

PommaLabs.Thrower
2.2.1

Generated by Doxygen 1.8.10

Fri Feb 26 2016 11:56:17

Contents

1	Namespace Index	1
1.1	Packages	1
2	Hierarchical Index	3
2.1	Class Hierarchy	3
3	Class Index	5
3.1	Class List	5
4	File Index	7
4.1	File List	7
5	Namespace Documentation	9
5.1	PommaLabs Namespace Reference	9
5.2	PommaLabs.Thrower Namespace Reference	9
6	Class Documentation	11
6.1	PommaLabs.Thrower.HttpException Class Reference	11
6.1.1	Detailed Description	12
6.1.2	Constructor & Destructor Documentation	12
6.1.2.1	HttpException(HttpStatusCode httpStatusCode)	12
6.1.2.2	HttpException(HttpStatusCode httpStatusCode, HttpExceptionInfo additionalInfo)	12
6.1.2.3	HttpException(HttpStatusCode httpStatusCode, string message)	13
6.1.2.4	HttpException(HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)	13
6.1.2.5	HttpException(HttpStatusCode httpStatusCode, string message, Exception innerException)	13
6.1.2.6	HttpException(HttpStatusCode httpStatusCode, string message, Exception innerException, HttpExceptionInfo additionalInfo)	13
6.1.3	Property Documentation	13
6.1.3.1	DefaultErrorCode	14
6.1.3.2	DefaultUserMessage	14
6.1.3.3	ErrorCode	14
6.1.3.4	HttpStatusCode	14

6.1.3.5	UserMessage	14
6.2	PommaLabs.Thrower.HttpExceptionInfo Struct Reference	14
6.2.1	Detailed Description	14
6.2.2	Constructor & Destructor Documentation	15
6.2.2.1	HttpExceptionInfo(object errorCode=null, string userMessage=null)	15
6.2.3	Property Documentation	15
6.2.3.1	ErrorCode	15
6.2.3.2	UserMessage	15
6.3	PommaLabs.Thrower.Raise< TEx > Class Template Reference	15
6.3.1	Detailed Description	19
6.3.2	Member Function Documentation	19
6.3.2.1	If(bool cond)	19
6.3.2.2	If(bool cond, string message)	20
6.3.2.3	IfAreEqual< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2)	21
6.3.2.4	IfAreEqual< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, string message)	21
6.3.2.5	IfAreNotEqual< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2)	22
6.3.2.6	IfAreNotEqual< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, string message)	22
6.3.2.7	IfAreNotSame< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2)	22
6.3.2.8	IfAreNotSame< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, string message)	23
6.3.2.9	IfAreSame< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2)	24
6.3.2.10	IfAreSame< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, string message)	24
6.3.2.11	IfIsAssignableFrom(object instance, Type type)	25
6.3.2.12	IfIsAssignableFrom(object instance, Type type, string message)	25
6.3.2.13	IfIsAssignableFrom< TType >(object instance)	25
6.3.2.14	IfIsAssignableFrom< TType >(object instance, string message)	26
6.3.2.15	IfIsContainedIn(object argument, System.Collections.IList collection)	26
6.3.2.16	IfIsContainedIn(object argument, System.Collections.IList collection, string message)	27
6.3.2.17	IfIsContainedIn< TArg >(TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)	27
6.3.2.18	IfIsContainedIn< TArg >(TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)	27
6.3.2.19	IfIsContainedIn< TArg >(TArg arg, System.Collections.IDictionary dictionary)	28
6.3.2.20	IfIsContainedIn< TArg >(TArg arg, System.Collections.IDictionary dictionary, string message)	28
6.3.2.21	IfIsContainedIn< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary)	29
6.3.2.22	IfIsContainedIn< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary, string message)	30
6.3.2.23	IfIsEmpty(string valueToCheck)	30
6.3.2.24	IfIsEmpty(string valueToCheck, string message)	31
6.3.2.25	IfIsEmpty(System.Collections.ICollection collection)	32

6.3.2.26	<code>IfIsEmpty(System.Collections.ICollection collection, string message)</code>	32
6.3.2.27	<code>IfIsEmpty< TArg >(System.Collections.Generic.IEnumerable< TArg > collection)</code>	33
6.3.2.28	<code>IfIsEmpty< TArg >(System.Collections.Generic.IEnumerable< TArg > collection, string message)</code>	34
6.3.2.29	<code>IfInstanceOf(object instance, Type type)</code>	34
6.3.2.30	<code>IfInstanceOf(object instance, Type type, string message)</code>	34
6.3.2.31	<code>IfInstanceOf< TType >(object instance)</code>	35
6.3.2.32	<code>IfInstanceOf< TType >(object instance, string message)</code>	35
6.3.2.33	<code>IfNaN(double number)</code>	36
6.3.2.34	<code>IfNaN(double number, string message)</code>	36
6.3.2.35	<code>IfNotAssignableFrom(object instance, Type type)</code>	36
6.3.2.36	<code>IfNotAssignableFrom(object instance, Type type, string message)</code>	37
6.3.2.37	<code>IfNotAssignableFrom< TType >(object instance)</code>	37
6.3.2.38	<code>IfNotAssignableFrom< TType >(object instance, string message)</code>	37
6.3.2.39	<code>IfNotContainedIn(object argument, System.Collections.IList collection)</code>	38
6.3.2.40	<code>IfNotContainedIn(object argument, System.Collections.IList collection, string message)</code>	38
6.3.2.41	<code>IfNotContainedIn< TArg >(TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)</code>	39
6.3.2.42	<code>IfNotContainedIn< TArg >(TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)</code>	39
6.3.2.43	<code>IfNotContainedIn< TArg >(TArg arg, System.Collections.IDictionary dictionary)</code>	39
6.3.2.44	<code>IfNotContainedIn< TArg >(TArg arg, System.Collections.IDictionary dictionary, string message)</code>	40
6.3.2.45	<code>IfNotContainedIn< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary)</code>	40
6.3.2.46	<code>IfNotContainedIn< TArg1, TArg2 >(TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary, string message)</code>	41
6.3.2.47	<code>IfNotEmpty(string valueToCheck)</code>	42
6.3.2.48	<code>IfNotEmpty(string valueToCheck, string message)</code>	42
6.3.2.49	<code>IfNotEmpty(System.Collections.ICollection collection)</code>	43
6.3.2.50	<code>IfNotEmpty(System.Collections.ICollection collection, string message)</code>	43
6.3.2.51	<code>IfNotEmpty< TArg >(System.Collections.Generic.IEnumerable< TArg > collection)</code>	43
6.3.2.52	<code>IfNotEmpty< TArg >(System.Collections.Generic.IEnumerable< TArg > collection, string message)</code>	44
6.3.2.53	<code>IfNotInstanceOf(object instance, Type type)</code>	44
6.3.2.54	<code>IfNotInstanceOf(object instance, Type type, string message)</code>	44
6.3.2.55	<code>IfNotInstanceOf< TType >(object instance)</code>	45
6.3.2.56	<code>IfNotInstanceOf< TType >(object instance, string message)</code>	45
6.3.2.57	<code>IfNotNaN(double number)</code>	46
6.3.2.58	<code>IfNotNaN(double number, string message)</code>	46
6.3.2.59	<code>IfNotNull< TArg >(TArg arg)</code>	46

6.3.2.60	IfsNotNull< TArg >(TArg arg, string message)	47
6.3.2.61	IfsNull< TArg >(TArg arg)	47
6.3.2.62	IfsNull< TArg >(TArg arg, string message)	47
6.3.2.63	IfNot(bool cond)	48
6.3.2.64	IfNot(bool cond, string message)	48
6.4	PommaLabs.Thrower.RaiseArgumentException Class Reference	48
6.4.1	Detailed Description	50
6.4.2	Member Function Documentation	50
6.4.2.1	If(bool condition)	50
6.4.2.2	If(bool condition, string argumentName, string message=null)	50
6.4.2.3	IfsNotValid< TArg >(TArg argument)	50
6.4.2.4	IfsNotValid< TArg >(TArg argument, string argumentName, string message=null)	51
6.4.2.5	IfsNotValidEmailAddress(string emailAddress)	51
6.4.2.6	IfsNotValidEmailAddress(string emailAddress, bool allowInternational)	51
6.4.2.7	IfsNotValidEmailAddress(string emailAddress, string argumentName, string message=null)	51
6.4.2.8	IfsNotValidEmailAddress(string emailAddress, bool allowInternational, string argumentName, string message=null)	52
6.4.2.9	IfsNotValidPhoneNumber(string phoneNumber)	52
6.4.2.10	IfsNotValidPhoneNumber(string phoneNumber, string argumentName, string message=null)	52
6.4.2.11	IfsNullOrEmpty(string value)	52
6.4.2.12	IfsNullOrEmpty(string value, string argumentName, string message=null)	53
6.4.2.13	IfsNullOrWhiteSpace(string value)	53
6.4.2.14	IfsNullOrWhiteSpace(string value, string argumentName, string message=null)	53
6.4.2.15	IfNot(bool condition)	53
6.4.2.16	IfNot(bool condition, string argumentName, string message=null)	53
6.5	PommaLabs.Thrower.RaiseArgumentNullException Class Reference	54
6.5.1	Detailed Description	55
6.5.2	Member Function Documentation	55
6.5.2.1	IfsNull< TArg >(TArg argument)	55
6.5.2.2	IfsNull< TArg >(TArg argument, string argumentName)	55
6.5.2.3	IfsNull< TArg >(TArg argument, string argumentName, string message)	55
6.6	PommaLabs.Thrower.RaiseArgumentOutOfRangeException Class Reference	56
6.6.1	Detailed Description	58
6.6.2	Member Function Documentation	58
6.6.2.1	If(bool condition, string argumentName=null)	58
6.6.2.2	If(bool condition, string argumentName, string message)	59
6.6.2.3	IfsEqual(Comparable argument1, Comparable argument2)	59
6.6.2.4	IfsEqual(Comparable argument1, Comparable argument2, string argumentName)	59

6.6.2.5	<code>IfIsEqual(IComparable argument1, IComparable argument2, string argument← Name, string message)</code>	59
6.6.2.6	<code>IfIsEqual< TArg >(TArg argument1, TArg argument2)</code>	60
6.6.2.7	<code>IfIsEqual< TArg >(TArg argument1, TArg argument2, string argumentName)</code> . .	60
6.6.2.8	<code>IfIsEqual< TArg >(TArg argument1, TArg argument2, string argumentName, string message)</code>	60
6.6.2.9	<code>IfIsGreater(IComparable argument1, IComparable argument2)</code>	61
6.6.2.10	<code>IfIsGreater(IComparable argument1, IComparable argument2, string argument← Name)</code>	61
6.6.2.11	<code>IfIsGreater(IComparable argument1, IComparable argument2, string argument← Name, string message)</code>	61
6.6.2.12	<code>IfIsGreater< TArg >(TArg argument1, TArg argument2)</code>	61
6.6.2.13	<code>IfIsGreater< TArg >(TArg argument1, TArg argument2, string argumentName)</code> .	62
6.6.2.14	<code>IfIsGreater< TArg >(TArg argument1, TArg argument2, string argumentName, string message)</code>	62
6.6.2.15	<code>IfIsGreaterOrEqual(IComparable argument1, IComparable argument2)</code>	63
6.6.2.16	<code>IfIsGreaterOrEqual(IComparable argument1, IComparable argument2, string argumentName)</code>	64
6.6.2.17	<code>IfIsGreaterOrEqual(IComparable argument1, IComparable argument2, string argumentName, string message)</code>	64
6.6.2.18	<code>IfIsGreaterOrEqual< TArg >(TArg argument1, TArg argument2)</code>	64
6.6.2.19	<code>IfIsGreaterOrEqual< TArg >(TArg argument1, TArg argument2, string argumentName)</code>	64
6.6.2.20	<code>IfIsGreaterOrEqual< TArg >(TArg argument1, TArg argument2, string argumentName, string message)</code>	65
6.6.2.21	<code>IfIsLess(IComparable argument1, IComparable argument2)</code>	65
6.6.2.22	<code>IfIsLess(IComparable argument1, IComparable argument2, string argumentName)</code>	65
6.6.2.23	<code>IfIsLess(IComparable argument1, IComparable argument2, string argument← Name, string message)</code>	66
6.6.2.24	<code>IfIsLess< TArg >(TArg argument1, TArg argument2)</code>	66
6.6.2.25	<code>IfIsLess< TArg >(TArg argument1, TArg argument2, string argumentName)</code> . .	66
6.6.2.26	<code>IfIsLess< TArg >(TArg argument1, TArg argument2, string argumentName, string message)</code>	67
6.6.2.27	<code>IfIsLessOrEqual(IComparable argument1, IComparable argument2)</code>	67
6.6.2.28	<code>IfIsLessOrEqual(IComparable argument1, IComparable argument2, string argumentName)</code>	67
6.6.2.29	<code>IfIsLessOrEqual(IComparable argument1, IComparable argument2, string argumentName, string message)</code>	67
6.6.2.30	<code>IfIsLessOrEqual< TArg >(TArg argument1, TArg argument2)</code>	68
6.6.2.31	<code>IfIsLessOrEqual< TArg >(TArg argument1, TArg argument2, string argument← Name)</code>	68
6.6.2.32	<code>IfIsLessOrEqual< TArg >(TArg argument1, TArg argument2, string argument← Name, string message)</code>	68
6.6.2.33	<code>IfIsNotEqual(IComparable argument1, IComparable argument2)</code>	69

6.6.2.34	IfsNotEqual(Comparable argument1, Comparable argument2, string argumentName)	69
6.6.2.35	IfsNotEqual(Comparable argument1, Comparable argument2, string argumentName, string message)	69
6.6.2.36	IfsNotEqual< TArg >(TArg argument1, TArg argument2)	69
6.6.2.37	IfsNotEqual< TArg >(TArg argument1, TArg argument2, string argumentName)	70
6.6.2.38	IfsNotEqual< TArg >(TArg argument1, TArg argument2, string argumentName, string message)	70
6.6.2.39	IfNot(bool condition, string argumentName=null)	71
6.6.2.40	IfNot(bool condition, string argumentName, string message)	72
6.7	PommaLabs.Thrower.RaiseBase Class Reference	72
6.7.1	Detailed Description	73
6.7.2	Member Data Documentation	73
6.7.2.1	NoCtorTypes	73
6.7.2.2	StrCtorType	74
6.7.2.3	StrExCtorTypes	74
6.8	PommaLabs.Thrower.RaiseHttpException Class Reference	74
6.8.1	Detailed Description	74
6.8.2	Member Function Documentation	74
6.8.2.1	If(bool condition, HttpStatusCode httpStatusCode, string message=null)	74
6.8.2.2	If(bool condition, HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)	75
6.8.2.3	IfNot(bool condition, HttpStatusCode httpStatusCode, string message=null)	75
6.8.2.4	IfNot(bool condition, HttpStatusCode httpStatusCode, string message, HttpExceptionInfo additionalInfo)	75
6.9	PommaLabs.Thrower.RaiseIndexOutOfRangeException Class Reference	75
6.9.1	Detailed Description	77
6.9.2	Member Function Documentation	77
6.9.2.1	IfsEqual(Comparable argument1, Comparable argument2)	77
6.9.2.2	IfsEqual(Comparable argument1, Comparable argument2, string message)	78
6.9.2.3	IfsEqual< TArg >(TArg argument1, TArg argument2)	78
6.9.2.4	IfsEqual< TArg >(TArg argument1, TArg argument2, string message)	78
6.9.2.5	IfsGreater(Comparable argument1, Comparable argument2)	79
6.9.2.6	IfsGreater(Comparable argument1, Comparable argument2, string message)	80
6.9.2.7	IfsGreater< TArg >(TArg argument1, TArg argument2)	80
6.9.2.8	IfsGreater< TArg >(TArg argument1, TArg argument2, string message)	80
6.9.2.9	IfsGreaterOrEqual(Comparable argument1, Comparable argument2)	81
6.9.2.10	IfsGreaterOrEqual(Comparable argument1, Comparable argument2, string message)	82
6.9.2.11	IfsGreaterOrEqual< TArg >(TArg argument1, TArg argument2)	82
6.9.2.12	IfsGreaterOrEqual< TArg >(TArg argument1, TArg argument2, string message)	82
6.9.2.13	IfsLess(Comparable argument1, Comparable argument2)	83

6.9.2.14	IfsLess(IComparable argument1, IComparable argument2, string message)	84
6.9.2.15	IfsLess< TArg >(TArg argument1, TArg argument2)	84
6.9.2.16	IfsLess< TArg >(TArg argument1, TArg argument2, string message)	84
6.9.2.17	IfsLessOrEqual(IComparable argument1, IComparable argument2)	85
6.9.2.18	IfsLessOrEqual(IComparable argument1, IComparable argument2, string message)	86
6.9.2.19	IfsLessOrEqual< TArg >(TArg argument1, TArg argument2)	86
6.9.2.20	IfsLessOrEqual< TArg >(TArg argument1, TArg argument2, string message)	86
6.9.2.21	IfsNotEqual(IComparable argument1, IComparable argument2)	87
6.9.2.22	IfsNotEqual(IComparable argument1, IComparable argument2, string message)	88
6.9.2.23	IfsNotEqual< TArg >(TArg argument1, TArg argument2)	88
6.9.2.24	IfsNotEqual< TArg >(TArg argument1, TArg argument2, string message)	88
6.10	PommaLabs.Thrower.RaiseInvalidOperationException Class Reference	89
6.10.1	Detailed Description	89
6.10.2	Member Function Documentation	90
6.10.2.1	If(bool condition, string message=null)	90
6.10.2.2	IfNot(bool condition, string message=null)	90
6.11	PommaLabs.Thrower.RaiseNotSupportedException Class Reference	90
6.11.1	Detailed Description	91
6.11.2	Member Function Documentation	91
6.11.2.1	If(bool condition, string message=null)	91
6.11.2.2	IfNot(bool condition, string message=null)	91
6.12	PommaLabs.Thrower.RaiseObjectDisposedException Class Reference	92
6.12.1	Detailed Description	93
6.12.2	Member Function Documentation	93
6.12.2.1	If(bool disposed, string objectName, string message=null)	93
6.13	PommaLabs.Thrower.ThrowerException Class Reference	93
6.13.1	Detailed Description	94
7	File Documentation	95
7.1	Raise.cs File Reference	95
7.2	Raise.cs	95
7.3	RaiseArgumentException.cs File Reference	108
7.4	RaiseArgumentException.cs	108
7.5	RaiseArgumentNullException.cs File Reference	111
7.6	RaiseArgumentNullException.cs	111
7.7	RaiseArgumentOutOfRangeException.cs File Reference	112
7.8	RaiseArgumentOutOfRangeException.cs	113
7.9	RaiseHttpException.cs File Reference	120
7.10	RaiseHttpException.cs	121

7.11 RaiseIndexOutOfRangeException.cs File Reference	123
7.12 RaiseIndexOutOfRangeException.cs	123
7.13 RaiseInvalidOperationException.cs File Reference	128
7.14 RaiseInvalidOperationException.cs	128
7.15 RaiseNotSupportedException.cs File Reference	129
7.16 RaiseNotSupportedException.cs	129
7.17 RaiseObjectDisposedException.cs File Reference	130
7.18 RaiseObjectDisposedException.cs	130
 Index	 131

Chapter 1

Namespace Index

1.1 Packages

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

PommaLabs	9
PommaLabs.Thrower	9

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Exception	
PommaLabs.Thrower.HttpException	11
PommaLabs.Thrower.ThrowerException	93
PommaLabs.Thrower.HttpExceptionInfo	14
PommaLabs.Thrower.RaiseBase	72
PommaLabs.Thrower.Raise< TEx >	15
PommaLabs.Thrower.RaiseArgumentException	48
PommaLabs.Thrower.RaiseArgumentNullException	54
PommaLabs.Thrower.RaiseArgumentOutOfRangeException	56
PommaLabs.Thrower.RaiseIndexOutOfRangeException	75
PommaLabs.Thrower.RaiseInvalidOperationException	89
PommaLabs.Thrower.RaiseNotSupportedException	90
PommaLabs.Thrower.RaiseObjectDisposedException	92
PommaLabs.Thrower.RaiseHttpException	74

Chapter 3

Class Index

3.1 Class List

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

PommaLabs.Thrower.HttpException	Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.	11
PommaLabs.Thrower.HttpExceptionInfo	Additional info which will be included into HttpException	14
PommaLabs.Thrower.Raise< TEx >	Contains methods that throw specified exception <i>TEx</i> if given conditions will be verified.	15
PommaLabs.Thrower.RaiseArgumentException	Utility methods which can be used to handle bad arguments.	48
PommaLabs.Thrower.RaiseArgumentNullException	Utility methods which can be used to handle null references.	54
PommaLabs.Thrower.RaiseArgumentOutOfRangeException	Utility methods which can be used to handle ranges.	56
PommaLabs.Thrower.RaiseBase	Stores items shared by various Raise<TEx> instances.	72
PommaLabs.Thrower.RaiseHttpException	Utility methods which can be used to handle error codes through HTTP.	74
PommaLabs.Thrower.RaiseIndexOutOfRangeException	Utility methods which can be used to handle indexes.	75
PommaLabs.Thrower.RaiseInvalidOperationException	Utility methods which can be used to handle bad object states.	89
PommaLabs.Thrower.RaiseNotSupportedException	Utility methods which can be used to handle unsupported operations.	90
PommaLabs.Thrower.RaiseObjectDisposedException	Utility methods which can be used to handle bad object states.	92
PommaLabs.Thrower.ThrowerException	Exception thrown by Raise<TEx> when the type parameter passed to that class has something invalid (missing constructors, etc).	93

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Raise.cs	95
RaiseArgumentException.cs	108
RaiseArgumentNullException.cs	111
RaiseArgumentOutOfRangeException.cs	112
RaiseHttpException.cs	120
RaiseIndexOutOfRangeException.cs	123
RaiseInvalidOperationException.cs	128
RaiseNotSupportedException.cs	129
RaiseObjectDisposedException.cs	130

Chapter 5

Namespace Documentation

5.1 PommaLabs Namespace Reference

Namespaces

- namespace [Thrower](#)

5.2 PommaLabs.Thrower Namespace Reference

Classes

- class [HttpException](#)
Represents an exception which contains an error message that should be delivered through the HTTP response, using given status code.
- struct [HttpExceptionInfo](#)
Additional info which will be included into [HttpException](#).
- class [Raise](#)
Contains methods that throw specified exception TEx if given conditions will be verified.
- class [RaiseArgumentException](#)
Utility methods which can be used to handle bad arguments.
- class [RaiseArgumentNullException](#)
Utility methods which can be used to handle null references.
- class [RaiseArgumentOutOfRangeException](#)
Utility methods which can be used to handle ranges.
- class [RaiseBase](#)
Stores items shared by various `Raise<TEx>` instances.
- class [RaiseHttpException](#)
Utility methods which can be used to handle error codes through HTTP.
- class [RaiseIndexOutOfRangeException](#)
Utility methods which can be used to handle indexes.
- class [RaiseInvalidOperationException](#)
Utility methods which can be used to handle bad object states.
- class [RaiseNotSupportedException](#)
Utility methods which can be used to handle unsupported operations.
- class [RaiseObjectDisposedException](#)
Utility methods which can be used to handle bad object states.
- class [ThrowerException](#)

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

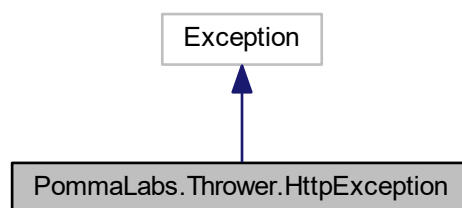
Chapter 6

Class Documentation

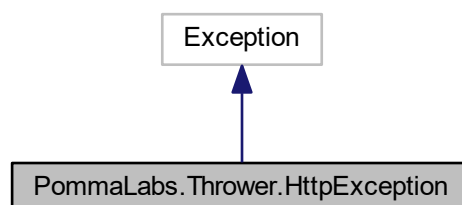
6.1 PommaLabs.Thrower.HttpException Class Reference

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

Inheritance diagram for PommaLabs.Thrower.HttpException:



Collaboration diagram for PommaLabs.Thrower.HttpException:



Public Member Functions

- [HttpException](#) ([HttpStatusCode](#) httpStatusCode)
Builds the exception using given status code.
- [HttpException](#) ([HttpStatusCode](#) httpStatusCode, [HttpExceptionInfo](#) additionalInfo)
Builds the exception using given status code.
- [HttpException](#) ([HttpStatusCode](#) httpStatusCode, string message)
Builds the exception using given status code and message.
- [HttpException](#) ([HttpStatusCode](#) httpStatusCode, string message, [HttpExceptionInfo](#) additionalInfo)
Builds the exception using given status code, message and error code.
- [HttpException](#) ([HttpStatusCode](#) httpStatusCode, string message, Exception innerException)
Builds the exception using given status code, message and inner exception.
- [HttpException](#) ([HttpStatusCode](#) httpStatusCode, string message, Exception innerException, [HttpExceptionInfo](#) additionalInfo)
Builds the exception using given status code, message, error code and inner exception.

Properties

- HttpStatusCode [HttpStatusCode](#) [get]
The HTTP status code assigned to this exception.
- object [ErrorCode](#) [get]
The application defined error code.
- static object [DefaultErrorCode](#) [get, set]
The default application defined error code, used when none has been specified.
- string [UserMessage](#) = "unspecified" [get]
An error message which can be shown to the user.
- static string [DefaultUserMessage](#) [get, set]
The default user message.

6.1.1 Detailed Description

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

Definition at line 143 of file [RaiseHttpException.cs](#).

6.1.2 Constructor & Destructor Documentation

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

Builds the exception using given status code.

Parameters

<i>httpStatusCode</i>	The HTTP status code.
-----------------------	-----------------------

Definition at line 149 of file [RaiseHttpException.cs](#).

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

Builds the exception using given status code.

Parameters

<i>httpStatusCode</i>	The HTTP status code.
<i>additionalInfo</i>	Additional exception info.

Definition at line 159 of file [RaiseHttpException.cs](#).

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

Builds the exception using given status code and message.

Parameters

<i>httpStatusCode</i>	The HTTP status code.
<i>message</i>	The exception message.

Definition at line 172 of file [RaiseHttpException.cs](#).

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

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

Parameters

<i>httpStatusCode</i>	The HTTP status code.
<i>message</i>	The exception message.
<i>additionalInfo</i>	Additional exception info.

Definition at line 183 of file [RaiseHttpException.cs](#).

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

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

Parameters

<i>httpStatusCode</i>	The HTTP status code.
<i>message</i>	The exception message.
<i>innerException</i>	The inner exception.

Definition at line 197 of file [RaiseHttpException.cs](#).

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

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

Parameters

<i>httpStatusCode</i>	The HTTP status code.
<i>message</i>	The exception message.
<i>innerException</i>	The inner exception.
<i>additionalInfo</i>	Additional exception info.

Definition at line 209 of file [RaiseHttpException.cs](#).

6.1.3 Property Documentation

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

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

Definition at line 230 of file [RaiseHttpException.cs](#).

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

The default user message.

Definition at line 240 of file [RaiseHttpException.cs](#).

6.1.3.3 object PommaLabs.Throwable.HttpException.ErrorCode [get]

The application defined error code.

Definition at line 225 of file [RaiseHttpException.cs](#).

6.1.3.4 HttpStatusCode PommaLabs.Throwable.HttpException.HttpStatusCode [get]

The HTTP status code assigned to this exception.

Definition at line 220 of file [RaiseHttpException.cs](#).

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

An error message which can be shown to the user.

Definition at line 235 of file [RaiseHttpException.cs](#).

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

- [RaiseHttpException.cs](#)

6.2 PommaLabs.Throwable.HttpExceptionInfo Struct Reference

Additional info which will be included into [HttpException](#).

Public Member Functions

- [HttpExceptionInfo](#) (object errorCode=null, string userMessage=null)
Builds the additional exception info.

Properties

- object [ErrorCode](#) [get, set]
The application defined error code.
- string [UserMessage](#) [get, set]
An error message which can be shown to user.

6.2.1 Detailed Description

Additional info which will be included into [HttpException](#).

Definition at line 113 of file [RaiseHttpException.cs](#).

6.2.2 Constructor & Destructor Documentation

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

Builds the additional exception info.

Parameters

<i>errorCode</i>	The application defined error code.
<i>userMessage</i>	The user message.

Definition at line 120 of file [RaiseHttpException.cs](#).

6.2.3 Property Documentation

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

The application defined error code.

Definition at line 130 of file [RaiseHttpException.cs](#).

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

An error message which can be shown to user.

Definition at line 136 of file [RaiseHttpException.cs](#).

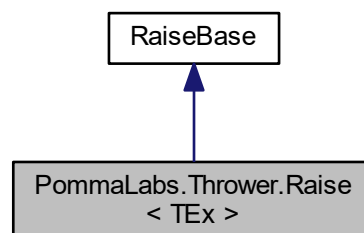
The documentation for this struct was generated from the following file:

- [RaiseHttpException.cs](#)

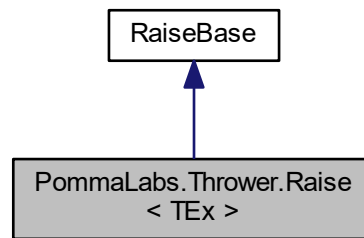
6.3 PommaLabs.Thrower.Raise< TEx > Class Template Reference

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

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



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



Static Public Member Functions

- static void `If` (bool cond)
Throws an exception of type TEx if and only if specified condition is true.
- static void `If` (bool cond, string message)
Throws an exception of type TEx with given message message if and only if specified condition is true.
- static void `IfNot` (bool cond)
Throws an exception of type TEx if and only if specified condition is false.
- static void `IfNot` (bool cond, string message)
Throws an exception of type TEx with given message message if and only if specified condition is false.
- static void `IfAreEqual`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)
Throws an exception of type TEx if and only if specified arguments are equal.
- static void `IfAreEqual`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)
Throws an exception of type TEx with given message message if and only if specified arguments are equal.
- static void `IfAreNotEqual`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)
Throws an exception of type TEx if and only if specified arguments are not equal.
- static void `IfAreNotEqual`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)
Throws an exception of type TEx with given message message if and only if specified arguments are not equal.
- static void `IfAreSame`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)
Throws an exception of type TEx if and only if specified arguments point to the same object.
- static void `IfAreSame`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)
Throws an exception of type TEx with given message message if and only if specified arguments point to the same object.
- static void `IfAreNotSame`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2)
Throws an exception of type TEx if and only if specified arguments do not point to the same object.
- static void `IfAreNotSame`< TArg1, TArg2 > (TArg1 arg1, TArg2 arg2, string message)
Throws an exception of type TEx with given message message if and only if specified arguments do not point to the same object.
- static void `IfIsAssignableFrom` (object instance, Type type)
Throws an exception of type TEx if and only if an instance of given type can be assigned to specified object.
- static void `IfIsAssignableFrom` (object instance, Type type, string message)
Throws an exception of type TEx with given message message if and only if an instance of given type can be assigned to specified object.
- static void `IfIsAssignableFrom`< TType > (object instance)
Throws an exception of type TEx if and only if an instance of given type can be assigned to specified object.

- static void [IfIsAssignableFrom< TType >](#) (object instance, string message)
Throws an exception of type TEx with given message message if and only if an instance of given type can be assigned to specified object.
- static void [IfIsNotAssignableFrom](#) (object instance, Type type)
Throws an exception of type TEx if and only if an instance of given type cannot be assigned to specified object.
- static void [IfIsNotAssignableFrom](#) (object instance, Type type, string message)
Throws an exception of type TEx with given message message if and only if an instance of given type cannot be assigned to specified object.
- static void [IfIsNotAssignableFrom< TType >](#) (object instance)
Throws an exception of type TEx if and only if an instance of given type cannot be assigned to specified object.
- static void [IfIsNotAssignableFrom< TType >](#) (object instance, string message)
Throws an exception of type TEx with given message message if and only if an instance of given type cannot be assigned to specified object.
- static void [IfIsContainedIn](#) (object argument, System.Collections.IList collection)
Throws an exception of type TEx if and only if specified argument is contained in given collection.
- static void [IfIsContainedIn](#) (object argument, System.Collections.IList collection, string message)
Throws an exception of type TEx with given message message if and only if specified argument is contained in given collection.
- static void [IfIsNotContainedIn](#) (object argument, System.Collections.IList collection)
Throws an exception of type TEx if and only if specified argument is not contained in given collection.
- static void [IfIsNotContainedIn](#) (object argument, System.Collections.IList collection, string message)
Throws an exception of type TEx with given message message if and only if specified argument is not contained in given collection.
- static void [IfIsContainedIn< TArg >](#) (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)
Throws an exception of type TEx if and only if specified argument is contained in given collection.
- static void [IfIsContainedIn< TArg >](#) (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)
Throws an exception of type TEx with given message message if and only if specified argument is contained in given collection.
- static void [IfIsNotContainedIn< TArg >](#) (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection)
Throws an exception of type TEx if and only if specified argument is not contained in given collection.
- static void [IfIsNotContainedIn< TArg >](#) (TArg arg, System.Collections.Generic.IEnumerable< TArg > collection, string message)
Throws an exception of type TEx with given message message if and only if specified argument is not contained in given collection.
- static void [IfIsContainedIn< TArg >](#) (TArg arg, System.Collections.IDictionary dictionary)
Throws an exception of type TEx if and only if specified argument is contained in given dictionary keys.
- static void [IfIsContainedIn< TArg >](#) (TArg arg, System.Collections.IDictionary dictionary, string message)
Throws an exception of type TEx with given message message if and only if specified argument is contained in given dictionary keys.
- static void [IfIsNotContainedIn< TArg >](#) (TArg arg, System.Collections.IDictionary dictionary)
Throws an exception of type TEx if and only if specified argument is not contained in given dictionary keys.
- static void [IfIsNotContainedIn< TArg >](#) (TArg arg, System.Collections.IDictionary dictionary, string message)
Throws an exception of type TEx with given message message if and only if specified argument is not contained in given dictionary keys.
- static void [IfIsContainedIn< TArg1, TArg2 >](#) (TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary)
Throws an exception of type TEx if and only if specified arguments are contained in given dictionary pairs.
- static void [IfIsContainedIn< TArg1, TArg2 >](#) (TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary, string message)
Throws an exception of type TEx with given message message if and only if specified arguments are contained in given dictionary pairs.

- static void [IfIsNotContainedIn< TArg1, TArg2 >](#) (TArg1 arg1, TArg2 arg2, System.Collections.Generic.IDictionary< TArg1, TArg2 > dictionary)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- static void [IfIsNaN](#) (double number)

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

- static void [IfIsNaN](#) (double number, string message)

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

- static void [IfIsNotNaN](#) (double number)

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

- static void [IfIsNotNaN](#) (double number, string message)

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

- static void [IfIsNull](#)< TArg > (TArg arg)

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

- static void [IfIsNull](#)< TArg > (TArg arg, string message)

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

- static void [IfIsNotNull](#)< TArg > (TArg arg)

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

- static void [IfIsNotNull](#)< TArg > (TArg arg, string message)

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

Additional Inherited Members

6.3.1 Detailed Description

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

Template Parameters

<i>TEx</i>	The type of the exceptions thrown if conditions will be satisfied.
------------	--

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

Type Constraints

***TEx* : Exception**

Definition at line 70 of file [Raise.cs](#).

6.3.2 Member Function Documentation

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

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

Parameters

<i>cond</i>	The condition that determines whether an exception will be thrown.
-------------	--

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *cond* is true, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 132 of file [Raise.cs](#).

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

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

Parameters

<i>cond</i>	The condition that determines whether an exception will be thrown.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

Definition at line 166 of file [Raise.cs](#).

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

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

Parameters

<i>arg1</i>	First argument to test for equality.
<i>arg2</i>	Second argument to test for equality.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

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

Definition at line 255 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

Definition at line 290 of file [Raise.cs](#).

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

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

Parameters

<i>arg1</i>	First argument to test for equality.
<i>arg2</i>	Second argument to test for equality.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If arguments are not equal, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 318 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If arguments are not equal, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 353 of file [Raise.cs](#).

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

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

Parameters

<i>arg1</i>	First argument to test for reference equality.
<i>arg2</i>	Second argument to test for reference equality.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If arguments do not point to the same object, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 444 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If arguments do not point to the same object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 479 of file [Raise.cs](#).

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

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

Parameters

<i>arg1</i>	First argument to test for reference equality.
<i>arg2</i>	Second argument to test for reference equality.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If arguments point to the same object, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 381 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If arguments point to the same object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 416 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
<i>type</i>	The type whose instance must be assigned to given object.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 507 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If an instance of given type can be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 543 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type whose instance must be assigned to given object.
--------------	---

Parameters

<i>instance</i>	The object to test.
-----------------	---------------------

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 572 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type whose instance must be assigned to given object.
--------------	---

Parameters

<i>instance</i>	The object to test.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If an instance of given type can be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 609 of file [Raise.cs](#).

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

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

Parameters

<i>argument</i>	The argument to check.
<i>collection</i>	The collection that must not contain given argument.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *argument* is contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 773 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

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

Definition at line 808 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to check.
<i>collection</i>	The collection that must not contain given argument.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg* is contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 899 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

Definition at line 934 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to check.
<i>dictionary</i>	The dictionary that must not contain given argument.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

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

Definition at line 1025 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

Definition at line 1060 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 1153 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If *arg1* and *arg2* are contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

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

Definition at line 1190 of file [Raise.cs](#).

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

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

Parameters

<i>valueToCheck</i>	The string to check for emptiness.
---------------------	------------------------------------

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 1286 of file [Raise.cs](#).

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

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

Parameters

<i>valueToCheck</i>	The string to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

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

Definition at line 1321 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
-------------------	--

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *collection* is null or empty, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1410 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

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

Definition at line 1444 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
-------------------	--

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *collection* is null or empty, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1532 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *collection* is null or empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1566 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
<i>type</i>	The type the object must have.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *instance* has given type, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1655 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
<i>type</i>	The type the object must have.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

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

Definition at line 1690 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type the object must have.
--------------	--------------------------------

Parameters

<i>instance</i>	The object to test.
-----------------	---------------------

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If *instance* has given type, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1719 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type the object must have.
--------------	--------------------------------

Parameters

<i>instance</i>	The object to test.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

or a constructor which takes a string as only parameter.

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

Definition at line 1755 of file [Raise.cs](#).

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

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

Parameters

<i>number</i>	The double to test for double.NaN equality.
---------------	---

Exceptions

ThrowerException	<code>TEx</code> has not a public or internal constructor with no parameters, or <code>TEx</code> is abstract.
----------------------------------	--

If *number* is double.NaN, then an exception of type `TEx` will be thrown.

In order to do that, `TEx` must have a constructor which doesn't take any arguments.

Definition at line 1910 of file [Raise.cs](#).

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

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

Parameters

<i>number</i>	The double to test for double.NaN equality.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *number* is double.NaN, then an exception of type `TEx`, with the message specified by *message*, will be thrown.

In order to do that, `TEx` must have either a constructor which takes a string and an `System.Exception` as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1944 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
<i>type</i>	The type whose instance must not be assigned to given object.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 637 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 674 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type whose instance must not be assigned to given object.
--------------	---

Parameters

<i>instance</i>	The object to test.
-----------------	---------------------

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 705 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type whose instance must not be assigned to given object.
--------------	---

Parameters

<i>instance</i>	The object to test.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If an instance of given type cannot be assigned to specified object, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 745 of file [Raise.cs](#).

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

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

Parameters

<i>argument</i>	The argument to check.
<i>collection</i>	The collection that must contain given argument.

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If *argument* is not contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 836 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an *System.Exception* as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 871 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to check.
<i>collection</i>	The collection that must contain given argument.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg* is not contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 962 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an *System.Exception* as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 997 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to check.
<i>dictionary</i>	The dictionary that must contain given argument.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg* is not contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1088 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1124 of file [Raise.cs](#).

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

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

Parameters

<i>arg1</i>	The key argument to check.
<i>arg2</i>	The value argument to check.
<i>dictionary</i>	The dictionary that must contain given argument.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg1* and *arg2* are not contained, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1220 of file [Raise.cs](#).

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

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

Parameters

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

Exceptions

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

If *arg1* and *arg2* are not contained, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1258 of file [Raise.cs](#).

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

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

Parameters

<i>valueToCheck</i>	The string to check for emptiness.
---------------------	------------------------------------

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If *valueToCheck* is not empty, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1348 of file [Raise.cs](#).

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

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

Parameters

<i>valueToCheck</i>	The string to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

the exception.

Definition at line 1383 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
-------------------	--

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *collection* is null or not empty, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1471 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *collection* is null or not empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1505 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
-------------------	--

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 1593 of file [Raise.cs](#).

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

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

Parameters

<i>collection</i>	The collection to check for emptiness.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *collection* is null or not empty, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1627 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
<i>type</i>	The type the object must not have.

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 1783 of file [Raise.cs](#).

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

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

Parameters

<i>instance</i>	The object to test.
-----------------	---------------------

<i>type</i>	The type the object must not have.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 1818 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type the object must not have.
--------------	------------------------------------

Parameters

<i>instance</i>	The object to test.
-----------------	---------------------

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If *instance* has not given type, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 1847 of file [Raise.cs](#).

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

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

Template Parameters

<i>TType</i>	The type the object must not have.
--------------	------------------------------------

Parameters

<i>instance</i>	The object to test.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

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

Definition at line 1883 of file [Raise.cs](#).

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

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

Parameters

<i>number</i>	The double to test for double.NaN equality.
---------------	---

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

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

Definition at line 1971 of file [Raise.cs](#).

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

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

Parameters

<i>number</i>	The double to test for double.NaN equality.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

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

Definition at line 2006 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to test for nullity.
------------	-----------------------------------

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg* is null, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 2094 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to test for nullity.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

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

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 2128 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to test for nullity.
------------	-----------------------------------

Exceptions

ThrowerException	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
----------------------------------	--

If *arg* is null, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 2033 of file [Raise.cs](#).

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

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

Parameters

<i>arg</i>	The argument to test for nullity.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *arg* is null, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 2067 of file [Raise.cs](#).

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

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

Parameters

<i>cond</i>	The condition that determines whether an exception will be thrown.
-------------	--

Exceptions

<i>ThrowerException</i>	<i>TEx</i> has not a public or internal constructor with no parameters, or <i>TEx</i> is abstract.
---	--

If *cond* is false, then an exception of type *TEx* will be thrown.

In order to do that, *TEx* must have a constructor which doesn't take any arguments.

Definition at line 193 of file [Raise.cs](#).

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

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

Parameters

<i>cond</i>	The condition that determines whether an exception will be thrown.
<i>message</i>	The message the thrown exception will have.

Exceptions

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

If *cond* is false, then an exception of type *TEx* , with the message specified by *message* , will be thrown.

In order to do that, *TEx* must have either a constructor which takes a string and an System.Exception as arguments, or a constructor which takes a string as only parameter.

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

Definition at line 227 of file [Raise.cs](#).

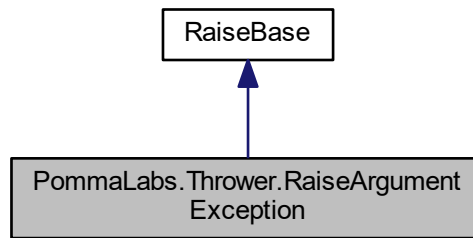
The documentation for this class was generated from the following file:

- [Raise.cs](#)

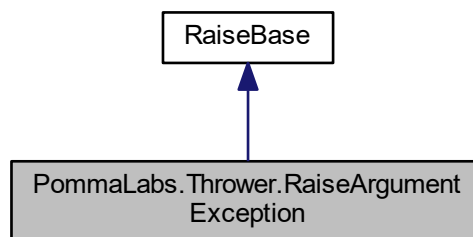
6.4 PommaLabs.Thrower.RaiseArgumentException Class Reference

Utility methods which can be used to handle bad arguments.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentException:



Static Public Member Functions

- static void **If** (bool condition)
Throws ArgumentException if given condition is true.
- static void **If** (bool condition, string argumentName, string message=null)
Throws ArgumentException if given condition is true.
- static void **IfNot** (bool condition)
Throws ArgumentException if given condition is false.
- static void **IfNot** (bool condition, string argumentName, string message=null)
Throws ArgumentException if given condition is false.
- static void **IfIsValid< TArg >** (TArg argument)
Throws ArgumentException if given argument is not valid.
- static void **IfIsValid< TArg >** (TArg argument, string argumentName, string message=null)
Throws ArgumentException if given argument is not valid.
- static void **IfIsValidEmailAddress** (string emailAddress)
Throws ArgumentException if given string is not a valid email address.
- static void **IfIsValidEmailAddress** (string emailAddress, bool allowInternational)
Throws ArgumentException if given string is not a valid email address.
- static void **IfIsValidEmailAddress** (string emailAddress, string argumentName, string message=null)

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

- static void [IfIsValidEmailAddress](#) (string emailAddress, bool allowInternational, string argumentName, string message=null)

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

- static void [IfIsValidPhoneNumber](#) (string phoneNumber)

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

- static void [IfIsValidPhoneNumber](#) (string phoneNumber, string argumentName, string message=null)

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

- static void [IfIsNullOrEmpty](#) (string value)

Throws ArgumentException if given string is null or empty.

- static void [IfIsNullOrEmpty](#) (string value, string argumentName, string message=null)

Throws ArgumentException if given string is null or empty.

- static void [IfIsNullOrWhiteSpace](#) (string value)

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

- static void [IfIsNullOrWhiteSpace](#) (string value, string argumentName, string message=null)

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

Additional Inherited Members

6.4.1 Detailed Description

Utility methods which can be used to handle bad arguments.

Definition at line 33 of file [RaiseArgumentException.cs](#).

6.4.2 Member Function Documentation

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

Throws ArgumentException if given condition is true.

Parameters

<i>condition</i>	The condition.
------------------	----------------

Definition at line 47 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 68 of file [RaiseArgumentException.cs](#).

6.4.2.3 static void PommaLabs.Thrower.RaiseArgumentException.IfIsValid< TArg > (TArg argument) [static]

Throws ArgumentException if given argument is not valid.

Template Parameters

<i>TArg</i>	The type of the argument.
-------------	---------------------------

Parameters

<i>argument</i>	The argument.
-----------------	---------------

Definition at line 130 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given argument is not valid.

Template Parameters

<i>TArg</i>	The type of the argument.
-------------	---------------------------

Parameters

<i>argument</i>	The argument.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 153 of file [RaiseArgumentException.cs](#).

6.4.2.5 `static void PommaLabs.Thrower.RaiseArgumentException.IfIsValidEmailAddress (string emailAddress) [static]`

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

Parameters

<i>emailAddress</i>	An email address.
---------------------	-------------------

Definition at line 176 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>emailAddress</i>	An email address.
<i>allowInternational</i>	true if the validator should allow international characters; otherwise,

false

.

Definition at line 196 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>emailAddress</i>	An email address.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 218 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>emailAddress</i>	An email address.
<i>allowInternational</i>	true if the validator should allow international characters; otherwise,

false

.

Parameters

<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 243 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>phoneNumber</i>	A phone number.
--------------------	-----------------

Definition at line 266 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>phoneNumber</i>	A phone number.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 288 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given string is null or empty.

Parameters

<i>value</i>	The string value.
--------------	-------------------

Definition at line 312 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given string is null or empty.

Parameters

<i>value</i>	The string value.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The optional message.

message and *argumentName* are strictly required arguments.

Definition at line 333 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>value</i>	The string value.
--------------	-------------------

Definition at line 349 of file [RaiseArgumentException.cs](#).

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

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

Parameters

<i>value</i>	The string value.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The optional message.

message and *argumentName* are strictly required arguments.

Definition at line 370 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given condition is false.

Parameters

<i>condition</i>	The condition.
------------------	----------------

Definition at line 88 of file [RaiseArgumentException.cs](#).

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

Throws ArgumentException if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 109 of file [RaiseArgumentException.cs](#).

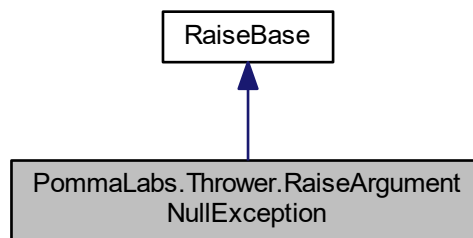
The documentation for this class was generated from the following file:

- [RaiseArgumentException.cs](#)

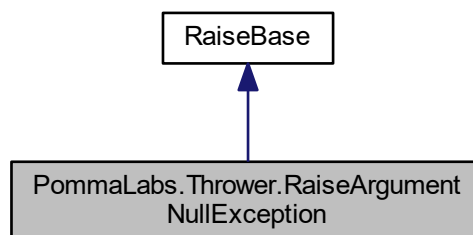
6.5 PommaLabs.Thrower.RaiseArgumentNullException Class Reference

Utility methods which can be used to handle null references.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentNullException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentNullException:



Static Public Member Functions

- static void [IfIsNull< TArg >](#) (TArg argument)
Throws ArgumentException if given argument is null.

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

Additional Inherited Members

6.5.1 Detailed Description

Utility methods which can be used to handle null references.

Definition at line 32 of file [RaiseArgumentNullException.cs](#).

6.5.2 Member Function Documentation

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

Throws ArgumentException if given argument if null.

Template Parameters

<i>TArg</i>	The type of the argument.
-------------	---------------------------

Parameters

<i>argument</i>	The argument.
-----------------	---------------

Definition at line 43 of file [RaiseArgumentNullException.cs](#).

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

Throws ArgumentException if given argument if null.

Template Parameters

<i>TArg</i>	The type of the argument.
-------------	---------------------------

Parameters

<i>argument</i>	The argument.
<i>argumentName</i>	The name of the argument.

Definition at line 61 of file [RaiseArgumentNullException.cs](#).

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

Throws ArgumentException if given argument if null.

Template Parameters

<i>TArg</i>	The type of the argument.
-------------	---------------------------

Parameters

<i>argument</i>	The argument.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message that should be put into the exception.

Definition at line 80 of file [RaiseArgumentNullException.cs](#).

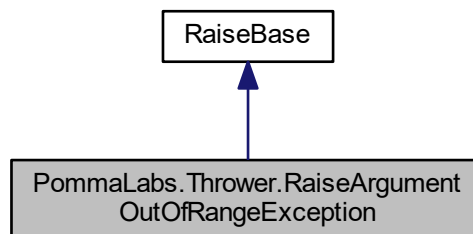
The documentation for this class was generated from the following file:

- [RaiseArgumentNullException.cs](#)

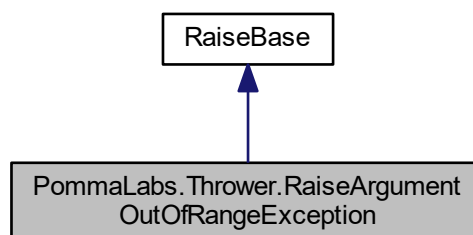
6.6 PommaLabs.Thrower.RaiseArgumentOutOfRangeException Class Reference

Utility methods which can be used to handle ranges.

Inheritance diagram for PommaLabs.Thrower.RaiseArgumentOutOfRangeException:



Collaboration diagram for PommaLabs.Thrower.RaiseArgumentOutOfRangeException:



Static Public Member Functions

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

- Throws ArgumentOutOfRangeException if given condition is true.*
- static void **IfNot** (bool condition, string argumentName=null)
Throws ArgumentOutOfRangeException if given condition is false.
- static void **IfNot** (bool condition, string argumentName, string message)
Throws ArgumentOutOfRangeException if given condition is false.
- static void **IfIsLess< TArg >** (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLess** (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLess< TArg >** (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLess** (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLess< TArg >** (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLess** (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is less than argument2 .
- static void **IfIsLessOrEqual< TArg >** (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsLessOrEqual** (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsLessOrEqual< TArg >** (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsLessOrEqual** (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsLessOrEqual< TArg >** (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsLessOrEqual** (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is less than or equal to argument2 .
- static void **IfIsGreater< TArg >** (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreater** (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreater< TArg >** (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreater** (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreater< TArg >** (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreater** (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is greater than argument2 .
- static void **IfIsGreaterOrEqual< TArg >** (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void **IfIsGreaterOrEqual** (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void **IfIsGreaterOrEqual< TArg >** (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .

- static void [IfIsGreaterOrEqual](#) (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void [IfIsGreaterOrEqual< TArg >](#) (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void [IfIsGreaterOrEqual](#) (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is greater than or equal to argument2 .
- static void [IfIsEqual< TArg >](#) (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsEqual](#) (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsEqual< TArg >](#) (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsEqual](#) (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsEqual< TArg >](#) (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsEqual](#) (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is equal to argument2 .
- static void [IfIsNotEqual< TArg >](#) (TArg argument1, TArg argument2)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void [IfIsNotEqual](#) (IComparable argument1, IComparable argument2)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void [IfIsNotEqual< TArg >](#) (TArg argument1, TArg argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void [IfIsNotEqual](#) (IComparable argument1, IComparable argument2, string argumentName)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void [IfIsNotEqual< TArg >](#) (TArg argument1, TArg argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .
- static void [IfIsNotEqual](#) (IComparable argument1, IComparable argument2, string argumentName, string message)
Throws ArgumentOutOfRangeException if argument1 is not equal to argument2 .

Additional Inherited Members

6.6.1 Detailed Description

Utility methods which can be used to handle ranges.

Definition at line 31 of file [RaiseArgumentOutOfRangeException.cs](#).

6.6.2 Member Function Documentation

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

Throws ArgumentOutOfRangeException if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The optional name of the argument.

Definition at line 44 of file [RaiseArgumentOutOfRangeException.cs](#).

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

Throws ArgumentOutOfRangeException if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 65 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 677 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 721 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

<i>argumentName</i>	The name of the argument.
<i>message</i>	The message that should be put into the exception.

Definition at line 767 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 658 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

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

Definition at line 701 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 746 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 413 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 457 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 503 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 394 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 437 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 482 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 545 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 589 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 635 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 526 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

***TArg* : *Comparable*< *TArg*>**

Definition at line 569 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*< *TArg*>**

Definition at line 614 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 149 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 193 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 239 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 130 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

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

Definition at line 173 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*< *TArg*>**

Definition at line 218 of file [RaiseArgumentOutOfRangeException.cs](#).

6.6.2.27 `static void PommaLabs.Thrower.RaiseArgumentOutOfRangeException.IfIsLessOrEqual (Comparable argument1, Comparable argument2) [static]`

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 281 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 325 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 371 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

***TArg* : *Comparable*< *TArg*>**

Definition at line 262 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

***TArg* : *Comparable*< *TArg*>**

Definition at line 305 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 350 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 809 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Definition at line 853 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 899 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 790 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.
<i>argumentName</i>	The name of the argument.

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 833 of file [RaiseArgumentOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

***TArg* : *Comparable*<*TArg*>**

Definition at line 878 of file [RaiseArgumentOutOfRangeException.cs](#).

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

Throws `ArgumentOutOfRangeException` if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The optional name of the argument.

Definition at line 86 of file [RaiseArgumentOutOfRangeException.cs](#).

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

Throws `ArgumentOutOfRangeException` if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>argumentName</i>	The name of the argument.
<i>message</i>	The message.

message and *argumentName* are strictly required arguments.

Definition at line 107 of file [RaiseArgumentOutOfRangeException.cs](#).

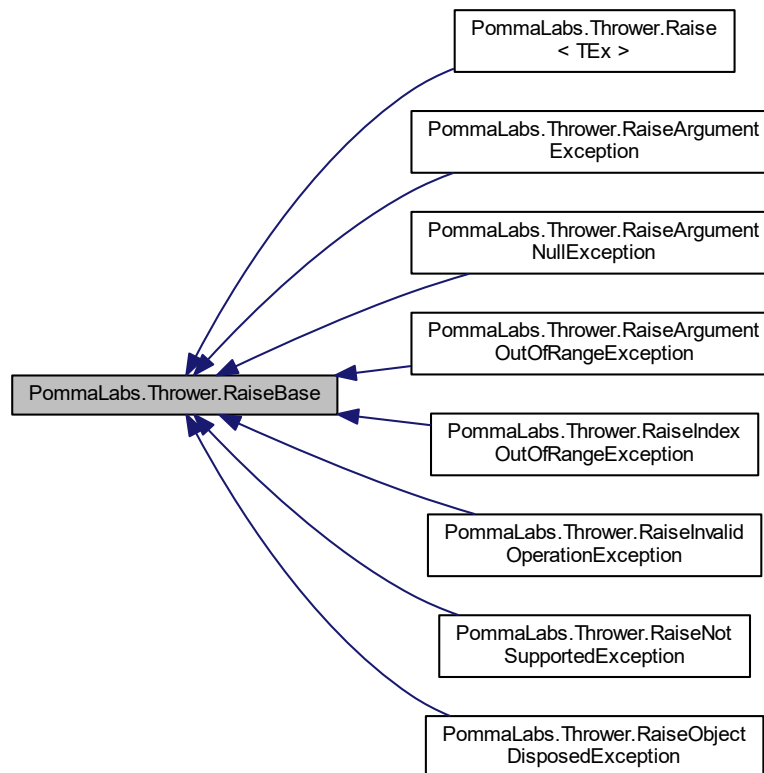
The documentation for this class was generated from the following file:

- [RaiseArgumentOutOfRangeException.cs](#)

6.7 PommaLabs.Thrower.RaiseBase Class Reference

Stores items shared by various `Raise<TEx>` instances.

Inheritance diagram for PommaLabs.Thrower.RaiseBase:



Static Protected Attributes

- static readonly Type[] [NoCtorTypes](#) = new Type[0]
Stores an empty array of System.Type used to seek constructors without parameters.
- static readonly Type[] [StrExCtorTypes](#) = { typeof(string), typeof(Exception) }
Stores the types needed to seek the constructor which takes a string and an exception as parameters to instance the exception.
- static readonly Type[] [StrCtorType](#) = { typeof(string) }
Stores the type needed to seek the constructor which takes a string as parameter to instance the exception.

6.7.1 Detailed Description

Stores items shared by various Raise<TEx> instances.

Definition at line 35 of file [Raise.cs](#).

6.7.2 Member Data Documentation

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

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

Definition at line 42 of file [Raise.cs](#).

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

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

Definition at line 58 of file [Raise.cs](#).

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

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

Definition at line 50 of file [Raise.cs](#).

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

- [Raise.cs](#)

6.8 PommaLabs.Thrower.RaiseHttpException Class Reference

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

Static Public Member Functions

- static void [If](#) (bool condition, HttpStatusCode httpStatusCode, string message=null)
Throws [HttpException](#) if given condition is true.
- static void [If](#) (bool condition, HttpStatusCode httpStatusCode, string message, [HttpExceptionInfo](#) 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.

6.8.1 Detailed Description

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

Definition at line 33 of file [RaiseHttpException.cs](#).

6.8.2 Member Function Documentation

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

Throws [HttpException](#) if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>statusCode</i>	The HTTP status code corresponding to the error.
<i>message</i>	The optional message.

Definition at line 45 of file [RaiseHttpException.cs](#).

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

Throws [HttpException](#) if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>statusCode</i>	The HTTP status code corresponding to the error.
<i>message</i>	The required message.
<i>additionalInfo</i>	Additional exception info.

Definition at line 64 of file [RaiseHttpException.cs](#).

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

Throws [HttpException](#) if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>statusCode</i>	The HTTP status code corresponding to the error.
<i>message</i>	The optional message.

Definition at line 82 of file [RaiseHttpException.cs](#).

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

Throws [HttpException](#) if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>statusCode</i>	The HTTP status code corresponding to the error.
<i>message</i>	The required message.
<i>additionalInfo</i>	Additional exception info.

Definition at line 101 of file [RaiseHttpException.cs](#).

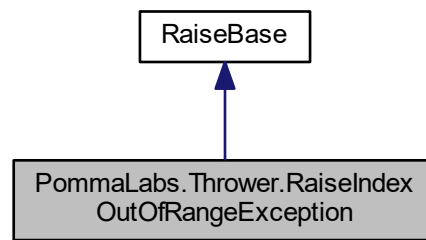
The documentation for this class was generated from the following file:

- [RaiseHttpException.cs](#)

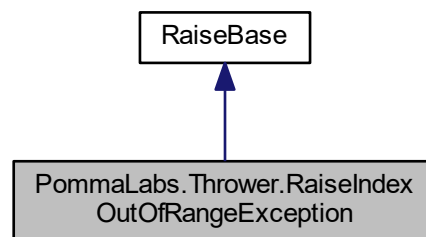
6.9 PommaLabs.Thrower.RaiseIndexOutOfRangeException Class Reference

Utility methods which can be used to handle indexes.

Inheritance diagram for PommaLabs.Thrower.RaiseIndexOutOfRangeException:



Collaboration diagram for PommaLabs.Thrower.RaiseIndexOutOfRangeException:



Static Public Member Functions

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

- Throws IndexOutOfRangeException if argument1 is greater than argument2 .*
- static void [IfIsGreater](#) (IComparable argument1, IComparable argument2)
- Throws IndexOutOfRangeException if argument1 is greater than argument2 .*
- static void [IfIsGreater< TArg >](#) (TArg argument1, TArg argument2, string message)
- Throws IndexOutOfRangeException if argument1 is greater than argument2 .*
- static void [IfIsGreater](#) (IComparable argument1, IComparable argument2, string message)
- Throws IndexOutOfRangeException if argument1 is greater than argument2 .*
- static void [IfIsGreaterOrEqual< TArg >](#) (TArg argument1, TArg argument2)
- Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .*
- static void [IfIsGreaterOrEqual](#) (IComparable argument1, IComparable argument2)
- Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .*
- static void [IfIsGreaterOrEqual< TArg >](#) (TArg argument1, TArg argument2, string message)
- Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .*
- static void [IfIsGreaterOrEqual](#) (IComparable argument1, IComparable argument2, string message)
- Throws IndexOutOfRangeException if argument1 is greater than or equal to argument2 .*
- static void [IfIsEqual< TArg >](#) (TArg argument1, TArg argument2)
- Throws IndexOutOfRangeException if argument1 is equal to argument2 .*
- static void [IfIsEqual](#) (IComparable argument1, IComparable argument2)
- Throws IndexOutOfRangeException if argument1 is equal to argument2 .*
- static void [IfIsEqual< TArg >](#) (TArg argument1, TArg argument2, string message)
- Throws IndexOutOfRangeException if argument1 is equal to argument2 .*
- static void [IfIsEqual](#) (IComparable argument1, IComparable argument2, string message)
- Throws IndexOutOfRangeException if argument1 is equal to argument2 .*
- static void [IfIsNotEqual< TArg >](#) (TArg argument1, TArg argument2)
- Throws IndexOutOfRangeException if argument1 is not equal to argument2 .*
- static void [IfIsNotEqual](#) (IComparable argument1, IComparable argument2)
- Throws IndexOutOfRangeException if argument1 is not equal to argument2 .*
- static void [IfIsNotEqual< TArg >](#) (TArg argument1, TArg argument2, string message)
- Throws IndexOutOfRangeException if argument1 is not equal to argument2 .*
- static void [IfIsNotEqual](#) (IComparable argument1, IComparable argument2, string message)
- Throws IndexOutOfRangeException if argument1 is not equal to argument2 .*

Additional Inherited Members

6.9.1 Detailed Description

Utility methods which can be used to handle indexes.

Definition at line 31 of file [RaiseIndexOutOfRangeException.cs](#).

6.9.2 Member Function Documentation

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 409 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 453 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 390 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

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

Definition at line 433 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 237 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 281 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 218 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

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

Definition at line 261 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 323 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 367 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 304 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

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

Definition at line 347 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 65 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 109 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

TArg* : *Comparable< *TArg* >

Definition at line 46 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

TArg* : *Comparable< *TArg* >

Definition at line 89 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 151 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 195 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 132 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

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

Definition at line 175 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Definition at line 495 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Parameters

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

Definition at line 539 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

<i>argument1</i>	The left side argument.
<i>argument2</i>	The right side argument.

Type Constraints

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

Definition at line 476 of file [RaiseIndexOutOfRangeException.cs](#).

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

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

Template Parameters

<i>TArg</i>	The type of the arguments.
-------------	----------------------------

Parameters

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

Type Constraints

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

Definition at line 519 of file [RaiseIndexOutOfRangeException.cs](#).

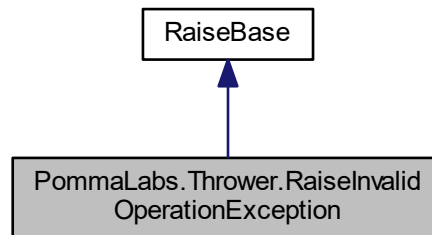
The documentation for this class was generated from the following file:

- [RaiseIndexOutOfRangeException.cs](#)

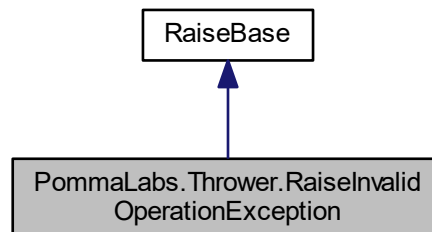
6.10 PommaLabs.Thrower.RaiseInvalidOperationException Class Reference

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

Inheritance diagram for PommaLabs.Thrower.RaiseInvalidOperationException:



Collaboration diagram for PommaLabs.Thrower.RaiseInvalidOperationException:



Static Public Member Functions

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

Additional Inherited Members

6.10.1 Detailed Description

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

Definition at line 31 of file [RaiseInvalidOperationException.cs](#).

6.10.2 Member Function Documentation

6.10.2.1 `static void PommaLabs.Thrower.RaiseInvalidOperationException.If (bool condition, string message = null)`
`[static]`

Throws InvalidOperationException if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>message</i>	The optional message.

Definition at line 42 of file [RaiseInvalidOperationException.cs](#).

6.10.2.2 `static void PommaLabs.Thrower.RaiseInvalidOperationException.IfNot (bool condition, string message = null)`
`[static]`

Throws InvalidOperationException if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>message</i>	The optional message.

Definition at line 59 of file [RaiseInvalidOperationException.cs](#).

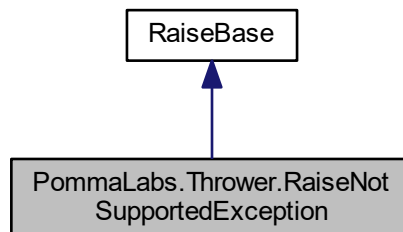
The documentation for this class was generated from the following file:

- [RaiseInvalidOperationException.cs](#)

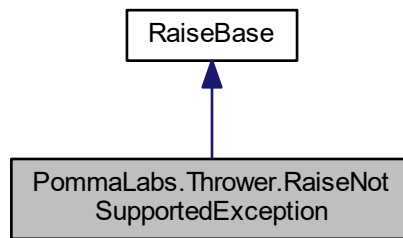
6.11 PommaLabs.Thrower.RaiseNotSupportedException Class Reference

Utility methods which can be used to handle unsupported operations.

Inheritance diagram for PommaLabs.Thrower.RaiseNotSupportedException:



Collaboration diagram for PommaLabs.Thrower.RaiseNotSupportedException:



Static Public Member Functions

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

Additional Inherited Members

6.11.1 Detailed Description

Utility methods which can be used to handle unsupported operations.

Definition at line 31 of file [RaiseNotSupportedException.cs](#).

6.11.2 Member Function Documentation

6.11.2.1 static void `PommaLabs.Thrower.RaiseNotSupportedException.If` (bool condition, string message = null)
 [static]

Throws `NotSupportedException` if given condition is true.

Parameters

<i>condition</i>	The condition.
<i>message</i>	The optional message.

Definition at line 42 of file [RaiseNotSupportedException.cs](#).

6.11.2.2 static void `PommaLabs.Thrower.RaiseNotSupportedException.IfNot` (bool condition, string message = null)
 [static]

Throws `NotSupportedException` if given condition is false.

Parameters

<i>condition</i>	The condition.
<i>message</i>	The optional message.

Definition at line 59 of file [RaiseNotSupportedException.cs](#).

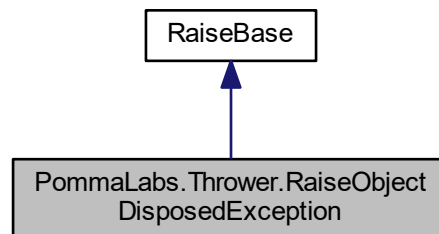
The documentation for this class was generated from the following file:

- [RaiseNotSupportedException.cs](#)

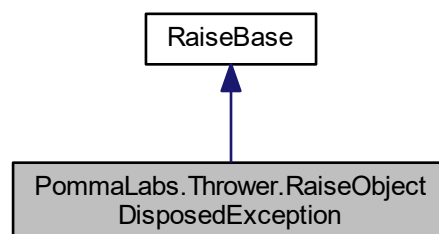
6.12 PommaLabs.Thrower.RaiseObjectDisposedException Class Reference

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

Inheritance diagram for PommaLabs.Thrower.RaiseObjectDisposedException:



Collaboration diagram for PommaLabs.Thrower.RaiseObjectDisposedException:



Static Public Member Functions

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

Additional Inherited Members

6.12.1 Detailed Description

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

Definition at line 31 of file [RaiseObjectDisposedException.cs](#).

6.12.2 Member Function Documentation

6.12.2.1 `static void PommaLabs.Thrower.RaiseObjectDisposedException.If (bool disposed, string objectName, string message = null) [static]`

Throws ObjectDisposedException if the object has been disposed.

Parameters

<i>disposed</i>	Whether the object has been disposed or not.
<i>objectName</i>	The required object name.
<i>message</i>	The optional message.

Definition at line 43 of file [RaiseObjectDisposedException.cs](#).

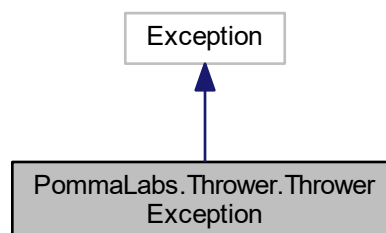
The documentation for this class was generated from the following file:

- [RaiseObjectDisposedException.cs](#)

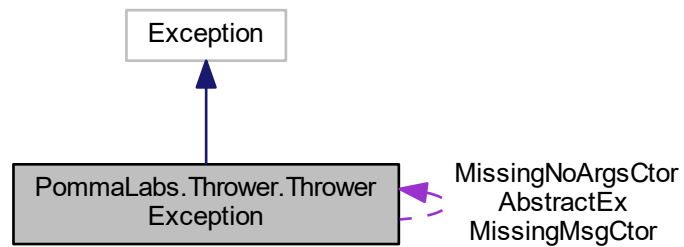
6.13 PommaLabs.Thrower.ThrowerException Class Reference

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

Inheritance diagram for PommaLabs.Thrower.ThrowerException:



Collaboration diagram for PommaLabs.Thrower.ThrowerException:



6.13.1 Detailed Description

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

Definition at line 2195 of file [Raise.cs](#).

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

- [Raise.cs](#)

Chapter 7

File Documentation

7.1 Raise.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseBase](#)
Stores items shared by various `Raise<TEx>` instances.
- class [PommaLabs.Thrower.Raise< TEx >](#)
Contains methods that throw specified exception `TEx` if given conditions will be verified.
- class [PommaLabs.Thrower.ThrowerException](#)
Exception thrown by `Raise<TEx>` when the type parameter passed to that class has something invalid (missing constructors, etc).

Namespaces

- namespace [PommaLabs.Thrower](#)

7.2 Raise.cs

```
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,
00022 // OUT 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 using System.Reflection;
00029
00030 namespace PommaLabs.Thrower
```

```

00031 {
00035     public abstract class RaiseBase
00036     {
00040         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00041         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00042         protected static readonly Type[] NoCtorTypes = new Type[0];
00043
00048         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00049         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00050         protected static readonly Type[] StrExCtorTypes = { typeof(string), typeof(Exception) };
00051
00056         [SuppressMessage("Microsoft.Naming", "CA1704:IdentifiersShouldBeSpelledCorrectly")]
00057         [SuppressMessage("Microsoft.Security", "CA2105:ArrayFieldsShouldNotBeReadOnly")]
00058         protected static readonly Type[] StrCtorType = { typeof(string) };
00059     }
00060
00070     public sealed class Raise<TE> : RaiseBase where TE : Exception
00071     {
00076         private static readonly bool ExTypeIsAbstract = PortableTypeInfo.IsAbstract(typeof(TE));
00077
00083         private static readonly ConstructorInfo NoArgsCtor = GetCtor(NoCtorTypes);
00084
00097         private static readonly ConstructorInfo MsgCtor = GetCtor(StrExCtorTypes) ?? GetCtor(StrCtorType);
00098
00103         private static readonly int MsgArgCount = (MsgCtor == null) ? 0 : MsgCtor.GetParameters().Length;
00104
00108         private Raise()
00109         {
00110             throw new InvalidOperationException("This class should not be instantiated");
00111         }
00112
00127         #if (NET45 || NET46)
00128         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00129             MethodImplOptions.AggressiveInlining)]
00129         #endif
00130
00131         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00132         public static void If(bool cond)
00133         {
00134             if (cond)
00135             {
00136                 DoThrow();
00137             }
00138         }
00139
00161         #if (NET45 || NET46)
00162         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00163             MethodImplOptions.AggressiveInlining)]
00163         #endif
00164
00165         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00166         public static void If(bool cond, string message)
00167         {
00168             if (cond)
00169             {
00170                 DoThrow(message);
00171             }
00172         }
00173
00188         #if (NET45 || NET46)
00189         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00190             MethodImplOptions.AggressiveInlining)]
00190         #endif
00191
00192         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00193         public static void IfNot(bool cond)
00194         {
00195             if (!cond)
00196             {
00197                 DoThrow();
00198             }
00199         }
00200
00222         #if (NET45 || NET46)
00223         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00224             MethodImplOptions.AggressiveInlining)]
00224         #endif
00225
00226         [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00227         public static void IfNot(bool cond, string message)
00228         {
00229             if (!cond)
00230             {
00231                 DoThrow(message);
00232             }
00233         }
00234

```

```

00250 #if (NET45 || NET46)
00251     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00252 #endif
00253
00254     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00255     public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00256     {
00257         if (Equals(arg1, arg2))
00258         {
00259             DoThrow();
00260         }
00261     }
00262
00285 #if (NET45 || NET46)
00286     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00287 #endif
00288
00289     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00290     public static void IfAreEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00291     {
00292         if (Equals(arg1, arg2))
00293         {
00294             DoThrow(message);
00295         }
00296     }
00297
00313 #if (NET45 || NET46)
00314     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00315 #endif
00316
00317     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00318     public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00319     {
00320         if (!Equals(arg1, arg2))
00321         {
00322             DoThrow();
00323         }
00324     }
00325
00348 #if (NET45 || NET46)
00349     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00350 #endif
00351
00352     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00353     public static void IfAreNotEqual<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00354     {
00355         if (!Equals(arg1, arg2))
00356         {
00357             DoThrow(message);
00358         }
00359     }
00360
00376 #if (NET45 || NET46)
00377     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00378 #endif
00379
00380     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00381     public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00382     {
00383         if (ReferenceEquals(arg1, arg2))
00384         {
00385             DoThrow();
00386         }
00387     }
00388
00411 #if (NET45 || NET46)
00412     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00413 #endif
00414
00415     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00416     public static void IfAreSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00417     {
00418         if (ReferenceEquals(arg1, arg2))
00419         {
00420             DoThrow(message);
00421         }
00422     }
00423
00439 #if (NET45 || NET46)
00440     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]

```

```

00441 #endif
00442
00443     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00444     public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2)
00445     {
00446         if (!ReferenceEquals(arg1, arg2))
00447         {
00448             DoThrow();
00449         }
00450     }
00451
00474 #if (NET45 || NET46)
00475     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00476 #endif
00477
00478     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00479     public static void IfAreNotSame<TArg1, TArg2>(TArg1 arg1, TArg2 arg2, string message)
00480     {
00481         if (!ReferenceEquals(arg1, arg2))
00482         {
00483             DoThrow(message);
00484         }
00485     }
00486
00502 #if (NET45 || NET46)
00503     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00504 #endif
00505
00506     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00507     public static void IfIsAssignableFrom(object instance, Type type)
00508     {
00509         if (PortableTypeInfo.IsAssignableFrom(instance, type))
00510         {
00511             DoThrow();
00512         }
00513     }
00514
00538 #if (NET45 || NET46)
00539     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00540 #endif
00541
00542     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00543     public static void IfIsAssignableFrom(object instance, Type type, string message)
00544     {
00545         if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, type))
00546         {
00547             DoThrow(message);
00548         }
00549     }
00550
00566 #if (NET45 || NET46)
00567     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00568 #endif
00569
00570     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00571     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00572     public static void IfIsAssignableFrom<TType>(object instance)
00573     {
00574         if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, typeof(TType
)))
00575         {
00576             DoThrow();
00577         }
00578     }
00579
00603 #if (NET45 || NET46)
00604     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00605 #endif
00606
00607     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00608     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00609     public static void IfIsAssignableFrom<TType>(object instance, string message)
00610     {
00611         if (ReferenceEquals(instance, null) || PortableTypeInfo.IsAssignableFrom(instance, typeof(TType
)))
00612         {
00613             DoThrow(message);
00614         }
00615     }
00616
00632 #if (NET45 || NET46)
00633     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.

```

```

       MethodImplOptions.AggressiveInlining)]
00634 #endif
00635
00636     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00637     public static void IfIsNotAssignableFrom(object instance, Type type)
00638     {
00639         if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, type))
00640         {
00641             DoThrow();
00642         }
00643     }
00644
00669 #if (NET45 || NET46)
00670     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00671 #endif
00672
00673     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00674     public static void IfIsNotAssignableFrom(object instance, Type type, string
        message)
00675     {
00676         if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, type))
00677         {
00678             DoThrow(message);
00679         }
00680     }
00681
00699 #if (NET45 || NET46)
00700     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00701 #endif
00702
00703     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00704     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00705     public static void IfIsNotAssignableFrom<TType>(object instance)
00706     {
00707         if (!PortableTypeInfo.IsAssignableFrom(instance, typeof(TType)))
00708         {
00709             DoThrow();
00710         }
00711     }
00712
00739 #if (NET45 || NET46)
00740     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00741 #endif
00742
00743     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00744     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
00745     public static void IfIsNotAssignableFrom<TType>(object instance, string message)
00746     {
00747         if (ReferenceEquals(instance, null) || !PortableTypeInfo.IsAssignableFrom(instance, typeof(
        TType)))
00748         {
00749             DoThrow(message);
00750         }
00751     }
00752
00768 #if (NET45 || NET46)
00769     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00770 #endif
00771
00772     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00773     public static void IfIsContainedIn(object argument,
        System.Collections.IList collection)
00774     {
00775         if (ReferenceEquals(collection, null) || collection.Contains(argument))
00776         {
00777             DoThrow();
00778         }
00779     }
00780
00803 #if (NET45 || NET46)
00804     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00805 #endif
00806
00807     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00808     public static void IfIsContainedIn(object argument,
        System.Collections.IList collection, string message)
00809     {
00810         if (ReferenceEquals(collection, null) || collection.Contains(argument))
00811         {
00812             DoThrow(message);
00813         }
00814     }

```

```

00815
00831 #if (NET45 || NET46)
00832     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00833 #endif
00834
00835     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00836     public static void IfIsNotContainedIn(object argument,
System.Collections.IList collection)
00837     {
00838         if (ReferenceEquals(collection, null) || !collection.Contains(argument))
00839         {
00840             DoThrow();
00841         }
00842     }
00843
00866 #if (NET45 || NET46)
00867     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00868 #endif
00869
00870     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00871     public static void IfIsNotContainedIn(object argument,
System.Collections.IList collection, string message)
00872     {
00873         if (ReferenceEquals(collection, null) || !collection.Contains(argument))
00874         {
00875             DoThrow(message);
00876         }
00877     }
00878
00894 #if (NET45 || NET46)
00895     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00896 #endif
00897
00898     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00899     public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection)
00900     {
00901         if (ReferenceEquals(collection, null) || collection.Contains(arg))
00902         {
00903             DoThrow();
00904         }
00905     }
00906
00929 #if (NET45 || NET46)
00930     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00931 #endif
00932
00933     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00934     public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection, string message)
00935     {
00936         if (ReferenceEquals(collection, null) || collection.Contains(arg))
00937         {
00938             DoThrow(message);
00939         }
00940     }
00941
00957 #if (NET45 || NET46)
00958     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00959 #endif
00960
00961     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00962     public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection)
00963     {
00964         if (ReferenceEquals(collection, null) || !collection.Contains(arg))
00965         {
00966             DoThrow();
00967         }
00968     }
00969
00992 #if (NET45 || NET46)
00993     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00994 #endif
00995
00996     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
00997     public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.Generic.IEnumerable<
TArg> collection, string message)
00998     {
00999         if (ReferenceEquals(collection, null) || !collection.Contains(arg))
01000     {

```

```

01001         DoThrow(message);
01002     }
01003 }
01004
01020 #if (NET45 || NET46)
01021     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01022 #endif
01023
01024     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01025     public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary)
01026     {
01027         if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
01028         {
01029             DoThrow();
01030         }
01031     }
01032
01055 #if (NET45 || NET46)
01056     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01057 #endif
01058
01059     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01060     public static void IfIsContainedIn<TArg>(TArg arg, System.Collections.IDictionary dictionary,
string message)
01061     {
01062         if (ReferenceEquals(dictionary, null) || dictionary.Contains(arg))
01063         {
01064             DoThrow(message);
01065         }
01066     }
01067
01083 #if (NET45 || NET46)
01084     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01085 #endif
01086
01087     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01088     public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary
dictionary)
01089     {
01090         if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
01091         {
01092             DoThrow();
01093         }
01094     }
01095
01119 #if (NET45 || NET46)
01120     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01121 #endif
01122
01123     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01124     public static void IfIsNotContainedIn<TArg>(TArg arg, System.Collections.IDictionary
dictionary, string message)
01125     {
01126         if (ReferenceEquals(dictionary, null) || !dictionary.Contains(arg))
01127         {
01128             DoThrow(message);
01129         }
01130     }
01131
01148 #if (NET45 || NET46)
01149     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01150 #endif
01151
01152     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01153     public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
01154     {
01155         if (ReferenceEquals(dictionary, null) || dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01156         {
01157             DoThrow();
01158         }
01159     }
01160
01185 #if (NET45 || NET46)
01186     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01187 #endif
01188
01189     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01190     public static void IfIsContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,

```

```

01191         string message)
01192     {
01193         if (ReferenceEquals(dictionary, null) || dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01194         {
01195             DoThrow(message);
01196         }
01197     }
01198
01215 #if (NET45 || NET46)
01216     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01217 #endif
01218
01219     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01220     public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary)
01221     {
01222         if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01223         {
01224             DoThrow();
01225         }
01226     }
01227
01253 #if (NET45 || NET46)
01254     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01255 #endif
01256
01257     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01258     public static void IfIsNotContainedIn<TArg1, TArg2>(TArg1 arg1, TArg2 arg2,
System.Collections.Generic.IDictionary<TArg1, TArg2> dictionary,
01259         string message)
01260     {
01261         if (ReferenceEquals(dictionary, null) || !dictionary.Contains(new
System.Collections.Generic.KeyValuePair<TArg1, TArg2>(arg1, arg2)))
01262         {
01263             DoThrow(message);
01264         }
01265     }
01266
01281 #if (NET45 || NET46)
01282     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01283 #endif
01284
01285     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01286     public static void IfIsEmpty(string valueToCheck)
01287     {
01288         if (IsNullOrWhiteSpace(valueToCheck))
01289         {
01290             DoThrow();
01291         }
01292     }
01293
01316 #if (NET45 || NET46)
01317     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01318 #endif
01319
01320     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01321     public static void IfIsEmpty(string valueToCheck, string message)
01322     {
01323         if (IsNullOrWhiteSpace(valueToCheck))
01324         {
01325             DoThrow(message);
01326         }
01327     }
01328
01343 #if (NET45 || NET46)
01344     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01345 #endif
01346
01347     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01348     public static void IfIsNotEmpty(string valueToCheck)
01349     {
01350         if (!IsNullOrWhiteSpace(valueToCheck))
01351         {
01352             DoThrow();
01353         }
01354     }
01355
01378 #if (NET45 || NET46)
01379     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]

```



```

01380 #endif
01381
01382     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01383     public static void IfIsNotEmpty(string valueToCheck, string message)
01384     {
01385         if (!IsNullOrWhiteSpace(valueToCheck))
01386         {
01387             DoThrow(message);
01388         }
01389     }
01390
01405 #if (NET45 || NET46)
01406     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01407 #endif
01408
01409     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01410     public static void IfIsEmpty(System.Collections.ICollection collection)
01411     {
01412         if (ReferenceEquals(collection, null) || collection.Count == 0)
01413         {
01414             DoThrow();
01415         }
01416     }
01417
01439 #if (NET45 || NET46)
01440     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01441 #endif
01442
01443     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01444     public static void IfIsEmpty(System.Collections.ICollection collection, string
message)
01445     {
01446         if (ReferenceEquals(collection, null) || collection.Count == 0)
01447         {
01448             DoThrow(message);
01449         }
01450     }
01451
01466 #if (NET45 || NET46)
01467     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01468 #endif
01469
01470     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01471     public static void IfIsNotEmpty(System.Collections.ICollection collection)
01472     {
01473         if (ReferenceEquals(collection, null) || collection.Count > 0)
01474         {
01475             DoThrow();
01476         }
01477     }
01478
01500 #if (NET45 || NET46)
01501     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01502 #endif
01503
01504     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01505     public static void IfIsNotEmpty(System.Collections.ICollection collection, string
message)
01506     {
01507         if (ReferenceEquals(collection, null) || collection.Count > 0)
01508         {
01509             DoThrow(message);
01510         }
01511     }
01512
01527 #if (NET45 || NET46)
01528     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01529 #endif
01530
01531     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01532     public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection)
01533     {
01534         if (ReferenceEquals(collection, null) || !collection.Any())
01535         {
01536             DoThrow();
01537         }
01538     }
01539
01561 #if (NET45 || NET46)
01562     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01563 #endif

```

```

01564
01565     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01566     public static void IfIsEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection,
string message)
01567     {
01568         if (ReferenceEquals(collection, null) || !collection.Any())
01569         {
01570             DoThrow(message);
01571         }
01572     }
01573
01588 #if (NET45 || NET46)
01589     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01590 #endif
01591
01592     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01593     public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
)
01594     {
01595         if (ReferenceEquals(collection, null) || collection.Any())
01596         {
01597             DoThrow();
01598         }
01599     }
01600
01622 #if (NET45 || NET46)
01623     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01624 #endif
01625
01626     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01627     public static void IfIsNotEmpty<TArg>(System.Collections.Generic.IEnumerable<TArg> collection
, string message)
01628     {
01629         if (ReferenceEquals(collection, null) || collection.Any())
01630         {
01631             DoThrow(message);
01632         }
01633     }
01634
01650 #if (NET45 || NET46)
01651     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01652 #endif
01653
01654     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01655     public static void IfIsInstanceOf(object instance, Type type)
01656     {
01657         if (PortableTypeInfo.IsInstanceOf(instance, type))
01658         {
01659             DoThrow();
01660         }
01661     }
01662
01685 #if (NET45 || NET46)
01686     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01687 #endif
01688
01689     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01690     public static void IfIsInstanceOf(object instance, Type type, string message)
01691     {
01692         if (PortableTypeInfo.IsInstanceOf(instance, type))
01693         {
01694             DoThrow(message);
01695         }
01696     }
01697
01713 #if (NET45 || NET46)
01714     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01715 #endif
01716
01717     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01718     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01719     public static void IfIsInstanceOf<TType>(object instance)
01720     {
01721         if (instance is TType)
01722         {
01723             DoThrow();
01724         }
01725     }
01726
01749 #if (NET45 || NET46)
01750     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]

```

```

01751 #endif
01752
01753     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01754     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01755     public static void IfIsInstanceOf<TType>(object instance, string message)
01756     {
01757         if (instance is TType)
01758         {
01759             DoThrow(message);
01760         }
01761     }
01762
01778 #if (NET45 || NET46)
01779     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01780 #endif
01781
01782     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01783     public static void IfIsNotInstanceOf(object instance, Type type)
01784     {
01785         if (!PortableTypeInfo.IsInstanceOf(instance, type))
01786         {
01787             DoThrow();
01788         }
01789     }
01790
01813 #if (NET45 || NET46)
01814     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01815 #endif
01816
01817     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01818     public static void IfIsNotInstanceOf(object instance, Type type, string message)
01819     {
01820         if (!PortableTypeInfo.IsInstanceOf(instance, type))
01821         {
01822             DoThrow(message);
01823         }
01824     }
01825
01841 #if (NET45 || NET46)
01842     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01843 #endif
01844
01845     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01846     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01847     public static void IfIsNotInstanceOf<TType>(object instance)
01848     {
01849         if (!(instance is TType))
01850         {
01851             DoThrow();
01852         }
01853     }
01854
01877 #if (NET45 || NET46)
01878     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01879 #endif
01880
01881     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01882     [SuppressMessage("Microsoft.Design", "CA1004:GenericMethodsShouldProvideTypeParameter")]
01883     public static void IfIsNotInstanceOf<TType>(object instance, string message)
01884     {
01885         if (!(instance is TType))
01886         {
01887             DoThrow(message);
01888         }
01889     }
01890
01905 #if (NET45 || NET46)
01906     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01907 #endif
01908
01909     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01910     public static void IfIsNaN(double number)
01911     {
01912         if (double.IsNaN(number))
01913         {
01914             DoThrow();
01915         }
01916     }
01917
01939 #if (NET45 || NET46)
01940     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]

```

```

01941 #endif
01942
01943     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01944     public static void IfIsNaN(double number, string message)
01945     {
01946         if (double.IsNaN(number))
01947         {
01948             DoThrow(message);
01949         }
01950     }
01951
01966 #if (NET45 || NET46)
01967     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
01968 #endif
01969
01970     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
01971     public static void IfIsNotNaN(double number)
01972     {
01973         if (!double.IsNaN(number))
01974         {
01975             DoThrow();
01976         }
01977     }
01978
02001 #if (NET45 || NET46)
02002     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
02003 #endif
02004
02005     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
02006     public static void IfIsNotNaN(double number, string message)
02007     {
02008         if (!double.IsNaN(number))
02009         {
02010             DoThrow(message);
02011         }
02012     }
02013
02028 #if (NET45 || NET46)
02029     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
02030 #endif
02031
02032     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
02033     public static void IfIsNull<TArg>(TArg arg)
02034     {
02035         if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(arg, null))
02036         {
02037             DoThrow();
02038         }
02039     }
02040
02062 #if (NET45 || NET46)
02063     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
02064 #endif
02065
02066     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
02067     public static void IfIsNull<TArg>(TArg arg, string message)
02068     {
02069         if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(arg, null))
02070         {
02071             DoThrow(message);
02072         }
02073     }
02074
02089 #if (NET45 || NET46)
02090     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
02091 #endif
02092
02093     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]
02094     public static void IfIsNotNull<TArg>(TArg arg)
02095     {
02096         if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(arg, null))
02097         {
02098             DoThrow();
02099         }
02100     }
02101
02123 #if (NET45 || NET46)
02124     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
02125 #endif
02126
02127     [SuppressMessage("Microsoft.Design", "CA1000:DoNotDeclareStaticMembersOnGenericTypes")]

```

```

02128     public static void IfIsNotNull<TArg>(TArg arg, string message)
02129     {
02130         if (PortableTypeInfo.IsValueType(typeof(TArg)) || !ReferenceEquals(arg, null))
02131         {
02132             DoThrow(message);
02133         }
02134     }
02135
02136     private static ConstructorInfo GetCtor(System.Collections.Generic.IList<Type> ctorTypes)
02137     {
02138         return (from c in PortableTypeInfo.GetConstructors(typeof(TEx))
02139             let args = c.GetParameters()
02140             let zipArgs = args.Zip(ctorTypes, (argType, ctorType) => new { argType, ctorType })
02141             where args.Length == ctorTypes.Count &&
02142                 (c.IsPublic || c.IsAssembly) &&
02143                 zipArgs.All(t => ReferenceEquals(t.argType.ParameterType, t.ctorType))
02144             select c).FirstOrDefault();
02145     }
02146
02147     #if (NET45 || NET46)
02148     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
02149     MethodImplOptions.AggressiveInlining)]
02149     #endif
02150
02151     private static void DoThrow()
02152     {
02153         // Checks whether the proper constructor exists. If not, then we produce an internal exception.
02154         if (ExTypeIsAbstract)
02155         {
02156             throw ThrowerException.AbstractEx;
02157         }
02158         if (NoArgsCtor == null)
02159         {
02160             throw ThrowerException.MissingNoArgsCtor;
02161         }
02162         // A proper constructor exists: therefore, we can throw the exception.
02163         throw (TEx) NoArgsCtor.Invoke(new object[0]);
02164     }
02165
02166     #if (NET45 || NET46)
02167     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
02168     MethodImplOptions.AggressiveInlining)]
02168     #endif
02169
02170     private static void DoThrow(string message)
02171     {
02172         // Checks whether the proper constructor exists. If not, then we produce an internal exception.
02173         if (ExTypeIsAbstract)
02174         {
02175             throw ThrowerException.AbstractEx;
02176         }
02177         if (MsgCtor == null)
02178         {
02179             throw ExTypeIsAbstract ? ThrowerException.AbstractEx : ThrowerException.MissingMsgCtor;
02180         }
02181         // A proper constructor exists: therefore, we can throw the exception.
02182         var messageArgs = new object[MsgArgCount];
02183         messageArgs[0] = message;
02184         throw (TEx) MsgCtor.Invoke(messageArgs);
02185     }
02186
02187     private static bool IsNullOrWhiteSpace(string value) => value == null || string.IsNullOrEmpty(value
02188     .Trim());
02189
02190     [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
02191     public sealed class ThrowerException : Exception
02192     {
02193         [SuppressMessage("Microsoft.Design", "CA1032:ImplementStandardExceptionConstructors")]
02194         private ThrowerException(string message)
02195             : base(message)
02196         {
02197         }
02198
02199         internal static ThrowerException AbstractEx => new ThrowerException("Given exception type is
02200         abstract");
02201
02202         internal static ThrowerException MissingNoArgsCtor => new ThrowerException("Given exception type
02203         has no parameterless constructor");
02204
02205         internal static ThrowerException MissingMsgCtor => new ThrowerException("Given exception type has
02206         not a valid message constructor");
02207     }
02208 }
02209 }

```

7.3 RaiseArgumentException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseArgumentException](#)
Utility methods which can be used to handle bad arguments.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.4 RaiseArgumentException.cs

```

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 // NON-INFRINGEMENT. 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,
00022 // OUT 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 {
00030     public sealed class RaiseArgumentException : RaiseBase
00031     {
00032         #region If
00033         const string DefaultIfMessage = "Argument is not valid";
00034
00035         #if (NET45 || NET46)
00036         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00037             MethodImplOptions.AggressiveInlining)]
00038         #endif
00039         public static void If(bool condition)
00040         {
00041             if (condition)
00042             {
00043                 throw new ArgumentException(DefaultIfMessage);
00044             }
00045         }
00046
00047         #if (NET45 || NET46)
00048         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00049             MethodImplOptions.AggressiveInlining)]
00050         #endif
00051         public static void If(bool condition, string argumentName, string message = null)
00052         {
00053             if (condition)
00054             {
00055                 throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00056             }
00057         }
00058     }
00059     #endregion If
00060 }

```

```

00078         #region IfNot
00079
00084 #if (NET45 || NET46)
00085     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00086 #endif
00087
00088     public static void IfNot(bool condition)
00089     {
00090         if (!condition)
00091         {
00092             throw new ArgumentException(DefaultIfMessage);
00093         }
00094     }
00095
00105 #if (NET45 || NET46)
00106     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00107 #endif
00108
00109     public static void IfNot(bool condition, string argumentName, string message = null)
00110     {
00111         if (!condition)
00112         {
00113             throw new ArgumentException(message ?? DefaultIfMessage, argumentName);
00114         }
00115     }
00116
00117     #endregion IfNot
00118
00119     #region IfIsValid
00120
00126 #if (NET45 || NET46)
00127     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00128 #endif
00129
00130     public static void IfIsValid<TArg>(TArg argument)
00131     {
00132         IList<ValidationError> validationErrors;
00133         if (!ObjectValidator.Validate(argument, out validationErrors))
00134         {
00135             throw new ArgumentException(ObjectValidator.FormatValidationErrors(validationErrors, null))
;
00136         }
00137     }
00138
00149 #if (NET45 || NET46)
00150     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00151 #endif
00152
00153     public static void IfIsValid<TArg>(TArg argument, string argumentName, string message = null)
00154     {
00155         IList<ValidationError> validationErrors;
00156         if (!ObjectValidator.Validate(argument, out validationErrors))
00157         {
00158             throw new ArgumentException(ObjectValidator.FormatValidationErrors(validationErrors,
message), argumentName);
00159         }
00160     }
00161
00162     #endregion IfIsValid
00163
00164     #region IfIsValidEmailAddress
00165
00166     const string DefaultIfIsValidEmailAddressMessage = "String \"{0}\" is not a valid email address"
;
00167
00172 #if (NET45 || NET46)
00173     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00174 #endif
00175
00176     public static void IfIsValidEmailAddress(string emailAddress)
00177     {
00178         if (!EmailAddressValidator.Validate(emailAddress))
00179         {
00180             var exceptionMsg = string.Format(DefaultIfIsValidEmailAddressMessage, emailAddress);
00181             throw new ArgumentException(exceptionMsg);
00182         }
00183     }
00184
00192 #if (NET45 || NET46)
00193     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00194 #endif

```

```

00195
00196     public static void IfIsValidEmailMessage(string emailAddress, bool
allowInternational)
00197     {
00198         if (!EmailAddressValidator.Validate(emailAddress, allowInternational))
00199         {
00200             var exceptionMsg = string.Format(DefaultIfIsValidEmailMessage, emailAddress);
00201             throw new ArgumentException(exceptionMsg);
00202         }
00203     }
00204
00214 #if (NET45 || NET46)
00215     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00216 #endif
00217
00218     public static void IfIsValidEmailMessage(string emailAddress, string
argumentName, string message = null)
00219     {
00220         if (!EmailAddressValidator.Validate(emailAddress))
00221         {
00222             var exceptionMsg = message ?? string.Format(DefaultIfIsValidEmailMessage,
emailAddress);
00223             throw new ArgumentException(exceptionMsg, argumentName);
00224         }
00225     }
00226
00239 #if (NET45 || NET46)
00240     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00241 #endif
00242
00243     public static void IfIsValidEmailMessage(string emailAddress, bool
allowInternational, string argumentName, string message = null)
00244     {
00245         if (!EmailAddressValidator.Validate(emailAddress, allowInternational))
00246         {
00247             var exceptionMsg = message ?? string.Format(DefaultIfIsValidEmailMessage,
emailAddress);
00248             throw new ArgumentException(exceptionMsg, argumentName);
00249         }
00250     }
00251
00252     #endregion IfIsValidEmailMessage
00253
00254     #region IfIsValidPhoneNumber
00255
00256     const string DefaultIfIsValidPhoneNumberMessage = "String \"{0}\" is not a valid phone number";
00257
00262 #if (NET45 || NET46)
00263     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00264 #endif
00265
00266     public static void IfIsValidPhoneNumber(string phoneNumber)
00267     {
00268         if (!PhoneNumberValidator.Validate(phoneNumber))
00269         {
00270             var exceptionMsg = string.Format(DefaultIfIsValidPhoneNumberMessage, phoneNumber);
00271             throw new ArgumentException(exceptionMsg);
00272         }
00273     }
00274
00284 #if (NET45 || NET46)
00285     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00286 #endif
00287
00288     public static void IfIsValidPhoneNumber(string phoneNumber, string
argumentName, string message = null)
00289     {
00290         if (!PhoneNumberValidator.Validate(phoneNumber))
00291         {
00292             var exceptionMsg = message ?? string.Format(DefaultIfIsValidPhoneNumberMessage,
phoneNumber);
00293             throw new ArgumentException(exceptionMsg, argumentName);
00294         }
00295     }
00296
00297     #endregion IfIsValidPhoneNumber
00298
00299     #region String validation
00300
00301     const string IsNullOrEmptyMessage = "Argument cannot be a null or empty string";
00302     const string IsNullOrWhiteSpaceMessage = "Argument cannot be a null, empty or blank string";
00303
00308 #if (NET45 || NET46)

```



```

00309         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00310     #endif
00311
00312     public static void IfIsNullOrEmpty(string value)
00313     {
00314         if (value == null || value == string.Empty)
00315         {
00316             throw new ArgumentException(IsNullOrEmptyMessage);
00317         }
00318     }
00319
00320 #if (NET45 || NET46)
00321     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00322 #endif
00323     public static void IfIsNullOrEmpty(string value, string argumentName, string message
= null)
00324     {
00325         if (value == null || value == string.Empty)
00326         {
00327             throw new ArgumentException(message ?? IsNullOrEmptyMessage, argumentName);
00328         }
00329     }
00330
00331 #if (NET45 || NET46)
00332     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00333 #endif
00334     public static void IfIsNullOrWhiteSpace(string value)
00335     {
00336         if (value == null || value.Trim() == string.Empty)
00337         {
00338             throw new ArgumentException(IsNullOrWhiteSpaceMessage);
00339         }
00340     }
00341
00342 #if (NET45 || NET46)
00343     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00344 #endif
00345     public static void IfIsNullOrWhiteSpace(string value, string argumentName,
string message = null)
00346     {
00347         if (value == null || value.Trim() == string.Empty)
00348         {
00349             throw new ArgumentException(message ?? IsNullOrWhiteSpaceMessage, argumentName);
00350         }
00351     }
00352
00353 #endregion String validation
00354 }
00355 }

```

7.5 RaiseArgumentNullException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseArgumentNullException](#)
Utility methods which can be used to handle null references.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.6 RaiseArgumentNullException.cs

```

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,
00022 // OUT 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 {
00032     public sealed class RaiseArgumentNullException :
        RaiseBase
00033     {
00039         #if (NET45 || NET46)
00040             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00041         #endif
00042
00043         public static void IfIsNull<TArg>(TArg argument)
00044         {
00045             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
00046             {
00047                 throw new ArgumentNullException();
00048             }
00049         }
00050
00057         #if (NET45 || NET46)
00058             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00059         #endif
00060
00061         public static void IfIsNull<TArg>(TArg argument, string argumentName)
00062         {
00063             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
00064             {
00065                 throw new ArgumentNullException(argumentName);
00066             }
00067         }
00068
00076         #if (NET45 || NET46)
00077             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00078         #endif
00079
00080         public static void IfIsNull<TArg>(TArg argument, string argumentName, string message)
00081         {
00082             if (!PortableTypeInfo.IsValueType(typeof(TArg)) && ReferenceEquals(argument, null))
00083             {
00084                 throw new ArgumentNullException(argumentName, message);
00085             }
00086         }
00087     }
00088 }

```

7.7 RaiseArgumentOutOfRangeException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseArgumentOutOfRangeException](#)

Utility methods which can be used to handle ranges.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.8 RaiseArgumentOutOfRangeException.cs

```

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,
00022 // OUT 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 {
00031     public sealed class RaiseArgumentOutOfRangeException :
        RaiseBase
00032     {
00033         #region If
00034
00040         #if (NET45 || NET46)
00041             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00042         #endif
00043
00044         public static void If(bool condition, string argumentName = null)
00045         {
00046             if (condition)
00047             {
00048                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
                ArgumentOutOfRangeException(argumentName);
00049             }
00050         }
00051
00061         #if (NET45 || NET46)
00062             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00063         #endif
00064
00065         public static void If(bool condition, string argumentName, string message)
00066         {
00067             if (condition)
00068             {
00069                 throw new ArgumentOutOfRangeException(argumentName, message);
00070             }
00071         }
00072
00073         #endregion If
00074
00075         #region IfNot
00076
00082         #if (NET45 || NET46)
00083             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00084         #endif
00085
00086         public static void IfNot(bool condition, string argumentName = null)
00087         {
00088             if (!condition)
00089             {
00090                 throw string.IsNullOrEmpty(argumentName) ? new ArgumentOutOfRangeException() : new
                ArgumentOutOfRangeException(argumentName);
00091             }
00092         }

```

```

00093
00103 #if (NET45 || NET46)
00104     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00105 #endif
00106
00107     public static void IfNot(bool condition, string argumentName, string message)
00108     {
00109         if (!condition)
00110         {
00111             throw new ArgumentOutOfRangeException(argumentName, message);
00112         }
00113     }
00114
00115     #endregion IfNot
00116
00117     #region Less - Without parameter name, without message
00118
00126 #if (NET45 || NET46)
00127     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00128 #endif
00129
00130     public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
00131         where TArg : IComparable<TArg>
00132     {
00133         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00134         {
00135             throw new ArgumentOutOfRangeException();
00136         }
00137     }
00138
00145 #if (NET45 || NET46)
00146     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00147 #endif
00148
00149     public static void IfIsLess(IComparable argument1, IComparable argument2)
00150     {
00151         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00152         {
00153             throw new ArgumentOutOfRangeException();
00154         }
00155     }
00156
00157     #endregion Less - Without parameter name, without message
00158
00159     #region Less - With parameter name, without message
00160
00169 #if (NET45 || NET46)
00170     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00171 #endif
00172
00173     public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName)
00174         where TArg : IComparable<TArg>
00175     {
00176         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00177         {
00178             throw new ArgumentOutOfRangeException(argumentName);
00179         }
00180     }
00181
00189 #if (NET45 || NET46)
00190     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00191 #endif
00192
00193     public static void IfIsLess(IComparable argument1, IComparable argument2, string
argumentName)
00194     {
00195         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00196         {
00197             throw new ArgumentOutOfRangeException(argumentName);
00198         }
00199     }
00200
00201     #endregion Less - With parameter name, without message
00202
00203     #region Less - With parameter name, with message
00204
00214 #if (NET45 || NET46)
00215     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00216 #endif
00217
00218     public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string argumentName, string

```

```

message)
00219     where TArg : IComparable<TArg>
00220     {
00221         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00222         {
00223             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00224         }
00225     }
00226
00235 #if (NET45 || NET46)
00236     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00237 #endif
00238
00239     public static void IfIsLess(IComparable argument1, IComparable argument2, string
argumentName, string message)
00240     {
00241         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00242         {
00243             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00244         }
00245     }
00246
00247     #endregion Less - With parameter name, with message
00248
00249     #region LessEqual - Without parameter name, without message
00250
00258 #if (NET45 || NET46)
00259     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00260 #endif
00261
00262     public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00263     where TArg : IComparable<TArg>
00264     {
00265         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00266         {
00267             throw new ArgumentOutOfRangeException();
00268         }
00269     }
00270
00277 #if (NET45 || NET46)
00278     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00279 #endif
00280
00281     public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00282     {
00283         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00284         {
00285             throw new ArgumentOutOfRangeException();
00286         }
00287     }
00288
00289     #endregion LessEqual - Without parameter name, without message
00290
00291     #region LessEqual - With parameter name, without message
00292
00301 #if (NET45 || NET46)
00302     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00303 #endif
00304
00305     public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00306     where TArg : IComparable<TArg>
00307     {
00308         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00309         {
00310             throw new ArgumentOutOfRangeException(argumentName);
00311         }
00312     }
00313
00321 #if (NET45 || NET46)
00322     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00323 #endif
00324
00325     public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
string argumentName)
00326     {
00327         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00328         {
00329             throw new ArgumentOutOfRangeException(argumentName);
00330         }
00331     }
00332
00333     #endregion LessEqual - With parameter name, without message

```

```

00334
00335     #region LessEqual - With parameter name, with message
00336
00346 #if (NET45 || NET46)
00347     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00348 #endif
00349
00350     public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
string message)
00351     {
00352         where TArg : IComparable<TArg>
00353         {
00354             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00355             {
00356                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00357             }
00358         }
00359     }
00367 #if (NET45 || NET46)
00368     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00369 #endif
00370
00371     public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
string argumentName, string message)
00372     {
00373         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00374         {
00375             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00376         }
00377     }
00378
00379     #endregion LessEqual - With parameter name, with message
00380
00381     #region Greater - Without parameter name, without message
00382
00390 #if (NET45 || NET46)
00391     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00392 #endif
00393
00394     public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00395     {
00396         where TArg : IComparable<TArg>
00397         {
00398             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00399             {
00400                 throw new ArgumentOutOfRangeException();
00401             }
00402         }
00403     }
00409 #if (NET45 || NET46)
00410     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00411 #endif
00412
00413     public static void IfIsGreater(IComparable argument1, IComparable argument2)
00414     {
00415         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00416         {
00417             throw new ArgumentOutOfRangeException();
00418         }
00419     }
00420
00421     #endregion Greater - Without parameter name, without message
00422
00423     #region Greater - With parameter name, without message
00424
00433 #if (NET45 || NET46)
00434     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00435 #endif
00436
00437     public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName)
00438     {
00439         where TArg : IComparable<TArg>
00440         {
00441             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00442             {
00443                 throw new ArgumentOutOfRangeException(argumentName);
00444             }
00445         }
00446     }
00453 #if (NET45 || NET46)
00454     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00455 #endif
00456
00457     public static void IfIsGreater(IComparable argument1, IComparable argument2, string

```

```

        argumentName)
00458     {
00459         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00460         {
00461             throw new ArgumentOutOfRangeException(argumentName);
00462         }
00463     }
00464
00465     #endregion Greater - With parameter name, without message
00466     #region Greater - With parameter name, with message
00467
00468     #if (NET45 || NET46)
00478     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00479     MethodImplOptions.AggressiveInlining)]
00480     #endif
00481
00482     public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string argumentName, string
    message)
00483     {
00484         where TArg : IComparable<TArg>
00485         {
00486             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00487             {
00488                 throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00489             }
00490         }
00491     }
00492     #if (NET45 || NET46)
00500     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00501     MethodImplOptions.AggressiveInlining)]
00502     #endif
00503     public static void IfIsGreater(IComparable argument1, IComparable argument2, string
    argumentName, string message)
00504     {
00505         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00506         {
00507             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00508         }
00509     }
00510
00511     #endregion Greater - With parameter name, with message
00512     #region GreaterEqual - Without parameter name, without message
00513
00514     #if (NET45 || NET46)
00522     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00523     MethodImplOptions.AggressiveInlining)]
00524     #endif
00525
00526     public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)
00527     {
00528         where TArg : IComparable<TArg>
00529         {
00530             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00531             {
00532                 throw new ArgumentOutOfRangeException();
00533             }
00534         }
00535     }
00536     #if (NET45 || NET46)
00542     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00543     MethodImplOptions.AggressiveInlining)]
00544     #endif
00545     public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
    argument2)
00546     {
00547         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00548         {
00549             throw new ArgumentOutOfRangeException();
00550         }
00551     }
00552
00553     #endregion GreaterEqual - Without parameter name, without message
00554     #region GreaterEqual - With parameter name, without message
00555
00556     #if (NET45 || NET46)
00566     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00567     MethodImplOptions.AggressiveInlining)]
00568     #endif
00569     public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00570     {
00571         where TArg : IComparable<TArg>
00572         {
00573             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00574         {

```

```

00574         throw new ArgumentOutOfRangeException(argumentName);
00575     }
00576 }
00577
00585 #if (NET45 || NET46)
00586     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00587 #endif
00588
00589     public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
argument2, string argumentName)
00590     {
00591         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00592         {
00593             throw new ArgumentOutOfRangeException(argumentName);
00594         }
00595     }
00596
00597     #endregion GreaterEqual - With parameter name, without message
00598
00599     #region GreaterEqual - With parameter name, with message
00600
00610 #if (NET45 || NET46)
00611     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00612 #endif
00613
00614     public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string argumentName,
string message)
00615     where TArg : IComparable<TArg>
00616     {
00617         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00618         {
00619             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00620         }
00621     }
00622
00631 #if (NET45 || NET46)
00632     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00633 #endif
00634
00635     public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
argument2, string argumentName, string message)
00636     {
00637         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00638         {
00639             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00640         }
00641     }
00642
00643     #endregion GreaterEqual - With parameter name, with message
00644
00645     #region Equal - Without parameter name, without message
00646
00654 #if (NET45 || NET46)
00655     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00656 #endif
00657
00658     public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00659     where TArg : IComparable<TArg>
00660     {
00661         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00662         {
00663             throw new ArgumentOutOfRangeException();
00664         }
00665     }
00666
00673 #if (NET45 || NET46)
00674     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00675 #endif
00676
00677     public static void IfIsEqual(IComparable argument1, IComparable argument2)
00678     {
00679         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00680         {
00681             throw new ArgumentOutOfRangeException();
00682         }
00683     }
00684
00685     #endregion Equal - Without parameter name, without message
00686
00687     #region Equal - With parameter name, without message
00688
00697 #if (NET45 || NET46)

```



```

00698         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00699     #endif
00700
00701     public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00702     where TArg : IComparable<TArg>
00703     {
00704         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00705         {
00706             throw new ArgumentOutOfRangeException(argumentName);
00707         }
00708     }
00709
00710     #if (NET45 || NET46)
00711     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00712     #endif
00713
00714     public static void IfIsEqual(IComparable argument1, IComparable argument2, string
argumentName)
00715     {
00716         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00717         {
00718             throw new ArgumentOutOfRangeException(argumentName);
00719         }
00720     }
00721
00722     #endregion Equal - With parameter name, without message
00723
00724     #region Equal - With parameter name, with message
00725
00726     #if (NET45 || NET46)
00727     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00728     #endif
00729
00730     public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
00731     where TArg : IComparable<TArg>
00732     {
00733         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00734         {
00735             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00736         }
00737     }
00738
00739     #if (NET45 || NET46)
00740     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00741     #endif
00742
00743     public static void IfIsEqual(IComparable argument1, IComparable argument2, string
argumentName, string message)
00744     {
00745         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00746         {
00747             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00748         }
00749     }
00750
00751     #endregion Equal - With parameter name, with message
00752
00753     #region NotEqual - Without parameter name, without message
00754
00755     #if (NET45 || NET46)
00756     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00757     #endif
00758
00759     public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00760     where TArg : IComparable<TArg>
00761     {
00762         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00763         {
00764             throw new ArgumentOutOfRangeException();
00765         }
00766     }
00767
00768     #if (NET45 || NET46)
00769     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00770     #endif
00771
00772     public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
00773     {
00774         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00775         {
00776             throw new ArgumentOutOfRangeException();
00777         }
00778     }

```

```

00813         throw new ArgumentOutOfRangeException();
00814     }
00815 }
00816
00817 #endregion NotEqual - Without parameter name, without message
00818
00819 #region NotEqual - With parameter name, without message
00820
00829 #if (NET45 || NET46)
00830     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00831 #endif
00832
00833     public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName)
00834     where TArg : IComparable<TArg>
00835     {
00836         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00837         {
00838             throw new ArgumentOutOfRangeException(argumentName);
00839         }
00840     }
00841
00849 #if (NET45 || NET46)
00850     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00851 #endif
00852
00853     public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
argumentName)
00854     {
00855         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00856         {
00857             throw new ArgumentOutOfRangeException(argumentName);
00858         }
00859     }
00860
00861 #endregion NotEqual - With parameter name, without message
00862
00863 #region NotEqual - With parameter name, with message
00864
00874 #if (NET45 || NET46)
00875     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00876 #endif
00877
00878     public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string argumentName, string
message)
00879     where TArg : IComparable<TArg>
00880     {
00881         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00882         {
00883             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00884         }
00885     }
00886
00895 #if (NET45 || NET46)
00896     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00897 #endif
00898
00899     public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
argumentName, string message)
00900     {
00901         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00902         {
00903             throw new ArgumentOutOfRangeException(argumentName, argument1, message);
00904         }
00905     }
00906
00907 #endregion NotEqual - With parameter name, with message
00908 }
00909 }

```

7.9 RaiseHttpException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseHttpException](#)
Utility methods which can be used to handle error codes through HTTP.
- struct [PommaLabs.Thrower.HttpExceptionInfo](#)

Additional info which will be included into [HttpException](#).

- class [PommaLabs.Thrower.HttpException](#)

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

Namespaces

- namespace [PommaLabs.Thrower](#)

7.10 RaiseHttpException.cs

```

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,
00022 // OUT 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.Net;
00027
00028 namespace PommaLabs.Thrower
00029 {
00030     public static class RaiseHttpException
00031     {
00041         #if (NET45 || NET46)
00042             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00043                 MethodImplOptions.AggressiveInlining)]
00044             #endif
00045             public static void If(bool condition, HttpStatusCode httpStatusCode, string message = null)
00046             {
00047                 if (condition)
00048                 {
00049                     throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
00050                     HttpException(httpStatusCode, message);
00051                 }
00052             }
00060             #if (NET45 || NET46)
00061                 [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00062                     MethodImplOptions.AggressiveInlining)]
00063             #endif
00064             public static void If(bool condition, HttpStatusCode httpStatusCode, string message,
00065                 HttpExceptionInfo additionalInfo)
00066             {
00067                 if (condition)
00068                 {
00069                     throw new HttpException(httpStatusCode, message, additionalInfo);
00070                 }
00071             }
00078             #if (NET45 || NET46)
00079                 [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00080                     MethodImplOptions.AggressiveInlining)]
00081             #endif
00082             public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message = null)
00083             {
00084                 if (!condition)

```

```

00085         {
00086             throw string.IsNullOrEmpty(message) ? new HttpException(httpStatusCode) : new
HttpException(httpStatusCode, message);
00087         }
00088     }
00089
00097 #if (NET45 || NET46)
00098     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00099 #endif
00100
00101     public static void IfNot(bool condition, HttpStatusCode httpStatusCode, string message,
HttpExceptionInfo additionalInfo)
00102     {
00103         if (!condition)
00104         {
00105             throw new HttpException(httpStatusCode, message, additionalInfo);
00106         }
00107     }
00108 }
00109
00113 public struct HttpExceptionInfo
00114 {
00120     public HttpExceptionInfo(object errorCode = null, string userMessage = null)
00121     {
00122         ErrorCode = errorCode ?? HttpException.DefaultErrorCode;
00123         UserMessage = userMessage ?? HttpException.
DefaultUserMessage;
00124     }
00125
00129     [Validate(Required = false)]
00130     public object ErrorCode { get; set; }
00131
00135     [Validate(Required = false)]
00136     public string UserMessage { get; set; }
00137 }
00138
00143 public sealed class HttpException : Exception
00144 {
00149     public HttpException(HttpStatusCode httpStatusCode)
00150         : this(httpStatusCode, new HttpExceptionInfo())
00151     {
00152     }
00153
00159     public HttpException(HttpStatusCode httpStatusCode,
HttpExceptionInfo additionalInfo)
00160         : base()
00161     {
00162         HttpStatusCode = httpStatusCode;
00163         ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00164         UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00165     }
00166
00172     public HttpException(HttpStatusCode httpStatusCode, string message)
00173         : this(httpStatusCode, message, new HttpExceptionInfo())
00174     {
00175     }
00176
00183     public HttpException(HttpStatusCode httpStatusCode, string message,
HttpExceptionInfo additionalInfo)
00184         : base(message)
00185     {
00186         HttpStatusCode = httpStatusCode;
00187         ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00188         UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00189     }
00190
00197     public HttpException(HttpStatusCode httpStatusCode, string message, Exception
innerException)
00198         : this(httpStatusCode, message, innerException, new
HttpExceptionInfo())
00199     {
00200     }
00201
00209     public HttpException(HttpStatusCode httpStatusCode, string message, Exception
innerException, HttpExceptionInfo additionalInfo)
00210         : base(message, innerException)
00211     {
00212         HttpStatusCode = httpStatusCode;
00213         ErrorCode = additionalInfo.ErrorCode ?? DefaultErrorCode;
00214         UserMessage = additionalInfo.UserMessage ?? DefaultUserMessage;
00215     }
00216
00220     public HttpStatusCode HttpStatusCode { get; }
00221
00225     public object ErrorCode { get; }
00226

```

```

00230         public static object DefaultErrorCode { get; set; } = "unspecified";
00231
00235         public string UserMessage { get; }
00236
00240         public static string DefaultUserMessage { get; set; } = "unspecified";
00241     }
00242 }

```

7.11 RaiseIndexOutOfRangeException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseIndexOutOfRangeException](#)

Utility methods which can be used to handle indexes.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.12 RaiseIndexOutOfRangeException.cs

```

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,
00022 // OUT 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 {
00031     public sealed class RaiseIndexOutOfRangeException :
        RaiseBase
00032     {
00033         #region Less - Without message
00034
00042 #if (NET45 || NET46)
00043         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00044 #endif
00045
00046         public static void IfIsLess<TArg>(TArg argument1, TArg argument2)
00047             where TArg : IComparable<TArg>
00048         {
00049             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00050             {
00051                 throw new IndexOutOfRangeException();
00052             }
00053         }
00054
00061 #if (NET45 || NET46)
00062         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
        MethodImplOptions.AggressiveInlining)]
00063 #endif
00064
00065         public static void IfIsLess(IComparable argument1, IComparable argument2)

```

```

00066         {
00067             if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00068             {
00069                 throw new IndexOutOfRangeException();
00070             }
00071         }
00072     #endregion Less - Without message
00073
00074     #region Less - With message
00075
00076 #if (NET45 || NET46)
00085     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00086     MethodImplOptions.AggressiveInlining)]
00087 #endif
00088
00089     public static void IfIsLess<TArg>(TArg argument1, TArg argument2, string message)
00090     where TArg : IComparable<TArg>
00091     {
00092         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00093         {
00094             throw new IndexOutOfRangeException(message);
00095         }
00096     }
00097
00105 #if (NET45 || NET46)
00106     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00107     MethodImplOptions.AggressiveInlining)]
00108 #endif
00109     public static void IfIsLess(IComparable argument1, IComparable argument2, string message)
00110     {
00111         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) < 0)
00112         {
00113             throw new IndexOutOfRangeException(message);
00114         }
00115     }
00116
00117     #endregion Less - With message
00118
00119     #region LessEqual - Without message
00120
00128 #if (NET45 || NET46)
00129     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00130     MethodImplOptions.AggressiveInlining)]
00131 #endif
00132     public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2)
00133     where TArg : IComparable<TArg>
00134     {
00135         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00136         {
00137             throw new IndexOutOfRangeException();
00138         }
00139     }
00140
00147 #if (NET45 || NET46)
00148     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00149     MethodImplOptions.AggressiveInlining)]
00150 #endif
00151     public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2)
00152     {
00153         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00154         {
00155             throw new IndexOutOfRangeException();
00156         }
00157     }
00158
00159     #endregion LessEqual - Without message
00160
00161     #region LessEqual - With message
00162
00171 #if (NET45 || NET46)
00172     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00173     MethodImplOptions.AggressiveInlining)]
00174 #endif
00175     public static void IfIsLessOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00176     where TArg : IComparable<TArg>
00177     {
00178         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00179         {
00180             throw new IndexOutOfRangeException(message);
00181         }
00182     }
00183

```

```

00191 #if (NET45 || NET46)
00192     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00193 #endif
00194
00195     public static void IfIsLessOrEqual(IComparable argument1, IComparable argument2,
string message)
00196     {
00197         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) <= 0)
00198         {
00199             throw new IndexOutOfRangeException(message);
00200         }
00201     }
00202
00203     #endregion LessEqual - With message
00204
00205     #region Greater - Without message
00206
00214 #if (NET45 || NET46)
00215     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00216 #endif
00217
00218     public static void IfIsGreater<TArg>(TArg argument1, TArg argument2)
00219         where TArg : IComparable<TArg>
00220     {
00221         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00222         {
00223             throw new IndexOutOfRangeException();
00224         }
00225     }
00226
00233 #if (NET45 || NET46)
00234     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00235 #endif
00236
00237     public static void IfIsGreater(IComparable argument1, IComparable argument2)
00238     {
00239         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00240         {
00241             throw new IndexOutOfRangeException();
00242         }
00243     }
00244
00245     #endregion Greater - Without message
00246
00247     #region Greater - With message
00248
00257 #if (NET45 || NET46)
00258     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00259 #endif
00260
00261     public static void IfIsGreater<TArg>(TArg argument1, TArg argument2, string message)
00262         where TArg : IComparable<TArg>
00263     {
00264         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00265         {
00266             throw new IndexOutOfRangeException(message);
00267         }
00268     }
00269
00277 #if (NET45 || NET46)
00278     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00279 #endif
00280
00281     public static void IfIsGreater(IComparable argument1, IComparable argument2, string
message)
00282     {
00283         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) > 0)
00284         {
00285             throw new IndexOutOfRangeException(message);
00286         }
00287     }
00288
00289     #endregion Greater - With message
00290
00291     #region GreaterEqual - Without message
00292
00300 #if (NET45 || NET46)
00301     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00302 #endif
00303
00304     public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2)

```

```

00305         where TArg : IComparable<TArg>
00306     {
00307         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00308         {
00309             throw new IndexOutOfRangeException();
00310         }
00311     }
00312
00319 #if (NET45 || NET46)
00320     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00321         MethodImplOptions.AggressiveInlining)]
00322 #endif
00323     public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
00324         argument2)
00325     {
00326         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00327         {
00328             throw new IndexOutOfRangeException();
00329         }
00330     }
00331     #endregion GreaterEqual - Without message
00332     #region GreaterEqual - With message
00333 #if (NET45 || NET46)
00334     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00335         MethodImplOptions.AggressiveInlining)]
00336 #endif
00337     public static void IfIsGreaterOrEqual<TArg>(TArg argument1, TArg argument2, string message)
00338         where TArg : IComparable<TArg>
00339     {
00340         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00341         {
00342             throw new IndexOutOfRangeException(message);
00343         }
00344     }
00345 #if (NET45 || NET46)
00346     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00347         MethodImplOptions.AggressiveInlining)]
00348 #endif
00349     public static void IfIsGreaterOrEqual(IComparable argument1, IComparable
00350         argument2, string message)
00351     {
00352         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) >= 0)
00353         {
00354             throw new IndexOutOfRangeException(message);
00355         }
00356     }
00357     #endregion GreaterEqual - With message
00358     #region Equal - Without message
00359 #if (NET45 || NET46)
00360     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00361         MethodImplOptions.AggressiveInlining)]
00362 #endif
00363     public static void IfIsEqual<TArg>(TArg argument1, TArg argument2)
00364         where TArg : IComparable<TArg>
00365     {
00366         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00367         {
00368             throw new IndexOutOfRangeException();
00369         }
00370     }
00371 #if (NET45 || NET46)
00372     [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00373         MethodImplOptions.AggressiveInlining)]
00374 #endif
00375     public static void IfIsEqual(IComparable argument1, IComparable argument2)
00376     {
00377         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00378         {
00379             throw new IndexOutOfRangeException();
00380         }
00381     }
00382     #endregion Equal - Without message

```



```

00419         #region Equal - With message
00420
00429     #if (NET45 || NET46)
00430         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00431             MethodImplOptions.AggressiveInlining)]
00432     #endif
00433     public static void IfIsEqual<TArg>(TArg argument1, TArg argument2, string message)
00434     {
00435         where TArg : IComparable<TArg>
00436     {
00437         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00438         {
00439             throw new IndexOutOfRangeException(message);
00440         }
00441     }
00442
00449     #if (NET45 || NET46)
00450         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00451             MethodImplOptions.AggressiveInlining)]
00452     #endif
00453     public static void IfIsEqual(IComparable argument1, IComparable argument2, string message)
00454     {
00455         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) == 0)
00456         {
00457             throw new IndexOutOfRangeException(message);
00458         }
00459     }
00460
00461     #endregion Equal - With message
00462
00463     #region NotEqual - Without message
00464
00472     #if (NET45 || NET46)
00473         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00474             MethodImplOptions.AggressiveInlining)]
00475     #endif
00476     public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2)
00477     {
00478         where TArg : IComparable<TArg>
00479     {
00480         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00481         {
00482             throw new IndexOutOfRangeException();
00483         }
00484     }
00485
00491     #if (NET45 || NET46)
00492         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00493             MethodImplOptions.AggressiveInlining)]
00494     #endif
00495     public static void IfIsNotEqual(IComparable argument1, IComparable argument2)
00496     {
00497         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00498         {
00499             throw new IndexOutOfRangeException();
00500         }
00501     }
00502
00503     #endregion NotEqual - Without message
00504
00505     #region NotEqual - With message
00506
00515     #if (NET45 || NET46)
00516         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00517             MethodImplOptions.AggressiveInlining)]
00518     #endif
00519     public static void IfIsNotEqual<TArg>(TArg argument1, TArg argument2, string message)
00520     {
00521         where TArg : IComparable<TArg>
00522     {
00523         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)
00524         {
00525             throw new IndexOutOfRangeException(message);
00526         }
00527     }
00528
00535     #if (NET45 || NET46)
00536         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
00537             MethodImplOptions.AggressiveInlining)]
00538     #endif
00539     public static void IfIsNotEqual(IComparable argument1, IComparable argument2, string
00540         message)
00541     {
00542         if (ReferenceEquals(argument1, null) || argument1.CompareTo(argument2) != 0)

```

```

00542         {
00543             throw new IndexOutOfRangeException(message);
00544         }
00545     }
00546
00547     #endregion NotEqual - With message
00548 }
00549 }

```

7.13 RaiseInvalidOperationException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseInvalidOperationException](#)
Utility methods which can be used to handle bad object states.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.14 RaiseInvalidOperationException.cs

```

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,
00022 // OUT 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 {
00031     public sealed class RaiseInvalidOperationException :
        RaiseBase
00032     {
00038         #if (NET45 || NET46)
00039             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00040         #endif
00041
00042         public static void If(bool condition, string message = null)
00043         {
00044             if (condition)
00045             {
00046                 throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new
                InvalidOperationException(message);
00047             }
00048         }
00049
00055         #if (NET45 || NET46)
00056             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
                MethodImplOptions.AggressiveInlining)]
00057         #endif
00058
00059         public static void IfNot(bool condition, string message = null)
00060         {

```

```

00061         if (!condition)
00062         {
00063             throw string.IsNullOrEmpty(message) ? new InvalidOperationException() : new
InvalidOperationException(message);
00064         }
00065     }
00066 }
00067 }

```

7.15 RaiseNotSupportedException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseNotSupportedException](#)

Utility methods which can be used to handle unsupported operations.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.16 RaiseNotSupportedException.cs

```

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,
00022 // OUT 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 {
00031     public sealed class RaiseNotSupportedException :
RaiseBase
00032     {
00038         #if (NET45 || NET46)
00039         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00040         #endif
00041
00042         public static void If(bool condition, string message = null)
00043         {
00044             if (condition)
00045             {
00046                 throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new
NotSupportedException(message);
00047             }
00048         }
00049
00055         #if (NET45 || NET46)
00056         [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00057         #endif
00058
00059         public static void IfNot(bool condition, string message = null)
00060         {

```

```

00061         if (!condition)
00062         {
00063             throw string.IsNullOrEmpty(message) ? new NotSupportedException() : new
NotSupportedException(message);
00064         }
00065     }
00066 }
00067 }

```

7.17 RaiseObjectDisposedException.cs File Reference

Classes

- class [PommaLabs.Thrower.RaiseObjectDisposedException](#)
Utility methods which can be used to handle bad object states.

Namespaces

- namespace [PommaLabs.Thrower](#)

7.18 RaiseObjectDisposedException.cs

```

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,
00022 // OUT 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 {
00031     public sealed class RaiseObjectDisposedException :
RaiseBase
00032     {
00039         #if (NET45 || NET46)
00040             [System.Runtime.CompilerServices.MethodImpl(System.Runtime.CompilerServices.
MethodImplOptions.AggressiveInlining)]
00041         #endif
00042
00043         public static void If(bool disposed, string objectName, string message = null)
00044         {
00045             if (disposed)
00046             {
00047                 throw string.IsNullOrEmpty(message) ? new ObjectDisposedException(objectName) : new
ObjectDisposedException(objectName, message);
00048             }
00049         }
00050     }
00051 }

```

Index

- DefaultErrorCode
 - PommaLabs::Thrower::HttpException, [13](#)
- DefaultUserMessage
 - PommaLabs::Thrower::HttpException, [14](#)
- ErrorCode
 - PommaLabs::Thrower::HttpException, [14](#)
 - PommaLabs::Thrower::HttpExceptionInfo, [15](#)
- HttpException
 - PommaLabs::Thrower::HttpException, [12](#), [13](#)
- HttpExceptionInfo
 - PommaLabs::Thrower::HttpExceptionInfo, [15](#)
- HttpStatusCode
 - PommaLabs::Thrower::HttpException, [14](#)
- If
 - PommaLabs::Thrower::Raise, [19](#)
 - PommaLabs::Thrower::RaiseArgumentException, [50](#)
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [58](#), [59](#)
 - PommaLabs::Thrower::RaiseHttpException, [74](#), [75](#)
 - PommaLabs::Thrower::RaiseInvalidOperationException, [90](#)
 - PommaLabs::Thrower::RaiseNotSupportedException, [91](#)
 - PommaLabs::Thrower::RaiseObjectDisposedException, [93](#)
- IfAreEqual< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [21](#)
- IfAreNotEqual< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [21](#), [22](#)
- IfAreNotSame< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [22](#)
- IfAreSame< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [24](#)
- IfIsAssignableFrom
 - PommaLabs::Thrower::Raise, [25](#)
- IfIsAssignableFrom< TType >
 - PommaLabs::Thrower::Raise, [25](#), [26](#)
- IfIsContainedIn
 - PommaLabs::Thrower::Raise, [26](#)
- IfIsContainedIn< TArg >
 - PommaLabs::Thrower::Raise, [27](#), [28](#)
- IfIsContainedIn< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [28](#), [30](#)
- IfIsEmpty
 - PommaLabs::Thrower::Raise, [30](#), [32](#)
- IfIsEmpty< TArg >
 - PommaLabs::Thrower::Raise, [32](#), [34](#)
- IfIsEqual
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [59](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [77](#), [78](#)
- IfIsEqual< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [60](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [78](#)
- IfIsGreater
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [61](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [78](#), [80](#)
- IfIsGreater< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [61](#), [62](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [80](#)
- IfIsGreaterOrEqual
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [62](#), [64](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [80](#), [82](#)
- IfIsGreaterOrEqual< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [64](#), [65](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [82](#)
- IfIsInstanceOf
 - PommaLabs::Thrower::Raise, [34](#)
- IfIsInstanceOf< TType >
 - PommaLabs::Thrower::Raise, [35](#)
- IfIsLess
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [65](#), [66](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [82](#), [84](#)
- IfIsLess< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [66](#), [67](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [84](#)
- IfIsLessOrEqual
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException, [67](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException, [84](#)

- Exception, [84](#), [86](#)
- IfIsLessOrEqual< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException↔
 - RangeException, [68](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException↔
 - Exception, [86](#)
- IfIsNaN
 - PommaLabs::Thrower::Raise, [36](#)
- IfIsNotAssignableFrom
 - PommaLabs::Thrower::Raise, [36](#), [37](#)
- IfIsNotAssignableFrom< TType >
 - PommaLabs::Thrower::Raise, [37](#)
- IfIsNotContainedIn
 - PommaLabs::Thrower::Raise, [38](#)
- IfIsNotContainedIn< TArg >
 - PommaLabs::Thrower::Raise, [39](#), [40](#)
- IfIsNotContainedIn< TArg1, TArg2 >
 - PommaLabs::Thrower::Raise, [40](#)
- IfIsNotEmpty
 - PommaLabs::Thrower::Raise, [42](#), [43](#)
- IfIsNotEmpty< TArg >
 - PommaLabs::Thrower::Raise, [43](#), [44](#)
- IfIsNotEqual
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException↔
 - RangeException, [69](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException↔
 - Exception, [86](#), [88](#)
- IfIsNotEqual< TArg >
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException↔
 - RangeException, [69](#), [70](#)
 - PommaLabs::Thrower::RaiseIndexOutOfRangeException↔
 - Exception, [88](#)
- IfIsNotInstanceOf
 - PommaLabs::Thrower::Raise, [44](#)
- IfIsNotInstanceOf< TType >
 - PommaLabs::Thrower::Raise, [45](#)
- IfIsNotNaN
 - PommaLabs::Thrower::Raise, [46](#)
- IfIsNotNull< TArg >
 - PommaLabs::Thrower::Raise, [46](#), [47](#)
- IfIsNotValid< TArg >
 - PommaLabs::Thrower::RaiseArgumentException, [50](#), [51](#)
- IfIsNotValidEmailAddress
 - PommaLabs::Thrower::RaiseArgumentException, [51](#), [52](#)
- IfIsNotValidPhoneNumber
 - PommaLabs::Thrower::RaiseArgumentException, [52](#)
- IfIsNull< TArg >
 - PommaLabs::Thrower::Raise, [47](#)
 - PommaLabs::Thrower::RaiseArgumentNullException↔
 - Exception, [55](#)
- IfIsNullOrEmpty
 - PommaLabs::Thrower::RaiseArgumentException, [52](#), [53](#)
- IfIsNullOrWhiteSpace
 - PommaLabs::Thrower::RaiseArgumentException, [53](#)
- IfNot
 - PommaLabs::Thrower::Raise, [48](#)
 - PommaLabs::Thrower::RaiseArgumentException, [53](#)
 - PommaLabs::Thrower::RaiseArgumentOutOfRangeException↔
 - RangeException, [70](#), [72](#)
 - PommaLabs::Thrower::RaiseHttpException, [75](#)
 - PommaLabs::Thrower::RaiseInvalidOperationException↔
 - Exception, [90](#)
 - PommaLabs::Thrower::RaiseNotSupported↔
 - Exception, [91](#)
- NoCtorTypes
 - PommaLabs::Thrower::RaiseBase, [73](#)
- PommaLabs, [9](#)
- PommaLabs.Thrower, [9](#)
- PommaLabs.Thrower.HttpException, [11](#)
- PommaLabs.Thrower.HttpExceptionInfo, [14](#)
- PommaLabs.Thrower.Raise< TEx >, [15](#)
- PommaLabs.Thrower.RaiseArgumentException, [48](#)
- PommaLabs.Thrower.RaiseArgumentNullException, [54](#)
- PommaLabs.Thrower.RaiseArgumentOutOfRangeException↔
- Exception, [56](#)
- PommaLabs.Thrower.RaiseBase, [72](#)
- PommaLabs.Thrower.RaiseHttpException, [74](#)
- PommaLabs.Thrower.RaiseIndexOutOfRangeException, [75](#)
- PommaLabs.Thrower.RaiseInvalidOperationException, [89](#)
- PommaLabs.Thrower.RaiseNotSupportedException, [90](#)
- PommaLabs.Thrower.RaiseObjectDisposedException, [92](#)
- PommaLabs.Thrower.ThrowerException, [93](#)
- PommaLabs::Thrower::HttpException
 - DefaultErrorCode, [13](#)
 - DefaultUserMessage, [14](#)
 - ErrorCode, [14](#)
 - HttpException, [12](#), [13](#)
 - HttpStatusCode, [14](#)
 - UserMessage, [14](#)
- PommaLabs::Thrower::HttpExceptionInfo
 - ErrorCode, [15](#)
 - HttpExceptionInfo, [15](#)
 - UserMessage, [15](#)
- PommaLabs::Thrower::Raise
 - If, [19](#)
 - IfAreEqual< TArg1, TArg2 >, [21](#)
 - IfAreNotEqual< TArg1, TArg2 >, [21](#), [22](#)
 - IfAreNotSame< TArg1, TArg2 >, [22](#)
 - IfAreSame< TArg1, TArg2 >, [24](#)
 - IfIsAssignableFrom, [25](#)
 - IfIsAssignableFrom< TType >, [25](#), [26](#)
 - IfIsContainedIn, [26](#)
 - IfIsContainedIn< TArg >, [27](#), [28](#)
 - IfIsContainedIn< TArg1, TArg2 >, [28](#), [30](#)
 - IfIsEmpty, [30](#), [32](#)
 - IfIsEmpty< TArg >, [32](#), [34](#)

- [IfInstanceOf](#), [34](#)
- [IfInstanceOf< TType >](#), [35](#)
- [IfIsNaN](#), [36](#)
- [IfIsNotAssignableFrom](#), [36](#), [37](#)
- [IfIsNotAssignableFrom< TType >](#), [37](#)
- [IfIsNotContainedIn](#), [38](#)
- [IfIsNotContainedIn< TArg >](#), [39](#), [40](#)
- [IfIsNotContainedIn< TArg1, TArg2 >](#), [40](#)
- [IfIsEmpty](#), [42](#), [43](#)
- [IfIsEmpty< TArg >](#), [43](#), [44](#)
- [IfIsNotInstanceOf](#), [44](#)
- [IfIsNotInstanceOf< TType >](#), [45](#)
- [IfIsNaN](#), [46](#)
- [IfIsNull< TArg >](#), [46](#), [47](#)
- [IfIsNull< TArg >](#), [47](#)
- [IfNot](#), [48](#)
- [PommaLabs::Thrower::RaiseArgumentException](#)
 - [If](#), [50](#)
 - [IfIsValid< TArg >](#), [50](#), [51](#)
 - [IfIsValidEmailAddress](#), [51](#), [52](#)
 - [IfIsValidPhoneNumber](#), [52](#)
 - [IfNullOrEmpty](#), [52](#), [53](#)
 - [IfNullOrEmptySpace](#), [53](#)
 - [IfNot](#), [53](#)
- [PommaLabs::Thrower::RaiseArgumentNullException](#)
 - [IfIsNull< TArg >](#), [55](#)
- [PommaLabs::Thrower::RaiseArgumentOutOfRangeException](#)
 - [If](#), [58](#), [59](#)
 - [IfIsEqual](#), [59](#)
 - [IfIsEqual< TArg >](#), [60](#)
 - [IfIsGreater](#), [61](#)
 - [IfIsGreater< TArg >](#), [61](#), [62](#)
 - [IfIsGreaterOrEqual](#), [62](#), [64](#)
 - [IfIsGreaterOrEqual< TArg >](#), [64](#), [65](#)
 - [IfIsLess](#), [65](#), [66](#)
 - [IfIsLess< TArg >](#), [66](#), [67](#)
 - [IfIsLessOrEqual](#), [67](#)
 - [IfIsLessOrEqual< TArg >](#), [68](#)
 - [IfIsNotEqual](#), [69](#)
 - [IfIsNotEqual< TArg >](#), [69](#), [70](#)
 - [IfNot](#), [70](#), [72](#)
- [PommaLabs::Thrower::RaiseBase](#)
 - [NoCtorTypes](#), [73](#)
 - [StrCtorType](#), [73](#)
 - [StrExCtorTypes](#), [74](#)
- [PommaLabs::Thrower::RaiseHttpException](#)
 - [If](#), [74](#), [75](#)
 - [IfNot](#), [75](#)
- [PommaLabs::Thrower::RaiseIndexOutOfRangeException](#)
 - [IfIsEqual](#), [77](#), [78](#)
 - [IfIsEqual< TArg >](#), [78](#)
 - [IfIsGreater](#), [78](#), [80](#)
 - [IfIsGreater< TArg >](#), [80](#)
 - [IfIsGreaterOrEqual](#), [80](#), [82](#)
 - [IfIsGreaterOrEqual< TArg >](#), [82](#)
 - [IfIsLess](#), [82](#), [84](#)
- [IfIsLess< TArg >](#), [84](#)
- [IfIsLessOrEqual](#), [84](#), [86](#)
- [IfIsLessOrEqual< TArg >](#), [86](#)
- [IfIsNotEqual](#), [86](#), [88](#)
- [IfIsNotEqual< TArg >](#), [88](#)
- [PommaLabs::Thrower::RaiseInvalidOperationException](#)
 - [If](#), [90](#)
 - [IfNot](#), [90](#)
- [PommaLabs::Thrower::RaiseNotSupportedException](#)
 - [If](#), [91](#)
 - [IfNot](#), [91](#)
- [PommaLabs::Thrower::RaiseObjectDisposedException](#)
 - [If](#), [93](#)
- [Raise.cs](#), [95](#)
- [RaiseArgumentException.cs](#), [108](#)
- [RaiseArgumentNullException.cs](#), [111](#)
- [RaiseArgumentOutOfRangeException.cs](#), [112](#)
- [RaiseHttpException.cs](#), [120](#)
- [RaiseIndexOutOfRangeException.cs](#), [123](#)
- [RaiseInvalidOperationException.cs](#), [128](#)
- [RaiseNotSupportedException.cs](#), [129](#)
- [RaiseObjectDisposedException.cs](#), [130](#)
- [StrCtorType](#)
 - [PommaLabs::Thrower::RaiseBase](#), [73](#)
- [StrExCtorTypes](#)
 - [PommaLabs::Thrower::RaiseBase](#), [74](#)
- [UserMessage](#)
 - [PommaLabs::Thrower::HttpException](#), [14](#)
 - [PommaLabs::Thrower::HttpExceptionInfo](#), [15](#)