String Extension Object Parser

The String Extension Object Parser is a utility class that provides extension methods to parse strings containing object references, nested objects, properties, fields, and methods. It allows you to dynamically resolve and replace placeholders in the string with actual values from the provided object.

Version

Current version: 0.9.0

Namespace

You will need to set access to the following Namespace in order to use this ParseObject string extension.

```
using DLS.Utilities.Extensions.StringExtensions;
```

Usage

The String Parsing Extension class provides the following method:

`ParseObject(this string text, object target)`

This method parses the given `text` string by replacing placeholders with the corresponding values from the `target` object. It returns the modified string with resolved placeholders.

Parameters

- `text` (string): The string containing placeholders to be replaced.
- `target` (object): The object containing the values to replace the placeholders.

Return Value

- The modified string with resolved placeholders.

Supported Syntax

The String Extension Object Parser supports the following syntax to represent object references, properties, fields, and methods in the input string:

Object References

You can reference an object by its name. The parser will look for a matching variable in the `target` object.

Example:

```
string text = "Hello, {user.Name}!";
User user = new User { Name = "John" };
```

```
string result = text.ParseObject(user);
// Output: "Hello, John!"
```

Nested Objects

You can access properties or fields of nested objects by chaining the property or field names using the dot notation.

Example:

```
string text = "{company.Worker.Name} is a worker for {company.Name}.";
Worker worker = new Worker { Name = "Jerry", Age = 31 };
Company company = new Company { Name = "ABC Corp", Worker = worker, };
string result = text.ParseObject(company);
// Output: "Jerry is a worker for ABC Corp."
```

Properties

You can directly reference properties of an object by their names.

Example:

```
string text = "The product name is {product.Name}.";
Product product = new Product { Name = "XYZ Widget" };
string result = text.ParseObject(product);
// Output: "The product name is XYZ Widget."
```

Fields

You can directly reference fields of an object by their names.

Example:

```
string text = "The temperature is {sensor.Temperature}.";
Sensor sensor = new Sensor { Temperature = 25.5 };
string result = text.ParseObject(sensor);
// Output: "The temperature is 25.5."
```

Methods

You can invoke methods on an object by appending the method name followed by parentheses () to the object reference.

Example:

```
string text = "The result is {calculator.Add(5, 3)}.";
Calculator calculator = new Calculator();
string result = text.ParseObject(calculator);
// Output: "The result is 8."
```

Method Parameters

You can pass parameters to methods by providing the parameter values within the parentheses `()`. The parameters can be constants, strings, numbers, or nested property references.

Example:

```
string text = "The result is {calculator.Add({a}, {b})}.";
Calculator calculator = new Calculator();
string result = text.ParseObject(new { a = 5, b = 3, calculator });
// Output: "The result is 8."
```

Indexers

You can access indexed properties or collections using square brackets `[]` notation.

Example:

```
string text = "The value at index 0 is {array[0]}.";
int[] array = { 10, 20, 30 };
string result = text.ParseObject(array);
// Output: "The value at index 0 is 10."
```

Nested Indexers

You can chain indexers to access nested indexed properties or collections.

Example:

```
string text = "The value at row 1, column 2 is {matrix[1][2]}.";
int[][] matrix = { new[] { 1, 2, 3 }, new[] { 4, 5, 6 } };
string result = text.ParseObject(matrix);
// Output: "The value at row 1, column 2 is 6."
```

Limitations

- The String Extension Object Parser supports public properties and fields only.
- Method parameters can be constants, strings, numbers, or nested property references. They cannot be expressions or complex calculations.
- The parser uses reflection to access properties, fields, and methods, which may impact performance for large objects or frequent parsing operations.

That's it! You now have the necessary information to use the String Extension Object Parser. Enjoy parsing and resolving object references in your strings dynamically!