01575                 DoThrow();
01576             }
01577         }
01578 
01601 #if (NET45 || NET46 || PORTABLE)
01602 
01603         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01604 #endif
01605 
01606         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01607         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01608         public static void IfIsInstanceOf<TType>(object instance, string message)
01609         {
01610             if (instance is TType)
01611             {
01612                 DoThrow(message);
01613             }
01614         }
01615 
01631 #if (NET45 || NET46 || PORTABLE)
01632 
01633         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01634 #endif
01635 
01636         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01637         public static void IfIsNotInstanceOf(object instance, Type type)
01638         {
01639             if (!PortableTypeInfo.IsInstanceOf(instance, type))
01640             {
01641                 DoThrow();
01642             }
01643         }
01644 
01667 #if (NET45 || NET46 || PORTABLE)
01668 
01669         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01670 #endif
01671 
01672         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01673         public static void IfIsNotInstanceOf(object instance, Type type, string message)
01674         {
01675             if (!PortableTypeInfo.IsInstanceOf(instance, type))
01676             {
01677                 DoThrow(message);
01678             }
01679         }
01680 
01696 #if (NET45 || NET46 || PORTABLE)
01697 
01698         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01699 #endif
01700 
01701         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01702         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01703         public static void IfIsNotInstanceOf<TType>(object instance)
01704         {
01705             if (!(instance is TType))
01706             {
01707                 DoThrow();
01708             }
01709         }
01710 
01733 #if (NET45 || NET46 || PORTABLE)
01734 
```

```
01735         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01736 #endif
01737
01738         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01739         [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01740         public static void IfIsNotInstanceOf<TType>(object instance, string message)
01741         {
01742             if (!(instance is TType))
01743             {
01744                 DoThrow(message);
01745             }
01746         }
01747
01762 #if (NET45 || NET46 || PORTABLE)
01763
01764         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01765 #endif
01766
01767         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01768         public static void IfIsNaN(double number)
01769         {
01770             if (double.IsNaN(number))
01771             {
01772                 DoThrow();
01773             }
01774         }
01775
01797 #if (NET45 || NET46 || PORTABLE)
01798
01799         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01800 #endif
01801
01802         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01803         public static void IfIsNaN(double number, string message)
01804         {
01805             if (double.IsNaN(number))
01806             {
01807                 DoThrow(message);
01808             }
01809         }
01810
01825 #if (NET45 || NET46 || PORTABLE)
01826
01827         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01828 #endif
01829
01830         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01831         public static void IfIsNotNaN(double number)
01832         {
01833             if (!double.IsNaN(number))
01834             {
01835                 DoThrow();
01836             }
01837         }
01838
01860 #if (NET45 || NET46 || PORTABLE)
01861
01862         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01863 #endif
01864
01865         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01866         public static void IfIsNotNaN(double number, string message)
01867         {
01868             if (!double.IsNaN(number))
01869             {
01870                 DoThrow(message);
01871             }
01872         }
01873
01888 #if (NET45 || NET46 || PORTABLE)
01889
01890         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01891 #endif
01892
01893         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01894         public static void IfIsNull<TArg>(TArg arg)
01895         {
01896             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      arg, null))
01897             {
01898                 DoThrow();
```

```
01899             }
01900         }
01901
01923 #if (NET45 || NET46 || PORTABLE)
01924
01925         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01926 #endif
01927
01928         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01929         public static void IfIsNull<TArg>(TArg arg, string message)
01930         {
01931             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      arg, null))
01932             {
01933                 DoThrow(message);
01934             }
01935         }
01936
01951 #if (NET45 || NET46 || PORTABLE)
01952
01953         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01954 #endif
01955
01956         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01957         public static void IfIsNotNull<TArg>(TArg arg)
01958         {
01959             if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(
      arg, null))
01960             {
01961                 DoThrow();
01962             }
01963         }
01964
01986 #if (NET45 || NET46 || PORTABLE)
01987
01988         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
01989 #endif
01990
01991         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01992         public static void IfIsNotNull<TArg>(TArg arg, string message)
01993         {
01994             if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(
      arg, null))
01995             {
01996                 DoThrow(message);
01997             }
01998         }
01999
02000         private static bool IsNullOrWhiteSpace(string value) => value == null || string.IsNullOrEmpty(value
      .Trim());
02001     }
02002 }
```

7.23 Raise.cs File Reference

Classes

- class PommaLabs.Thrower.Raise< TEx >

  New exception handling mechanism, which is more fluent than the old ones.

Namespaces

- namespace PommaLabs.Thrower

7.24 Raise.cs

```
00001 // File name: Raise.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.ExceptionHandlers;
00025 using System.Diagnostics.CodeAnalysis;
00026
00027 namespace PommaLabs.Thrower
00028 {
00032     public static class Raise
00033     {
00037         public static ArgumentExceptionHandler ArgumentException { get; } = new
      ArgumentExceptionHandler();
00038
00042         public static ArgumentNullExceptionHandler ArgumentNullException { get;
       } = new ArgumentNullExceptionHandler();
00043
00047         public static ArgumentOutOfRangeExceptionHandler
      ArgumentOutOfRangeException { get; } = new ArgumentOutOfRangeExceptionHandler();
00048
00052         public static HttpExceptionHandler HttpException { get; } = new
      HttpExceptionHandler();
00053
00057         public static IndexOutOfRangeExceptionHandler
      IndexOutOfRangeException { get; } = new IndexOutOfRangeExceptionHandler();
00058
00062         public static InvalidOperationExceptionHandler
      InvalidOperationException { get; } = new InvalidOperationExceptionHandler();
00063
00067         public static NotSupportedExceptionHandler NotSupportedException { get;
       } = new NotSupportedExceptionHandler();
00068
00072         public static ObjectDisposedExceptionHandler ObjectDisposedException
      { get; } = new ObjectDisposedExceptionHandler();
00073     }
00074 }
```

7.25 Obsolete/RaiseArgumentException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentException

  Utility methods which can be used to handle bad arguments.

Namespaces

- namespace PommaLabs.Thrower

7.26 RaiseArgumentException.cs

```
00001 // File name: RaiseArgumentException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Validation;
00025 using System;
00026 using System.Collections.Generic;
00027
00028 namespace PommaLabs.Thrower
00029 {
00036     [Obsolete("Please use Raise.ArgumentException.If* overloads, this class has been deprecated")]
00037     public sealed class RaiseArgumentException : RaiseBase
00038     {
00039         #region If
00040
00041         private const string DefaultIfMessage = "Argument is not valid";
00042
00047 #if (NET45 || NET46 || PORTABLE)
00048         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00049 #endif
00050
00051         public static void If(bool condition)
00052         {
00053             if (condition)
00054             {
00055                 throw new ArgumentException(DefaultIfMessage);
00056             }
00057         }
00058
00068 #if (NET45 || NET46 || PORTABLE)
00069         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00070 #endif
00071
00072         public static void If(bool condition, string argumentName, string message = null)
00073         {
00074             if (condition)
00075             {
00076                 throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00077             }
00078         }
00079
00080         #endregion If
00081
00082         #region IfNot
00083
00088 #if (NET45 || NET46 || PORTABLE)
00089         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00090 #endif
00091
00092         public static void IfNot(bool condition)
00093         {
00094             if (!condition)
00095             {
00096                 throw new ArgumentException(DefaultIfMessage);
00097             }
00098         }
00099
00109 #if (NET45 || NET46 || PORTABLE)
00110         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00111 #endif
00112
```

```
00113         public static void IfNot(bool condition, string argumentName, string message = null)
00114         {
00115             if (!condition)
00116             {
00117                 throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00118             }
00119         }
00120 
00121         #endregion IfNot
00122 
00123         #region IfIsNotValid
00124 
00130 #if (NET45 || NET46 || PORTABLE)
00131         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00132 #endif
00133 
00134         public static void IfIsNotValid<TArg>(TArg argument)
00135         {
00136             IList<ValidationError> validationErrors;
00137             if (!ObjectValidator.Validate(argument, out validationErrors))
00138             {
00139                 throw new ArgumentException(ObjectValidator.
      FormatValidationErrors(validationErrors, null));
00140             }
00141         }
00142 
00153 #if (NET45 || NET46 || PORTABLE)
00154         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00155 #endif
00156 
00157         public static void IfIsNotValid<TArg>(TArg argument, string argumentName, string message = null)
00158         {
00159             IList<ValidationError> validationErrors;
00160             if (!ObjectValidator.Validate(argument, out validationErrors))
00161             {
00162                 throw new ArgumentException(ObjectValidator.
      FormatValidationErrors(validationErrors, message), argumentName);
00163             }
00164         }
00165 
00166         #endregion IfIsNotValid
00167 
00168         #region IfIsNotValidEmailAddress
00169 
00170         private const string DefaultIfIsNotValidEmailAddressMessage = "String \"{0}\" is not a valid email
       address";
00171 
00176 #if (NET45 || NET46 || PORTABLE)
00177         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00178 #endif
00179 
00180         public static void IfIsNotValidEmailAddress(string emailAddress)
00181         {
00182             if (!EmailAddressValidator.Validate(emailAddress))
00183             {
00184                 var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
00185                 throw new ArgumentException(exceptionMsg);
00186             }
00187         }
00188 
00194 #if (NET45 || NET46 || PORTABLE)
00195         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00196 #endif
00197 
00198         public static void IfIsNotValidEmailAddress(string emailAddress,
      EmailAddressValidator.Options validatorOptions)
00199         {
00200             if (!EmailAddressValidator.Validate(emailAddress, validatorOptions
      ))
00201             {
00202                 var exceptionMsg = string.Format(DefaultIfIsNotValidEmailAddressMessage, emailAddress);
00203                 throw new ArgumentException(exceptionMsg);
00204             }
00205         }
00206 
00216 #if (NET45 || NET46 || PORTABLE)
00217         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00218 #endif
00219 
00220         public static void IfIsNotValidEmailAddress(string emailAddress, string
      argumentName, string message = null)
00221         {
```

```
00222             if (!EmailAddressValidator.Validate(emailAddress))
00223             {
00224                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage,
      emailAddress);
00225                 throw new ArgumentException(exceptionMsg, argumentName);
00226             }
00227         }
00228 
00239 #if (NET45 || NET46 || PORTABLE)
00240         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00241 #endif
00242 
00243         public static void IfIsNotValidEmailAddress(string emailAddress, string
      argumentName, EmailAddressValidator.Options validatorOptions, string message = null
      )
00244         {
00245             if (!EmailAddressValidator.Validate(emailAddress, validatorOptions
      ))
00246             {
00247                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidEmailAddressMessage,
      emailAddress);
00248                 throw new ArgumentException(exceptionMsg, argumentName);
00249             }
00250         }
00251 
00252         #endregion IfIsNotValidEmailAddress
00253 
00254         #region IfIsNotValidPhoneNumber
00255 
00256         private const string DefaultIfIsNotValidPhoneNumberMessage = "String \"{0}\" is not a valid phone
       number";
00257 
00262 #if (NET45 || NET46 || PORTABLE)
00263         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00264 #endif
00265 
00266         public static void IfIsNotValidPhoneNumber(string phoneNumber)
00267         {
00268             if (!PhoneNumberValidator.Validate(phoneNumber))
00269             {
00270                 var exceptionMsg = string.Format(DefaultIfIsNotValidPhoneNumberMessage, phoneNumber);
00271                 throw new ArgumentException(exceptionMsg);
00272             }
00273         }
00274 
00284 #if (NET45 || NET46 || PORTABLE)
00285         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00286 #endif
00287 
00288         public static void IfIsNotValidPhoneNumber(string phoneNumber, string
      argumentName, string message = null)
00289         {
00290             if (!PhoneNumberValidator.Validate(phoneNumber))
00291             {
00292                 var exceptionMsg = message ?? string.Format(DefaultIfIsNotValidPhoneNumberMessage,
      phoneNumber);
00293                 throw new ArgumentException(exceptionMsg, argumentName);
00294             }
00295         }
00296 
00297         #endregion IfIsNotValidPhoneNumber
00298 
00299         #region String validation
00300 
00301         private const string StringIsNullOrEmptyMessage = "Argument cannot be a null or empty string";
00302         private const string StringIsNullOrWhiteSpaceMessage = "Argument cannot be a null, empty or blank
       string";
00303 
00308 #if (NET45 || NET46 || PORTABLE)
00309         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00310 #endif
00311 
00312         public static void IfIsNullOrEmpty(string value)
00313         {
00314             if (ReferenceEquals(value, null) || string.Empty.Equals(value))
00315             {
00316                 throw new ArgumentException(StringIsNullOrEmptyMessage);
00317             }
00318         }
00319 
00329 #if (NET45 || NET46 || PORTABLE)
00330         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
```

```
00331 #endif
00332
00333         public static void IfIsNullOrEmpty(string value, string argumentName, string message
       = null)
00334         {
00335             if (ReferenceEquals(value, null) || string.Empty.Equals(value))
00336             {
00337                 throw new ArgumentException(message ?? StringIsNullOrEmptyMessage, argumentName);
00338             }
00339         }
00340
00345 #if (NET45 || NET46 || PORTABLE)
00346         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00347 #endif
00348
00349         public static void IfIsNullOrWhiteSpace(string value)
00350         {
00351             if (ReferenceEquals(value, null) || string.Empty.Equals(value.Trim()))
00352             {
00353                 throw new ArgumentException(StringIsNullOrWhiteSpaceMessage);
00354             }
00355         }
00356
00366 #if (NET45 || NET46 || PORTABLE)
00367         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00368 #endif
00369
00370         public static void IfIsNullOrWhiteSpace(string value, string argumentName,
      string message = null)
00371         {
00372             if (ReferenceEquals(value, null) || string.Empty.Equals(value.Trim()))
00373             {
00374                 throw new ArgumentException(message ?? StringIsNullOrWhiteSpaceMessage, argumentName);
00375             }
00376         }
00377
00378         #endregion String validation
00379
00380         #region Collection validation
00381
00382         internal const string CollectionIsNullOrEmptyMessage = "Argument cannot be a null or empty
       collection";
00383
00389 #if (NET45 || NET46 || PORTABLE)
00390         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00391 #endif
00392
00393         public static void IfIsNullOrEmpty<TItem>(ICollection<TItem> value)
00394         {
00395             if (ReferenceEquals(value, null) || value.Count == 0)
00396             {
00397                 throw new ArgumentException(CollectionIsNullOrEmptyMessage);
00398             }
00399         }
00400
00411 #if (NET45 || NET46 || PORTABLE)
00412         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00413 #endif
00414
00415         public static void IfIsNullOrEmpty<TItem>(ICollection<TItem> value, string argumentName, string
      message = null)
00416         {
00417             if (ReferenceEquals(value, null) || value.Count == 0)
00418             {
00419                 throw new ArgumentException(message ?? CollectionIsNullOrEmptyMessage, argumentName);
00420             }
00421         }
00422
00423         #endregion Collection validation
00424     }
00425 }
```

7.27 Obsolete/RaiseArgumentNullException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentNullException

  Utility methods which can be used to handle null references.

Namespaces

- namespace PommaLabs.Thrower

7.28 RaiseArgumentNullException.cs

```
00001 // File name: RaiseArgumentNullException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Reflection;
00025 using System;
00026
00027 namespace PommaLabs.Thrower
00028 {
00035     [Obsolete("Please use Raise.ArgumentNullException.If* overloads, this class has been deprecated")]
00036     public sealed class RaiseArgumentNullException :
      RaiseBase
00037     {
00038         #region Classes
00039
00045 #if (NET45 || NET46 || PORTABLE)
00046         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00047 #endif
00048
00049         public static void IfIsNull<TArg>(TArg argument)
00050         {
00051             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      argument, null))
00052             {
00053                 throw new ArgumentNullException();
00054             }
00055         }
00056
00063 #if (NET45 || NET46 || PORTABLE)
00064         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00065 #endif
00066
00067         public static void IfIsNull<TArg>(TArg argument, string argumentName)
00068         {
00069             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      argument, null))
00070             {
00071                 throw new ArgumentNullException(argumentName);
00072             }
00073         }
00074
00082 #if (NET45 || NET46 || PORTABLE)
00083         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00084 #endif
00085
00086         public static void IfIsNull<TArg>(TArg argument, string argumentName, string message)
00087         {
00088             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(
      argument, null))
00089             {
00090                 throw new ArgumentNullException(argumentName, message);
00091             }
00092         }
```

```
00093
00094         #endregion Classes
00095
00096         #region Nullable structs
00097
00103 #if (NET45 || NET46 || PORTABLE)
00104         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00105 #endif
00106
00107         public static void IfIsNull<TArg>(TArg? argument)
00108             where TArg : struct
00109         {
00110             if (!argument.HasValue)
00111             {
00112                 throw new ArgumentNullException();
00113             }
00114         }
00115
00122 #if (NET45 || NET46 || PORTABLE)
00123         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00124 #endif
00125
00126         public static void IfIsNull<TArg>(TArg? argument, string argumentName)
00127             where TArg : struct
00128         {
00129             if (!argument.HasValue)
00130             {
00131                 throw new ArgumentNullException(argumentName);
00132             }
00133         }
00134
00142 #if (NET45 || NET46 || PORTABLE)
00143         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00144 #endif
00145
00146         public static void IfIsNull<TArg>(TArg? argument, string argumentName, string message)
00147             where TArg : struct
00148         {
00149             if (!argument.HasValue)
00150             {
00151                 throw new ArgumentNullException(argumentName, message);
00152             }
00153         }
00154
00155         #endregion Nullable structs
00156     }
00157 }
```

7.29 Obsolete/RaiseArgumentOutOfRangeException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseArgumentOutOfRangeException

  Utility methods which can be used to handle ranges.

Namespaces

- namespace PommaLabs.Thrower

7.30 RaiseArgumentOutOfRangeException.cs

```
00001 // File name: RaiseArgumentOutOfRangeException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
```

```
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower
00027 {
00034     [Obsolete("Please use Raise.ArgumentOutOfRangeException.If* overloads, this class has been deprecated")
      ]
00035     public sealed class RaiseArgumentOutOfRangeException :
      RaiseBase
00036     {
00037         #region If
00038
00044 #if (NET45 || NET46 || PORTABLE)
00045         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00046 #endif
00047
00048         public static void If(bool condition, string argumentName = null)
00049         {
00050             if (condition)
00051             {
00052                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
      ArgumentOutOfRangeException(argumentName);
00053             }
00054         }
00055
00065 #if (NET45 || NET46 || PORTABLE)
00066         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00067 #endif
00068
00069         public static void If(bool condition, string argumentName, string message)
00070         {
00071             if (condition)
00072             {
00073                 throw new ArgumentOutOfRangeException(argumentName, message);
00074             }
00075         }
00076
00077         #endregion If
00078
00079         #region IfNot
00080
00086 #if (NET45 || NET46 || PORTABLE)
00087         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00088 #endif
00089
00090         public static void IfNot(bool condition, string argumentName = null)
00091         {
00092             if (!condition)
00093             {
00094                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
      ArgumentOutOfRangeException(argumentName);
00095             }
00096         }
00097
00107 #if (NET45 || NET46 || PORTABLE)
00108         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00109 #endif
00110
00111         public static void IfNot(bool condition, string argumentName, string message)
00112         {
00113             if (!condition)
00114             {
00115                 throw new ArgumentOutOfRangeException(argumentName, message);
00116             }
00117         }
00118
00119         #endregion IfNot
```

```
00120
00121         #region Less - Without parameter name, without message
00122
00130 #if (NET45 || NET46 || PORTABLE)
00131         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00132 #endif
00133
00134         public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
00135             where TArg : IComparable<TArg>
00136         {
00137             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00138             {
00139                 throw new ArgumentOutOfRangeException();
00140             }
00141         }
00142
00149 #if (NET45 || NET46 || PORTABLE)
00150         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00151 #endif
00152
00153         public static void IfIsLess(IComparable argument1, IComparable argument2)
00154         {
00155             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00156             {
00157                 throw new ArgumentOutOfRangeException();
00158             }
00159         }
00160
00161         #endregion Less - Without parameter name, without message
00162
00163         #region Less - With parameter name, without message
00164
00173 #if (NET45 || NET46 || PORTABLE)
00174         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00175 #endif
00176
00177         public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName)
00178             where TArg : IComparable<TArg>
00179         {
00180             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00181             {
00182                 throw new ArgumentOutOfRangeException(argumentName);
00183             }
00184         }
00185
00193 #if (NET45 || NET46 || PORTABLE)
00194         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00195 #endif
00196
00197         public static void IfIsLess(IComparable argument1, IComparable argument2, string
      argumentName)
00198         {
00199             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00200             {
00201                 throw new ArgumentOutOfRangeException(argumentName);
00202             }
00203         }
00204
00205         #endregion Less - With parameter name, without message
00206
00207         #region Less - With parameter name, with message
00208
00218 #if (NET45 || NET46 || PORTABLE)
00219         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00220 #endif
00221
00222         public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName, string
      message)
00223             where TArg : IComparable<TArg>
00224         {
00225             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00226             {
00227                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00228             }
00229         }
00230
00239 #if (NET45 || NET46 || PORTABLE)
00240         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00241 #endif
00242
00243         public static void IfIsLess(IComparable argument1, IComparable argument2, string
```

```
    argumentName, string message)
00244         {
00245             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00246             {
00247                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00248             }
00249         }
00250 
00251         #endregion Less - With parameter name, with message
00252 
00253         #region LessEqual - Without parameter name, without message
00254 
00262 #if (NET45 || NET46 || PORTABLE)
00263         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
    MethodImplOptions.AggressiveInlining)]
00264 #endif
00265 
00266         public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00267             where TArg : IComparable<TArg>
00268         {
00269             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00270             {
00271                 throw new ArgumentOutOfRangeException();
00272             }
00273         }
00274 
00281 #if (NET45 || NET46 || PORTABLE)
00282         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
    MethodImplOptions.AggressiveInlining)]
00283 #endif
00284 
00285         public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00286         {
00287             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00288             {
00289                 throw new ArgumentOutOfRangeException();
00290             }
00291         }
00292 
00293         #endregion LessEqual - Without parameter name, without message
00294 
00295         #region LessEqual - With parameter name, without message
00296 
00305 #if (NET45 || NET46 || PORTABLE)
00306         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
    MethodImplOptions.AggressiveInlining)]
00307 #endif
00308 
00309         public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00310             where TArg : IComparable<TArg>
00311         {
00312             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00313             {
00314                 throw new ArgumentOutOfRangeException(argumentName);
00315             }
00316         }
00317 
00325 #if (NET45 || NET46 || PORTABLE)
00326         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
    MethodImplOptions.AggressiveInlining)]
00327 #endif
00328 
00329         public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
    string argumentName)
00330         {
00331             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00332             {
00333                 throw new ArgumentOutOfRangeException(argumentName);
00334             }
00335         }
00336 
00337         #endregion LessEqual - With parameter name, without message
00338 
00339         #region LessEqual - With parameter name, with message
00340 
00350 #if (NET45 || NET46 || PORTABLE)
00351         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
    MethodImplOptions.AggressiveInlining)]
00352 #endif
00353 
00354         public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
    string message)
00355             where TArg : IComparable<TArg>
00356         {
00357             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00358             {
00359                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
```

```
00360             }
00361         }
00362 
00371 #if (NET45 || NET46 || PORTABLE)
00372         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00373 #endif
00374 
00375         public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
      string argumentName, string message)
00376         {
00377             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00378             {
00379                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00380             }
00381         }
00382 
00383         #endregion LessEqual - With parameter name, with message
00384 
00385         #region Greater - Without parameter name, without message
00386 
00394 #if (NET45 || NET46 || PORTABLE)
00395         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00396 #endif
00397 
00398         public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00399             where TArg : IComparable<TArg>
00400         {
00401             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00402             {
00403                 throw new ArgumentOutOfRangeException();
00404             }
00405         }
00406 
00413 #if (NET45 || NET46 || PORTABLE)
00414         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00415 #endif
00416 
00417         public static void IfIsGreater(IComparable argument1, IComparable argument2)
00418         {
00419             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00420             {
00421                 throw new ArgumentOutOfRangeException();
00422             }
00423         }
00424 
00425         #endregion Greater - Without parameter name, without message
00426 
00427         #region Greater - With parameter name, without message
00428 
00437 #if (NET45 || NET46 || PORTABLE)
00438         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00439 #endif
00440 
00441         public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName)
00442             where TArg : IComparable<TArg>
00443         {
00444             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00445             {
00446                 throw new ArgumentOutOfRangeException(argumentName);
00447             }
00448         }
00449 
00457 #if (NET45 || NET46 || PORTABLE)
00458         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00459 #endif
00460 
00461         public static void IfIsGreater(IComparable argument1, IComparable argument2, string
      argumentName)
00462         {
00463             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00464             {
00465                 throw new ArgumentOutOfRangeException(argumentName);
00466             }
00467         }
00468 
00469         #endregion Greater - With parameter name, without message
00470 
00471         #region Greater - With parameter name, with message
00472 
00482 #if (NET45 || NET46 || PORTABLE)
00483         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
```

```
00484 #endif
00485
00486         public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName, string
      message)
00487             where TArg : IComparable<TArg>
00488         {
00489             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00490             {
00491                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00492             }
00493         }
00494
00503 #if (NET45 || NET46 || PORTABLE)
00504         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00505 #endif
00506
00507         public static void IfIsGreater(IComparable argument1, IComparable argument2, string
      argumentName, string message)
00508         {
00509             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00510             {
00511                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00512             }
00513         }
00514
00515         #endregion Greater - With parameter name, with message
00516
00517         #region GreaterEqual - Without parameter name, without message
00518
00526 #if (NET45 || NET46 || PORTABLE)
00527         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00528 #endif
00529
00530         public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
00531             where TArg : IComparable<TArg>
00532         {
00533             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00534             {
00535                 throw new ArgumentOutOfRangeException();
00536             }
00537         }
00538
00545 #if (NET45 || NET46 || PORTABLE)
00546         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00547 #endif
00548
00549         public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
      argument2)
00550         {
00551             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00552             {
00553                 throw new ArgumentOutOfRangeException();
00554             }
00555         }
00556
00557         #endregion GreaterEqual - Without parameter name, without message
00558
00559         #region GreaterEqual - With parameter name, without message
00560
00569 #if (NET45 || NET46 || PORTABLE)
00570         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00571 #endif
00572
00573         public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00574             where TArg : IComparable<TArg>
00575         {
00576             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00577             {
00578                 throw new ArgumentOutOfRangeException(argumentName);
00579             }
00580         }
00581
00589 #if (NET45 || NET46 || PORTABLE)
00590         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00591 #endif
00592
00593         public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
      argument2, string argumentName)
00594         {
00595             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00596             {
00597                 throw new ArgumentOutOfRangeException(argumentName);
```

```
00598             }
00599         }
00600 
00601         #endregion GreaterEqual - With parameter name, without message
00602 
00603         #region GreaterEqual - With parameter name, with message
00604 
00614 #if (NET45 || NET46 || PORTABLE)
00615         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00616 #endif
00617 
00618         public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
      string message)
00619             where TArg : IComparable<TArg>
00620         {
00621             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00622             {
00623                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00624             }
00625         }
00626 
00635 #if (NET45 || NET46 || PORTABLE)
00636         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00637 #endif
00638 
00639         public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
      argument2, string argumentName, string message)
00640         {
00641             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00642             {
00643                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00644             }
00645         }
00646 
00647         #endregion GreaterEqual - With parameter name, with message
00648 
00649         #region Equal - Without parameter name, without message
00650 
00658 #if (NET45 || NET46 || PORTABLE)
00659         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00660 #endif
00661 
00662         public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00663             where TArg : IComparable<TArg>
00664         {
00665             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00666             {
00667                 throw new ArgumentOutOfRangeException();
00668             }
00669         }
00670 
00677 #if (NET45 || NET46 || PORTABLE)
00678         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00679 #endif
00680 
00681         public static void IfIsEqual(IComparable argument1, IComparable argument2)
00682         {
00683             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00684             {
00685                 throw new ArgumentOutOfRangeException();
00686             }
00687         }
00688 
00689         #endregion Equal - Without parameter name, without message
00690 
00691         #region Equal - With parameter name, without message
00692 
00701 #if (NET45 || NET46 || PORTABLE)
00702         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00703 #endif
00704 
00705         public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00706             where TArg : IComparable<TArg>
00707         {
00708             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00709             {
00710                 throw new ArgumentOutOfRangeException(argumentName);
00711             }
00712         }
00713 
00721 #if (NET45 || NET46 || PORTABLE)
00722         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
```

```
      MethodImplOptions.AggressiveInlining)]
00723 #endif
00724
00725         public static void IfIsEqual(IComparable argument1, IComparable argument2, string
      argumentName)
00726         {
00727             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00728             {
00729                 throw new ArgumentOutOfRangeException(argumentName);
00730             }
00731         }
00732
00733         #endregion Equal - With parameter name, without message
00734
00735         #region Equal - With parameter name, with message
00736
00746 #if (NET45 || NET46 || PORTABLE)
00747         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00748 #endif
00749
00750         public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
      message)
00751             where TArg : IComparable<TArg>
00752         {
00753             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00754             {
00755                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00756             }
00757         }
00758
00767 #if (NET45 || NET46 || PORTABLE)
00768         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00769 #endif
00770
00771         public static void IfIsEqual(IComparable argument1, IComparable argument2, string
      argumentName, string message)
00772         {
00773             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00774             {
00775                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00776             }
00777         }
00778
00779         #endregion Equal - With parameter name, with message
00780
00781         #region NotEqual - Without parameter name, without message
00782
00790 #if (NET45 || NET46 || PORTABLE)
00791         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00792 #endif
00793
00794         public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00795             where TArg : IComparable<TArg>
00796         {
00797             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00798             {
00799                 throw new ArgumentOutOfRangeException();
00800             }
00801         }
00802
00809 #if (NET45 || NET46 || PORTABLE)
00810         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00811 #endif
00812
00813         public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
00814         {
00815             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00816             {
00817                 throw new ArgumentOutOfRangeException();
00818             }
00819         }
00820
00821         #endregion NotEqual - Without parameter name, without message
00822
00823         #region NotEqual - With parameter name, without message
00824
00833 #if (NET45 || NET46 || PORTABLE)
00834         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00835 #endif
00836
00837         public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00838             where TArg : IComparable<TArg>
```

```
00839         {
00840             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00841             {
00842                 throw new ArgumentOutOfRangeException(argumentName);
00843             }
00844         }
00845 
00853 #if (NET45 || NET46 || PORTABLE)
00854         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00855 #endif
00856 
00857         public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
      argumentName)
00858         {
00859             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00860             {
00861                 throw new ArgumentOutOfRangeException(argumentName);
00862             }
00863         }
00864 
00865         #endregion NotEqual - With parameter name, without message
00866 
00867         #region NotEqual - With parameter name, with message
00868 
00878 #if (NET45 || NET46 || PORTABLE)
00879         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00880 #endif
00881 
00882         public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
      message)
00883             where TArg : IComparable<TArg>
00884         {
00885             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00886             {
00887                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00888             }
00889         }
00890 
00899 #if (NET45 || NET46 || PORTABLE)
00900         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00901 #endif
00902 
00903         public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
      argumentName, string message)
00904         {
00905             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00906             {
00907                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00908             }
00909         }
00910 
00911         #endregion NotEqual - With parameter name, with message
00912     }
00913 }
```

7.31 Obsolete/RaiseHttpException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseHttpException

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

Namespaces

- namespace PommaLabs.Thrower

7.32 RaiseHttpException.cs

```
00001 // File name: RaiseHttpException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025 using System.Net;
00026
00027 namespace PommaLabs.Thrower
00028 {
00033     [Obsolete("Please use Raise.HttpException.If* overloads, this class has been deprecated")]
00034     public static class RaiseHttpException
00035     {
00042 #if (NET45 || NET46 || PORTABLE)
00043
00044         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00045 #endif
00046
00047         public static void If(bool condition, HttpStatusCode httpStatusCode, string message = null)
00048         {
00049             if (condition)
00050             {
00051                 throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
      HttpException(httpStatusCode, message);
00052             }
00053         }
00054
00062 #if (NET45 || NET46 || PORTABLE)
00063
00064         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00065 #endif
00066
00067         public static void If(bool condition, HttpStatusCode httpStatusCode, string message,
      HttpExceptionInfo additionalInfo)
00068         {
00069             if (condition)
00070             {
00071                 throw new HttpException(httpStatusCode, message, additionalInfo);
00072             }
00073         }
00074
00081 #if (NET45 || NET46 || PORTABLE)
00082
00083         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00084 #endif
00085
00086         public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message = null)
00087         {
00088             if (!condition)
00089             {
00090                 throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
      HttpException(httpStatusCode, message);
00091             }
00092         }
00093
00101 #if (NET45 || NET46 || PORTABLE)
00102
00103         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00104 #endif
00105
00106         public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message,
      HttpExceptionInfo additionalInfo)
```

```
00107         {
00108             if (!condition)
00109             {
00110                 throw new HttpException(httpStatusCode, message, additionalInfo);
00111             }
00112         }
00113     }
00114 }
```

7.33 Obsolete/RaiseIndexOutOfRangeException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseIndexOutOfRangeException

  Utility methods which can be used to handle indexes.

Namespaces

- namespace PommaLabs.Thrower

7.34 RaiseIndexOutOfRangeException.cs

```
00001 // File name: RaiseIndexOutOfRangeException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower
00027 {
00034     [Obsolete("Please use Raise.IndexOutOfRangeException.If* overloads, this class has been deprecated")]
00035     public sealed class RaiseIndexOutOfRangeException :
      RaiseBase
00036     {
00037         #region Less - Without message
00038
00046 #if (NET45 || NET46 || PORTABLE)
00047         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00048 #endif
00049
00050         public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
00051             where TArg : IComparable<TArg>
00052         {
00053             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00054             {
00055                 throw new IndexOutOfRangeException();
00056             }
00057         }
00058
00065 #if (NET45 || NET46 || PORTABLE)
```

```
00066         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00067 #endif
00068
00069         public static void IfIsLess(IComparable argument1, IComparable argument2)
00070         {
00071             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00072             {
00073                 throw new IndexOutOfRangeException();
00074             }
00075         }
00076
00077         #endregion Less - Without message
00078
00079         #region Less - With message
00080
00089 #if (NET45 || NET46 || PORTABLE)
00090         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00091 #endif
00092
00093         public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string message)
00094             where TArg : IComparable<TArg>
00095         {
00096             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00097             {
00098                 throw new IndexOutOfRangeException(message);
00099             }
00100         }
00101
00109 #if (NET45 || NET46 || PORTABLE)
00110         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00111 #endif
00112
00113         public static void IfIsLess(IComparable argument1, IComparable argument2, string message)
00114         {
00115             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00116             {
00117                 throw new IndexOutOfRangeException(message);
00118             }
00119         }
00120
00121         #endregion Less - With message
00122
00123         #region LessEqual - Without message
00124
00132 #if (NET45 || NET46 || PORTABLE)
00133         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00134 #endif
00135
00136         public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00137             where TArg : IComparable<TArg>
00138         {
00139             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00140             {
00141                 throw new IndexOutOfRangeException();
00142             }
00143         }
00144
00151 #if (NET45 || NET46 || PORTABLE)
00152         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00153 #endif
00154
00155         public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00156         {
00157             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00158             {
00159                 throw new IndexOutOfRangeException();
00160             }
00161         }
00162
00163         #endregion LessEqual - Without message
00164
00165         #region LessEqual - With message
00166
00175 #if (NET45 || NET46 || PORTABLE)
00176         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00177 #endif
00178
00179         public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00180             where TArg : IComparable<TArg>
00181         {
00182             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
```

```
00183             {
00184                 throw new IndexOutOfRangeException(message);
00185             }
00186         }
00187 
00195 #if (NET45 || NET46 || PORTABLE)
00196         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00197 #endif
00198 
00199         public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
      string message)
00200         {
00201             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00202             {
00203                 throw new IndexOutOfRangeException(message);
00204             }
00205         }
00206 
00207         #endregion LessEqual - With message
00208 
00209         #region Greater - Without message
00210 
00218 #if (NET45 || NET46 || PORTABLE)
00219         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00220 #endif
00221 
00222         public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00223             where TArg : IComparable<TArg>
00224         {
00225             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00226             {
00227                 throw new IndexOutOfRangeException();
00228             }
00229         }
00230 
00237 #if (NET45 || NET46 || PORTABLE)
00238         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00239 #endif
00240 
00241         public static void IfIsGreater(IComparable argument1, IComparable argument2)
00242         {
00243             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00244             {
00245                 throw new IndexOutOfRangeException();
00246             }
00247         }
00248 
00249         #endregion Greater - Without message
00250 
00251         #region Greater - With message
00252 
00261 #if (NET45 || NET46 || PORTABLE)
00262         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00263 #endif
00264 
00265         public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string message)
00266             where TArg : IComparable<TArg>
00267         {
00268             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00269             {
00270                 throw new IndexOutOfRangeException(message);
00271             }
00272         }
00273 
00281 #if (NET45 || NET46 || PORTABLE)
00282         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00283 #endif
00284 
00285         public static void IfIsGreater(IComparable argument1, IComparable argument2, string
      message)
00286         {
00287             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00288             {
00289                 throw new IndexOutOfRangeException(message);
00290             }
00291         }
00292 
00293         #endregion Greater - With message
00294 
00295         #region GreaterEqual - Without message
00296 
00304 #if (NET45 || NET46 || PORTABLE)
```

```
00305         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00306 #endif
00307
00308         public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
00309             where TArg : IComparable<TArg>
00310         {
00311             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00312             {
00313                 throw new IndexOutOfRangeException();
00314             }
00315         }
00316
00323 #if (NET45 || NET46 || PORTABLE)
00324         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00325 #endif
00326
00327         public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
      argument2)
00328         {
00329             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00330             {
00331                 throw new IndexOutOfRangeException();
00332             }
00333         }
00334
00335         #endregion GreaterEqual - Without message
00336
00337         #region GreaterEqual - With message
00338
00347 #if (NET45 || NET46 || PORTABLE)
00348         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00349 #endif
00350
00351         public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00352             where TArg : IComparable<TArg>
00353         {
00354             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00355             {
00356                 throw new IndexOutOfRangeException(message);
00357             }
00358         }
00359
00367 #if (NET45 || NET46 || PORTABLE)
00368         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00369 #endif
00370
00371         public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
      argument2, string message)
00372         {
00373             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00374             {
00375                 throw new IndexOutOfRangeException(message);
00376             }
00377         }
00378
00379         #endregion GreaterEqual - With message
00380
00381         #region Equal - Without message
00382
00390 #if (NET45 || NET46 || PORTABLE)
00391         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00392 #endif
00393
00394         public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00395             where TArg : IComparable<TArg>
00396         {
00397             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00398             {
00399                 throw new IndexOutOfRangeException();
00400             }
00401         }
00402
00409 #if (NET45 || NET46 || PORTABLE)
00410         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00411 #endif
00412
00413         public static void IfIsEqual(IComparable argument1, IComparable argument2)
00414         {
00415             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00416             {
00417                 throw new IndexOutOfRangeException();
```

```
00418             }
00419         }
00420 
00421         #endregion Equal - Without message
00422 
00423         #region Equal - With message
00424 
00433 #if (NET45 || NET46 || PORTABLE)
00434         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00435 #endif
00436 
00437         public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string message)
00438             where TArg : IComparable<TArg>
00439         {
00440             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00441             {
00442                 throw new IndexOutOfRangeException(message);
00443             }
00444         }
00445 
00453 #if (NET45 || NET46 || PORTABLE)
00454         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00455 #endif
00456 
00457         public static void IfIsEqual(IComparable argument1, IComparable argument2, string message)
00458         {
00459             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00460             {
00461                 throw new IndexOutOfRangeException(message);
00462             }
00463         }
00464 
00465         #endregion Equal - With message
00466 
00467         #region NotEqual - Without message
00468 
00476 #if (NET45 || NET46 || PORTABLE)
00477         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00478 #endif
00479 
00480         public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00481             where TArg : IComparable<TArg>
00482         {
00483             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00484             {
00485                 throw new IndexOutOfRangeException();
00486             }
00487         }
00488 
00495 #if (NET45 || NET46 || PORTABLE)
00496         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00497 #endif
00498 
00499         public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
00500         {
00501             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00502             {
00503                 throw new IndexOutOfRangeException();
00504             }
00505         }
00506 
00507         #endregion NotEqual - Without message
00508 
00509         #region NotEqual - With message
00510 
00519 #if (NET45 || NET46 || PORTABLE)
00520         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00521 #endif
00522 
00523         public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string message)
00524             where TArg : IComparable<TArg>
00525         {
00526             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00527             {
00528                 throw new IndexOutOfRangeException(message);
00529             }
00530         }
00531 
00539 #if (NET45 || NET46 || PORTABLE)
00540         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00541 #endif
```

```
00542
00543         public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
      message)
00544         {
00545             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00546             {
00547                 throw new IndexOutOfRangeException(message);
00548             }
00549         }
00550
00551         #endregion NotEqual - With message
00552     }
00553 }
```

7.35 Obsolete/RaiseInvalidOperationException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseInvalidOperationException

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

Namespaces

- namespace PommaLabs.Thrower

7.36 RaiseInvalidOperationException.cs

```
00001 // File name: RaiseInvalidOperationException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower
00027 {
00034     [Obsolete("Please use Raise.InvalidOperationException.If* overloads, this class has been deprecated")]
00035     public sealed class RaiseInvalidOperationException :
      RaiseBase
00036     {
00042 #if (NET45 || NET46 || PORTABLE)
00043         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00044 #endif
00045
00046         public static void If(bool condition, string message = null)
00047         {
00048             if (condition)
00049             {
00050                 throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new
      InvalidOperationException(message);
00051             }
```

```
00052         }
00053
00059 #if (NET45 || NET46 || PORTABLE)
00060         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00061 #endif
00062
00063         public static void IfNot(bool condition, string message = null)
00064         {
00065             if (!condition)
00066             {
00067                 throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new
      InvalidOperationException(message);
00068             }
00069         }
00070     }
00071 }
```

7.37 Obsolete/RaiseNotSupportedException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseNotSupportedException

  Utility methods which can be used to handle unsupported operations.

Namespaces

- namespace PommaLabs.Thrower

7.38 RaiseNotSupportedException.cs

```
00001 // File name: RaiseNotSupportedException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower
00027 {
00034     [Obsolete("Please use Raise.NotSupportedException.If* overloads, this class has been deprecated")]
00035     public sealed class RaiseNotSupportedException :
      RaiseBase
00036     {
00042 #if (NET45 || NET46 || PORTABLE)
00043         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00044 #endif
00045
00046         public static void If(bool condition, string message = null)
00047         {
00048             if (condition)
```

```
00049             {
00050                 throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new
      NotSupportedException(message);
00051             }
00052         }
00053
00059 #if (NET45 || NET46 || PORTABLE)
00060         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00061 #endif
00062
00063         public static void IfNot(bool condition, string message = null)
00064         {
00065             if (!condition)
00066             {
00067                 throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new
      NotSupportedException(message);
00068             }
00069         }
00070     }
00071 }
```

7.39 Obsolete/RaiseObjectDisposedException.cs File Reference

Classes

- class PommaLabs.Thrower.RaiseObjectDisposedException

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

Namespaces

- namespace PommaLabs.Thrower

7.40 RaiseObjectDisposedException.cs

```
00001 // File name: RaiseObjectDisposedException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower
00027 {
00034     [Obsolete("Please use Raise.ObjectDisposedException.If* overloads, this class has been deprecated")]
00035     public sealed class RaiseObjectDisposedException :
      RaiseBase
00036     {
00043 #if (NET45 || NET46 || PORTABLE)
00044         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00045 #endif
```

```
00046
00047         public static void If(bool disposed, string objectName, string message = null)
00048         {
00049             if (disposed)
00050             {
00051                 throw string.IsNullOrEmpty(message) ? new ObjectDisposedException(objectName) : new
      ObjectDisposedException(objectName, message);
00052             }
00053         }
00054     }
00055 }
```

7.41 RaiseGeneric.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.

Namespaces

- namespace PommaLabs.Thrower

7.42 RaiseGeneric.cs

```
00001 // File name: RaiseGeneric.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Reflection;
00025 using System;
00026 using System.Collections.Generic;
00027 using System.Diagnostics.CodeAnalysis;
00028 using System.Linq;
00029 using System.Reflection;
00030
00031 namespace PommaLabs.Thrower
00032 {
00036     public abstract class RaiseBase
00037     {
00041         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00042         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00043         protected static readonly Type[] NoCtorTypes = new Type[0];
00044
00049         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00050         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00051         protected static readonly Type[] StrExCtorTypes = { typeof(string), typeof(Exception) };
```

```
00052
00057         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00058         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00059         protected static readonly Type[] StrCtorType = { typeof(string) };
00060     }
00061
00071     public sealed partial class Raise<TEx> : RaiseBase where TEx : Exception
00072     {
00073 #pragma warning disable RECS0108 // Warns about static fields in generic types
00074
00079         private static readonly bool ExTypeIsAbstract = PortableTypeInfo.
      IsAbstract(typeof(TEx));
00080
00085         private static readonly ConstructorInfo NoArgsCtor = GetCtor(NoCtorTypes);
00086
00099         private static readonly ConstructorInfo MsgCtor = GetCtor(StrExCtorTypes) ?? GetCtor(StrCtorType);
00100
00105         private static readonly int MsgArgCount = (MsgCtor == null) ? 0 : MsgCtor.GetParameters().Length;
00106
00107 #pragma warning restore RECS0108 // Warns about static fields in generic types
00108
00112         private Raise()
00113         {
00114             throw new InvalidOperationException("This class should not be instantiated");
00115         }
00116
00131         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00132         public static void If(bool cond)
00133         {
00134             if (cond)
00135             {
00136                 DoThrow();
00137             }
00138         }
00139
00161         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00162         public static void If(bool cond, string message)
00163         {
00164             if (cond)
00165             {
00166                 DoThrow(message);
00167             }
00168         }
00169
00184         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00185         public static void IfNot(bool cond)
00186         {
00187             if (!cond)
00188             {
00189                 DoThrow();
00190             }
00191         }
00192
00214         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00215         public static void IfNot(bool cond, string message)
00216         {
00217             if (!cond)
00218             {
00219                 DoThrow(message);
00220             }
00221         }
00222
00223         private static ConstructorInfo GetCtor(System.Collections.Generic.IList<Type> ctorTypes)
00224         {
00225             return (from c in PortableTypeInfo.GetConstructors(typeof(TEx))
00226                     let args = c.GetParameters()
00227                     let zipArgs = MyZip(args, ctorTypes, (argType, ctorType) => new { argType, ctorType })
00228                     where args.Length == ctorTypes.Count &&
00229                           (c.IsPublic || c.IsAssembly) &&
00230                           zipArgs.All(t => ReferenceEquals(t.argType.ParameterType, t.ctorType))
00231                     select c).FirstOrDefault();
00232         }
00233
00234         private static IEnumerable<TResult> MyZip<TFirst, TSecond, TResult>(IEnumerable<TFirst> first,
      IEnumerable<TSecond> second, Func<TFirst, TSecond, TResult> resultSelector)
00235         {
00236             Raise.ArgumentNullException.IfIsNull(first, nameof(first));
00237             Raise.ArgumentNullException.IfIsNull(second, nameof(second));
00238             Raise.ArgumentNullException.IfIsNull(resultSelector, nameof(resultSelector));
00239
00240             using (IEnumerator<TFirst> e1 = first.GetEnumerator())
00241             using (IEnumerator<TSecond> e2 = second.GetEnumerator())
00242             {
00243                 while (e1.MoveNext() && e2.MoveNext())
00244                 {
00245 #pragma warning disable CC0031 // Check for null before calling a delegate
00246                     yield return resultSelector(e1.Current, e2.Current);
```

```
00247 #pragma warning restore CC0031 // Check for null before calling a delegate
00248                 }
00249             }
00250         }
00251 
00252         private static void DoThrow()
00253         {
00254             // Checks whether the proper constructor exists. If not, then we produce an internal exception.
00255             if (ExTypeIsAbstract)
00256             {
00257                 throw ThrowerException.AbstractEx;
00258             }
00259             if (NoArgsCtor == null)
00260             {
00261                 throw ThrowerException.MissingNoArgsCtor;
00262             }
00263             // A proper constrctor exists: therefore, we can throw the exception.
00264             throw (TEx) NoArgsCtor.Invoke(new object[0]);
00265         }
00266 
00267         private static void DoThrow(string message)
00268         {
00269             // Checks whether the proper constructor exists. If not, then we produce an internal exception.
00270             if (ExTypeIsAbstract)
00271             {
00272                 throw ThrowerException.AbstractEx;
00273             }
00274             if (MsgCtor == null)
00275             {
00276                 throw ExTypeIsAbstract ? ThrowerException.AbstractEx :
    ThrowerException.MissingMsgCtor;
00277             }
00278             // A proper constrctor exists: therefore, we can throw the exception.
00279             var messageArgs = new object[MsgArgCount];
00280             messageArgs[0] = message;
00281             throw (TEx) MsgCtor.Invoke(messageArgs);
00282         }
00283     }
00284 }
```

7.43 Reflection/FastMember/CallSiteCache.cs File Reference

Classes

- class **PommaLabs.Thrower.Reflection.FastMember.CallSiteCache**

Namespaces

- namespace PommaLabs.Thrower.Reflection.FastMember

7.44 CallSiteCache.cs

```
00001 // Copyright 2013 Marc Gravell
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
00004 // in compliance with the License. You may obtain a copy of the License at:
00005 //
00006 // "http://www.apache.org/licenses/LICENSE-2.0"
00007 //
00008 // Unless required by applicable law or agreed to in writing, software distributed under the License
00009 // is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
00010 // or implied. See the License for the specific language governing permissions and limitations under
00011 // the License.
00012 
00013 #if !NET35 && !PORTABLE
00014 
00015 using Microsoft.CSharp.RuntimeBinder;
00016 using System;
00017 using System.Collections;
00018 using System.Runtime.CompilerServices;
```

```
00019
00020 namespace PommaLabs.Thrower.Reflection.FastMember
00021 {
00022     internal static class CallSiteCache
00023     {
00024         private static readonly Hashtable Getters = new Hashtable(), Setters = new Hashtable();
00025
00026         internal static object GetValue(string name, object target)
00027         {
00028             var callSite = (CallSite<Func<CallSite, object, object>>) Getters[name];
00029             if (callSite == null)
00030             {
00031                 var newSite = CallSite<Func<CallSite, object, object>>.Create(Binder.GetMember(
      CSharpBinderFlags.None, name, typeof(CallSiteCache), new CSharpArgumentInfo[] { CSharpArgumentInfo.Create(
      CSharpArgumentInfoFlags.None, null) }));
00032                 lock (Getters)
00033                 {
00034                     callSite = (CallSite<Func<CallSite, object, object>>) Getters[name];
00035                     if (callSite == null)
00036                     {
00037                         Getters[name] = callSite = newSite;
00038                     }
00039                 }
00040             }
00041             return callSite.Target(callSite, target);
00042         }
00043
00044         internal static void SetValue(string name, object target, object value)
00045         {
00046             var callSite = (CallSite<Func<CallSite, object, object, object>>) Setters[name];
00047             if (callSite == null)
00048             {
00049                 var newSite = CallSite<Func<CallSite, object, object, object>>.Create(Binder.SetMember(
      CSharpBinderFlags.None, name, typeof(CallSiteCache), new CSharpArgumentInfo[] { CSharpArgumentInfo.Create(
      CSharpArgumentInfoFlags.None, null), CSharpArgumentInfo.Create(CSharpArgumentInfoFlags.UseCompileTimeType, null) }
      ));
00050                 lock (Setters)
00051                 {
00052                     callSite = (CallSite<Func<CallSite, object, object, object>>) Setters[name];
00053                     if (callSite == null)
00054                     {
00055                         Setters[name] = callSite = newSite;
00056                     }
00057                 }
00058             }
00059             callSite.Target(callSite, target, value);
00060         }
00061     }
00062 }
00063
00064 #endif
```

7.45 Reflection/FastMember/MemberSet.cs File Reference

Classes

- class PommaLabs.Thrower.Reflection.FastMember.MemberSet

  Represents an abstracted view of the members defined for a type.

- class PommaLabs.Thrower.Reflection.FastMember.Member

  Represents an abstracted view of an individual member defined for a type.

Namespaces

- namespace PommaLabs.Thrower.Reflection.FastMember

7.46 MemberSet.cs

```
00001 // Copyright 2013 Marc Gravell
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
00004 // in compliance with the License. You may obtain a copy of the License at:
00005 //
00006 // "http://www.apache.org/licenses/LICENSE-2.0"
00007 //
00008 // Unless required by applicable law or agreed to in writing, software distributed under the License
00009 // is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
00010 // or implied. See the License for the specific language governing permissions and limitations under
00011 // the License.
00012
00013 #if !PORTABLE
00014
00015 using System;
00016 using System.Collections.Generic;
00017 using System.Linq;
00018 using System.Reflection;
00019
00020 namespace PommaLabs.Thrower.Reflection.FastMember
00021 {
00025     public sealed class MemberSet : IEnumerable<Member>, IList<Member>
00026     {
00027         private Member[] members;
00028
00029         internal MemberSet(Type type)
00030         {
00031             members = type
00032                 .GetProperties()
00033                 .Cast<MemberInfo>()
00034                 .Concat(type.GetFields().Cast<MemberInfo>()).OrderBy(x => x.Name, StringComparer.
      InvariantCulture)
00035                 .Select(member => new Member(member))
00036                 .ToArray();
00037         }
00038
00042         public IEnumerator<Member> GetEnumerator()
00043         {
00044             foreach (var member in members) yield return member;
00045         }
00046
00050         public Member this[int index] => members[index];
00051
00055         public int Count => members.Length;
00056
00057         Member IList<Member>.this[int index]
00058         {
00059             get { return members[index]; }
00060             set { throw new NotSupportedException(); }
00061         }
00062
00063         System.Collections.IEnumerator System.Collections.IEnumerable.GetEnumerator() =>
      GetEnumerator();
00064
00065         bool ICollection<Member>.Remove(Member item)
00066         {
00067             throw new NotSupportedException();
00068         }
00069
00070         void ICollection<Member>.Add(Member item)
00071         {
00072             throw new NotSupportedException();
00073         }
00074
00075         void ICollection<Member>.Clear()
00076         {
00077             throw new NotSupportedException();
00078         }
00079
00080         void IList<Member>.RemoveAt(int index)
00081         {
00082             throw new NotSupportedException();
00083         }
00084
00085         void IList<Member>.Insert(int index, Member item)
00086         {
00087             throw new NotSupportedException();
00088         }
00089
00090         bool ICollection<Member>.Contains(Member item) => members.Contains(item);
00091
00092         void ICollection<Member>.CopyTo(Member[] array, int arrayIndex)
00093         {
00094             members.CopyTo(array, arrayIndex);
```

```
00095         }
00096
00097         bool ICollection<Member>.IsReadOnly => true;
00098
00099         int IList<Member>.IndexOf(Member member) => Array.IndexOf<Member>(members, member);
00100     }
00101
00105     public sealed class Member
00106     {
00107         private readonly MemberInfo member;
00108
00109         internal Member(MemberInfo member)
00110         {
00111             this.member = member;
00112         }
00113
00117         public string Name => member.Name;
00118
00122         public Type Type
00123         {
00124             get
00125             {
00126                 switch (member.MemberType)
00127                 {
00128                     case MemberTypes.Field: return ((FieldInfo) member).FieldType;
00129                     case MemberTypes.Property: return ((PropertyInfo) member).PropertyType;
00130                     default: throw new NotSupportedException(member.MemberType.ToString());
00131                 }
00132             }
00133         }
00134
00139         public bool IsDefined(Type attributeType)
00140         {
00141             if (attributeType == null) throw new ArgumentNullException(nameof(attributeType));
00142             return Attribute.IsDefined(member, attributeType);
00143         }
00144     }
00145 }
00146
00147 #endif
```

7.47 Reflection/FastMember/ObjectAccessor.cs File Reference

Classes

- class PommaLabs.Thrower.Reflection.FastMember.ObjectAccessor

  Represents an individual object, allowing access to members by-name.

Namespaces

- namespace PommaLabs.Thrower.Reflection.FastMember

7.48 ObjectAccessor.cs

```
00001 // Copyright 2013 Marc Gravell
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
00004 // in compliance with the License. You may obtain a copy of the License at:
00005 //
00006 // "http://www.apache.org/licenses/LICENSE-2.0"
00007 //
00008 // Unless required by applicable law or agreed to in writing, software distributed under the License
00009 // is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
00010 // or implied. See the License for the specific language governing permissions and limitations under
00011 // the License.
00012
00013 #if !PORTABLE
00014
00015 using System;
00016 using System.Collections;
```

```
00017 using System.Collections.Generic;
00018 using System.Linq;
00019
00020 #if !NET35
00021 using System.Dynamic;
00022 #endif
00023
00024 namespace PommaLabs.Thrower.Reflection.FastMember
00025 {
00029     public abstract class ObjectAccessor : IDictionary<string, object>
00030     {
00034         public abstract object this[string name] { get; set; }
00035
00039         public abstract object Target { get; }
00040
00045         public override bool Equals(object obj) => Target.Equals(obj);
00046
00050         public override int GetHashCode() => Target.GetHashCode();
00051
00055         public override string ToString() => Target.ToString();
00056
00061         public static ObjectAccessor Create(object target) => Create(target, false);
00062
00068         public static ObjectAccessor Create(object target, bool allowNonPublicAccessors
      )
00069         {
00070             if (target == null) throw new ArgumentNullException(nameof(target));
00071 #if !NET35
00072             var dlr = target as IDynamicMetaObjectProvider;
00073             if (dlr != null) return new DynamicWrapper(dlr); // use the DLR
00074 #endif
00075             return new TypeAccessorWrapper(target, TypeAccessor.
      Create(target.GetType(), allowNonPublicAccessors));
00076         }
00077
00078         #region IDictionary<string, object> members
00079
00088         public abstract ICollection<string> Keys { get; }
00089
00098         public abstract ICollection<object> Values { get; }
00099
00104         public abstract int Count { get; }
00105
00114         public bool IsReadOnly => true;
00115
00126         public abstract bool ContainsKey(string key);
00127
00140         public void Add(string key, object value)
00141         {
00142             throw new NotSupportedException();
00143         }
00144
00157         public bool Remove(string key)
00158         {
00159             throw new NotSupportedException();
00160         }
00161
00177         public abstract bool TryGetValue(string key, out object value);
00178
00186         public void Add(KeyValuePair<string, object> item)
00187         {
00188             throw new NotSupportedException();
00189         }
00190
00197         public void Clear()
00198         {
00199             throw new NotSupportedException();
00200         }
00201
00211         public abstract bool Contains(KeyValuePair<string, object> item);
00212
00236         public void CopyTo(KeyValuePair<string, object>[] array, int arrayIndex)
00237         {
00238             foreach (var kv in this)
00239             {
00240                 array[arrayIndex++] = kv;
00241             }
00242         }
00243
00256         public bool Remove(KeyValuePair<string, object> item)
00257         {
00258             throw new NotSupportedException();
00259         }
00260
00266         public abstract IEnumerator<KeyValuePair<string, object>> GetEnumerator();
00267
00276         IEnumerator IEnumerable.GetEnumerator() => GetEnumerator();
```

```
00277
00278         #endregion IDictionary<string, object> members
00279
00280         private sealed class TypeAccessorWrapper : ObjectAccessor, IDictionary<string, object
    >
00281         {
00282             private readonly TypeAccessor _accessor;
00283             private readonly MemberSet _members;
00284
00285             public TypeAccessorWrapper(object target, TypeAccessor accessor)
00286             {
00287                 Target = target;
00288                 _accessor = accessor;
00289                 _members = accessor.GetMembers();
00290             }
00291
00292             public override object this[string name]
00293             {
00294                 get { return _accessor[Target, name]; }
00295                 set { _accessor[Target, name] = value; }
00296             }
00297
00298             public override object Target { get; }
00299
00300             #region IDictionary<string, object> members
00301
00302             public override ICollection<string> Keys => _members.Select(x => x.Name).ToArray();
00303
00304             public override ICollection<object> Values => _members.Select(x => _accessor[Target, x.Name]).
    ToArray();
00305
00306             public override int Count => _members.Count;
00307
00308             public override bool ContainsKey(string key) => _members.Any(x => x.Name == key);
00309
00310             public override bool TryGetValue(string key, out object value)
00311             {
00312                 if (ContainsKey(key))
00313                 {
00314                     value = _accessor[Target, key];
00315                     return true;
00316                 }
00317                 value = null;
00318                 return false;
00319             }
00320
00321             public override bool Contains(KeyValuePair<string, object> item) => _members.Any(x => x.Name ==
     item.Key && _accessor[Target, item.Key] == item.Value);
00322
00323             public override IEnumerator<KeyValuePair<string, object>> GetEnumerator()
00324             {
00325                 foreach (var m in _members)
00326                 {
00327                     yield return new KeyValuePair<string, object>(m.Name, _accessor[Target, m.Name]);
00328                 }
00329             }
00330
00331             #endregion IDictionary<string, object> members
00332         }
00333
00334 #if !NET35
00335         sealed class DynamicWrapper : ObjectAccessor
00336         {
00337             public DynamicWrapper(IDynamicMetaObjectProvider target)
00338             {
00339                 Target = target;
00340             }
00341
00342             public override object this[string name]
00343             {
00344                 get { return CallSiteCache.GetValue(name, Target); }
00345                 set { CallSiteCache.SetValue(name, Target, value); }
00346             }
00347
00348             public override object Target { get; }
00349
00350         #region IDictionary<string, object> members
00351
00352             public override ICollection<string> Keys
00353             {
00354                 get { throw new NotSupportedException(); }
00355             }
00356
00357             public override ICollection<object> Values
00358             {
00359                 get { throw new NotSupportedException(); }
00360             }
```

```
00361
00362             public override int Count
00363             {
00364                 get { throw new NotSupportedException(); }
00365             }
00366
00367             public override bool ContainsKey(string key)
00368             {
00369                 throw new NotSupportedException();
00370             }
00371
00372             public override bool TryGetValue(string key, out object value)
00373             {
00374                 throw new NotSupportedException();
00375             }
00376
00377             public override bool Contains(KeyValuePair<string, object> item)
00378             {
00379                 throw new NotSupportedException();
00380             }
00381
00382             public override IEnumerator<KeyValuePair<string, object>> GetEnumerator()
00383             {
00384                 throw new NotSupportedException();
00385             }
00386
00387         #endregion IDictionary<string, object> members
00388         }
00389 #endif
00390     }
00391 }
00392
00393 #endif
```

7.49 Reflection/FastMember/ObjectReader.cs File Reference

Classes

- class **PommaLabs.Thrower.Reflection.FastMember.ObjectReader**

  Provides a means of reading a sequence of objects as a data-reader, for example for use with SqlBulkCopy or other data-base oriented code

Namespaces

- namespace PommaLabs.Thrower.Reflection.FastMember

7.50 ObjectReader.cs

```
00001 // Copyright 2013 Marc Gravell
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
00004 // in compliance with the License. You may obtain a copy of the License at:
00005 //
00006 // "http://www.apache.org/licenses/LICENSE-2.0"
00007 //
00008 // Unless required by applicable law or agreed to in writing, software distributed under the License
00009 // is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
00010 // or implied. See the License for the specific language governing permissions and limitations under
00011 // the License.
00012
00013 #if !PORTABLE
00014
00015 using System;
00016 using System.Collections;
00017 using System.Collections.Generic;
00018 using System.Data;
00019
00020 namespace PommaLabs.Thrower.Reflection.FastMember
00021 {
```

```
00026     internal class ObjectReader : IDataReader
00027     {
00028         private IEnumerator source;
00029         private readonly TypeAccessor accessor;
00030         private readonly string[] memberNames;
00031         private readonly Type[] effectiveTypes;
00032         private readonly BitArray allowNull;
00033 
00039         public static ObjectReader Create<T>(IEnumerable<T> source, params string[] members)
00040         {
00041             return new ObjectReader(typeof(T), source, members);
00042         }
00043 
00050         public ObjectReader(Type type, IEnumerable source, params string[] members)
00051         {
00052             if (source == null) throw new ArgumentOutOfRangeException(nameof(source));
00053 
00054             var allMembers = members == null || members.Length == 0;
00055 
00056             this.accessor = TypeAccessor.Create(type);
00057             if (accessor.GetMembersSupported)
00058             {
00059                 var typeMembers = this.accessor.GetMembers();
00060 
00061                 if (allMembers)
00062                 {
00063                     members = new string[typeMembers.Count];
00064                     for (int i = 0; i < members.Length; i++)
00065                     {
00066                         members[i] = typeMembers[i].Name;
00067                     }
00068                 }
00069 
00070                 this.allowNull = new BitArray(members.Length);
00071                 this.effectiveTypes = new Type[members.Length];
00072                 for (int i = 0; i < members.Length; i++)
00073                 {
00074                     Type memberType = null;
00075                     var allowNull = true;
00076                     var hunt = members[i];
00077                     foreach (var member in typeMembers)
00078                     {
00079                         if (member.Name == hunt)
00080                         {
00081                             if (memberType == null)
00082                             {
00083                                 var tmp = member.Type;
00084                                 memberType = Nullable.GetUnderlyingType(tmp) ?? tmp;
00085 
00086                                 allowNull = !(memberType.IsValueType && memberType == tmp);
00087 
00088                                 // but keep checking, in case of duplicates
00089                             }
00090                             else
00091                             {
00092                                 memberType = null; // duplicate found; say nothing
00093                                 break;
00094                             }
00095                         }
00096                     }
00097                     this.allowNull[i] = allowNull;
00098                     this.effectiveTypes[i] = memberType ?? typeof(object);
00099                 }
00100             }
00101             else if (allMembers)
00102             {
00103                 throw new InvalidOperationException("Member information is not available for this type; the
      required members must be specified explicitly");
00104             }
00105 
00106             this.current = null;
00107             this.memberNames = (string[]) members.Clone();
00108 
00109             this.source = source.GetEnumerator();
00110         }
00111 
00112         private object current;
00113 
00114         void IDataReader.Close()
00115         {
00116             Dispose();
00117         }
00118 
00119         int IDataReader.Depth
00120         {
00121             get { return 0; }
00122         }
```

```
00123
00124         DataTable IDataReader.GetSchemaTable()
00125         {
00126             // these are the columns used by DataTable load
00127             var table = new DataTable
00128             {
00129                 Columns =
00130                 {
00131                     {"ColumnOrdinal", typeof(int)},
00132                     {"ColumnName", typeof(string)},
00133                     {"DataType", typeof(Type)},
00134                     {"ColumnSize", typeof(int)},
00135                     {"AllowDBNull", typeof(bool)}
00136                 }
00137             };
00138             var rowData = new object[5];
00139             for (int i = 0; i < memberNames.Length; i++)
00140             {
00141                 rowData[0] = i;
00142                 rowData[1] = memberNames[i];
00143                 rowData[2] = effectiveTypes == null ? typeof(object) : effectiveTypes[i];
00144                 rowData[3] = -1;
00145                 rowData[4] = allowNull == null ? true : allowNull[i];
00146                 table.Rows.Add(rowData);
00147             }
00148             return table;
00149         }
00150
00151         bool IDataReader.IsClosed
00152         {
00153             get { return source == null; }
00154         }
00155
00156         bool IDataReader.NextResult()
00157         {
00158             return false;
00159         }
00160
00161         bool IDataReader.Read()
00162         {
00163             var tmp = source;
00164             if (tmp != null && tmp.MoveNext())
00165             {
00166                 current = tmp.Current;
00167                 return true;
00168             }
00169             current = null;
00170             return false;
00171         }
00172
00173         int IDataReader.RecordsAffected
00174         {
00175             get { return 0; }
00176         }
00177
00181         public void Dispose()
00182         {
00183             current = null;
00184             var tmp = source as IDisposable;
00185             source = null;
00186             if (tmp != null) tmp.Dispose();
00187         }
00188
00189         int IDataRecord.FieldCount
00190         {
00191             get { return memberNames.Length; }
00192         }
00193
00194         bool IDataRecord.GetBoolean(int i)
00195         {
00196             return (bool) this[i];
00197         }
00198
00199         byte IDataRecord.GetByte(int i)
00200         {
00201             return (byte) this[i];
00202         }
00203
00204         long IDataRecord.GetBytes(int i, long fieldOffset, byte[] buffer, int bufferoffset, int length)
00205         {
00206             var s = (byte[]) this[i];
00207             var available = s.Length - (int) fieldOffset;
00208             if (available <= 0) return 0;
00209
00210             var count = Math.Min(length, available);
00211             Buffer.BlockCopy(s, (int) fieldOffset, buffer, bufferoffset, count);
00212             return count;
```

```
00213         }
00214
00215         char IDataRecord.GetChar(int i)
00216         {
00217             return (char) this[i];
00218         }
00219
00220         long IDataRecord.GetChars(int i, long fieldoffset, char[] buffer, int bufferoffset, int length)
00221         {
00222             var s = (string) this[i];
00223             var available = s.Length - (int) fieldoffset;
00224             if (available <= 0) return 0;
00225
00226             var count = Math.Min(length, available);
00227             s.CopyTo((int) fieldoffset, buffer, bufferoffset, count);
00228             return count;
00229         }
00230
00231         IDataReader IDataRecord.GetData(int i)
00232         {
00233             throw new NotSupportedException();
00234         }
00235
00236         string IDataRecord.GetDataTypeName(int i)
00237         {
00238             return (effectiveTypes == null ? typeof(object) : effectiveTypes[i]).Name;
00239         }
00240
00241         DateTime IDataRecord.GetDateTime(int i)
00242         {
00243             return (DateTime) this[i];
00244         }
00245
00246         decimal IDataRecord.GetDecimal(int i)
00247         {
00248             return (decimal) this[i];
00249         }
00250
00251         double IDataRecord.GetDouble(int i)
00252         {
00253             return (double) this[i];
00254         }
00255
00256         Type IDataRecord.GetFieldType(int i)
00257         {
00258             return effectiveTypes == null ? typeof(object) : effectiveTypes[i];
00259         }
00260
00261         float IDataRecord.GetFloat(int i)
00262         {
00263             return (float) this[i];
00264         }
00265
00266         Guid IDataRecord.GetGuid(int i)
00267         {
00268             return (Guid) this[i];
00269         }
00270
00271         short IDataRecord.GetInt16(int i)
00272         {
00273             return (short) this[i];
00274         }
00275
00276         int IDataRecord.GetInt32(int i)
00277         {
00278             return (int) this[i];
00279         }
00280
00281         long IDataRecord.GetInt64(int i)
00282         {
00283             return (long) this[i];
00284         }
00285
00286         string IDataRecord.GetName(int i)
00287         {
00288             return memberNames[i];
00289         }
00290
00291         int IDataRecord.GetOrdinal(string name)
00292         {
00293             return Array.IndexOf(memberNames, name);
00294         }
00295
00296         string IDataRecord.GetString(int i)
00297         {
00298             return (string) this[i];
00299         }
```

```
00300
00301         object IDataRecord.GetValue(int i)
00302         {
00303             return this[i];
00304         }
00305
00306         int IDataRecord.GetValues(object[] values)
00307         {
00308             // duplicate the key fields on the stack
00309             var members = this.memberNames;
00310             var current = this.current;
00311             var accessor = this.accessor;
00312
00313             var count = Math.Min(values.Length, members.Length);
00314             for (int i = 0; i < count; i++) values[i] = accessor[current, members[i]] ?? DBNull.Value;
00315             return count;
00316         }
00317
00318         bool IDataRecord.IsDBNull(int i)
00319         {
00320             return this[i] is DBNull;
00321         }
00322
00323         object IDataRecord.this[string name]
00324         {
00325             get { return accessor[current, name] ?? DBNull.Value; }
00326         }
00327
00331         public object this[int i]
00332         {
00333             get { return accessor[current, memberNames[i]] ?? DBNull.Value; }
00334         }
00335     }
00336 }
00337
00338 #endif
```

7.51 Reflection/FastMember/TypeAccessor.cs File Reference

Classes

- class PommaLabs.Thrower.Reflection.FastMember.TypeAccessor

  Provides by-name member-access to objects of a given type.
- class PommaLabs.Thrower.Reflection.FastMember.TypeAccessor.RuntimeTypeAccessor

  A TypeAccessor based on a Type implementation, with available member metadata

Namespaces

- namespace PommaLabs.Thrower.Reflection.FastMember

7.52 TypeAccessor.cs

```
00001 // Copyright 2013 Marc Gravell
00002 //
00003 // Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except
00004 // in compliance with the License. You may obtain a copy of the License at:
00005 //
00006 // "http://www.apache.org/licenses/LICENSE-2.0"
00007 //
00008 // Unless required by applicable law or agreed to in writing, software distributed under the License
00009 // is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express
00010 // or implied. See the License for the specific language governing permissions and limitations under
00011 // the License.
00012
00013 #if !PORTABLE
00014
00015 using System;
00016 using System.Collections;
```

```
00017 using System.Collections.Generic;
00018 using System.Reflection;
00019 using System.Reflection.Emit;
00020 using System.Threading;
00021
00022 #if !NET35
00023 using System.Dynamic;
00024 #endif
00025
00026 namespace PommaLabs.Thrower.Reflection.FastMember
00027 {
00031     public abstract class TypeAccessor
00032     {
00033         // hash-table has better read-without-locking semantics than dictionary
00034         private static readonly Hashtable publicAccessorsOnly = new Hashtable(), nonPublicAccessors = new
      Hashtable();
00035
00039         public virtual bool CreateNewSupported => false;
00040
00044         public virtual object CreateNew() { throw new NotSupportedException(); }
00045
00049         public virtual bool GetMembersSupported => false;
00050
00054         public virtual MemberSet GetMembers() { throw new NotSupportedException(); }
00055
00061         public static TypeAccessor Create(Type type) => Create(type, false);
00062
00067         public static TypeAccessor Create<T>() => Create(typeof(T), false);
00068
00075         public static TypeAccessor Create(Type type, bool allowNonPublicAccessors)
00076         {
00077             if (type == null) throw new ArgumentNullException(nameof(type));
00078             var lookup = allowNonPublicAccessors ? nonPublicAccessors : publicAccessorsOnly;
00079             var obj = (TypeAccessor) lookup[type];
00080             if (obj != null) return obj;
00081
00082             lock (lookup)
00083             {
00084                 // double-check
00085                 obj = (TypeAccessor) lookup[type];
00086                 if (obj != null) return obj;
00087
00088                 obj = CreateNew(type, allowNonPublicAccessors);
00089
00090                 lookup[type] = obj;
00091                 return obj;
00092             }
00093         }
00094
00100         public static TypeAccessor Create<T>(bool allowNonPublicAccessors) => Create(typeof(T),
       allowNonPublicAccessors);
00101
00102 #if !NET35
00103         sealed class DynamicAccessor : TypeAccessor
00104         {
00105             public static readonly DynamicAccessor Singleton = new DynamicAccessor();
00106             private DynamicAccessor() { }
00107             public override object this[object target, string name]
00108             {
00109                 get { return CallSiteCache.GetValue(name, target); }
00110                 set { CallSiteCache.SetValue(name, target, value); }
00111             }
00112         }
00113 #endif
00114
00115         private static AssemblyBuilder assembly;
00116         private static ModuleBuilder module;
00117         private static int counter;
00118
00119         private static readonly MethodInfo tryGetValue = typeof(Dictionary<string, int>).GetMethod("
      TryGetValue");
00120
00121         private static void WriteMapImpl(ILGenerator il, Type type, List<MemberInfo> members, FieldBuilder
      mapField, bool allowNonPublicAccessors, bool isGet)
00122         {
00123             OpCode obj, index, value;
00124
00125             var fail = il.DefineLabel();
00126             if (mapField == null)
00127             {
00128                 index = OpCodes.Ldarg_0;
00129                 obj = OpCodes.Ldarg_1;
00130                 value = OpCodes.Ldarg_2;
00131             }
00132             else
00133             {
00134                 il.DeclareLocal(typeof(int));
```

```
00135                 index = OpCodes.Ldloc_0;
00136                 obj = OpCodes.Ldarg_1;
00137                 value = OpCodes.Ldarg_3;
00138 
00139                 il.Emit(OpCodes.Ldarg_0);
00140                 il.Emit(OpCodes.Ldfld, mapField);
00141                 il.Emit(OpCodes.Ldarg_2);
00142                 il.Emit(OpCodes.Ldloca_S, (byte) 0);
00143                 il.EmitCall(OpCodes.Callvirt, tryGetValue, null);
00144                 il.Emit(OpCodes.Brfalse, fail);
00145             }
00146             var labels = new Label[members.Count];
00147             for (int i = 0; i < labels.Length; i++)
00148             {
00149                 labels[i] = il.DefineLabel();
00150             }
00151             il.Emit(index);
00152             il.Emit(OpCodes.Switch, labels);
00153             il.MarkLabel(fail);
00154             il.Emit(OpCodes.Ldstr, "name");
00155             il.Emit(OpCodes.Newobj, typeof(ArgumentOutOfRangeException).GetConstructor(new Type[] { typeof(
      string) }));
00156             il.Emit(OpCodes.Throw);
00157             for (int i = 0; i < labels.Length; i++)
00158             {
00159                 il.MarkLabel(labels[i]);
00160                 var member = members[i];
00161                 var isFail = true;
00162                 switch (member.MemberType)
00163                 {
00164                     case MemberTypes.Field:
00165                         var field = (FieldInfo) member;
00166                         il.Emit(obj);
00167                         Cast(il, type, true);
00168                         if (isGet)
00169                         {
00170                             il.Emit(OpCodes.Ldfld, field);
00171                             if (field.FieldType.IsValueType) il.Emit(OpCodes.Box, field.FieldType);
00172                         }
00173                         else
00174                         {
00175                             il.Emit(value);
00176                             Cast(il, field.FieldType, false);
00177                             il.Emit(OpCodes.Stfld, field);
00178                         }
00179                         il.Emit(OpCodes.Ret);
00180                         isFail = false;
00181                         break;
00182 
00183                     case MemberTypes.Property:
00184                         var prop = (PropertyInfo) member;
00185                         MethodInfo accessor;
00186                         if (prop.CanRead && (accessor = isGet ? prop.GetGetMethod(allowNonPublicAccessors)
      : prop.GetSetMethod(allowNonPublicAccessors)) != null)
00187                         {
00188                             il.Emit(obj);
00189                             Cast(il, type, true);
00190                             if (isGet)
00191                             {
00192                                 il.EmitCall(type.IsValueType ? OpCodes.Call : OpCodes.Callvirt, accessor,
      null);
00193                                 if (prop.PropertyType.IsValueType) il.Emit(OpCodes.Box, prop.PropertyType);
00194                             }
00195                             else
00196                             {
00197                                 il.Emit(value);
00198                                 Cast(il, prop.PropertyType, false);
00199                                 il.EmitCall(type.IsValueType ? OpCodes.Call : OpCodes.Callvirt, accessor,
      null);
00200                             }
00201                             il.Emit(OpCodes.Ret);
00202                             isFail = false;
00203                         }
00204                         break;
00205                 }
00206                 if (isFail) il.Emit(OpCodes.Br, fail);
00207             }
00208         }
00209 
00210         private static readonly MethodInfo stringEquals = typeof(string).GetMethod("op_Equality", new Type[
      ] { typeof(string), typeof(string) });
00211 
00215         protected abstract class RuntimeTypeAccessor :
      TypeAccessor
00216         {
00220             protected abstract Type Type { get; }
00221 
```

```
00225             public override bool GetMembersSupported => true;
00226
00227             private MemberSet members;
00228
00232             public override MemberSet GetMembers() => members ?? (members = new
      MemberSet(Type));
00233         }
00234
00235         private sealed class DelegateAccessor : RuntimeTypeAccessor
00236         {
00237             private readonly Dictionary<string, int> map;
00238             private readonly Func<int, object, object> getter;
00239             private readonly Action<int, object, object> setter;
00240             private readonly Func<object> ctor;
00241             private readonly Type type;
00242             protected override Type Type => type;
00243
00244             public DelegateAccessor(Dictionary<string, int> map, Func<int, object, object> getter,
      Action<int, object, object> setter, Func<object> ctor, Type type)
00245             {
00246                 this.map = map;
00247                 this.getter = getter;
00248                 this.setter = setter;
00249                 this.ctor = ctor;
00250                 this.type = type;
00251             }
00252
00253             public override bool CreateNewSupported => ctor != null;
00254
00255             public override object CreateNew() => ctor != null ? ctor() : base.CreateNew();
00256
00257             public override object this[object target, string name]
00258             {
00259                 get
00260                 {
00261                     int index;
00262                     if (map.TryGetValue(name, out index)) return getter(index, target);
00263                     else throw new ArgumentOutOfRangeException(nameof(name));
00264                 }
00265                 set
00266                 {
00267                     int index;
00268                     if (map.TryGetValue(name, out index)) setter(index, target, value);
00269                     else throw new ArgumentOutOfRangeException(nameof(name));
00270                 }
00271             }
00272         }
00273
00274         private static bool IsFullyPublic(Type type, PropertyInfo[] props, bool allowNonPublicAccessors)
00275         {
00276             while (type.IsNestedPublic) type = type.DeclaringType;
00277             if (!type.IsPublic) return false;
00278
00279             if (allowNonPublicAccessors)
00280             {
00281                 for (int i = 0; i < props.Length; i++)
00282                 {
00283                     if (props[i].GetGetMethod(true) != null && props[i].GetGetMethod(false) == null) return
       false; // non-public getter
00284                     if (props[i].GetSetMethod(true) != null && props[i].GetSetMethod(false) == null) return
       false; // non-public setter
00285                 }
00286             }
00287
00288             return true;
00289         }
00290
00291         private static TypeAccessor CreateNew(Type type, bool allowNonPublicAccessors)
00292         {
00293 #if !NET35
00294             if (typeof(IDynamicMetaObjectProvider).IsAssignableFrom(type))
00295             {
00296                 return DynamicAccessor.Singleton;
00297             }
00298 #endif
00299
00300             var props = type.GetProperties(BindingFlags.Public | BindingFlags.Instance);
00301             var fields = type.GetFields(BindingFlags.Public | BindingFlags.Instance);
00302             var map = new Dictionary<string, int>(StringComparer.Ordinal);
00303             var members = new List<MemberInfo>(props.Length + fields.Length);
00304             var i = 0;
00305             foreach (var prop in props)
00306             {
00307                 if (!map.ContainsKey(prop.Name) && prop.GetIndexParameters().Length == 0)
00308                 {
00309                     map.Add(prop.Name, i++);
00310                     members.Add(prop);
```

```
00311                 }
00312             }
00313             foreach (var field in fields) if (!map.ContainsKey(field.Name)) { map.Add(field.Name, i++);
      members.Add(field); }
00314
00315             ConstructorInfo ctor = null;
00316             if (type.IsClass && !type.IsAbstract)
00317             {
00318                 ctor = type.GetConstructor(Type.EmptyTypes);
00319             }
00320             ILGenerator il;
00321             if (!IsFullyPublic(type, props, allowNonPublicAccessors))
00322             {
00323                 var dynGetter = new DynamicMethod(type.FullName + "_get", typeof(object), new Type[] {
      typeof(int), typeof(object) }, type, true);
00324                 var dynSetter = new DynamicMethod(type.FullName + "_set", null, new Type[] { typeof(int), t
      ypeof(object), typeof(object) }, type, true);
00325                 WriteMapImpl(dynGetter.GetILGenerator(), type, members, null, allowNonPublicAccessors, true
      );
00326                 WriteMapImpl(dynSetter.GetILGenerator(), type, members, null, allowNonPublicAccessors,
      false);
00327                 DynamicMethod dynCtor = null;
00328                 if (ctor != null)
00329                 {
00330                     dynCtor = new DynamicMethod(type.FullName + "_ctor", typeof(object), Type.EmptyTypes,
      type, true);
00331                     il = dynCtor.GetILGenerator();
00332                     il.Emit(OpCodes.Newobj, ctor);
00333                     il.Emit(OpCodes.Ret);
00334                 }
00335                 return new DelegateAccessor(
00336                     map,
00337                     (Func<int, object, object>) dynGetter.CreateDelegate(typeof(Func<int, object, object>))
      ,
00338                     (Action<int, object, object>) dynSetter.CreateDelegate(typeof(Action<int, object,
       object>)),
00339                     dynCtor == null ? null : (Func<object>) dynCtor.CreateDelegate(typeof(Func<object>)),
      type);
00340             }
00341
00342             // note this region is synchronized; only one is being created at a time so we don't need
00343             // to stress about the builders
00344             if (assembly == null)
00345             {
00346                 var name = new AssemblyName("FastMember_dynamic");
00347                 assembly = AppDomain.CurrentDomain.DefineDynamicAssembly(name, AssemblyBuilderAccess.Run);
00348                 module = assembly.DefineDynamicModule(name.Name);
00349             }
00350             var tb = module.DefineType("FastMember_dynamic." + type.Name + "_" + Interlocked.Increment(ref
      counter),
00351                 (typeof(TypeAccessor).Attributes | TypeAttributes.Sealed | TypeAttributes.
      Public) & ~(TypeAttributes.Abstract | TypeAttributes.NotPublic), typeof(
      RuntimeTypeAccessor));
00352
00353             il = tb.DefineConstructor(MethodAttributes.Public, CallingConventions.Standard, new[] {
00354                 typeof(Dictionary<string,int>)
00355             }).GetILGenerator();
00356             il.Emit(OpCodes.Ldarg_0);
00357             il.Emit(OpCodes.Ldarg_1);
00358             var mapField = tb.DefineField("_map", typeof(Dictionary<string, int>), FieldAttributes.InitOnly
       | FieldAttributes.Private);
00359             il.Emit(OpCodes.Stfld, mapField);
00360             il.Emit(OpCodes.Ret);
00361
00362             var indexer = typeof(TypeAccessor).GetProperty("Item");
00363             var baseGetter = indexer.GetGetMethod();
00364             var baseSetter = indexer.GetSetMethod();
00365             var body = tb.DefineMethod(baseGetter.Name, baseGetter.Attributes & ~MethodAttributes.Abstract,
       typeof(object), new Type[] { typeof(object), typeof(string) });
00366             il = body.GetILGenerator();
00367             WriteMapImpl(il, type, members, mapField, allowNonPublicAccessors, true);
00368             tb.DefineMethodOverride(body, baseGetter);
00369
00370             body = tb.DefineMethod(baseSetter.Name, baseSetter.Attributes & ~MethodAttributes.Abstract,
      null, new Type[] { typeof(object), typeof(string), typeof(object) });
00371             il = body.GetILGenerator();
00372             WriteMapImpl(il, type, members, mapField, allowNonPublicAccessors, false);
00373             tb.DefineMethodOverride(body, baseSetter);
00374
00375             MethodInfo baseMethod;
00376             if (ctor != null)
00377             {
00378                 baseMethod = typeof(TypeAccessor).GetProperty(nameof(CreateNewSupported)).
      GetGetMethod();
00379                 body = tb.DefineMethod(baseMethod.Name, baseMethod.Attributes, baseMethod.ReturnType, Type.
      EmptyTypes);
00380                 il = body.GetILGenerator();
```

```
00381                 il.Emit(OpCodes.Ldc_I4_1);
00382                 il.Emit(OpCodes.Ret);
00383                 tb.DefineMethodOverride(body, baseMethod);
00384 
00385                 baseMethod = typeof(TypeAccessor).GetMethod(nameof(CreateNew));
00386                 body = tb.DefineMethod(baseMethod.Name, baseMethod.Attributes, baseMethod.ReturnType, Type.
      EmptyTypes);
00387                 il = body.GetILGenerator();
00388                 il.Emit(OpCodes.Newobj, ctor);
00389                 il.Emit(OpCodes.Ret);
00390                 tb.DefineMethodOverride(body, baseMethod);
00391             }
00392 
00393             baseMethod = typeof(RuntimeTypeAccessor).GetProperty(nameof(Type),
      BindingFlags.NonPublic | BindingFlags.Instance).GetGetMethod(true);
00394             body = tb.DefineMethod(baseMethod.Name, baseMethod.Attributes & ~MethodAttributes.Abstract,
      baseMethod.ReturnType, Type.EmptyTypes);
00395             il = body.GetILGenerator();
00396             il.Emit(OpCodes.Ldtoken, type);
00397             il.Emit(OpCodes.Call, typeof(Type).GetMethod("GetTypeFromHandle"));
00398             il.Emit(OpCodes.Ret);
00399             tb.DefineMethodOverride(body, baseMethod);
00400 
00401             var accessor = (TypeAccessor) Activator.CreateInstance(tb.CreateType(), map);
00402             return accessor;
00403         }
00404 
00405         private static void Cast(ILGenerator il, Type type, bool valueAsPointer)
00406         {
00407             if (type == typeof(object)) { }
00408             else if (type.IsValueType)
00409             {
00410                 if (valueAsPointer)
00411                 {
00412                     il.Emit(OpCodes.Unbox, type);
00413                 }
00414                 else
00415                 {
00416                     il.Emit(OpCodes.Unbox_Any, type);
00417                 }
00418             }
00419             else
00420             {
00421                 il.Emit(OpCodes.Castclass, type);
00422             }
00423         }
00424 
00428         public abstract object this[object target, string name]
00429         {
00430             get;
00431             set;
00432         }
00433     }
00434 }
00435 
00436 #endif
```

7.53 Reflection/PortableSerializationAttributes.cs File Reference

7.54 PortableSerializationAttributes.cs

```
00001 // File name: PortableSerializationAttributes.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
```

```
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 #if PORTABLE
00025
00026 namespace System
00027 {
00031     [AttributeUsage(AttributeTargets.Class | AttributeTargets.Struct, AllowMultiple = true, Inherited =
      false)]
00032     public sealed class SerializableAttribute : Attribute
00033     {
00034         // This does nothing and should do nothing.
00035     }
00036
00040     [AttributeUsage(AttributeTargets.Field, Inherited = false)]
00041     public sealed class NonSerializedAttribute : Attribute
00042     {
00043         // This does nothing and should do nothing.
00044     }
00045 }
00046
00047 #endif
```

7.55 Reflection/PortableTypeInfo.cs File Reference

Classes

- class PommaLabs.Thrower.Reflection.PortableTypeInfo

  Portable version of some useful reflection methods.

Namespaces

- namespace PommaLabs.Thrower.Reflection

7.56 PortableTypeInfo.cs

```
00001 // File name: PortableTypeInfo.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025 using System.Collections.Generic;
00026 using System.Linq;
00027 using System.Reflection;
00028
00029 #if !PORTABLE
00030
00031 using PommaLabs.Thrower.Reflection.FastMember;
00032
```

```
00033 #endif
00034
00035 namespace PommaLabs.Thrower.Reflection
00036 {
00040     public static class PortableTypeInfo
00041     {
00042 #if !PORTABLE
00043         internal const BindingFlags PublicAndPrivateInstanceFlags = BindingFlags.Public | BindingFlags.
      NonPublic | BindingFlags.Instance;
00044         internal const BindingFlags PublicInstanceFlags = BindingFlags.Public | BindingFlags.Instance;
00045 #endif
00046
00047         private static readonly object[] EmptyObjectArray = new object[0];
00048
00057 #if (NET45 || NET46 || PORTABLE)
00058         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00059 #endif
00060
00061         public static IList<Attribute> GetCustomAttributes(MemberInfo memberInfo, bool
      inherit)
00062         {
00063 #if PORTABLE
00064             return memberInfo.GetCustomAttributes(inherit).ToArray();
00065 #else
00066             return memberInfo.GetCustomAttributes(inherit).Cast<Attribute>().ToArray();
00067 #endif
00068         }
00069
00075 #if (NET45 || NET46 || PORTABLE)
00076         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00077 #endif
00078
00079         public static IList<ConstructorInfo> GetConstructors(Type type)
00080         {
00081 #if PORTABLE
00082             return IntrospectionExtensions.GetTypeInfo(type).DeclaredConstructors.ToArray();
00083 #else
00084             return type.GetConstructors(PublicAndPrivateInstanceFlags);
00085 #endif
00086         }
00087
00093 #if (NET45 || NET46 || PORTABLE)
00094         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00095 #endif
00096
00097         public static IList<ConstructorInfo> GetConstructors<T>() => GetConstructors(typeof(T));
00098
00104 #if (NET45 || NET46 || PORTABLE)
00105         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00106 #endif
00107
00108         public static Type GetBaseType(Type type)
00109         {
00110 #if PORTABLE
00111             return IntrospectionExtensions.GetTypeInfo(type).BaseType;
00112 #else
00113             return type.BaseType;
00114 #endif
00115         }
00116
00122 #if (NET45 || NET46 || PORTABLE)
00123         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00124 #endif
00125
00126         public static Type GetGenericTypeDefinition(Type type)
00127         {
00128 #if PORTABLE
00129             return IntrospectionExtensions.GetTypeInfo(type).GetGenericTypeDefinition();
00130 #else
00131             return type.GetGenericTypeDefinition();
00132 #endif
00133         }
00134
00140 #if (NET45 || NET46 || PORTABLE)
00141         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00142 #endif
00143
00144         public static IList<Type> GetGenericTypeArguments(Type type)
00145         {
00146 #if PORTABLE
00147             return IntrospectionExtensions.GetTypeInfo(type).GenericTypeArguments;
```

```
00148 #else
00149             return type.GetGenericArguments();
00150 #endif
00151         }
00152 
00158 #if (NET45 || NET46 || PORTABLE)
00159         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00160 #endif
00161 
00162         public static IList<Type> GetInterfaces(Type type)
00163         {
00164 #if PORTABLE
00165             return IntrospectionExtensions.GetTypeInfo(type).ImplementedInterfaces.ToArray();
00166 #else
00167             return type.GetInterfaces();
00168 #endif
00169         }
00170 
00176 #if (NET45 || NET46 || PORTABLE)
00177         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00178 #endif
00179 
00180         public static IList<PropertyInfo> GetPublicProperties(Type type)
00181         {
00182 #if PORTABLE
00183             var properties = new List<PropertyInfo>();
00184             while (type != null)
00185             {
00186                 var typeInfo = IntrospectionExtensions.GetTypeInfo(type);
00187                 properties.AddRange(typeInfo.DeclaredProperties.Where(p => p.GetMethod.IsPublic));
00188                 type = typeInfo.BaseType;
00189             }
00190             return properties;
00191 #else
00192             return type.GetProperties(PublicInstanceFlags);
00193 #endif
00194         }
00195 
00201 #if (NET45 || NET46 || PORTABLE)
00202         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00203 #endif
00204 
00205         public static IList<PropertyInfo> GetPublicProperties<T>() => GetPublicProperties(typeof(T));
00206 
00207         #region GetPublicPropertyValue
00208 
00215 #if (NET45 || NET46 || PORTABLE)
00216         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00217 #endif
00218 
00219         public static object GetPublicPropertyValue(object instance, PropertyInfo
      propertyInfo)
00220         {
00221             Raise.ArgumentNullException.IfIsNull(instance, nameof(instance), "Instance cannot be null"
      );
00222             Raise.ArgumentException.IfNot(propertyInfo.CanRead, nameof(propertyInfo), "Given property
       cannot be read");
00223             return propertyInfo.GetValue(instance, EmptyObjectArray);
00224         }
00225 
00226 #if !PORTABLE
00227 #if (NET45 || NET46 || PORTABLE)
00235         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00236 #endif
00237 
00238         public static object GetPublicPropertyValue(
      TypeAccessor typeAccessor, object instance, PropertyInfo propertyInfo)
00239         {
00240             Raise.ArgumentNullException.IfIsNull(instance, nameof(instance), "Instance cannot be null"
      );
00241             Raise.ArgumentException.IfNot(propertyInfo.CanRead, nameof(propertyInfo), "Given property
       cannot be read");
00242             return typeAccessor[instance, propertyInfo.Name];
00243         }
00244 
00245 #endif
00246 
00247         #endregion GetPublicPropertyValue
00248 
00249         #region IsAbstract
00250 
00256 #if (NET45 || NET46 || PORTABLE)
```

```
00257         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00258 #endif
00259
00260         public static bool IsAbstract(Type type)
00261         {
00262 #if PORTABLE
00263             return IntrospectionExtensions.GetTypeInfo(type).IsAbstract;
00264 #else
00265             return type.IsAbstract;
00266 #endif
00267         }
00268
00274 #if (NET45 || NET46 || PORTABLE)
00275         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00276 #endif
00277
00278         public static bool IsAbstract<T>() => IsAbstract(typeof(T));
00279
00280         #endregion IsAbstract
00281
00282         #region IsClass
00283
00289 #if (NET45 || NET46 || PORTABLE)
00290         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00291 #endif
00292
00293         public static bool IsClass(Type type)
00294         {
00295 #if PORTABLE
00296             return IntrospectionExtensions.GetTypeInfo(type).IsClass;
00297 #else
00298             return type.IsClass;
00299 #endif
00300         }
00301
00307 #if (NET45 || NET46 || PORTABLE)
00308         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00309 #endif
00310
00311         public static bool IsClass<T>() => IsClass(typeof(T));
00312
00313         #endregion IsClass
00314
00325 #if (NET45 || NET46 || PORTABLE)
00326         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00327 #endif
00328
00329         public static bool IsAssignableFrom(object obj, Type type)
00330         {
00331             if (ReferenceEquals(obj, null) || ReferenceEquals(type, null))
00332             {
00333                 return false;
00334             }
00335
00336 #if PORTABLE
00337             return IntrospectionExtensions.GetTypeInfo(obj.GetType()).IsAssignableFrom(
      IntrospectionExtensions.GetTypeInfo(type));
00338 #else
00339             return obj.GetType().IsAssignableFrom(type);
00340 #endif
00341         }
00342
00343         #region IsEnum
00344
00350 #if (NET45 || NET46 || PORTABLE)
00351         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00352 #endif
00353
00354         public static bool IsEnum(Type type)
00355         {
00356 #if PORTABLE
00357             return IntrospectionExtensions.GetTypeInfo(type).IsEnum;
00358 #else
00359             return type.IsEnum;
00360 #endif
00361         }
00362
00368 #if (NET45 || NET46 || PORTABLE)
00369         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00370 #endif
```

```
00371
00372         public static bool IsEnum<T>() => IsEnum(typeof(T));
00373
00374         #endregion IsEnum
00375
00376         #region IsGenericType
00377
00383 #if (NET45 || NET46 || PORTABLE)
00384         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00385 #endif
00386
00387         public static bool IsGenericType(Type type)
00388         {
00389 #if PORTABLE
00390             return IntrospectionExtensions.GetTypeInfo(type).IsGenericType;
00391 #else
00392             return type.IsGenericType;
00393 #endif
00394         }
00395
00401 #if (NET45 || NET46 || PORTABLE)
00402         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00403 #endif
00404
00405         public static bool IsGenericType<T>() => IsGenericType(typeof(T));
00406
00407         #endregion IsGenericType
00408
00409         #region IsGenericTypeDefinition
00410
00416 #if (NET45 || NET46 || PORTABLE)
00417         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00418 #endif
00419
00420         public static bool IsGenericTypeDefinition(Type type)
00421         {
00422 #if PORTABLE
00423             return IntrospectionExtensions.GetTypeInfo(type).IsGenericTypeDefinition;
00424 #else
00425             return type.IsGenericTypeDefinition;
00426 #endif
00427         }
00428
00434 #if (NET45 || NET46 || PORTABLE)
00435         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00436 #endif
00437
00438         public static bool IsGenericTypeDefinition<T>() => IsGenericTypeDefinition(typeof(T));
00439
00440         #endregion IsGenericTypeDefinition
00441
00448 #if (NET45 || NET46 || PORTABLE)
00449         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00450 #endif
00451
00452         public static bool IsInstanceOf(object obj, Type type)
00453         {
00454             if (ReferenceEquals(obj, null) || ReferenceEquals(type, null))
00455             {
00456                 return false;
00457             }
00458
00459 #if PORTABLE
00460             return IntrospectionExtensions.GetTypeInfo(type).IsAssignableFrom(IntrospectionExtensions.
      GetTypeInfo(obj.GetType()));
00461 #else
00462             return type.IsInstanceOfType(obj);
00463 #endif
00464         }
00465
00466         #region IsInterface
00467
00473 #if (NET45 || NET46 || PORTABLE)
00474         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00475 #endif
00476
00477         public static bool IsInterface(Type type)
00478         {
00479 #if PORTABLE
00480             return IntrospectionExtensions.GetTypeInfo(type).IsInterface;
00481 #else
```

```
00482             return type.IsInterface;
00483 #endif
00484         }
00485
00491 #if (NET45 || NET46 || PORTABLE)
00492         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00493 #endif
00494
00495         public static bool IsInterface<T>() => IsInterface(typeof(T));
00496
00497         #endregion IsInterface
00498
00499         #region IsPrimitive
00500
00506 #if (NET45 || NET46 || PORTABLE)
00507         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00508 #endif
00509
00510         public static bool IsPrimitive(Type type)
00511         {
00512 #if PORTABLE
00513             return IntrospectionExtensions.GetTypeInfo(type).IsPrimitive;
00514 #else
00515             return type.IsPrimitive;
00516 #endif
00517         }
00518
00524 #if (NET45 || NET46 || PORTABLE)
00525         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00526 #endif
00527
00528         public static bool IsPrimitive<T>() => IsPrimitive(typeof(T));
00529
00530         #endregion IsPrimitive
00531
00532         #region IsValueType
00533
00539 #if (NET45 || NET46 || PORTABLE)
00540         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00541 #endif
00542
00543         public static bool IsValueType(Type type)
00544         {
00545 #if PORTABLE
00546             return IntrospectionExtensions.GetTypeInfo(type).IsValueType;
00547 #else
00548             return type.IsValueType;
00549 #endif
00550         }
00551
00557 #if (NET45 || NET46 || PORTABLE)
00558         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
      MethodImplOptions.AggressiveInlining)]
00559 #endif
00560
00561         public static bool IsValueType<T>() => IsValueType(typeof(T));
00562
00563         #endregion IsValueType
00564     }
00565 }
```

7.57 ThrowerException.cs File Reference

Classes

- 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

7.58 ThrowerException.cs

```
00001 // File name: ThrowerException.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025 using System.Diagnostics.CodeAnalysis;
00026
00027 namespace PommaLabs.Thrower
00028 {
00033     [Serializable]
00034     [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
00035     public sealed class ThrowerException : Exception
00036     {
00037         [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
00038         private ThrowerException(string message)
00039             : base(message)
00040         {
00041         }
00042
00043         internal static ThrowerException AbstractEx => new ThrowerException("Given exception type is
      abstract");
00044
00045         internal static ThrowerException MissingNoArgsCtor => new ThrowerException("Given exception type
      has no parameterless constructor");
00046
00047         internal static ThrowerException MissingMsgCtor => new ThrowerException("Given exception type has
      not a valid message constructor");
00048     }
00049 }
```

7.59 Validation/EmailAddressValidator.cs File Reference

Classes

- class PommaLabs.Thrower.Validation.EmailAddressValidator

  An email address validator.

Namespaces

- namespace PommaLabs.Thrower.Validation

7.60 EmailAddressValidator.cs

```
00001 // File name: EmailAddressValidator.cs
00002 //
00003 // Author(s): Jeffrey Stedfast <jeff@xamarin.com>
00004 //
00005 // Copyright (c) 2013 Xamarin Inc.
00006 //
00007 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00008 // associated documentation files (the "Software"), to deal in the Software without restriction,
00009 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00010 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00011 // furnished to do so, subject to the following conditions:
00012 //
00013 // The above copyright notice and this permission notice shall be included in all copies or
00014 // substantial portions of the Software.
00015 //
00016 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00017 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00018 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00019 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00020 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00021
00022 using System;
00023
00024 namespace PommaLabs.Thrower.Validation
00025 {
00030     public static class EmailAddressValidator
00031     {
00032         private const string AtomCharacters = "!#$%&'*+-/=?^_`{|}~";
00033
00034         private static bool IsLetterOrDigit(char c)
00035         {
00036             return (c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z') || (c >= '0' && c <= '9');
00037         }
00038
00039         private static bool IsAtom(char c, bool allowInternational)
00040         {
00041             return c < 128 ? IsLetterOrDigit(c) || AtomCharacters.IndexOf(c) != -1 : allowInternational;
00042         }
00043
00044         private static bool IsDomain(char c, bool allowInternational)
00045         {
00046             return c < 128 ? IsLetterOrDigit(c) || c == '-' : allowInternational;
00047         }
00048
00049         private static bool SkipAtom(string text, ref int index, bool allowInternational)
00050         {
00051             var startIndex = index;
00052
00053             while (index < text.Length && IsAtom(text[index], allowInternational))
00054                 index++;
00055
00056             return index > startIndex;
00057         }
00058
00059         private static bool SkipSubDomain(string text, ref int index, bool allowInternational)
00060         {
00061             var startIndex = index;
00062
00063             if (!IsDomain(text[index], allowInternational) || text[index] == '-')
00064                 return false;
00065
00066             index++;
00067
00068             while (index < text.Length && IsDomain(text[index], allowInternational))
00069                 index++;
00070
00071             return (index - startIndex) < 64 && text[index - 1] != '-';
00072         }
00073
00074         private static bool SkipDomain(string text, ref int index, bool allowInternational, bool
      allowTopLevelDomains)
00075         {
00076             if (!SkipSubDomain(text, ref index, allowInternational))
00077                 return false;
00078
00079             if (index < text.Length && text[index] == '.')
00080             {
00081                 do
00082                 {
00083                     index++;
00084
00085                     if (index == text.Length)
00086                         return false;
00087
```

```
00088                   if (!SkipSubDomain(text, ref index, allowInternational))
00089                       return false;
00090               } while (index < text.Length && text[index] == '.');
00091           }
00092           else if (!allowTopLevelDomains)
00093           {
00094               return false;
00095           }
00096
00097           return true;
00098       }
00099
00100       private static bool SkipQuoted(string text, ref int index, bool allowInternational)
00101       {
00102           var escaped = false;
00103
00104           // skip over leading '"'
00105           index++;
00106
00107           while (index < text.Length)
00108           {
00109               if (text[index] >= 128 && !allowInternational)
00110                   return false;
00111
00112               if (text[index] == '\\')
00113               {
00114                   escaped = !escaped;
00115               }
00116               else if (!escaped)
00117               {
00118                   if (text[index] == '"')
00119                       break;
00120               }
00121               else
00122               {
00123                   escaped = false;
00124               }
00125
00126               index++;
00127           }
00128
00129           if (index >= text.Length || text[index] != '"')
00130               return false;
00131
00132           index++;
00133
00134           return true;
00135       }
00136
00137       private static bool SkipWord(string text, ref int index, bool allowInternational)
00138       {
00139           if (text[index] == '"')
00140               return SkipQuoted(text, ref index, allowInternational);
00141
00142           return SkipAtom(text, ref index, allowInternational);
00143       }
00144
00145       private static bool SkipIPv4Literal(string text, ref int index)
00146       {
00147           var groups = 0;
00148
00149           while (index < text.Length && groups < 4)
00150           {
00151               var startIndex = index;
00152               var value = 0;
00153
00154               while (index < text.Length && text[index] >= '0' && text[index] <= '9')
00155               {
00156                   value = (value * 10) + (text[index] - '0');
00157                   index++;
00158               }
00159
00160               if (index == startIndex || index - startIndex > 3 || value > 255)
00161                   return false;
00162
00163               groups++;
00164
00165               if (groups < 4 && index < text.Length && text[index] == '.')
00166                   index++;
00167           }
00168
00169           return groups == 4;
00170       }
00171
00172       private static bool IsHexDigit(char c)
00173       {
00174           return (c >= 'A' && c <= 'F') || (c >= 'a' && c <= 'f') || (c >= '0' && c <= '9');
```

```
00175         }
00176
00177         // This needs to handle the following forms:
00178         //
00179         // IPv6-addr = IPv6-full / IPv6-comp / IPv6v4-full / IPv6v4-comp IPv6-hex = 1*4HEXDIG
00180         // IPv6-full = IPv6-hex 7(":" IPv6-hex) IPv6-comp = [IPv6-hex *5(":" IPv6-hex)] "::"
00181         // [IPv6-hex *5(":" IPv6-hex)] ; The "::" represents at least 2 16-bit groups of zeros ; No
00182         // more than 6 groups in addition to the "::" may be ; present IPv6v4-full = IPv6-hex 5(":"
00183         // IPv6-hex) ":" IPv4-address-literal IPv6v4-comp = [IPv6-hex *3(":" IPv6-hex)] "::"
00184         // [IPv6-hex *3(":" IPv6-hex) ":"] IPv4-address-literal ; The "::" represents at least 2
00185         // 16-bit groups of zeros ; No more than 4 groups in addition to the "::" and ;
00186         // IPv4-address-literal may be present
00187         private static bool SkipIPv6Literal(string text, ref int index)
00188         {
00189             var compact = false;
00190             var colons = 0;
00191
00192             while (index < text.Length)
00193             {
00194                 var startIndex = index;
00195
00196                 while (index < text.Length && IsHexDigit(text[index]))
00197                     index++;
00198
00199                 if (index >= text.Length)
00200                     break;
00201
00202                 if (index > startIndex && colons > 2 && text[index] == '.')
00203                 {
00204                     // IPv6v4
00205                     index = startIndex;
00206
00207                     if (!SkipIPv4Literal(text, ref index))
00208                         return false;
00209
00210                     return compact ? colons < 6 : colons == 6;
00211                 }
00212
00213                 var count = index - startIndex;
00214                 if (count > 4)
00215                     return false;
00216
00217                 if (text[index] != ':')
00218                     break;
00219
00220                 startIndex = index;
00221                 while (index < text.Length && text[index] == ':')
00222                     index++;
00223
00224                 count = index - startIndex;
00225                 if (count > 2)
00226                     return false;
00227
00228                 if (count == 2)
00229                 {
00230                     if (compact)
00231                         return false;
00232
00233                     compact = true;
00234                     colons += 2;
00235                 }
00236                 else
00237                 {
00238                     colons++;
00239                 }
00240             }
00241
00242             if (colons < 2)
00243                 return false;
00244
00245             return compact ? colons < 7 : colons == 7;
00246         }
00247
00265         public static bool Validate(string emailAddress, Options options =
      Options.None)
00266         {
00267             var allowInternational = ((options & Options.AllowInternational) ==
      Options.AllowInternational);
00268             var allowTopLevelDomains = ((options & Options.AllowTopLevelDomains) ==
      Options.AllowTopLevelDomains);
00269
00270             var index = 0;
00271
00272             if (emailAddress == null)
00273                 throw new ArgumentNullException(nameof(emailAddress));
00274
00275             if (emailAddress.Length == 0 || emailAddress.Length >= 255)
```

```
00276                 return false;
00277
00278             if (!SkipWord(emailAddress, ref index, allowInternational) || index >= emailAddress.Length)
00279                 return false;
00280
00281             while (emailAddress[index] == '.')
00282             {
00283                 index++;
00284
00285                 if (index >= emailAddress.Length)
00286                     return false;
00287
00288                 if (!SkipWord(emailAddress, ref index, allowInternational))
00289                     return false;
00290
00291                 if (index >= emailAddress.Length)
00292                     return false;
00293             }
00294
00295             if (index + 1 >= emailAddress.Length || index > 64 || emailAddress[index++] != '@')
00296                 return false;
00297
00298             if (emailAddress[index] != '[')
00299             {
00300                 // domain
00301                 if (!SkipDomain(emailAddress, ref index, allowInternational, allowTopLevelDomains))
00302                     return false;
00303
00304                 return index == emailAddress.Length;
00305             }
00306
00307             // address literal
00308             index++;
00309
00310             // we need at least 8 more characters
00311             if (index + 8 >= emailAddress.Length)
00312                 return false;
00313
00314             var ipv6 = emailAddress.Substring(index, 5);
00315             if (ipv6.ToLowerInvariant() == "ipv6:")
00316             {
00317                 index += "IPv6:".Length;
00318                 if (!SkipIPv6Literal(emailAddress, ref index))
00319                     return false;
00320             }
00321             else
00322             {
00323                 if (!SkipIPv4Literal(emailAddress, ref index))
00324                     return false;
00325             }
00326
00327             if (index >= emailAddress.Length || emailAddress[index++] != ']')
00328                 return false;
00329
00330             return index == emailAddress.Length;
00331         }
00332
00336         [Flags]
00337         public enum Options
00338         {
00342             None = 0,
00343
00347             AllowInternational = 1,
00348
00352             AllowTopLevelDomains = 2
00353         }
00354     }
00355 }
```

7.61 Validation/ObjectValidator.cs File Reference

Classes

- class PommaLabs.Thrower.Validation.ObjectValidator

  Validates an object public properties that have been decorated with the ValidateAttribute custom attribute.

Namespaces

- namespace PommaLabs.Thrower.Validation

7.62 ObjectValidator.cs

```
00001 // File name: ObjectValidator.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using PommaLabs.Thrower.Reflection;
00025 using System;
00026 using System.Collections;
00027 using System.Collections.Generic;
00028 using System.Linq;
00029 using System.Text;
00030
00031 namespace PommaLabs.Thrower.Validation
00032 {
00037     public static class ObjectValidator
00038     {
00042         public const string RootPlaceholder = "$";
00043
00044         private static readonly ValidateAttribute DefaultValidation = new
      ValidateAttribute();
00045
00046         private static readonly HashSet<Type> AlwaysValidTypes = new HashSet<Type>
00047         {
00048             typeof(bool),
00049             typeof(char),
00050             typeof(byte),
00051             typeof(short),
00052             typeof(ushort),
00053             typeof(int),
00054             typeof(uint),
00055             typeof(long),
00056             typeof(ulong),
00057             typeof(float),
00058             typeof(double),
00059             typeof(decimal),
00060             typeof(string)
00061         };
00062
00069         public static string FormatValidationErrors(IEnumerable<ValidationError>
      validationErrors, string startMessage = null)
00070         {
00071             var builder = new StringBuilder();
00072             if (!string.IsNullOrEmpty(startMessage))
00073             {
00074                 builder.Append(startMessage);
00075                 builder.Append(" - ");
00076             }
00077             builder.AppendLine("Following paths failed the validation checks:");
00078             foreach (var ve in validationErrors)
00079             {
00080                 builder.AppendLine($" >> {ve.Path}: {ve.Reason}");
00081             }
00082             return builder.ToString();
00083         }
00084
00092         public static bool Validate(object obj, out IList<ValidationError> validationErrors)
00093         {
00094             // The list of errors which will be populated during the validation process.
00095             validationErrors = new List<ValidationError>();
00096
00097 #if (!NET35 && !PORTABLE)
00098
00099             // Applies standard .NET validation.
00100             var netValidationErrors = new List<System.ComponentModel.DataAnnotations.ValidationResult
      >();
00101             var netValidationContext = new System.ComponentModel.DataAnnotations.ValidationContext(
```

```
obj, null, null);
00102             if (!System.ComponentModel.DataAnnotations.Validator.TryValidateObject(obj,
netValidationContext, netValidationErrors, true))
00103             {
00104                 foreach (var netValidationError in netValidationErrors)
00105                 foreach (var memberName in netValidationError.MemberNames)
00106                 {
00107                     validationErrors.Add(new ValidationError
00108                     {
00109                         Path = $"{RootPlaceholder}.{memberName}",
00110                         Reason = netValidationError.ErrorMessage
00111                     });
00112                 }
00113             }
00114 
00115 #endif
00116 
00117             // Apply the final Thrower validation.
00118             return ValidateInternal(obj, RootPlaceholder, DefaultValidation, validationErrors);
00119         }
00120 
00121         private static bool ValidateInternal(object obj, string path,
ValidateAttribute validation, IList<ValidationError> validationErrors)
00122         {
00123             if (ReferenceEquals(obj, null))
00124             {
00125                 if (validation.Required)
00126                 {
00127                     validationErrors.Add(new ValidationError { Path = path, Reason = "
Property is required, found null" });
00128                     return false;
00129                 }
00130 
00131                 // If object is null, we cannot do anything else.
00132                 return true;
00133             }
00134 
00135             var objType = obj.GetType();
00136 
00137             if (AlwaysValidTypes.Contains(objType) || PortableTypeInfo.
IsEnum(objType))
00138             {
00139                 return true;
00140             }
00141 
00142             var isValueType = PortableTypeInfo.IsValueType(objType);
00143 
00144             // Check whether this type is nullable.
00145             if (validation.Required && isValueType && PortableTypeInfo.
IsGenericType(objType) && objType.GetGenericTypeDefinition() == typeof(Nullable<>))
00146             {
00147                 var nullableProps = PortableTypeInfo.
GetPublicProperties(objType);
00148                 var nullableHasValueProp = nullableProps.First(p => p.Name == nameof(Nullable<bool>.
HasValue));
00149 
00150                 if ((bool) PortableTypeInfo.
GetPublicPropertyValue(obj, nullableHasValueProp))
00151                 {
00152                     validationErrors.Add(new ValidationError { Path = path, Reason = "
Property is required, found null" });
00153                     return false;
00154                 }
00155 
00156                 var nullableValueProp = nullableProps.First(p => p.Name == nameof(Nullable<bool>.Value));
00157                 var nullableValue = PortableTypeInfo.
GetPublicPropertyValue(obj, nullableValueProp);
00158                 return ValidateInternal(nullableValue, path, validation, validationErrors);
00159             }
00160 
00161             var collection = obj as ICollection;
00162             if (collection != null)
00163             {
00164                 var c = collection.Count;
00165                 if (c < validation.CollectionItemsMinCount)
00166                 {
00167                     validationErrors.Add(new ValidationError { Path = path, Reason = $"
Minimum item count is {validation.CollectionItemsMinCount}, found {c}" });
00168                 }
00169                 if (c > validation.CollectionItemsMaxCount)
00170                 {
00171                     validationErrors.Add(new ValidationError { Path = path, Reason = $"
Maximum item count is {validation.CollectionItemsMaxCount}, found {c}" });
00172                 }
00173             }
00174 
00175             var enumerable = obj as IEnumerable;
```

```
00176             if (enumerable != null && validation.Enumerable)
00177             {
00178                 var itemValidation = new ValidateAttribute { Required = validation.
      EnumerableItemsRequired };
00179                 var index = 0;
00180                 foreach (var item in enumerable)
00181                 {
00182                     var indexedNewPath = $"{path}[{index++}]";
00183                     ValidateInternal(item, indexedNewPath, itemValidation, validationErrors);
00184                 }
00185             }
00186 
00187             if (PortableTypeInfo.IsClass(objType) || isValueType)
00188             {
00189                 var props = PortableTypeInfo.GetPublicProperties(objType
      );
00190                 var reqProps = from p in props
00191                                from a in PortableTypeInfo.
      GetCustomAttributes(p, false)
00192                                let v = a as ValidateAttribute
00193                                where v != null
00194                                select new { PropertyInfo = p, Validation = v };
00195 
00196 #if !PORTABLE
00197                 var typeAccessor = Reflection.FastMember.TypeAccessor.Create(objType);
00198 #endif
00199 
00200                 foreach (var rp in reqProps)
00201                 {
00202                     var propertyInfo = rp.PropertyInfo;
00203 
00204 #if PORTABLE
00205                     var propertyValue = PortableTypeInfo.
      GetPublicPropertyValue(obj, propertyInfo);
00206 #else
00207                     var propertyValue = PortableTypeInfo.
      GetPublicPropertyValue(typeAccessor, obj, propertyInfo);
00208 #endif
00209 
00210                     var newPath = $"{path}.{propertyInfo.Name}";
00211                     ValidateInternal(propertyValue, newPath, rp.Validation, validationErrors);
00212                 }
00213 
00214                 return validationErrors.Count == 0;
00215             }
00216 
00217             // Non dovrei mai finire qui!
00218             return true;
00219         }
00220     }
00221 }
```

7.63 Validation/PhoneNumberValidator.cs File Reference

Classes

- class PommaLabs.Thrower.Validation.PhoneNumberValidator

  A phone number validator.

Namespaces

- namespace PommaLabs.Thrower.Validation

7.64 PhoneNumberValidator.cs

```
00001 // Taken from:
       http://referencesource.microsoft.com/#System.ComponentModel.DataAnnotations/DataAnnotations/PhoneAttribute.cs
00002 
00003 using System.Text.RegularExpressions;
00004 
```

```
00005 namespace PommaLabs.Thrower.Validation
00006 {
00011     public static class PhoneNumberValidator
00012     {
00013         private static readonly Regex PhoneNumberRegex = CreatePhoneNumberRegex();
00014 
00020         public static bool Validate(string phoneNumber)
00021         {
00022             // Preconditions
00023             Raise.ArgumentException.IfIsNullOrWhiteSpace(phoneNumber, nameof(phoneNumber));
00024 
00025             return PhoneNumberRegex.IsMatch(phoneNumber);
00026         }
00027 
00028         private static Regex CreatePhoneNumberRegex()
00029         {
00030             const string pattern = @"
      ^(\+\s?)?((?<!\+.*)\(\+?\d+([\s\-\/\.]?\d+)?\)|\d+)([\s\-\/\.]?(\(\d+([\s\-\/\.]?\d+)?\)|\d+))*(\s?(x|ext\.?)\s?\d+)?$";
00031 #if PORTABLE
00032             const RegexOptions options = RegexOptions.IgnoreCase | RegexOptions.ExplicitCapture;
00033 #else
00034             const RegexOptions options = RegexOptions.Compiled | RegexOptions.IgnoreCase | RegexOptions.
      ExplicitCapture;
00035 #endif
00036             return new Regex(pattern, options);
00037         }
00038     }
00039 }
```

7.65 Validation/ValidateAttribute.cs File Reference

Classes

- class PommaLabs.Thrower.Validation.ValidateAttribute

  Indicates that the property should be validated.

Namespaces

- namespace PommaLabs.Thrower.Validation

7.66 ValidateAttribute.cs

```
00001 // File name: ValidateAttribute.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023 
00024 using System;
00025 using System.Collections;
00026 
00027 namespace PommaLabs.Thrower.Validation
00028 {
```

```
00032     [AttributeUsage(AttributeTargets.Property, AllowMultiple = false, Inherited = false)]
00033     public sealed class ValidateAttribute : Attribute
00034     {
00040         public bool Required { get; set; } = false;
00041
00048         public bool Enumerable { get; set; } = true;
00049
00056         public bool EnumerableItemsRequired { get; set; } = false;
00057
00068         public long CollectionItemsMinCount { get; set; } = 0L;
00069
00076         public long CollectionItemsMaxCount { get; set; } = long.MaxValue;
00077     }
00078 }
```

7.67 Validation/ValidationError.cs File Reference

Classes

- struct PommaLabs.Thrower.Validation.ValidationError

  Represents an error found while validating an object.

Namespaces

- namespace PommaLabs.Thrower.Validation

7.68 ValidationError.cs

```
00001 // File name: ValidationError.cs
00002 //
00003 // Author(s): Alessio Parma <alessio.parma@gmail.com>
00004 //
00005 // The MIT License (MIT)
00006 //
00007 // Copyright (c) 2013-2016 Alessio Parma <alessio.parma@gmail.com>
00008 //
00009 // Permission is hereby granted, free of charge, to any person obtaining a copy of this software and
00010 // associated documentation files (the "Software"), to deal in the Software without restriction,
00011 // including without limitation the rights to use, copy, modify, merge, publish, distribute,
00012 // sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is
00013 // furnished to do so, subject to the following conditions:
00014 //
00015 // The above copyright notice and this permission notice shall be included in all copies or
00016 // substantial portions of the Software.
00017 //
00018 // THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
00019 // NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND
00020 // NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
00021 // DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT
00022 // OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
00023
00024 using System;
00025
00026 namespace PommaLabs.Thrower.Validation
00027 {
00031     [Serializable]
00032     public struct ValidationError
00033     {
00037         public string Path { get; set; }
00038
00042         public string Reason { get; set; }
00043     }
00044 }
```

Index