List<KeyValuePair<TKey, TValue>> (C#)

The List<KeyValuePair<TKey, TValue>> is a generic list of key-value pairs in C#, which is used to store associations between keys and values in a list structure. This structure doesn't have specific built-in subtypes, but it can be used in various ways depending on the types of keys and values specified.

Here are some variations based on type usage that basically cover the primary ways List<KeyValuePair<TKey, TValue>> can be customized:

Simple Type Pairings:

List<KeyValuePair<string, string>>: Common for key-value pairs where both the key and value are strings.

List<KeyValuePair<string, int>>: Useful for mapping strings to integers (e.g., a list of names with associated numeric IDs).

List<KeyValuePair<int, double>>: Stores integer keys with floating-point values.

List<KeyValuePair<int, bool>>: Associates integer keys with boolean flags.

Complex Object Pairings:

List<KeyValuePair<string, object>>: Allows for flexible values (objects), where each value could potentially be a different type.

List<KeyValuePair<string, MyCustomClass>>: Pairs string keys with instances of a custom class you define.

List<KeyValuePair<int, AnotherCustomType>>: Useful for pairing integers with instances of a custom class.

Nested Pairings:

List<KeyValuePair<string, List<string>>>: A key with a list of values, often used for mapping a single key to multiple values.

List<KeyValuePair<string, Dictionary<int, string>>>: Maps keys to dictionaries, useful in scenarios where you need complex mappings.

List<KeyValuePair<int, KeyValuePair<int, string>>>: A nested key-value pair structure, good for hierarchical or multi-level mappings.

Nullable Types:

List<KeyValuePair<int?, string>>: Allows nullable keys (e.g., keys that can be either an integer or null).

List<KeyValuePair<string, double?>>: Allows nullable values.

An example of passing a list of <string> type key-value pairs to a method and then read and display the list of key-value pairs in the method.

```
using System;
using System.Collections.Generic;

class Program
{
  // define a function that accepts a list of key-value pairs
  static void ProcessKeyValuePairs(List<KeyValuePair<string, string>> keyValuePairs)
  {
    // iterate over the list and print the key-value pairs
    foreach (var kvp in keyValuePairs)
    {
      Console.WriteLine($"Key: {kvp.Key}, Value: {kvp.Value}");
    }
  }


  static void Main()
  {
    // create a list of key-value pairs
    List<KeyValuePair<string, string>> lst = new List<KeyValuePair<string, string>>()
    {
      new KeyValuePair<string, string>("Key1", "Value1"),
      new KeyValuePair<string, string>("Key2", "Value2"),
      new KeyValuePair<string, string>("Key3", "Value3")
    };

    // pass the list to the function (call the function)
    ProcessKeyValuePairs(lst);
  }
}
```

An example of extracting specific pairs from a List<KeyValuePair<string, string>> in C# using LINQ to filter the list based on a condition.

```
using System;
using System.Collections.Generic;
using System.Linq;

class Program
{
    static void Main()
    {
    // populate a sample list of KeyValuePairs
    List<KeyValuePair<string, string>> keyValuePairs = new List<KeyValuePair<string, string>>()
    {
      new KeyValuePair<string, string>("Key1", "Value1"),
      new KeyValuePair<string, string>("Key2", "Value2"),
      new KeyValuePair<string, string>("Key3", "Value3"),
      new KeyValuePair<string, string>("Key4", "Value4")
    };

    // define a condition to filter where key contains "Key2" or contains "Key4"
    var specificPairs = keyValuePairs
      .Where(kvp => kvp.Key == "Key2" || kvp.Key == "Key4")
      .ToList();

    // output the results
    foreach (var pair in specificPairs)
    {
      Console.WriteLine($"Key: {pair.Key}, Value: {pair.Value}");
    }
  }
}
```

An example of populating a List<KeyValuePair<string, string>> in C# using the SQLDataClient and stored procedure.

```
using System.Data.SqlClient;
using System.Data;
using System;
using System.Collections.Generic;

class Program
{
  static void Main()
  {
    List<KeyValuePair<string, string>> sqlret = GetMsgTypeByConv(param1, param2);
  }


  public static List<KeyValuePair<string, string>> GetMsgTypeByConv(string  param1, string param2)
  {
    List<KeyValuePair<string, string>>? sqldata = new List<KeyValuePair<string, string>>();

    try
    {
      // configuration["sqlsettings:cphdbintegrated"] contains the SQL connection string
      string? _sql_con_str = configuration["sqlsettings:cphdbintegrated"];
      System.Data.SqlClient.SqlConnection _sql_con = new System.Data.SqlClient.SqlConnection(_sql_con_str);
      System.Data.SqlClient.SqlCommand _sql_cmd = new System.Data.SqlClient.SqlCommand("TheStoredProcedure", _sql_con);
      _sql_cmd.CommandType = System.Data.CommandType.StoredProcedure;
      _sql_cmd.Parameters.AddWithValue("@param1", param1);
      _sql_cmd.Parameters.AddWithValue("@param2", param2);
      _sql_con.Open();
      System.Data.SqlClient.SqlDataReader _sql_dr = _sql_cmd.ExecuteReader();
      while (_sql_dr.Read())
      {
        sqldata.Add(new KeyValuePair<string, string>("sqldata1", _sql_dr["sqldata1"].ToString()));
        sqldata.Add(new KeyValuePair<string, string>("sqldata2", _sql_dr["sqldata2"].ToString()));
      }
      _sql_dr.Close();
      _sql_con.Close();
    }
    catch (Exception ex)
    {
      Console.WriteLine($"error: {(string)ex.Message}");
    }

    return sqldata;
  }
}
```