LinksPlatform's Platform.Interfaces Class Library

./IConverter[T].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a converter between two values of the same type.</para>
    /// <para>Определяет конвертер между двумя значениями одного типа.</para>
    /// </summary>
    /// <typeparam name="T"><para>Type of value to convert.</para><para>Тип преобразуемого
    ↪  значения.</para></typeparam>
    public interface IConverter<T> : IConverter<T, T>
    {
    }
}
```

./IConverter[TSource, TTarget].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a converter between two types (TSource and TTarget).</para>
    /// <para>Определяет конвертер между двумя типами (исходным TSource и целевым
    ↪  TTarget).</para>
    /// </summary>
    /// <typeparam name="TSource"><para>Source type of conversion.</para><para>Исходный тип
    ↪  конверсии.</para></typeparam>
    /// <typeparam name="TTarget"><para>Target type of conversion.</para><para>Целевой тип
    ↪  конверсии.</para></typeparam>
    public interface IConverter<in TSource, out TTarget>
    {
        /// <summary>
        /// <para>Converts the value of the source type (TSource) to the value of the target
        ↪  type.</para>
        /// <para>Конвертирует значение исходного типа (TSource) в значение целевого типа.</para>
        /// </summary>
        /// <param name="source"><para>The source type value (TSource).</para><para>Значение
        ↪  исходного типа (TSource).</para></param>
        /// <returns><para>The value is converted to the target type
        ↪  (TTarget).</para><para>Значение ковертированное в целевой тип
        ↪  (TTarget).</para></returns>
        TTarget Convert(TSource source);
    }
}
```

./ICounter[TArgument, TResult].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a counter that requires passing an argument to perform a count.</para>
    /// <para>Определяет счетчик, который требует передачи аргумента для выполнения
    ↪  подсчёта.</para>
    /// </summary>
    /// <typeparam name="TArgument"><para>The argument type.</para><para>Тип
    ↪  аргумента.</para></typeparam>
    /// <typeparam name="TResult"><para>The count result type.</para><para>Тип результата
    ↪  подсчета.</para></typeparam>
    public interface ICounter<in TArgument, out TResult>
    {
        /// <summary>
        /// <para>Performs a count that requires passing an argument.</para>
        /// <para>Выполняет посчёт, для которого требуется передача аргумент.</para>
        /// </summary>
        /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
        /// <returns><para>The count result.</para><para>Результат подсчёта.</para></returns>
        TResult Count(TArgument argument);
    }
}
```

./ICounter[TResult].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a counter.</para>
    /// <para>Определяет счетчик.</para>
    /// </summary>
    /// <typeparam name="TResult"><para>The count result type.</para><para>Тип результата
    ↪  подсчета.</para></typeparam>
```

```
    public interface ICounter<out TResult>
    {
        /// <summary>
        /// <para>Performs a count.</para>
        /// <para>Выполняет посчёт.</para>
        /// </summary>
        /// <returns><para>The count result.</para><para>Результат подсчёта.</para></returns>
        TResult Count();
    }
}
```

./ICriterionMatcher.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a criterion matcher, that contains a specific method of determining whether the argument matches criterion or not.</para>
    /// <para>Определяет объект который проверяет соответствие критерию и содержащий конкретный метод определения, соответствует ли аргумент критерию или нет.</para>
    /// </summary>
    /// <typeparam name="TArgument"><para>Argument type.</para><para>Тип аргумента.</para></typeparam>
    public interface ICriterionMatcher<in TArgument>
    {
        /// <summary>
        /// <para>Determines whether the argument matches the criterion.</para>
        /// <para>Определяет, соответствует ли аргумент критерию.</para>
        /// </summary>
        /// <param name="argument"><para>The argument.</para><para>Аргумент.</para></param>
        /// <returns><para>A value that determines whether the argument matches the criterion.</para><para>Значение, определяющие соответствует ли аргумент критерию.</para></returns>
        bool IsMatched(TArgument argument);
    }
}
```

./IFactory.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a factory that produces instances of a specific type.</para>
    /// <para>Определяет фабрику, которая производит экземпляры определенного типа.</para>
    /// </summary>
    /// <typeparam name="TProduct"><para>Type of produced instances.</para><para>Тип производимых экземпляров.</para></typeparam>
    public interface IFactory<out TProduct>
    {
        /// <summary>
        /// <para>Creates an instance of TProduct type.</para>
        /// <para>Создает экземпляр типа TProduct.</para>
        /// </summary>
        /// <returns><para>The instance of TProduct type.</para><para>Экземпляр типа TProduct.</para></returns>
        TProduct Create();
    }
}
```

./IIncrementer.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines an incrementer that increments any number with a specific type.</para>
    /// <para>Определяет инкрементер, который выполняет приращение любого числа определенного типа.</para>
    /// </summary>
    /// <typeparam name="TNumber"><para>Type of incremented number.</para><para>Тип увеличиваемого числа.</para></typeparam>
    public interface IIncrementer<TNumber>
    {
        /// <summary>
        /// <para>Increments the number by a specific value.</para>
        /// <para>Увеличивает число на определённое значение.</para>
        /// </summary>
        /// <param name="number"><para>The number to be incremented.</para><para>Увеличиваемое число.</para></param>
        /// <returns><para>The incremented number.</para><para>Увеличенное число.</para></returns>
```

```
        TNumber Increment(TNumber number);
    }
}
```

./IPropertiesOperator.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a properties operator that is able to get or set values of properties of a object of a specific type.</para>
    /// <para>Определяет оператор свойств, который может получать или устанавливать значения свойств объекта определенного типа.</para>
    /// </summary>
    /// <typeparam name="TObject"><para>Object type.</para><para>Тип объекта.</para></typeparam>
    /// <typeparam name="TProperty"><para>Property reference type.</para><para>Тип ссылки на свойство.</para></typeparam>
    /// <typeparam name="TValue"><para>Property value type.</para><para>Тип значения свойства.</para></typeparam>
    public interface IPropertiesOperator<in TObject, in TProperty, TValue>
    {
        /// <summary>
        /// <para>Gets the value of the property in the specified object.</para>
        /// <para>Получает значение свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="object"><para>The object reference.</para><para>Ссылка на объект.</para></param>
        /// <param name="property"><para>The property reference.</para><para>Ссылка на свойство.</para></param>
        /// <returns><para>The value of the property.</para><para>Значение свойства.</para></returns>
        TValue GetValue(TObject @object, TProperty property);

        /// <summary>
        /// <para>Sets the value of a property in the specified object.</para>
        /// <para>Устанавливает значение свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="object"><para>The object reference.</para><para>Ссылка на объект.</para></param>
        /// <param name="property"><para>The property reference.</para><para>Ссылка на свойство.</para></param>
        /// <param name="value"><para>The value.</para><para>Значение</para></param>
        void SetValue(TObject @object, TProperty property, TValue value);
    }
}
```

./IPropertyOperator.cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines a specific property operator that is able to get or set values of that property.</para>
    /// <para>Определяет оператор определённого свойства, который может получать или устанавливать его значения.</para>
    /// </summary>
    /// <typeparam name="TObject"><para>Object type.</para><para>Тип объекта.</para></typeparam>
    /// <typeparam name="TValue"><para>Property value type.</para><para>Тип значения свойства.</para></typeparam>
    public interface IPropertyOperator<in TObject, TValue> : IProvider<TValue, TObject>
    {
        /// <summary>
        /// <para>Sets the value of a specific property in the specified object.</para>
        /// <para>Устанавливает значение определённого свойства в указанном объекте.</para>
        /// </summary>
        /// <param name="object"><para>The object reference.</para><para>Ссылка на объект.</para></param>
        /// <param name="value"><para>The value.</para><para>Значение.</para></param>
        void Set(TObject @object, TValue value);
    }
}
```

./IProvider[TProvided].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines the provider of objects.</para>
    /// <para>Определяет поставщика объектов.</para>
```

```
    /// </summary>
    /// <typeparam name="TProvided"><para>Type of provided object.</para><para>Тип
    ↪  предоставляемого объекта.</para></typeparam>
    public interface IProvider<out TProvided>
    {
        /// <summary>
        /// <para>Provides an object.</para>
        /// </summary>
        /// <returns><para>The provided object.</para></returns>
        TProvided Get();
    }
}
```

./IProvider[TProvided, TArgument].cs

```
namespace Platform.Interfaces
{
    /// <summary>
    /// <para>Defines the provider of objects for which an argument must be specified.</para>
    /// <para>Определяет поставщика объектов, для получения которых необходимо указать
    ↪  аргумент.</para>
    /// </summary>
    /// <typeparam name="TProvided"><para>Type of provided objects.</para><para>Тип
    ↪  предоставляемых объектов.</para></typeparam>
    /// <typeparam name="TArgument"><para>Argument type.</para><para>Тип
    ↪  аргумента.</para></typeparam>
    public interface IProvider<out TProvided, in TArgument>
    {
        /// <summary>
        /// <para>Provides an object.</para>
        /// <para>Предоставляет объект.</para>
        /// </summary>
        /// <param name="argument"><para>The argument required to acquire the
        ↪  object.</para><para>Аргумент, необходимый для получения объекта.</para></param>
        /// <returns><para>The object.</para><para>Объект.</para></returns>
        TProvided Get(TArgument argument);
    }
}
```

Index