CM-05 入力値検証機能

■ 概要

カスタム属性によりバリデータを定義することで、ユーザ定義クラスのプロパティの値を検証する機能を提供する。本機能は、Validation Application Block（以下、Validation AB）[1]を拡張し、必須入力チェック、半角カナ文字列チェック、int 型範囲チェックなど、業務アプリケーションで必要な各種検証ルールを提供している。各検証ルールの詳細は後述の『入力値検証ルール解説』を参照のこと。

なお、本機能は「CL-01 画面データ機能」と「SV-02 サーバ入力値検証機能」で利用される共通機能である。両機能における本機能の利用方法は以下の通りである。

- 画面データ機能
  画面項目の入力値検証に利用される。ロストフォーカス時（即値チェック）や「CL-03 イベント処理実行機能」を使ったイベント処理実行時のタイミングで入力値検証処理を実施する。
- サーバ入力値検証機能
  サーバ入力 DTO の入力値検証に利用。WCF サービスの実行開始時に入力値検証処理を実施する。

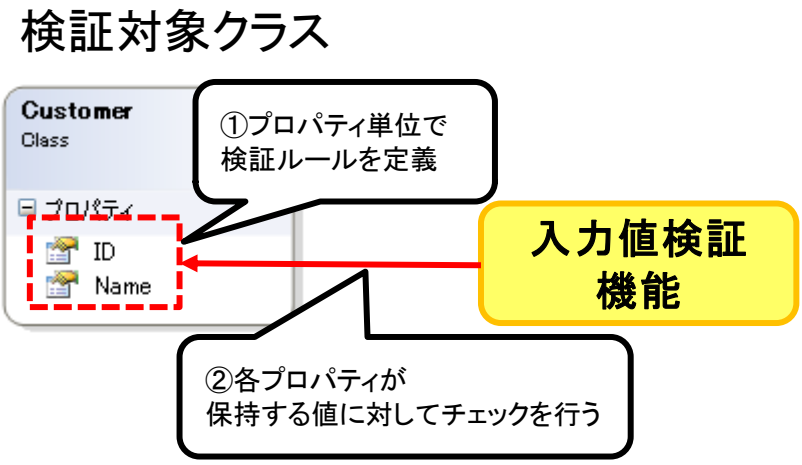

図 1 動作概念図

[1] マイクロソフトが推進するオープンソース・ライブラリ「Enterprise Library」に含まれる、入力値検証用の Application Block [http://www.codeplex.com/entlib]

■ 使用方法

入力値検証は、「単項目チェック」と「相関項目チェック等のカスタム入力チェック」の 2 つに分類され、どちらの場合も適用する検証ルールを属性で定義する。以降で使用方法について説明する。

◆単項目チェック

単項目チェックは、検証対象項目（プロパティ）に対して、本機能が提供する各種 Validator を属性で定義する。このとき、1 つのプロパティに複数の検証ルールを適用することも可能である（複数の Validator 属性を記述）。例えば、ある画面項目の入力値が必須かつ全角文字列でなければならない場合、「必須入力チェック」と「全角文字列チェック」を組み合わせることで実現できる。

以下に、単項目チェックの記述例を示す。

```
/// <summary>
/// 入力必須チェックを行う
/// 半角英数文字列チェックを行う
/// </summary>
[RequiredValidator (Ruleset = "RS01")]
[AlphaNumericStringValidator(Ruleset = "RS01")]
public string ID { get; set; }

/// <summary>
/// 文字列長チェックを行う
/// </summary>
[StringLengthRangeValidator (LowerBound = 1, UpperBound = 10,Tag = "名前",
        Ruleset = "RS01")]
public string Name { get; set; }
```

リスト 1 単項目チェックの記述例

Validator の属性には、適用する検証ルールに応じて、必要なパラメータを設定する。各 Validator に共通のパラメータを以下に示す。

表 1 各 Validator 属性に設定する共通のパラメータ

項番	属性名	必須	説明	デフォルト値
1	Negated		true を設定すると検証ルールを反転してチェックする。true もしくは false を設定する。	false
2	MessageTemplate		検証エラーメッセージのテンプレートとして使用する文字列。	各検証ルールに定義されている既定のメッセージ
3	MessageTemplateResourceName		検証エラーメッセージのテンプレートをメッセージリソースから読み込む際の名前。	なし
4	MessageTemplateResourceType		検証エラーメッセージのテンプレートをメッセージリソースから読み込む際のリソースの型。	なし

5	Tag		メッセージテンプレートのプレースホルダ{2}に渡される文字列。検証対象項目の論理名を設定する。	なし
6	Ruleset		ルールセット名	(空文字列)

MessageTemplate パラメータに設定した文字列、または MessageTemplateResourceType/MessageTemprateResourceName パラメータの設定によりメッセージリソースから読み込んだ文字列は、検証エラー時のエラーメッセージ生成に用いられるテンプレートである。これらを指定しなければフレームワークがもつ既定のメッセージが出力される。プレースホルダ{0}～{2}には固定で以下の値が格納される。

表 2 メッセージテンプレートに渡される文字列

プレースホルダ	説明
{0}	検証対象のプロパティに格納されている値の文字列表現
{1}	検証対象のプロパティ名(Key パラメータ)
{2}	Validator 属性の Tag パラメータの値

{3}以降のプレースホルダについては、検証ルールによって利用形態が異なる。詳細については後述の『入力値検証ルール解説』を参照のこと。

全バリデータ共通のパラメータ以外に、検証ルール固有のパラメータが存在する。例えば範囲チェックを行うルールならば上限値と下限値を指定するための属性がある。固有のパラメータの詳細については、後述の『入力値検証ルール解説』を参照のこと。

また、検証対象のクラスに対してどの入力チェックの組み合わせを実行するかを識別する名前として、「ルールセット名」(RuleSet)を使用する。

ルールセット名により実施する入力チェックルールの組み合わせを定義することができるので、入力値検証処理実行時に指定するルールセット名を切り替えることで、実行したいチェック処理を切り替えることができる。

◆相関項目チェック等のカスタム入力チェック

相関項目チェックを実施したい場合や、フレームワークが標準で提供している単項目チェックでは検証できないような複雑な検証を実施したい場合などは、Valildation Application Block が提供する SelfValidation 機能を利用する。

表 3 カスタム入力チェックに関して付与する属性

項番	属性名	ターゲット	説明
1	Microsoft.Practices.Enterprise Library.Validation.Validators. HasSelfValidationAttribute	クラス	SelfValidation 属性のついたメソッドがある場合に付与する
2	Microsoft.Practices.Enterprise Library.Validation.Validators. SelfValidationAttribute	メソッド	相関項目チェック等のカスタム入力チェック

SelfValidation 属性を付けたメソッドのシグニチャは、引数を Microsoft.Practices.EnterpriseLibrary.Validation.ValidationResults クラス、戻り値を void としなければならない。

入力値検証エラーの場合、Microsoft.Practices.EnterpriseLibrary.Validation.ValidationResult クラスのインスタンスを生成し、エラーメッセージを格納後、引数の ValidationResults オブジェクトに Add メソッドで追加する。

以下に、ValidationResult クラスのコンストラクタのパラメータを示す。

表 4 ValidationResult クラスのコンストラクタのパラメーター一覧

項番	パラメータ名	解説
1	message （string 型）	検証失敗時のエラーメッセージ。
2	target （object 型）	検証対象オブジェクト。
3	key （string 型）	検証対象のキー値（プロパティ名）。 キーを指定できない場合は、空文字（String.Empty）をセットする。
4	tag （string 型）	ValidationResults からのフィルタリングに用いる文字列。 SelfValidationValidator クラスでは、メッセージのプレースホルダは利用されないため、通常 String.Empty または null をセットする。

5	validator (Validator 型)	実行する Validator クラス。 「CL-01 画面データ機能」を使った画面データに対して、即値チェックでカスタム入力チェックエラーのメッセージがクリアされないようにするには、Validator 型の値として、「EventSpecificValidator.DefaultInstance」プロパティをセットする。(詳細は、「CL-01 画面データ機能」の機能説明書を参照) サーバ側入力値検証の場合は、Validation AB 標準の SelfValidation の利用方法と同様に、null をセットする。

以下に、カスタム入力チェックの記述例を示す。

```
[HasSelfValidation]                                   クラスに付与
public class SelfCustomer
{
   ・・・
    /// <summary>                                     メソッドに付与
    /// 相関項目チェックの例
    /// </summary>
    [SelfValidation(Ruleset = "RS01")]
    public void CustomValidator01(ValidationResults results)
    {
        // FirstNameとLastNameの両方が入力されていることをチェックする
        if (string.IsNullOrEmpty(FirstName) || string.IsNullOrEmpty(LastName))
        {
            ValidationResult result = new ValidationResult(
                "姓と名は入力必須項目です", this, String.Empty,
                null, EventSpecificValidator.DefaultInstance );
            results.AddResult(result);
        }
    }                                                 出力されるエラーメッセージ

    /// <summary>
    /// カスタム単項目チェックの例
    /// </summary>
    [SelfValidation(Ruleset = "RS01")]
    public void CustomValidator02(ValidationResults results)
    {
        ///NumberFieldに対して業務個有のチェックを実施
        if (CheckNumberField(NumberField))
        {
            ValidationResult result =
                new ValidationResult(Resource1.MSG0003, this, "NumberField" ,
                  null, EventSpecificValidator.DefaultInstance );
            results.AddResult(result);
        }
    }
}
```

リスト 2 「画面データ」クラスにおけるカスタム入力チェックの記述例

◆入力値検証処理の実行

検証処理を実施するには、Validation AB の標準の使用方法と同様である。まず、Microsoft.Practices.EnterpriseLibrary.Validation.ValidationFactory クラスを利用して検証対象の Validator クラスを生成し、Validator.Validate メソッドにより検証処理を実行する。以下に、実行例を示す。

なお、「CL-01 画面データ機能」における画面データの即値チェック、「CL-03 イベント処理実行機能」によるイベント実行時の画面データの検証、「SV-02 サーバ入力値検証機能」によるサーバ入力 DTO の検証では、フレームワークが Validator.Validate メソッドを実行するため、通常、開発者が明示的に Validator.Validate メソッドを呼び出す必要はない。

```
class Program
{
    public static void Main(string[] args)
    {
        ///入力検証対象オブジェクト
        TargetClass target = new TargetClass() { Id = null, Name = "aaaa" };
        ///App.configで定義したチェックルールをもとにバリデータを生成
        Validator valiator = ValidationFactory.CreateValidator<TargetClass>("RS01");
        ///入力値検証
        ValidationResults results = valiator.Validate(target);
        if (results.IsValid)
        {
            Console.WriteLine("入力値検証成功");
        }
        else
        {
            Console.WriteLine("入力値検証エラー");
            StringBuilder sb = new StringBuilder();
            foreach (ValidationResult result in results)
            {
                sb.AppendLine(result.Message);
            }
            Console.WriteLine(sb.ToString());
        }
            Console.ReadLine();
    }
}

///入力検証対象クラス
public class TargetClass
{
    [RequiredValidator(Tag = "Id", Ruleset = "RS01")]
    public string Id { get; set; }

    [RequiredValidator(Tag = "名前", Ruleset = "RS01")]
    [ZenkakuStringValidator(Tag = "名前", Ruleset = "RS01")]
    public string Name { get; set; }
}
```

リスト 3 入力値検証の実行例

■ 入力値検証ルール解説

◆検証ルール一覧

以下に、各提供元が提供する検証ルールとその検証クラスを示す。

表 5 TERASOLUNA フレームワークが提供する検証ルール

項番	検証ルール	検証クラス	概要
1	必須入力チェック	RequiredValidator	検証対象が null またはホワイトスペース(半角空白、全角空白、改行、タブ文字)でないか検証する。
2	半角数字文字列チェック	NumericStringValidator	検証対象が半角数字文字のみで構成されているか検証する。
3	半角英数文字列チェック	AlphaNumericStringValidator	検証対象が半角英数文字のみで構成されているか検証する。
4	半角英数大文字列チェック	CapAlphaNumericStringValidator	検証対象が大文字の半角英数文字のみで構成されているか検証する。
5	半角文字列チェック	HankakuStringValidator	検証対象が半角文字のみで構成されているか検証する。
6	半角カナ文字列チェック	HankakuKanaStringValidator	検証対象が半角カナ文字のみで構成されているか検証する。
7	全角文字列チェック	ZenkakuStringValidator	検証対象が全角文字のみで構成されているか検証する。
8	全角カナ文字列チェック	ZenkakuKanaStringValidator	検証対象が全角カナ文字のみで構成されているか検証する。
9	正規表現一致チェック	RegexValidatorEx	正規表現パターンを指定して、検証対象文字列がパターンとマッチするかを検証する。
10	日付形式チェック	DateTimeFormatValidator	検証対象が日付形式の文字列であるか検証する。
11	文字列存在チェック	ContainsCharactersValidatorEx	指定した文字列を含んでいるかどうかを検証する。
12	数値チェック	NumberValidator	検証対象が数値に変換可能であり、整数部・小数部がそれぞれ指定した桁数であるか検証する。
13	URL 形式チェック	UrlValidator	検証対象が URL 形式の文字列であるか検証する。
14	byte 列長範囲チェック	ByteRangeValidator	検証対象の文字列を指定したエンコーディングでバイト列に展開した際のバイト長が指定した範囲であるか検証する。
15	文字列長チェック	StringLengthRangeValidator	文字列が指定した文字数の範囲であるかを検証する。

16	日付型範囲チェック	DateTimeRangeValidatorEx	検証対象が dateTime 型に変換可能であり、指定した範囲の日時であるかを検証する。
17	int 型範囲チェック	IntRangeValidator	検証対象が int 型に変換可能であり、指定した範囲であるか検証する。
18	float 型範囲チェック	FloatRangeValidator	検証対象が float 型に変換可能であり、指定した範囲であるか検証する。
19	double 型範囲チェック	DoubleRangeValidator	検証対象が double 型に変換可能であり、指定した範囲であるか検証する。
20	decimal 型範囲チェック	DecimalRangeValidator	検証対象が decimal 型に変換可能であり、指定した範囲であるか検証する。

表 6 Validation Application Block が提供する検証ルール

項番	検証ルール	検証クラス	概要
1	byte 型チェック	TypeConversionValidator	検証対象が byte（Byte）型に変換可能か検証する。
2	short 型チェック		検証対象が short(Int16)型に変換可能か検証する。
3	int 型チェック		検証対象が int(Int32)型に変換可能か検証する。
4	long 型チェック		検証対象が long(Int64)型に変換可能か検証する。
5	float 型チェック		検証対象が float(Single)型に変換可能か検証する。
6	double 型チェック		検証対象が double（Double）型に変換可能か検証する。
7	decimal 型チェック		検証対象が decimal（Decimal）型に変換可能か検証する。
8	プロパティ比較チェック	PropertyComparisonValidator	他のプロパティ値と比較を行い、特定の条件を満たしているか否かを検証する。
9	カスタム入力チェック	SelfValidationValidator	カスタム入力チェックの際に利用する。 ※具体的な使用方法については、前述の『相関項目チェック等のカスタム入力チェック』を参照。
10	参照オブジェクト入力チェック	ObjectValidator	検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたオブジェクトを検証する。 ※「CL-01 画面データ機能」では利用しないこと。

11	参照コレクション入力チェック	ObjectCollectionValidator	検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたコレクションオブジェクトを検証する。 ※「CL-01 画面データ機能」では利用しないこと。

◆カスタムスニペットの利用

TERASOLUNA フレームワークでは、各入力値検証ルールのカスタム属性に対して、カスタムスニペットが用意されており、通常スニペットを使用して、カスタム属性を追加する。
なお、カスタム入力チェック(SelfValidationValidator については)メソッドのひな形も生成される。

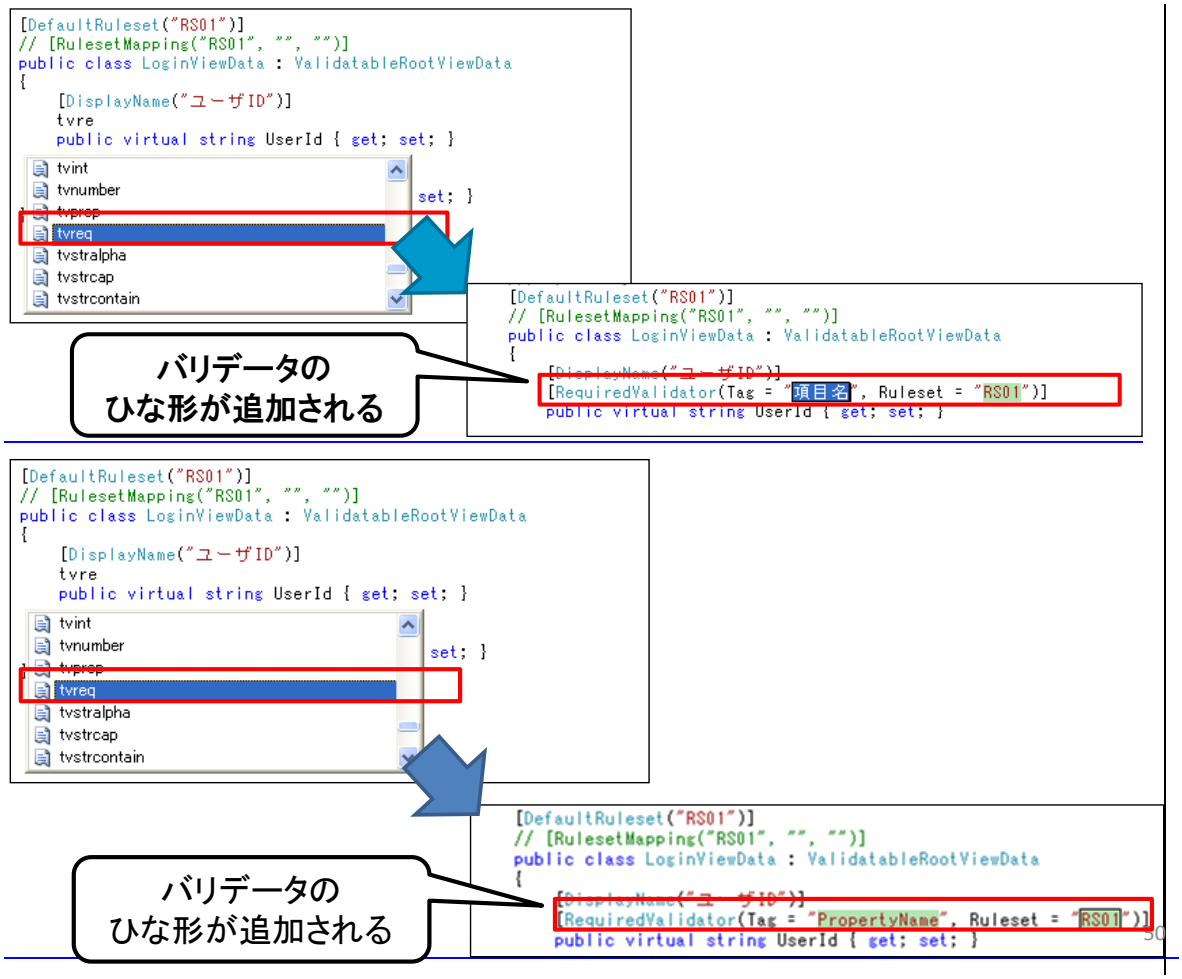

図 2 カスタムスニペットの利用イメージ

以下に、スニペットの一覧を示す。スニペットは、全て「tv」から始まる文字列になっている。

表 7 TERASOLUNA フレームワークが提供する検証ルールのスニペット

項番	検証ルール	検証クラス	スニペット
1	必須入力チェック	RequiredValidator	tvreq
2	半角数字文字列チェック	NumericStringValidator	tvstrnumeric
3	半角英数文字列チェック	AlphaNumericStringValidator	tvstralpha
4	半角英数大文字列チェック	CapAlphaNumericStringValidator	tvstrcap
5	半角文字列チェック	HankakuStringValidator	tvstrhan
6	半角カナ文字列チェック	HankakuKanaStringValidator	tvhankana
7	全角文字列チェック	ZenkakuStringValidator	tvstrzen
8	全角カナ文字列チェック	ZenkakuKanaStringValidator	tvstrzenkana
9	正規表現一致チェック	RegexValidatorEx	tvstrregex
10	日付形式チェック	DateTimeFormatValidator	tvdate
11	文字列存在チェック	ContainsCharactersValidatorEx	tvstrcontain
12	数値チェック	NumberValidator	tvnumber
13	URL 形式チェック	UrlValidator	tvurl
14	byte 列長範囲チェック	ByteRangeValidator	tvbyte
15	文字列長チェック	StringLengthRangeValidator	tvstrrange
16	日付型範囲チェック	DateTimeRangeValidatorEx	tvdaterange
17	int 型範囲チェック	IntRangeValidator	tvint
18	float 型範囲チェック	FloatRangeValidator	tvfloat
19	double 型範囲チェック	DoubleRangeValidator	tvdouble
20	decimal 型範囲チェック	DecimalRangeValidator	tvdecimal

表 7 Validation Application Block が提供する検証ルールのスニペット

項番	検証ルール	検証クラス	スニペット
1	byte 型チェック	TypeConversionValidator	tvtype
2	short 型チェック		
3	int 型チェック		
4	long 型チェック		
5	float 型チェック		
6	double 型チェック		
7	decimal 型チェック		
8	プロパティ比較チェック	PropertyComparisonValidator	tvcomparison
9	カスタム入力チェック	SelfValidationValidator	tvcustom
10	参照オブジェクト入力チェック	ObjectValidator	tvobj
11	参照コレクション入力チェック	ObjectCollectionValidator	tvobjcol

◆フレームワークが提供する入力値検証ルール

TERASOLUNA フレームワークが提供する入力値検証ルールの機能と利用方法について解説する。

- 必須入力チェック(RequiredValidator)
  検証対象値が未入力でないかを検証する。

  以下に、検証成功となるパターンを示す。
  - 通常時
    検証対象が null ではなく、検証対象値の文字列表現からホワイトスペース(半角空白、全角空白、改行、タブ文字)を取り除いた文字列の長さが 0 より大きい場合。
  - 反転時(Negated = true の場合)
    検証対象が null であるか、または検証対象値の文字列表現からホワイトスペース(半角空白、全角空白、改行、タブ文字)を取り除いた文字列の長さが 0 の場合。

  フレームワークで提供する入力値検証ルールは原則として null または空文字列に対して検証成功を返す。Negated が true である場合も同様。null または空文字列を許容しない場合には、各検証クラスにこの必須入力チェックを併用する。

表 8 構成クラス

項番	クラス名	説明
1	RequiredValidator	必須入力チェックを行う検証クラス
2	RequiredValidatorAttribute	RequiredValidator のインスタンスを生成し、プロパティに付与するための属性

必須入力チェックには、特に固有の設定項目はない。

表 9 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”は入力必須項目です。
2	反転時	“{2}”は入力しないでください。

以下に RequiredValidator 属性を使用した設定例を示す。

```
[RequiredValidator(Ruleset = "RS01")]
public string ID { get; set; }
```

リスト 4 RequiredValidator 属性の設定例

- 半角数字文字列チェック(NumericStringValidator)
  検証対象の文字列表現が、半角数字 (0～9)のみで構成されているか検証する。

  以下に、検証成功となるパターンを示す。
  - 通常時
    検証対象の文字列表現が半角数字(0～9)のみで構成されている場合。
  - 反転時(Negated = true の場合)
    検証対象の文字列表現に半角数字(0～9)以外の文字が含まれている場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合は、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 10 構成クラス

項番	クラス名	説明
1	NumericStringValidator	半角数字チェックを行う検証クラス
2	NumericStringValidatorAttribute	NumericStringValidator のインスタンスを生成し、プロパティに付与するための属性

半角数字チェックには、特に固有の設定項目はない。

表 11 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には数字を入力してください。
2	反転時	“{2}”には数字のみで入力しないでください。

以下に NumericStringValidator 属性を使用した設定例を示す。

```
[RequiredValidator(Ruleset = "RS01")]
[NumericStringValidator (Ruleset = "RS01")]
public string ID { get; set; }
```

リスト 5 NumericStringValidator 属性の設定例

- 半角英数文字列チェック(AlphaNumericStringValidator)

検索対象の文字列表現が、半角英数文字のみで構成されているか検証する。半角英数文字とは、半角数字(0～9)と半角の英字(a～z、A～Z)を併せた文字集合である。

以下に、検証成功となるパターンを示す。

➢ 通常時
検証対象の文字列表現が半角英数文字のみで構成されている場合。

➢ 反転時(Negated = true の場合)
検証対象の文字列表現に半角英数文字以外の文字が含まれている場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合は、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 12 構成クラス

項番	クラス名	説明
1	AlphaNumericStringValidator	半角英数文字列チェックを行う検証クラス
2	AlphaNumericStringValidatorAttribut	AlphaNumericStringValidator のインスタンスを生成し、プロパティに付与するための属性

半角英数文字列チェックには、特に固有の設定項目はない。

表 13 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には半角英数字を入力してください。
2	反転時	“{2}”には半角英数字のみを入力しないでください。

以下に AlphaNumericStringValidator 属性を使用した設定例を示す。

```
[RequiredValidator(Tag = "ユーザID", Ruleset = "RS01")]
[AlphaNumericStringValidator (Tag = "ユーザID", Ruleset = "RS01")]
public string ID { get; set; }
```

リスト 6 AlphaNumericStringValidator 属性の設定例

- 半角英数大文字列チェック(CapAlphaNumericStringValidator)
  検証対象の文字列表現が、半角英数大文字のみで構成されているか検証する。半角英数大文字とは、半角数字(0～9)と半角大文字の英字(A～Z)を併せた文字集合である。

以下に、検証成功となるパターンを示す。

- 通常時
  検証対象の文字列表現が半角英数大文字のみで構成されている場合。
- 反転時(Negated = true の場合)
  検証対象の文字列表現に半角英数大文字以外の文字が含まれている場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合は、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 14 構成クラス

項番	クラス名	説明
1	CapAlphaNumericStringValidator	半角英数大文字列チェックを行う検証クラス
2	CapAlphaNumericStringValidatorAttribute	CapAlphaNumericStringValidator のインスタンスを生成し、プロパティに付与するための属性

半角英数大文字列チェックには、特に固有の設定項目はない。

表 15 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には大文字の半角英数字で入力してください。
2	反転時	“{2}”には大文字の半角英数字のみで入力しないでください。

以下に CapAlphaNumericStringValidator 属性を使用した設定例を示す。

```
[RequiredValidator(Ruleset = "RS01")]
[CapAlphaNumericStringValidator (Ruleset = "RS01")]
public string ID { get; set; }
```

リスト 7 CapAlphaNumericStringValidator 属性の設定例

- 半角文字列チェック(HankakuStringValidator)

  検証対象の文字列表現が、半角文字列のみで構成されているか検証する。半角文字とは、“＼ ¢ £ §¨¬°±´¶×÷”を除く unicode の 0 から 255 番目まで(拡張 ASCII コードの範囲)の文字に半角カナを加えた文字集合である。

  以下に、検証成功となるパターンを示す。

  ➢ 通常時
  検証対象の文字列表現が半角文字のみで構成されている場合。

  ➢ 反転時(Negated = true の場合)
  検証対象の文字列表現に半角文字以外の文字が含まれている場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 16 構成クラス

項番	クラス名	説明
1	HankakuStringValidator	半角文字列チェックを行う検証クラス
2	HankakuStringValidator Attribute	HankakuStringValidator のインスタンスを生成し、プロパティに付与するための属性

半角文字列チェックには、特に固有の設定項目はない。

表 17 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には半角文字を入力してください。
2	反転時	“{2}”には半角文字のみを入力しないでください。

以下に HankakuStringValidator 属性を使用した設定例を示す。

```
[HankakuStringValidator (Tag = "半角データ", Ruleset = "RS01")]
public string HankakuField { get; set; }
```

リスト 8 HankakuStringValidator 属性の設定例

- 半角カナ文字列チェック(HankakuKanaStringValidator)

  検証対象の文字列表現が、半角カナのみで構成されているか検証する。半角カナ文字とは、次に示す文字集合である。"ｱｲｳｴｵｧｨｩｪｫｶｷｸｹｺｻｼｽｾｿﾀﾁﾂｯﾃﾄﾅﾆﾇﾈﾉﾊﾋﾌﾍﾎﾏﾐﾑﾒﾓﾔﾕﾖｬｭｮﾗﾘﾙﾚﾛﾜﾝﾞﾟｰ・、。「」"

  以下に、検証成功となるパターンを示す。

  ➢ 通常時
  検証対象の文字列表現が半角カナ文字のみで構成されている場合。

  ➢ 反転時(Negated = true の場合)
  検証対象の文字列表現に半角カナ文字以外の文字が含まれている場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 18 構成クラス

項番	クラス名	説明
1	HankakuKanaStringValidator	半角カナ文字列チェックを行う検証クラス
2	HankakuKanaStringValidator Attribute	HankakuKanaStringValidator のインスタンスを生成し、プロパティに付与するための属性

半角カナ文字列チェックには、特に固有の設定項目はない。

表 19 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には半角カナ文字を入力してください。
2	反転時	“{2}”には半角カナ文字のみを入力しないでください。

以下に HankakuKanaStringValidator 属性を使用した設定例を示す。

```
[HankakuKanaStringValidator (Ruleset = "RS01")]
public string HankakuKanaField { get; set; }
```

リスト 9 HankakuKanaStringValidator 属性の設定例

- 全角文字列チェック(ZenkakuStringValidator)

  検証対象の文字列表現が、全角文字のみで構成されているか検証する。全角文字列とは、半角文字(半角文字列チェックの定義に準じる)ではない文字のみで構成された文字列のことである。

  以下に、検証成功となるパターンを示す。

  - 通常時
    検証対象の文字列表現が全角文字のみで構成されている場合。
  - 反転時(Negated = true の場合)
    検証対象の文字列表現に全角文字以外の文字が含まれている場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 20 構成クラス

項番	クラス名	説明
1	ZenkakuStringValidator	全角文字列チェックを行う検証クラス
2	ZenkakuStringValidator Attribute	ZenkakuStringValidator のインスタンスを生成し、プロパティに付与するための属性

全角文字列チェックには、特に固有の設定項目はない。

表 21 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には全角文字を入力してください。
2	反転時	“{2}”には全角文字のみを入力しないでください。

以下に ZenkakuStringValidator 属性を使用した設定例を示す。

```
[ZenkakuStringValidator (Tag = "全角データ", Ruleset = "RS01")]
public string ZenkakuField { get; set; }
```

リスト 10 ZenkakuStringValidator 属性の設定例

- 全角カナ文字列チェック(ZenkakuKanaStringValidator)

  検証対象の文字列表現が、全角カナ文字列のみで構成されているか検証する。全角カナ文字とは、次に示す文字集合である。"アイウヴエオァィゥェォカキクケコヵヶガギグゲゴサシスセソザジズゼゾタチツテトダヂヅデドナニヌネノハヒフヘホバビブベボパピプペポマミムメモヤユヨャュョラリルレロワヮヰヱヲッンー"

  以下に、検証成功となるパターンを示す。

  - 通常時
    検証対象の文字列表現が全角カナ文字のみで構成されている場合。
  - 反転時(Negated = true の場合)
    検証対象の文字列表現に全角カナ文字以外の文字が含まれている場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 22 構成クラス

項番	クラス名	説明
1	ZenkakuKanaStringValidator	全角カナ文字列チェックを行う検証クラス
2	ZenkakuKanaStringValidator Attribute	ZenkakuKanaStringValidator のインスタンスを生成し、プロパティに付与するための属性

全角カナ文字列チェックには、特に固有の設定項目はない。

表 23 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	“{2}”には全角カナ文字を入力してください。
2	反転時	“{2}”には全角カナ文字のみを入力しないでください。

以下に ZenkakuStringValidator 属性を使用した設定例を示す。

```
[ZenkakuKanaStringValidator (Tag = "全角カナデータ", Ruleset = "RS01")]
public string ZenkakuKanaField { get; set; }
```

リスト 11 ZenkakuStringValidator 属性の設定例

- 正規表現一致チェック(RegexValidatorEx)

  検証対象値が指定した正規表現文字列にマッチするかどうかを検証する。検証には、正規表現パターンとともに、.NET Framework で提供される正規表現エンジンの各種オプション(RegexOptions 列挙体)を利用することができる。

  以下に、検証成功となるパターンを示す。

  - 通常時
    検証対象となる文字列が、指定されたパターンと正規表現オプションによって表される正規表現にマッチする場合。
  - 反転時(Negated = true の場合)
    検証対象となる文字列が、指定されたパターンと正規表現オプションによって表される正規表現にマッチしない場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 24 構成クラス

項番	クラス名	説明
1	RegexValidatorEx	正規表現による入力値検証を行う検証クラス
2	RegexValidatorEx Attribute	RegexValidatorEx のインスタンスを生成し、プロパティに付与するための属性

表 25 固有設定項目

項番	パラメータ名	必須	解説	デフォルト値
1	pattern	△※	正規表現のパターン	なし
2	patternResourceName	△※	正規表現パターンを読み込む際に利用するリソース名	なし
3	patternResourceType		正規表現のパターンを読み込むリソースの型名	なし

小文字から始まるパラメータ
→コンストラクタ引数で指定

4	options	-	正規表現オプション(RegexOptions 列挙体) を指定する。列挙体の値は以下の値の 1 または複数の組み合わせとなる ・None ・IgnoreCase ・Multiline ・ExplicitCapture ・Compiled ・Singleline ・IgnorePatternWhiteSpace ・RighttoLeft 複数の値を指定する場合、コンマで区切って記述する	None

※pattern と patternResourceName/ patternResourceType には以下のルールがある

- patternResourceName と patternResourceType はセットで扱われ、どちらか一方のみを指定することは不可。
- pattern と patternResourceName/patternResourceType セットのどちらかは必ず指定しなければならない。

表 26 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時／反転時(共通)	“{2}”は不正な形式です。

以下に RegexValidatorEx 属性を使用した設定例を示す。

```
[DisplayName("メールアドレス")]
[RegexValidatorEx (@"¥w+([-+.']¥w+)*@¥w+([-.]¥w+)*¥.¥w+([-.]¥w+)*",
        RegexOptions.IgnoreCase, Ruleset = "RS01")]
public string Email { get; set; }
```

リスト 12 RegexValidatorEx 属性の設定例

※以下に options パラメータで設定する、正規表現オプションについて示す。

表 27 RegexOptions 列挙体

項番	メンバ名	説明
1	None	何も設定しない
2	IgnoreCase	検索時に大文字と小文字を区別しない
3	Multiline	複数行モード
4	ExplicitCapture	明示的に名前または番号を指定されたグループだけが有効なキャプチャとなる。 名前のないかっこは非キャプチャ グループとして機能。
5	Compiled	正規表現をコンパイルしてアセンブリを作成 （実行速度は速くなるが、起動にかかる時間は長くなる）
6	Singleline	単一行モード
7	IgnorePatternWhiteSpace	パターンから、エスケープが解除された空白を削除し、#でマークされたコメントを有効化
8	RighttoLeft	検索を右から左に行う

- 日付形式チェック(DateTimeFormatValidator)

  検証対象の文字列表現が、指定した形式の日付文字列であるかを検証する。日付文字列の形式は、DateTimeFormatInfo で利用可能な形式指定文字列の組み合わせで指定する。

  例)"yyyy/MM/dd" を形式として指定する場合、
  "2005/08/12"(成功) "2005-08-12"(失敗) "2005 年 08 月 12 日"(失敗)

  以下に、検証成功となるパターンを示す。

  - 通常時
    検証対象となる文字表現が、指定した日付書式文字列に従って DateTime 型に変換可能である場合。
  - 反転時(Negated = true の場合)
    検証対象となる文字表現が、指定した日付書式文字列に従って DateTime 型に変換可能でない場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 28 構成クラス

項番	クラス名	説明
1	DateTimeFormatValidator	日付形式チェックを行う検証クラス
2	DateTimeFormatValidator Attribute	DateTimeFormatValidator のインスタンスを生成し、プロパティに付与するための属性

表 29 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	dateTimeFormat	○	検証対象である日付形式文字列を指定する ※設定値がプレースホルダの{3}に代入される。	なし

表 30 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	"{2}"には"{3}"の形式の日付を入力してください。
2	反転時	"{2}"には"{3}"の形式の日付を入力しないでください。

以下に DateTimeFormatValidator 属性を使用した設定例を示す。

```
[DateTimeFormatValidator ("yyyy/MM/dd", Ruleset = "RS01")]
public string BirthDay { get; set; }
```

リスト 13 DateTimeFormatValidator 属性の設定例

● 文字列存在チェック(ContainsCharacterValidatorEx)
検証対象文字列が、指定した文字を含んでいるかどうかの検証を行う。

Negated パラメータが false の場合、characterSet パラメータで指定した文字が含まれていない場合に検証エラーとなる。逆に Negated パラメータを true と指定することで、指定した文字が含まれる場合に検証エラーとする禁止文字列チェックとして利用することができる。

以下に、検証成功となるパターンを示す。

➢ 通常時
検証対象が文字列であり、かつ、containsCharacter で「Any」が指定されている場合には、characterSet で指定された文字のうちいずれかを含んでいれば検証成功となる。containsCharacter で「All」が指定されている場合には、characterSet で指定された文字を全て含んでいれば検証成功となる。

➢ 反転時(Negated = true の場合)
検証対象が文字列であり、かつ、containsCharacter で「Any」が指定されている場合には、characterSet で指定された文字が 1 も含まれていなければ検証成功となる。containsCharacter で「All」が指定されている場合には、characterSet で指定された文字を全て含んでいる場合以外に検証成功となる。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 31 構成クラス

項番	クラス名	説明
1	ContainsCharactersValidatorEx	指定した文字が含まれているかどうかを検証する検証クラス
2	ContainsCharactersValidatorExAttribute	ContainsCharactersValidator のインスタンスを生成し、プロパティに付与するための属性

表 32 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	characterSet	○	検証対象の文字列中で存在をチェックする文字。(複数指定する場合、コンマなどで区切らず、"AB."と続けて記述する。) ※設定値がプレースホルダの{3}に代入される。	なし

2	containsCharacter	-	characterSet で指定した文字の存在をチェックする方式を指定する ContainsCharacters 列挙体。 以下の値のいずれかとなる。 ・Any (いずれか 1 を含む) ・All (全ての文字を含む)	Any

表 33 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時かつ Any	“{2}”は”{3}”の文字いずれかを含む文字列を入力してください。
2	通常時かつ All	“{2}”は”{3}”の文字全てを含む文字列を入力してください。
3	反転時かつ Any	“{2}”は”{3}”の文字いずれも含まない文字列を入力してください。
4	反転時かつ All	“{2}”は”{3}”の文字が全て含まれない文字列を入力してください。

以下に ContainsCharactersValidatorEx 属性を使用した設定例を示す。

```
[ContainsCharactersValidatorEx ("emp", ContainsCharacters.All, Ruleset = "RS01")]
public string EmployeeID { get; set; }
```

リスト 14 ContainsCharactersValidatorEx 属性の設定例

- 数値チェック(NumberValidator)

検証対象の文字列表現が、整数部と小数部がそれぞれ指定した桁数である数値の形式であるかを検証する。桁数のチェックには、指定した値と等しいか調べる一致チェックと、指定した以内の範囲に含まれるか調べる範囲チェックを利用することができる。

以下に、検証成功となるパターンを示す。

➢ 通常時
検証対象の文字列表現が整数または小数の数値形式であり、桁数がそれぞれ指定した値・範囲に含まれる場合。

➢ 反転時(Negated = true の場合)
検証対象の文字列表現が整数または小数の数値形式でない場合、または、桁数がそれぞれ指定した値・範囲に含まれない場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 34 構成クラス

項番	クラス名	説明
1	NumberValidator	数値チェックを行う検証クラス
2	NumberValidatorAttribute	NumberValidator のインスタンスを生成し、プロパティに付与するための属性

表 35 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	IntegerLength		数値文字列の整数部の桁数を int 型の値で指定する。最小値は 1 ※設定値がプレースホルダの{3}に代入される。	1
2	IsAccordedInteger	-	整数部の桁数一致チェックを行うかどうかを指定する。 ・true の場合、integerLength で指定した桁数と一致しているかどうかを検証する。 ・false の場合、integerLength で指定した桁数以下であるか検証する。	false

3	Scale	-	数値文字列の小数部の桁数を int 型の値で指定する。最小値は 0 であり、0 を指定した際には整数チェックとなる ※設定値がプレースホルダの{4}に代入される。	0
4	IsAccordedScale	-	IsAccordedInteger と同様に小数部の桁数一致チェックを行うかどうかを指定する	false

表 36 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	"{2}" には整数部 {3} 桁、小数部 {4} 桁までの数値を入力してください。
2	反転時	"{2}" には整数部 {3} 桁、小数部 {4} 桁までの数値以外を入力してください。

以下に NumberValidator 属性を使用した設定例を示す。

```
[NumberValidator (IntegerLength = 2, IsAccordedInteger = true, Scale = 3,
        IsAccordedScale = false, Ruleset = "RS01")]
public string Average{ get; set; }
```

リスト 15 NumberValidator 属性の設定例

- URL 形式文字列チェック(UrlValidator)

  検証対象の文字列表現が、URL 形式の文字列であるかを検証する。URL の形式は、以下に示す通り。

  <プロトコル>://<ホスト名>:<ポート番号>/<パス文字列>

表 37 URL 形式文字列の構成要素

項番	名称	説明
1	プロトコル	http or https のみ対応。小文字と大文字は区別しない
2	ホスト名	“/“及び”:“以外の任意の文字列である
3	ポート番号	任意桁数の数値。省略可能であり、省略時は”:”を記述しないこと
4	パス	任意の文字列

以下に、検証成功となるパターンを示す。

- 通常時
  検証対象の文字列表現が URL 形式である場合。
- 反転時(Negated = true の場合)
  検証対象の文字列表現が URL 形式ではない場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 38 構成クラス

項番	クラス名	説明
1	UrlValidator	URL 形式文字列チェックを行う検証クラス
2	UrlValidatorAttribute	UrlValidator のインスタンスを生成し、プロパティに付与するための属性

URL 形式文字列チェックには、特に固有の設定項目はない。

表 39 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	"{2}" には適切な URL 形式で入力してください。
2	反転時	"{2}" には URL 形式で入力しないでください

以下に UrlValidator 属性を使用した設定例を示す。

```
[UrlValidator (Tag = "HPアドレス", Ruleset = "RS01")]
public string HpAddress{ get; set; }
```

リスト 16 UrlValidator 属性の設定例

- byte 列長範囲チェック(ByteRangeValidator)

  検証対象の文字列を指定したエンコード形式でバイト列にデコードした際、バイト列長が指定した範囲であるかどうかを検証する。

  以下に、検証成功となるパターンを示す。

  - 通常時

    検証対象が文字列であり、かつ、指定したエンコーディングでデコードしたバイト列の長さが指定した最小値と最大値の間に含まれる場合。
  - 反転時(Negated = true の場合)

    検証対象が文字列であり、かつ、指定したエンコーディングでデコードしたバイト列の長さが指定した最小値と最大値の間に含まれない場合。

  検証対象が null である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 40 構成クラス

項番	クラス名	説明
1	ByteRangeValidator	byte 列長範囲チェックを行う検証クラス
2	ByteRangeValidatorAttribute	ByteRangeValidator のインスタンスを生成し、プロパティに付与するための属性

表 41 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	MinByteLength	-	検証対象であるバイト長の最小値を int 型の値で指定する。範囲には指定した値自体が含まれる。 ※設定値がプレースホルダの{3}に代入される。	0
2	MaxByteLength	-	検証対象であるバイト長の最大値を int 型の値で指定する。範囲には指定した値自体が含まれる。 ※設定値がプレースホルダの{4}に代入される。	2,147,483,647

3	EncodingName	-	検証対象文字列をバイト列にデコードするために利用するエンコーディング名。System.Text.Encoding クラスでサポートされているエンコーディング名が利用可能。	utf-8

表 42 デフォルトテンプレート文字列

項番	検証ルール	メッセージ
1	通常時	"{2}" には {3} byte 以上 {4} byte 以下の長さの文字列を入力してください。
2	反転時	"{2}" には {3} byte 未満または {4} byte より大きい長さの文字列を入力してください。

以下に ByteRangeValidator 属性を使用した設定例を示す。

```
[ByteRangeValidator (MinByteLength = 5, MaxByteLength = 10,
        EncodingName = "iso-2022-jp", Ruleset = "RS01")]
public string ByteField{ get; set; }
```

リスト 17 ByteRangeValidator 属性の設定例

- 文字列長チェック(StringLengthRangeValidator)
  検証対象の文字列表現の文字列長が指定した範囲であるかどうかを検証する。

  最小値を Ignore(無視)指定することで最大文字数制限として、最大値を Ignore 指定することで最小文字数制限として機能する。また、最大値と最小値として同じ値を指定し、いずれも Inclusive(値を含む)として指定すれば、固定文字数チェックとして機能する。

  以下に、検証成功となるパターンを示す。

  - 通常時
    検証対象の文字列表現の文字列長が最小値と最大値の間に含まれる場合。
  - 反転時(Negated = true の場合)
    検証対象の文字列表現の文字列長が最小値と最大値の間に含まれない場合。

  検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 43 構成クラス

項番	クラス名	説明
1	StringLengthRangeValidator	文字列の長さが指定した範囲内であるかどうかを検証する検証クラス
2	StringLengthRangeValidatorAttribute	StringLengthRangeValidator のインスタンスを生成し、プロパティに付与するための属性

表 44 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	LowerBound	-	検証対象である文字列長範囲の最小値を int 型の値で指定する。 ※設定値がプレースホルダの{3}に代入される。	なし(未入力ならば無視される)
2	LowerBoundType	-	LowerBound で指定した最小値の値の扱いを指定する RangeBoundaryType 列挙体。以下の値のいずれかとなる。 ・Ignore (値を無視する) ・Inclusive (値を含む) ・Exclusive (値を含まない)	※「 LowerBoundType 、UpperBoundType の取りうる値」参照

3	UpperBound	-	検証対象である文字列長範囲の最大値を int 型の値で指定する。 ※設定値がプレースホルダの{5}に代入される。	なし(未入力ならば無視される)
4	UpperBoundType		UpperBound で指定した最大値の値の扱いを指定する RangeBoundaryType 列挙体	※「 LowerBoundType 、UpperBoundType の取りうる値」参照

LowerBoundType と UpperBoundType の組合せにより、デフォルトメッセージテンプレート文字列が異なる。以下にそれぞれのケースの出力値を示す。

表 45 デフォルトテンプレート文字列

(検証ルール:通常時)

項番	LowerBoundType	UpperBoundType	メッセージ
1	Ignore	Inclusive	"{2}" には {5}文字以下の文字列を入力してください。
2	Ignore	Exclusive	"{2}" には {5}文字未満の文字列を入力してください。
3	Inclusive	Ignore	"{2}" には {3}文字以上の文字列を入力してください。
4	Inclusive	Inclusive	"{2}" には {3}文字以上{5}文字以下の文字列を入力してください。
5	Inclusive	Exclusive	"{2}" には {3}文字以上{5}文字未満の文字列を入力してください。
6	Exclusive	Ignore	"{2}" には {3}文字より大きい文字列を入力してください。
7	Exclusive	Inclusive	"{2}" には {3}文字より大きい{5}文字以下の文字列を入力してください。
8	Exclusive	Exclusive	"{2}" には {3}文字より大きい{5}文字未満の文字列を入力してください。

(検証ルール:反転時)

項番	LowerBoundType	UpperBoundType	メッセージ
9	Ignore	Inclusive	"{2}" には {5}文字より大きい文字列を入力してください。
10	Ignore	Exclusive	"{2}" には {5}文字以上の文字列を入力してください。

11	Inclusive	Ignore	"{2}" には {3}文字未満の文字列を入力してください。
12	Inclusive	Inclusive	"{2}" には {3}文字未満または{5}文字より大きい文字列を入力してください。
13	Inclusive	Exclusive	"{2}" には {3}文字未満または{5}文字以上の文字列を入力してください。
14	Exclusive	Ignore	"{2}" には {3}文字以下の文字列を入力してください。
15	Exclusive	Inclusive	"{2}" には {3}文字以下または{5}文字より大きい文字列を入力してください。
16	Exclusive	Exclusive	"{2}" には {3}文字以下または{5}文字以上の文字列を入力してください。

以下に StringLengthRangeValidator 属性を使用した設定例を示す。

```
[StringLengthRangeValidator (
        LowerBound = 1, LowerBoundType = RangeBoundaryType.Inclusive,
        UpperBound = 8, UpperBoundType = RangeBoundaryType.Inclusive,
        Ruleset = "RS01")]
public string FirstName{ get; set; }
```

リスト 18 StringLengthRangeValidator 属性の設定例

- 範囲チェックで利用するパラメータのルール

StirngLengthRangeValidator に限らず、本フレームの範囲チェックの仕組みでは、lowerBound(最小値)、lowerBoundType(指定した最小値の扱い)、upperBound(最大値)、upperBoundType(指定した最大値の扱い)の 4 つのパラメータを利用して範囲を指定する。これらのパラメータに関して、共通して以下のルールがあるので注意すること。

- LowerBound <= UpperBound である必要がある
- LowerBoundType と UpperBoundType の両方が Ignore であってはならない
- LowerBound、UpperBound に指定する文字列は検証しようとするデータの型に変換できなければならない
- Negated = true と指定した場合、有効となる範囲が反転する。このとき、Inclusive /Exclusive の意味も逆転する。

- LowerBoundType、UpperBoundType の取りうる値

StirngLengthRangeValidator に限らず、本フレームの範囲チェックの仕組みでは、LowerBoundType、UpperBoundType は以下の条件によって決定される。

- LowerBound(UpperBound)が指定されているとき
  LowerBoundType(UpperBoundType)が指定されていれば、指定された値
  LowerBoundType(UpperBoundType)が指定されていなければ、Inclusive
- LowerBound(UpperBound)が指定されていないとき
  LowerBoundType(UpperBoundType)の指定に関わらず、Ignore

- 日付型範囲チェック(DateTimeRangeValidatorEx)

検証対象が日付(System.DateTime)型の数値または日付型に変換可能な文字列であり、指定した範囲に含まれているか検証する。

日付型範囲チェックでは、日付型に変換可能な文字列に対する範囲チェックも可能であるが、記述形式を指定することはできない。DateTimeConverter が対応した形式であれば検証対象となる。日付文字列の形式を指定するには、フレームワークで提供する日付文字列フォーマットチェック(DateTimeFormatValidator)と併用する。

また、属性のパラメータとして指定する日付文字列の形式は決まっている。その形式は.NET Framework 標準の書式指定文字列(“d”)に基づいており、日本語環境では”yyyy/MM/dd”となる。(比較時に扱われる日付値は、yyyy/MM/dd 0:00:00 となる)

以下に、検証成功となるパターンを示す。

- 通常時
  検証対象が日付型の値または日付型の値に変換可能な文字列であり、かつ、最小値と最大値の間に含まれる場合。
- 反転時(Negated = true の場合)
  検証対象が日付型の値または日付型の値に変換可能な文字列であり、かつ、最小値と最大値の間に含まれない場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 46 構成クラス

項番	クラス名	説明
1	DateTimeRangeValidatorEx	日付型範囲チェックを行う検証クラス
2	DateTimeRangeValidatorExAttribute	DateTimeRangeValidatorEx のインスタンスを生成し、プロパティに付与するための属性

表 47 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	LowerBound	-	検証対象である日付の最小値を”yyyy/MM/dd”形式で指定する。 ※設定値がプレースホルダの{3}に代入される。	なし(未入力ならば無視される)

2	LowerBoundType	-	LowerBound で指定した最小値の値の扱いを指定する RangeBoundaryType 列挙体	※「 LowerBoundType 、UpperBoundType の取りうる値」参照
3	UpperBound	-	検証対象である日付の最大値を”yyyy/MM/dd”形式で指定する。 ※設定値がプレースホルダの{5}に代入される。	なし(未入力ならば無視される)
4	UpperBoundType	-	UpperBound で指定した最小値の値の扱いを指定する RangeBoundaryType 列挙体	※「 LowerBoundType 、UpperBoundType の取りうる値」参照

LowerBoundType と UpperBoundType の組合せにより、デフォルトメッセージテンプレート文字列が異なる。以下にそれぞれのケースの出力値を示す。

表 48 デフォルトテンプレート文字列

(検証ルール:通常時)

項番	LowerBoundType	UpperBoundType	メッセージ
1	Ignore	Inclusive	"{2}" には “{5}” 以前の範囲の日付を入力してください。
2	Ignore	Exclusive	"{2}" には “{5}” より前の範囲の日付を入力してください。
3	Inclusive	Ignore	"{2}" には “{3}” 以後の範囲の日付を入力してください。
4	Inclusive	Inclusive	"{2}" には “{3}” 以後かつ “{5}” 以前の範囲の日付を入力してください。
5	Inclusive	Exclusive	"{2}" には “{3}” 以後かつ “{5}” より前の範囲の日付を入力してください。
6	Exclusive	Ignore	"{2}" には “{3}” より後の範囲の日付を入力してください。
7	Exclusive	Inclusive	"{2}" には “{3}” より後かつ “{5}” 以前の範囲の日付を入力してください。
8	Exclusive	Exclusive	"{2}" には “{3}” より後かつ “{5}” より後の範囲の日付を入力してください。

(検証ルール:反転時)

項番	LowerBoundType	UpperBoundType	メッセージ
9	Ignore	Inclusive	"{2}" には “{5}” より後の範囲の日付を入力してください。

10	Ignore	Exclusive	"{2}" には “{5}” 以後の範囲の日付を入力してください。
11	Inclusive	Ignore	"{2}" には “{3}” より前の範囲の日付を入力してください。
12	Inclusive	Inclusive	"{2}" には “{3}” より前または “{5}” より後の範囲の日付を入力してください。
13	Inclusive	Exclusive	"{2}" には “{3}” より前または “{5}” 以後の範囲の日付を入力してください。
14	Exclusive	Ignore	"{2}" には “{3}” 以前の範囲の日付を入力してください。
15	Exclusive	Inclusive	"{2}" には “{3}” 以前または “{5}” より後の範囲の日付を入力してください。
16	Exclusive	Exclusive	"{2}" には “{3}” 以前または “{5}” 以後の範囲の日付を入力してください。

以下に DateTimeRangeValidatorEx 属性を使用した設定例を示す。

```
[DateTimeRangeValidatorEx (
        LowerBound = "2009/04/01", LowerBoundType = RangeBoundaryType.Inclusive,
        UpperBound ="2010/04/01", UpperBoundType = RangeBoundaryType.Exclusive,
        Ruleset = "RS01")]
public string ShinseiDate{ get; set; }
```

リスト 19 DateTimeRangeValidatorEx 属性の設定例

- int 型範囲チェック(IntRangeValidator)
- float 型範囲チェック(FloatRangeValidator)
- double 型範囲チェック(DoubleRangeValidator)
- decimal 型範囲チェック(DecimalRangeValidator)

検証対象が、int(System.Int32)／float(System.Single)／double(System.Double)／decimal(System.Decimal)型(以下、各数値型)の数値または各数値型に変換可能な文字列であり、指定した範囲に含まれているか検証する。これら 4 つの検証クラスはフレームワークで提供する NumberRangeValidator を継承する。

以下に、検証成功となるパターンを示す。

- 通常時
  検証対象が、各数値型の値または各数値型の値に変換可能な文字列であり、かつ、最小値と最大値の間に含まれる場合。
- 反転時(Negated = true の場合)
  検証対象が、各数値型の値または各数値型の値に変換可能な文字列であり、かつ、最小値と最大値の間に含まれない場合。

検証対象が null または空文字列である場合、Negated の値に関わらず、検証処理は実行されない。検証結果は必ず成功となる。null または空文字列を許容しない場合には、フレームワークで提供する必須入力チェック(RequiredValidator)と併用する。

表 49 構成クラス

項番	クラス名	説明
1	IntRangeValidator	int 型範囲チェックを行う検証クラス
2	IntRangeValidatorAttribute	IntRangeValidator のインスタンスを生成し、プロパティに付与するための属性
3	FloatRangeValidator	float 型範囲チェックを行う検証クラス
4	FloatRangeValidatorAttribute	FloatRangeValidator のインスタンスを生成し、プロパティに付与するための属性
5	DoubleRangeValidator	double 型範囲チェックを行う検証クラス
6	DoubleRangeValidatorAttribute	DoubleRangeValidator のインスタンスを生成し、プロパティに付与するための属性
7	DecimalRangeValidator	decimal 型範囲チェックを行う検証クラス
8	DecimalRangeValidatorAttribute	DecimalRangeValidator のインスタンスを生成し、プロパティに付与するための属性

表 50 固有設定項目(共通)

項番	パラメータ名	必須	説明	デフォルト値
1	LowerBound	-	検証対象である数値型の値の最小値を指定する。 ※設定値がプレースホルダの{3}に代入される。	なし(未入力ならば無視される)
2	LowerBoundType	-	LowerBound で指定した最小値の値の扱いを指定する RangeBoundaryType 列挙体	※「 LowerBoundType 、UpperBoundType の取りうる値」参照
3	UpperBound	-	検証対象である数値型の値の最大値を指定する。 ※設定値がプレースホルダの{5}に代入される。	なし(未入力ならば無視される)
4	UpperBoundType	-	UpperBound で指定した最小値の値の扱いを指定する RangeBoundaryType 列挙体	※「 LowerBoundType 、UpperBoundType の取りうる値」参照

LowerBoundType と UpperBoundType の組合せにより、デフォルトメッセージテンプレート文字列が異なる。以下にそれぞれのケースの出力値を示す。

表 51 デフォルトテンプレート文字列

(検証ルール:通常時)

項番	LowerBoundType	UpperBoundType	メッセージ
1	Ignore	Inclusive	"{2}" には {5} 以下の範囲の数値を入力してください。
2	Ignore	Exclusive	"{2}" には {5} 未満の範囲の数値を入力してください。
3	Inclusive	Ignore	"{2}" には {3} 以上の範囲の数値を入力してください。
4	Inclusive	Inclusive	"{2}" には {3} 以上 {5} 以下の範囲の数値を入力してください。
5	Inclusive	Exclusive	"{2}" には {3} 以上 {5} 未満の範囲の数値を入力してください。
6	Exclusive	Ignore	"{2}" には {3} より大きい範囲の数値を入力してください。
7	Exclusive	Inclusive	"{2}" には {3} より大きい {5} 以下の範囲の数値を入力してください。
8	Exclusive	Exclusive	"{2}" には {3} より大きい {5} 未満の範囲の数値を入力してください。

(検証ルール:反転時)

項番	LowerBoundType	UpperBoundType	メッセージ
9	Ignore	Inclusive	"{2}" には {5} より大きい範囲の数値を入力してください。
10	Ignore	Exclusive	"{2}" には {5} 以上の範囲の数値を入力してください。
11	Inclusive	Ignore	"{2}" には {3} 未満の範囲の数値を入力してください。
12	Inclusive	Inclusive	"{2}" には {3} 未満または {5} より大きい範囲の数値を入力してください。
13	Inclusive	Exclusive	"{2}" には {3} 未満または {5} より大きい範囲の数値を入力してください。
14	Exclusive	Ignore	"{2}" には {3} 以下の範囲の数値を入力してください。
15	Exclusive	Inclusive	"{2}" には {3} 以下または {5} より大きい範囲の数値を入力してください。
16	Exclusive	Exclusive	"{2}" には {3} 以下または {5} 以上の範囲の数値を入力してください。

以下に 4 つのうちの 1 つ IntRangeValidator 属性を使用した設定例を示す。

```
[IntRangeValidator (
        LowerBound = 0, LowerBoundType = RangeBoundaryType.Inclusive,
        UpperBound = 120, UpperBoundType = RangeBoundaryType.Exclusive,
        Ruleset = "RS01")]
public virtual string Age{ get; set; }
```

リスト 20 IntRangeValidator 属性の設定例

◆Validation Application Block が提供する入力値検証ルール

Enterprise Library の ValidationAB が提供する入力値検証ルールの機能と利用方法について解説する。

- byte 型チェック
- short 型チェック
- int 型チェック
- long 型チェック
- float 型チェック
- double 型チェック
- decimal 型チェック

  検証対象値が byte／short／int／long／float／double／decimal 値に変換可能かを検証する。これら 7 つの検証ルールは全て Validation AB が提供する TypeConversionValidator を利用する。

表 52 構成クラス

項番	クラス名	説明
1	TypeConversionValidator	指定した型に変換可能かを検証する検証クラス
2	TypeConversionValidatorAttribute	TypeConversionValidator のインスタンスを生成し、プロパティに付与するための属性

表 53 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	targetType	○	チェック対象の型を typeof(型名)で指定する。例えば byte 型チェックでは、typeof(byte)を指定する	なし

表 54 メッセージテンプレート文字列

項番	プレースホルダ	説明
1	{3}	変換対象の型名（targetType）

以下に TypeConversionValidator 属性（double 型チェック）を使用した設定例を示す。

```
[TypeConversionValidator (typeof(double), Ruleset = "RS01")]
public string DiscountString{ get; set; }
```

リスト 21 TypeConversionValidator 属性の設定例

- プロパティ比較チェック(PropertyComparisonValidator)
  他のプロパティ値と比較を行い、特定の条件を満たしているか否かを検証する。

表 55 構成クラス

項番	クラス名	説明
1	PropertyComparisonValidator	他のプロパティ値と比較を行い、条件を満たしているかを検証する検証クラス
2	PropertyComparisonValidatorAttribute	PropertyComparisonValidator のインスタンスを生成し、プロパティに付与するための属性

表 56 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	propertyToCompare	○	比較対象のプロパティ名	なし
2	comparisonOperator	○	二つのプロパティの比較方法を指定する ComparisonOperator 列挙体。 以下の値のいずれかとなる ・Equal(等しい) ・NotEqual(等しくない) ・GreaterThan(より大きい) ・GreaterThanEqual(以上) ・LessThan(より小さい) ・LessThanEqual(以下)	なし

表 57 メッセージテンプレート文字列

項番	プレースホルダ	説明
1	{3}	比較対象のプロパティの値
2	{4}	比較対象のプロパティ名(propertyToCompare)
3	{5}	比較方法(comparisonOperator)

以下に PropertyComparisonValidator 属性を使用した設定例を示す。

```
[PropertyComparisonValidator ("Password", ComparisonOperator.Equal,
        Ruleset = "RS01")]
public string ConfirmPassword{ get; set; }
```

リスト 22 PropertyComparisonValidator 属性の設定例

- 参照オブジェクト入力チェック(ObjectValidator)
  検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたオブジェクトを検証する。
  本ルールは、「SV-02 サーバ入力値検証機能」での利用を想定している。
  「CL-01 画面データ機能」では、RulesetMapping 属性を使用するため、本検証ルールは使用しないこと。

表 58 構成クラス

項番	クラス名	説明
1	ObjectValidator	検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたオブジェクトを検証する検証クラス
2	ObjectValidatorAttribute	ObjectValidator のインスタンスを生成し、プロパティに付与するための属性

表 59 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	targetRuleset		ネストオブジェクトに対して適用するルールセット名	なし

以下に ObjectValidator 属性を使用した設定例を示す。
この例では、ルールセット名"RS01"を適用した場合に、ネストした SampleClass オブジェクトにはルールセット名"CRS01"が適用される。

```
[ObjectValidator("CRS01", Ruleset="RS01")]
public SampleClass NestedProp { get; set; }
```

リスト 23 ObjectValidator 属性の設定例

- 参照コレクション入力チェック(ObjectCollectionValidator)
  検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたコレクションオブジェクトを検証する。
  本ルールは、「SV-02 サーバ入力値検証機能」での利用を想定している。
  「CL-01 画面データ機能」では、RulesetMapping 属性を使用するため、本検証ルールは使用しないこと。

表 60 構成クラス

項番	クラス名	説明
1	ObjectCollectionValidator	検証対象クラスが構造化されている場合に、指定されたルールセット名でネストしたコレクションオブジェクトを検証する検証クラス
2	ObjectValidatorAttribute	ObjectValidatorAttribute のインスタンスを生成し、プロパティに付与するための属性

表 61 固有設定項目

項番	パラメータ名	必須	説明	デフォルト値
1	targetType	○	検証したいネストオブジェクトの型	なし
2	targetRuleset		ネストオブジェクトに対して適用するルールセット名	なし

以下に ObjectCollectionValidator 属性を使用した設定例を示す。
この例では、ルールセット名"RS01"を適用した場合に、ネストした SampleClass オブジェクトにはルールセット名"CRS01"が適用される。

```
[ObjectCollectionValidator(typeof(SampleClass), "CRS01", Ruleset="RS01")]
public List<SampleClass> NestedCollectionProp { get; set; }
```

リスト 24 ObjectValidator 属性の設定例

■ 内部構成

◆構成クラス

本機能を構成するクラスを以下に示す。

表 62 構成クラス一覧

項番	クラス名	説明
Terasoluna.Validation.Configuration 名前空間		
1	AlphaNumericStringValidatorData	AlphaNumericStringValidator について、設定情報から Validator を生成するクラス。
2	ByteRangeValidatorData	ByteRangeValidator について、設定情報から Validator を生成するクラス。
3	CapAlphaNumericStringValidatorData	CapAlphaNumericStringValidator について、設定情報から Validator を生成するクラス。
4	ContainsCharactersValidatorExData	ContainsCharactersValidatorEx について、設定情報から Validator を生成するクラス。
5	DateTimeFormatValidatorData	DateTimeFormatValidator について、設定情報から Validator を生成するクラス。
6	DateTimeRangeValidatorExData	DateTimeRangeValidatorEx について、設定情報から Validator を生成するクラス。
7	DecimalRangeValidatorData	DecimalRangeValidator について、設定情報から Validator を生成するクラス。
8	DoubleRangeValidatorData	DoubleRangeValidator について、設定情報から Validator を生成するクラス。
9	FloatRangeValidatorData	FloatRangeValidator について、設定情報から Validator を生成するクラス。
10	HankakuKanaStringValidatorData	HankakuKanaStringValidator について、設定情報から Validator を生成するクラス。
11	HankakuStringValidatorData	HankakuStringValidator について、設定情報から Validator を生成するクラス。
12	IntRangeValidatorData	IntRangeValidator について、設定情報から Validator を生成するクラス。
13	NumberValidatorData	NumberValidator について、設定情報から Validator を生成するクラス。
14	NumericStringValidatorData	umericStringValidator について、設定情報から Validator を生成するクラス。
15	RegexValidatorExData	RegexValidatorEx について、設定情報から Validator を生成するクラス。
16	RequiredValidatorData	RequiredValidator について、設定情報から

		Validator を生成するクラス。
17	StringLengthRangeValidatorData	AlphaNumericStringValidator について、設定情報から Validator を生成するクラス。
18	UrlValidatorData	AlphaNumericStringValidator について、設定情報から Validator を生成するクラス。
19	ZenkakuKanaStringValidatorData	kakuKanaStringValidator について、設定情報から Validator を生成するクラス。
20	ZenkakuStringValidatorData	ZenkakuStringValidator について、設定情報から Validator を生成するクラス。
Terasoluna.Validation.Validators 名前空間		
21	AlphaNumericStringValidator	半角英数文字列チェックを行う検証クラス
22	AlphaNumericStringValidatorAttribute	AlphaNumericStringValidator について属性から Validator を生成するクラス。
23	ByteRangeValidator	エンコーディングを指定して文字列のバイト長範囲チェックを行う検証クラス
24	ByteRangeValidator Attribute	ByteRangeValidator について属性から Validator を生成するクラス。
25	CapAlphaNumericStringValidator	半角英数大文字列チェックを行う検証クラス
26	CapAlphaNumericStringValidator Attribute	CapAlphaNumericStringValidator について属性から Validator を生成するクラス。
27	ContainsCharactersValidatorEx	指定した文字列を含んでいるかどうかを検証する
28	ContainsCharactersValidatorEx Attribute	ContainsCharactersValidatorEx について属性から Validator を生成するクラス。
29	DateTimeFormatValidator	日付形式文字列チェックを行う検証クラス
30	DateTimeFormatValidatorAttribute	DateTimeFormatValidator について属性から Validator を生成するクラス。
31	DateTimeRangeValidatorEx	日付の範囲チェックを行う検証クラス
32	DateTimeRangeValidatorExAttribute	DateTimeRangeValidatorEx について属性から Validator を生成するクラス。
33	DecimalRangeValidator	Decimal 値の範囲チェックを行う検証クラス
34	DecimalRangeValidatorAttribute	DecimalRangeValidator について属性から Validator を生成するクラス。
35	DoubleRangeValidator	Double 値の範囲チェックを行う検証クラス
36	DoubleRangeValidatorAttribute	DoubleRangeValidator について属性から Validator を生成するクラス。

37	EventSpecificValidator	イベント処理実行時にのみ実施する検証処理に付与するマーカクラス。カスタム入力チェックエラー情報が即値チェック時に消えないようにする。
38	EventSpecificValidationAttribute	EventSpecificValidator について属性から Validator を生成するクラス。
39	FieldValueAcces	フィールド情報を保持し、指定された値にアクセスするクラス
40	FloatRangeValidator	Float 値の範囲チェックを行う検証クラス
41	FloatRangeValidatorAttribute	FloatRangeValidator について属性から Validator を生成するクラス。
42	HankakuKanaStringValidator	半角カナ文字列チェックを行う検証クラス
43	HankakuKanaStringValidatorAttribute	HankakuKanaStringValidator について属性から Validator を生成するクラス。
44	HankakuStringValidator	半角文字列チェックを行う検証クラス
45	HankakuStringValidatorAttribute	HankakuStringValidator について属性から Validator を生成するクラス。
46	IntRangeValidator	Int 値の範囲チェックを行う検証クラス
47	IntRangeValidatorAttribute	IntRangeValidator について属性から Validator を生成するクラス。
48	MethodValueAccess	メソッド情報を保持し、指定された値にアクセスするクラス
49	NumberRangeValidator	数値型の範囲チェックを行う Validator の基底クラス
50	NumberValidator	数値の桁数チェックを行う検証クラス
51	NumberValidatorAttribute	NumberValidator について属性から Validator を生成するクラス。
52	NumericStringValidator	半角数字文字列チェックを行う検証クラス
53	NumericStringValidatorAttribute	NumericStringValidator について属性から Validator を生成するクラス。
54	PropertyValueAccess	プロパティ情報を保持し、指定された値にアクセスするクラス
55	ReflectionMemberValueAccessBuilder	FieldValueAcces 、MethodValueAccess 、PropertyValueAccess を生成するクラス
56	RegexValidatorEx	正規表現チェックを行う検証クラス

57	RegexValidatorExAttribute	RegexValidatorEx について属性から Validator を生成するクラス。
58	RequiredValidator	必須チェックを行う検証クラス
59	RequiredValidatorAttribute	RequiredValidator について属性から Validator を生成するクラス。
60	StringLengthRangeValidator	文字列長チェックを行う検証クラス
61	StringLengthRangeValidatorAttribute	StringLengthRangeValidator について属性から Validator を生成するクラス。
62	TypeRangeValidator	型指定した範囲チェックを行う Validator の基底クラス
63	UrlValidator	URL 形式文字列チェックを行う検証クラス
64	UrlValidatorAttribute	UrlValidator について属性から Validator を生成するクラス。
65	ZenkakuKanaStringValidator	全角カナ文字列チェックを行う検証クラス
66	ZenkakuKanaStringValidatorAttribute	ZenkakuKanaStringValidator について属性から Validator を生成するクラス。
67	ZenkakuStringValidator	全角文字列チェックを行う検証クラス
68	ZenkakuStringValidatorAttribute	ZenkakuStringValidator について属性から Validator を生成するクラス。

■ 拡張ポイント

◆デフォルトメッセージテンプレートの変更

入力値検証エラー時に表示されるメッセージは、フレームワーク内で定義されたリソースファイル(Terasoluna.Validation.dll の ValidationResources.resx)でデフォルトメッセージを管理している。メッセージを変更する場合は、「CM-07 メッセージ管理機能」により、デフォルトメッセージの置き換えを実施する。
詳細な手順については、「CM-07 メッセージ管理機能」の機能説明書を参照のこと。

◆カスタム検証ルールの作成

カスタム検証ルールを作成するには、入力値検証を実施する検証クラス(Validator 継承クラス)と、Validator インスタンスを初期化し、プロパティ毎に付与するカスタム属性(ValidatorAttribute 継承クラス)を実装する。

実装方法の詳細は、ValidationAB のドキュメントおよび QuickStartsのサンプルコードを参照のこと。

表 63 入力値検証クラスの構成

項番	名称	説明
1	Validator クラス	Microsoft.Practices.EnterpriseLibrary.Validation.Validator を継承した入力値検証の実装クラス。 検証を行う DoValidate メソッドを実装する。
2	ValidatorAttribute クラス	Microsoft.Practices.EnterpriseLibrary.Validation.Validators .ValidatorAttribute を継承した、Validator を生成・初期化し、属性として付与できるようにするクラス。 属性のパラメータから Validator を生成するための DoCreateValidator メソッドを実装する。 AttributeUsage 属性をクラスに付与する。

数字文字列のみを許可するカスタム検証ルールの実装例を、以下に示す。Validation AB の提供する正規表現チェックを拡張して実装し、対応する ValidatorAttribute クラスを作成する。

この例では、既存の RegexValidator クラスを継承し、SampleValidator は数字文字列を表す固定の正規表現を用いて入力値検証を行う。RegexValidator はコンストラクタの第一引数で正規表現パターン文字列を受け取るため、SampleValidator は親クラスのコンストラクタに固定の正規表現パターンを渡すことで機能を実現している。

```
public class SampleValidator : RegexValidator
{
    /// <summary>
    /// 正規表現を利用するValidatorのコンストラクタでパターンを固定
    /// </summary>
    public SampleValidator(string messageTemplate, bool negated)
        : base(@"[0-9]*", messageTemplate, negated)
    {
    }
}
```

リスト 25 Validator 継承クラスの実装例

```
[AttributeUsage(AttributeTargets.Property
| AttributeTargets.Field
| AttributeTargets.Method
| AttributeTargets.Class
| AttributeTargets.Parameter,
AllowMultiple = true,
Inherited = false)]
public class SampleValidatorAttribute : ValueValidatorAttribute
{
    protected override Validator DoCreateValidator(Type targetType)
    {
        // Validatorを生成
        return new SampleValidator(MessageTemplate, Negated);
    }
}
```

リスト 26 ValidatorAttribute 継承クラスの実装例

作成したカスタム検証ルールは以下のように利用する。

```
[SampleValidator(MessageTemplate="{2}は、数値を入力してください", Ruleset="RS01")]
public virtual string NumStr1 { get; set; }
```

リスト 27 カスタムバリデータの利用例

また、バリデータに固有の設定項目を追加する場合、ValidatorAttribute のパラメータ(引数)を定義する。そして、パラメータで設定された値を、Validator のコンストラクタとして利用する。

```
[AttributeUsage(AttributeTargets.Property
| AttributeTargets.Field
| AttributeTargets.Method
| AttributeTargets.Class
| AttributeTargets.Parameter,
AllowMultiple = true,
Inherited = false)]
public class SampleValidatorAttribute : ValueValidatorAttribute
{
    /// バリデータに追加する固有設定項目のパラメータ
    private string customSettingField;

    /// コンストラクタ
    /// バリデータに追加する固有設定項目のパラメータ
    public SampleValidatorAttribute(string customSettingField) : base()
    {
        this.customSettingField = customSettingField;
    }

    protected override Validator DoCreateValidator(Type targetType)
    {
        // StringValidatorを生成
        return new SampleValidator(customSettingField, MessageTemplate, Negated);
    }
}
```

リスト 28 ValidatorAttribute クラス(固有設定項目あり)

作成したカスタム検証ルールは以下のように利用する。カスタム属性に、コンスラクタに追加したパラメータを指定することができるようになる。

```
[SampleValidator("追加されたパラメータ",
        MessageTemplate = "{2}は、数値を入力してください", Ruleset = "RS01")]
public string NumStr1 { get; set; }
```

リスト 29 カスタムバリデータの利用例(固有設定項目あり)

入力値検証のロジックを独自に定義するには、Validator を実装する際、DoValidate メソッドをオーバーライドする。その際、ValueValidator を親クラスに持つ場合には、negated が true と指定されている場合の検証条件の反転を考慮すること。

■ 構成ファイルによるチェックルール定義

本機能説明書では、属性によりチェックルールを定義する方法を示してきた。
Validation AB ではアプリケーション構成ファイル（App.config）や Web 構成ファイル(Web.config)にチェックルールを定義することができるが、本機能が提供する Validator を併用することも可能である。
ただし、TERASOLUNA フレームワークでは属性によるルール定義を推奨している。
これは、ソースの可読性の高さや共通化設計などの保守性の観点、インテリセンスやカスタムスニペット、コンパイルエラーでの誤り検出などの生産性の観点で、属性による定義の方が有利なためである。
また、「CL-01 画面データ機能」は属性によるルール定義を想定しており、構成ファイルによるルール定義はサーバ側でのみ利用可能である。
以下に、構成ファイルおよび入力値検証処理の実装例を示す。
構成ファイルの設定方法は、Validation AB のドキュメントを参照すること。

```
class Program
{
    public static void Main(string[] args)
    {
        ///検証対象オブジェクト(Idは、必須入力チェックエラー、Nameは全角文字チェックエラー)
        TargetClass target = new TargetClass() { Id = null, Name = "aaaa" };
        ///App.configで定義したチェックルールをもとにバリデータを生成
        Validator valiator = ValidationFactory.CreateValidator<TargetClass>("RS01");
        ///入力値検証
        ValidationResults results = valiator.Validate(target);
        if (results.IsValid)
        {
            Console.WriteLine("入力値検証成功");
        }
        else
        {
            Console.WriteLine("入力値検証エラー");
            StringBuilder sb = new StringBuilder();
            foreach (ValidationResult result in results)
            {
                sb.AppendLine(result.Message);
            }
            Console.WriteLine(sb.ToString());
        }
            Console.ReadLine();
    }
}

///入力検証対象クラス
public class TargetClass
{
    public string Id { get; set; }
    public string Name { get; set; }
}
```

リスト 30 入力値検証処理の実行例

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="validation"
type="Microsoft.Practices.EnterpriseLibrary.Validation.Configuration.ValidationSettings,
Microsoft.Practices.EnterpriseLibrary.Validation, Version=4.1.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
  </configSections>
  <validation>
    <!-- 対象クラス(TargetClass) のチェックルール定義 -->
    <type assemblyName="ConsoleApplication5, Version=1.0.0.0, Culture=neutral,
PublicKeyToken=null" name="ConsoleApplication5.TargetClass">
      <!-- ルールセット RS01の定義-->
      <ruleset name="RS01">
        <properties>
          <!-- Idプロパティのチェックルール -->
          <property name="Id">
            <!-- 必須入力チェック(RequiredValidator) -->
            <validator messageTemplate="" messageTemplateResourceName=""
              messageTemplateResourceType="" tag="Id"
                  type="Terasoluna.Validation.Validators.RequiredValidator,
                        Terasoluna.Validation"
              name="Required Validator" />
          </property>
          <!-- Nameプロパティのチェックルール -->
          <property name="Name">
            <!-- 必須入力チェック(RequiredValidator) -->
            <validator messageTemplate="" messageTemplateResourceName=""
              messageTemplateResourceType="" tag="名前"
                  type="Terasoluna.Validation.Validators.RequiredValidator,
                        Terasoluna.Validation"
              name="Required Validator" />
            <!-- 全角文字列チェック(ZenkakuStringValidator) -->
            <validator messageTemplate="" messageTemplateResourceName=""
              messageTemplateResourceType="" tag="名前"
                  type="Terasoluna.Validation.Validators.ZenkakuStringValidator,
                        Terasoluna.Validation"
              name="ZenkakuStringValidator" />
          </property>
        </properties>
      </ruleset>
    </type>
  </validation>
</configuration>
```

リスト 31 アプリケーション構成ファイル(App.config)の設定例

```
入力値検証エラー
"Id" は入力必須項目です。
"名前" には全角文字を入力してください。
```

リスト 32 サンプルの実行結果

■ 関連機能

- CL-01 画面データ機能
- CL-03 イベント処理実行機能
- SV-02 サーバ入力値検証機能
- CM-07 メッセージ管理機能