**Introduction to the C# Regex Library**

The .NET framework provides a powerful library for working with regular expressions in C# through the **System.Text.RegularExpressions** namespace. Regular expressions, often referred to as regex, are patterns used to match and manipulate text. The C# **Regex** class allows you to define these patterns and perform various operations like searching for matches, extracting data, and replacing text based on these patterns. Below, we'll delve into the functioning of the C# Regex library with detailed explanations and examples.

**Creating a Regex Object**

A **Regex** object is your entry point into using regular expressions in C#. To create one, you define a regex pattern and compile it using the **Regex** class.

**Csharp code:**

using System.Text.RegularExpressions; string pattern = @"\d{3}-\d{2}-\d{4}"; Regex regex = new Regex(pattern);

In this example, we've created a **Regex** object to match the pattern for a social security number.

**Matching with the Match Method**

You can use the **Match** method to find the first occurrence of the regex pattern in an input string.

**csharp code:**

string input = "My social security number is 123-45-6789."; Match match = regex.Match(input); if (match.Success) { Console.WriteLine("Match found: " + match.Value); }

Here, we search for the pattern within the **input** string and, if a match is found, we retrieve and print the matched text.

**Matching All Occurrences with the Matches Method**

To find all occurrences of a pattern in an input string, you can use the **Matches** method.

**Csharp code:**

string input = "Phone numbers: 123-456-7890 and 987-654-3210."; MatchCollection matches = regex.Matches(input); foreach (Match match in matches) { Console.WriteLine("Match found: " + match.Value); }

The **Matches** method returns a collection of all matches found in the input string, which you can iterate through to access each match.

**Replacing with the Replace Method**

The **Regex.Replace** method allows you to replace all occurrences of a pattern in an input string with a specified replacement.

**csharp** **code:**

string input = "Hello, my email is user@example.com."; string replacement = "REDACTED"; string result = regex.Replace(input, replacement); Console.WriteLine("Modified input: " + result);

In this example, any matches of the regex pattern in the **input** string are replaced with the **REDACTED** text, producing the modified string.

**Common Regex Patterns and Elements**

Regular expressions consist of various elements that define patterns for matching text. Here are common elements:

**Literals**

Match literal characters directly, e.g., **abc** matches the string "abc."

**Character Classes**

Match any character from a set, e.g., **[aeiou]** matches any vowel.

**Quantifiers**

Specify how many times a character or group should appear, e.g., **\d{3}** matches exactly three digits.

**Anchors**

Indicate where the match should occur within the input string:

* **^** matches the start of a string.
* **$** matches the end of a string.
* **\b** matches word boundaries.

**Escaping Special Characters**

To match special characters like **.** or **\*** as literals, you need to escape them with a backslash, e.g., **\.** matches a period.

**Alternation**

Use the **|** symbol to specify alternatives, e.g., **cat|dog** matches either "cat" or "dog."

**Grouping**

Parentheses create groups, allowing you to apply quantifiers to multiple characters, e.g., **(abc)+** matches one or more repetitions of "abc."

**Character Escapes**

Backslashes followed by special characters match specific character types, e.g., **\d** matches digits, **\s** matches whitespace, and **\w** matches word characters.

Regular expressions can be simple or complex, depending on your specific needs. Complex patterns may require practice, but they are incredibly powerful for text processing and pattern matching tasks in C#.