Dictionary<string, Action>()

A dictionary<TKey, TValue> is a collection that stores key-value pairs, where each unique key is associated with one value and is optimized for fast lookups, making it ideal for scenarios where quick retrieval of values by a key is required. Dictionaries use hashing to organize and access data, allowing O(1) average time complexity for lookup, insertion, and deletion operations. Keys must be unique, and both keys and values can be of any specified type.

An example of using a dictionary object to specify an Action delegate (referencing a method for execution) based on Key value.

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

class Program
{


  ContentResult MethodToExecute(string param1, string param1)
  {
    // your code here
    return Content("execution complete");
  }


  ContentResult AnotherMethodToExecute(string param1, string param1)
  {
    // your code here
    return Content("execution complete");
  }



  static void Main()
  {
    string stringtocheck = "string item 1 to check";
    var actionMap = new Dictionary<string, Action>
    {
      { "string item 1 to check", () => MethodToExecute(param1, param2) },
      { "string item 2 to check", () => AnotherMethodToExecute(param1, param2) },
      { "string item 3 to check", () => MethodToExecute(param1, param2) },
      { "string item 4 to check", () => MethodToExecute(param1, param2) },
    };

    // Check if the key exists and invoke the associated action
    if (actionMap.TryGetValue(stringtocheck, out var action))
    {
      action(); // Execute the corresponding "MethodToExecute"
    }
    else
    {
      Console.WriteLine($"error");
    }
  }
}
```

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;
  }
}
```