# Désérialisation

## Désérialisation d’un string en objet C#

|  |
| --- |
| **Modèle**  public class Account  {  public string Email { get; set; }  public bool Active { get; set; }  public DateTime CreatedDate { get; set; }  public IList<string> Roles { get; set; }  } |
| **Utilisation**  string json = @"{  'Email': 'james@example.com',  'Active': true,  'CreatedDate': '2013-01-20T00:00:00Z',  'Roles': [  'User',  'Admin'  ]  }";  Account account = JsonConvert.DeserializeObject<Account>(json);  Console.WriteLine(account.Email);  // james@example.com |

## Désérialisation d’un fichier en objet C#

|  |
| --- |
| static void deserialize\_to\_object()  {  var filename = "person.json";  if (!File.Exists(filename)) save\_to\_file(filename);  using (StreamReader sr = File.OpenText(filename))  {  var fileContent = sr.ReadToEnd();  Person p = JsonConvert.DeserializeObject<Person>(fileContent);  Console.WriteLine($"\*\*\*\*\* Contenu de {filename} \*\*\*\*\*");  Console.WriteLine(fileContent);  Console.WriteLine($"\*\*\*\*\* {nameof(Person)}.toString() \*\*\*\*\*");  Console.WriteLine(p);  }  } |

## Désérialisation d’un dictionnaire Json en C#

## Requis

Il faut avoir le package Newtonsoft.Json d’installer dans le projet.

## Exemple

Supposons que l’on se retrouve devant un dictionnaire Json dont les clés sont variables. Par exemple un format qui ressemble à ce qui suit.

|  |
| --- |
| {  "familles": {  "Denoncourt": ["David", "Olivier"],  "Grenier": [],  "Phaneuf": ["Charles", "Mia"],  "Landry": ["Jérôme", "William", "Thomas", "Xavier", "Juliette"],  }  } |

Première chose que l’on remarque, c’est que les clés sont variables. Deuxièmement, certains ont des valeurs vides.

Dans cette situation, une solution est d’utiliser un dictionnaire pour récupérer toutes les valeurs.

|  |
| --- |
| public class FamiliesModel  {  public Dictionary<string, List<string>> Families;  } |

Il sera ensuite possible de travailler avec la propriété Families.

|  |
| --- |
| using (HttpResponseMessage response = await ApiHelper.ApiClient.GetAsync(url))  {  if (response.IsSuccessStatusCode)  {  FamiliesModel result = await response.Content.ReadAsAsync<FamiliesModel>();  var enfant = result.Families.Values.ToList();  var familles = result.Families.Keys.ToList();  return familles;  }  else  {  throw new Exception(response.ReasonPhrase);  }  } |

# Sérialisation en format JSON

La sérialisation est le concept de convertir d’un format de donnée vers un format qui peut être transmis ou sauvegardé.

Un format populaire est le Json qui est un fichier texte structuré.

Dans ce document, on utilisera le paquet NuGet **NewtonSoft.Json** dans les divers projets

## Sérialisation d’un objet

La méthode **JsonConvert.SerializeObject()** permet de convertir la plupart des objets dans un string en format Json.

|  |
| --- |
| public static void simple\_object\_test()  {  var p = new Person  {  FirstName = "Nick",  LastName = "B",  City = "Shawinigan",  Birthday = DateTime.Parse("1980-01-01")  };  string resultat = JsonConvert.SerializeObject(p);  Console.WriteLine("Exemple de conversion d'un objet simple");  Console.WriteLine(resultat);  Console.WriteLine("Appuyez sur pause");  Console.ReadLine();  /\*\*  \* Sortie  \* Exemple de conversion d'un objet simple  \* {"FirstName":"Nick","LastName":"B","FullName":"B, Nick","Birthday":"1980-01-01T00:00:00","City":"Shawinigan","Province":null,"Email":null,"Mobile":null}  \* Appuyez sur pause  \* \*/  } |

La méthode **JsonConvert.SerializeObject()** peut prendre un seul paramètre soit l’objet à convertir, mais il prend aussi un formatage si l’on désire une présentation plus lisible.

|  |
| --- |
| //… Autre code  string resultat = JsonConvert.SerializeObject(monObjet, Formatting.Indented);  // … Autre code  /\*\*  \* Sortie :  \* Exemple de conversion avec formatage  {  "FirstName": "Nick",  "LastName": "B",  "FullName": "B, Nick",  "Birthday": "1980-01-01T00:00:00",  "City": "Shawinigan",  "Province": null,  "Email": null,  "Mobile": null  }  Appuyez sur pause  \*/ |

## Sérialisation d’une collection

Pour sérialiser une collection, on utilise encore **JsonConvert.SerializeObject()**.

|  |
| --- |
| static void serialize\_array()  {  var data = new List<Person>  {  new Person { FirstName = "Ayanna", LastName = "Vargas", Birthday = DateTime.Parse("1967-12-25"), City = "Pickering", Province = "ON", Email = "purus.in@semvitae.edu", Mobile = "647-142-8014" },  new Person { FirstName = "Whitney", LastName = "Parks", Birthday = DateTime.Parse("1978-03-24"), City = "Greater Sudbury", Province = "ON", Email = "consectetuer.euismod@adipiscingelit.net", Mobile = "624-767-4994" },  new Person { FirstName = "Louis", LastName = "Watts", Birthday = DateTime.Parse("1974-07-09"), City = "Fredericton", Province = "NB", Email = "at@gravidamolestie.ca", Mobile = "253-179-3847" },  new Person { FirstName = "Pamela", LastName = "Knapp", Birthday = DateTime.Parse("1985-03-13"), City = "Mission", Province = "BC", Email = "eget.dictum@Aliquamvulputate.ca", Mobile = "501-312-8343" },  };  var resultat = JsonConvert.SerializeObject(data, Formatting.Indented);  Console.WriteLine(resultat);  Console.WriteLine("Appuyez sur une touche.");  } |

## Écriture dans un fichier

Étant donné que le résultat est un string, il suffit d’écrire le Json dans un fichier texte à l’aide d’un **StreamWriter**.

|  |
| --- |
| string resultat = JsonConvert.SerializeObject(p, Formatting.Indented);  using (var tw = new StreamWriter(filename, true))  {  tw.WriteLine(resultat);  tw.Close();  } |
| {  "FirstName": "Nick",  "LastName": "B",  "FullName": "B, Nick",  "Birthday": "1980-01-01T00:00:00",  "City": "Shawinigan",  "Province": null,  "Email": null,  "Mobile": null  } |