PROGRAMMEZ!

Le magazine des développeurs

SPÉCIAL AUTOMNE
2020

SPÉCIAL

.NET 5
+
AZURE
+
WINDOWS

M 01642 - 2H - F: 6,50 € - RD

Le seul magazine écrit par et pour les développeurs

Printed in EU - Imprimé en UE - BELGIQUE 7 € - Canada 9,80 $ CAN - SUISSE 13,10 FS - DOM Surf 7,50 € - TOM 1020 XPF - MAROC 55 DH

NOUVEAU !

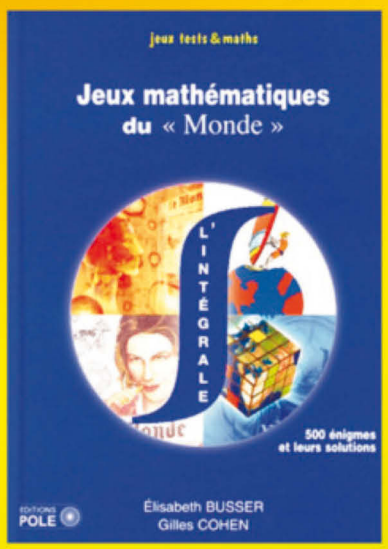
l'abonnement numérique

INTÉGRAL MATHS

Et aussi...

sur

affairedelogique.com



400 problèmes du Monde et leurs solutions

28 nos de Tangente

sur

tangente-mag.com

17 hors séries

53 nos de Tangente Éducation

sur

tangente-education.com

pour seulement

3,99 €

par mois

www.infinimath.com/librairie

.Net est mort, vive .Net
(et tout ce qui va avec)

Être développeur Windows, Xamarin, .Net, UWP… n'a jamais été un long code tranquille. Il faut bien avouer que Microsoft n'a pas facilité les choses : multiplication des couches, multiplication des noms, annonces et contre-annonces donnant une impression de flottements, abandon de technologies parfois après une très très très très longue agonie alors qu'il aurait fallu couper dans le vif.

Tout cela trouble le développeur : quelle technologie choisir ? Laquelle a le plus de chance de rester en vie dans les 5 ans à venir ? Quel avenir pour telle ou telle couche ? Un jour, on nous annonce la mort pure et simple de Win32, mais finalement, il faut le garder, car trop d'apps l'utilisent et que les développeurs sont mécontents. Par contre, quand IE6 fut définitivement arrêté : on pouvait entendre des cris de joie !

.Net, .Net Core, .Net Standard… Profusion de noms, incompréhension sur les noms, les équipes ont eu parfois du mal à préciser les choses. Il était temps de redonner un peu de cohérence et de logique à toutes ces briques techniques et surtout, fournir une vision d'ensemble à des développeurs parfois un peu perdus.

.Net 5 n'est finalement qu'une première étape mineure, mais importante. Mineure dans le sens où il s'agit d'une évolution très douce. Importante, dans le sens, que .Net 5 fournit la fondation nécessaire pour les grosses évolutions annoncées pour 2021 : .Net 6, intégration et dilution de Xamarin, projet Reunion, .Net MAUI, etc.

Finalement, la véritable rupture n'est pas .Net 5 et 2020, mais 2021. La dernière BUILD a dévoilé les grandes lignes. Aux équipes Microsoft de mener la mue.

Au-delà, on peut même se poser la question sur l'utilité, en 2020, du développeur desktop et donc du développeur .Net et Windows. C'est un peu comme si Apple rayait purement et simplement le développeur macOS. Même si l'OS desktop a perdu de son aura au profit du web, du mobile et du cloud, il reste primordial pour des millions d'applications, d'utilisateurs et de développeurs.

Rebootons la matrice.

François Tonic - ftonic@programmez.com
Boss de niveau 42

SOMMAIRE # Spécial automne 2020

Prochain numéro

Programmez! 243 disponible dès le 30 octobre

MEETUPS PROGRAMMEZ!

10 novembre : parlons serverless. Avec Aymeric Weinbach

22 décembre : C++20, avec Christophe Pichaud

19 janvier 2021 : être développeur en 2021, en partenariat avec Arolla

23 février : Java, en partenariat avec Arolla

DEVCON#9 : SPÉCIALE .NET 5 + AZURE + WINDOWS

+10 sessions & ateliers

19 novembre, à partir de 13h30

TOUS NOS ÉVÉNEMENTS SONT, POUR LE MOMENT, VIRTUELS.

Informations & inscription : programmez.com

NOVEMBRE

1	2	3	4	5	6	7
8	9	**10** Meetup Programmez!/	11	12	13	14
15	16	**17** DevDay (Belgique) / en ligne	**18** Sophi.A	**19** Sophi.A DevCon spéciale .Net 5 / en ligne Cloudnord 2020 / en ligne	**20** Sophi.A	21
22	**23** Voice Tech Paris	**24** Voice Tech Paris	25	26	27	28
29	30					

DÉCEMBRE

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22 Meetup Programmez!	23	24	25	26	27	28
29	30	31				

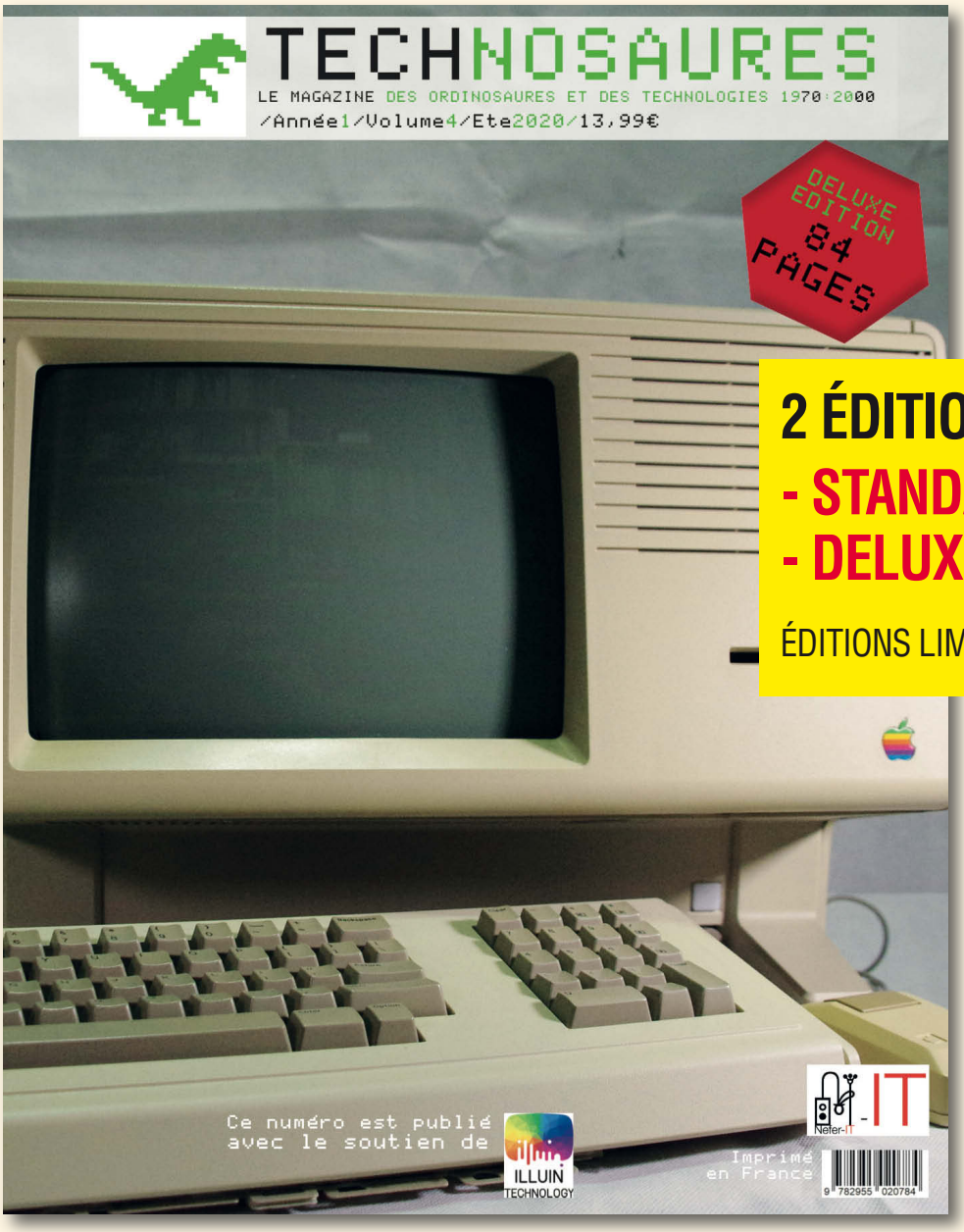
JANVIER 2021

1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	**19** Meetup Programmez!	20	21
22	23	24	25	26	27	28
29	30	31				

Merci à Aurélie Vache pour la liste 2020/2021, consultable sur son GitHub :
https://github.com/scraly/developers-conferences-agenda/blob/master/README.md

NUMÉRO EXCEPTIONNEL

100 % APPLE LISA



2 ÉDITIONS :

- STANDARD *52 PAGES*

- DELUXE *84 PAGES*

ÉDITIONS LIMITÉES

Commandez directement sur **www.programmez.com**

Standard édition : **8,99 €** *

Deluxe édition : **13,99 €** *

** Frais de port : 1,01 €*

TECHNOSAURES

le magazine du **rétro-computing**

ARRÊTONS DE PARLER DE DÉVELOPPEUR .NET ! PARLONS DÉVELOPPEUR C#

Le développeur .Net, ou plus largement sur les technologies Microsoft, n'est pas un profil rare. La popularité de C# prouve une certaine vivacité du langage. Mais parler de développeur .Net est pour nous une aberration, on devrait parler de développeur C# au sens large. Car si vous savez coder en C#, potentiellement vous savez coder avec toutes les technologies incluant C# (.Net, ASP.Net, Xamarin, Windows, etc.).

Malheureusement les cabinets de recrutement et les annonces, ainsi que les études sur les salaires, gardent cette terminologie obsolète.

Quel salaire peut-on espérer ? Comme toujours, les écarts moyens sont importants. La base serait :

- 28-34 k€ en profil junior donc -3 ans d'expérience
- 35-42 k€ en profil confirmé
- 42-53 k€ en profil senior hors Ile de France. 52-62 k€ en région Paris.

Comme toujours les fourchettes sont larges et nous retrouvons l'écart région parisienne et province. Nous sommes dans les grandes moyennes salariales. Nous ne tenons pas compte ici du variable ou des primes. Selon indeed, le salaire pivot serait 39 955 € bruts.

Le constat est grosso modo le même chez chooseyourboss :

- 34-40 k€ en junior
- 40-49 k€ en expérimenté
- 50-60 k€ en senior

Attention tout de même : la dynamique dans les régions et métropoles n'est pas homogène. La demande C# est en effet très diverse. Et il ne sera pas étonnant de devoir déménager ou faire plus de transport selon votre région / ville.

Comme nous l'a confié un responsable d'ESN dans le monde Microsoft, *« la rémunération est assez variable selon le candidat et le recruteur. En gros, quand on sort de reconversion, on démarre entre 30 et 34 k€. Quand on sort de l'école, on voit de tout mais globalement ce sera entre 36 et 42. J'ai un profil sorti d'alternative embauché à 51 k€. Pour des profils très précis ou un lead dev, on peut vite monter à 60 k€ ou plus. »*

Des entreprises, particulièrement les ESN, peuvent être prêtes à proposer des bonus élevés, des conditions de travail aménagés, voire un niveau salarial sortant de la norme, pour pouvoir recruter des profils, même s'ils ne correspondent pas totalement au besoin. Cette situation n'est pas nouvelle, ce qui change c'est son ampleur. La pénurie n'est pas générale, il faut le rappeler. Certains profils sont plus recherchés que d'autres.

Attention aux certifications

Dans de nombreuses technologies, il existe des certifications. Le monde Microsoft en propose de nombreuses pour tous les profils : développeur, DevOps, sécurité, data, admin, etc. Microsoft a décidé de supprimer plusieurs certifications : MCSA, MCSD, MCSE. Certaines le sont depuis cet été, d'autres prendront fin en janvier 2021.

Liste complète des certifications et examens retirés :
https://techcommunity.microsoft.com/t5/microsoft-learn-blog/mcsa-mcsd-mcse-certifications-retire-with-continued-investment/ba-p/1489670?BlogId=8&Id=375282

Certaines ESN ou entreprises demandent aux nouveaux développeurs recrutés de passer rapidement les certifications pour valider les acquis. C'est particulièrement le cas pour les nouveaux diplômés et les développeurs juniors. Mais comme nous l'ont précisé plusieurs développeurs, la qualité et le niveau des certifications varient. Attention : ces certifications sont payantes. Et certaines nécessitent plusieurs examens pour être validées. Avec l'évolution des outils et des technologies, il faut aussi repasser les examens de temps en temps.

La réelle reconnaissance vient quand un développeur devient MVP. Cela indique une implication dans la communauté et une connaissance de son domaine (C#, Visual Studio, etc.). Un MVP peut apporter une vrai valeur à une ESN, une équipe. L'entreprise doit aussi jouer le jeu. Et c'est aussi un argument pour négocier le salaire.

A partir de 2021, il y aura des changements majeurs. Xamarin se fondra dans .Net MAUI et disparaîtra comme technologie indépendante. Avec .Net 5, Microsoft adopte une mise à jour majeure annuelle.

Les principales annonces suite à la BUILD 2020 :

Fin 2020	2021	2022+
Août : Uno Platform 3.0	WinUI 3.0	WinUI 3.x
Septembre / octobre : Xamarin.Forms 5.0	Project Reunion	.Net 7, 8
Novembre : .Net 5, WinUI 3 Preview 3, C# 9	Visual Studio 16.x (?)	
Visual Studio 16.8	Automne (?) : .Net MAUI	
C#/WinRT	Novembre : .Net 6	
Project Reunion Preview		
.Net MAUI Preview		
WebView2		

Support : LTS et current

Soyez vigilant(e) sur les cycles de support de .Net. Officiellement :

.Net Core 2.1.x : LTS	.Net 6.0 : LTS
.Net Core 3.1.x : LTS	.Net 7.0 : current
.Net 5.0 : current	.Net 8.0 : LTS

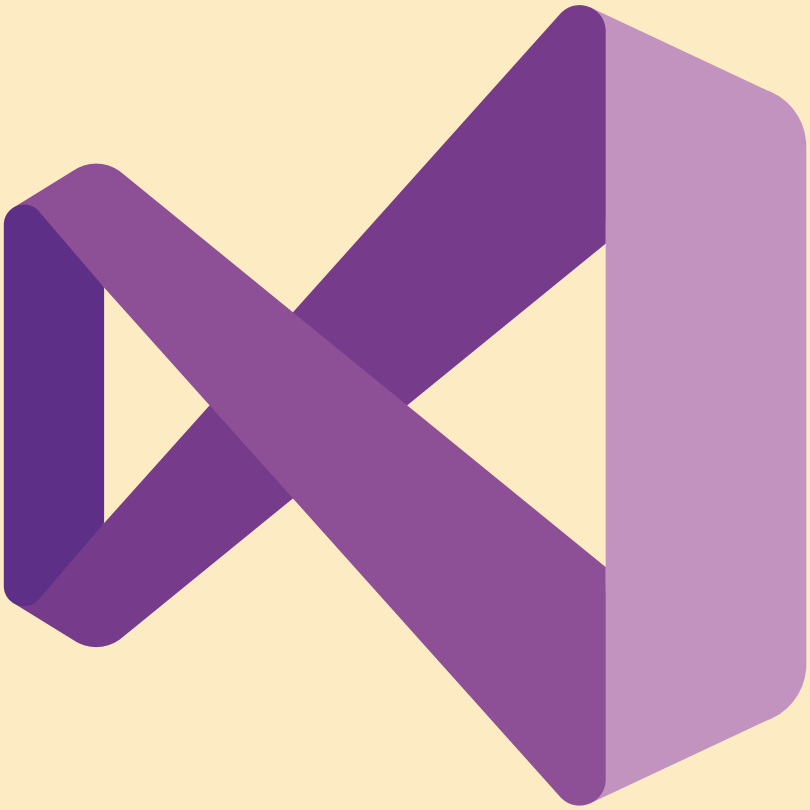
Le fait que .Net 5 ne soit pas LTS montre bien qu'il s'agit d'une version de transition. On bénéficiera d'un LTS une version sur deux. Rappelons que le LTS garantit un support actif sur 3 ans (après la disponibilité de la version finale). Les versions dites courantes (current) sont supportées durant 3 mois ou jusqu'à la prochaine version LTS.

Historique des versions

Année	Version	Principales nouveautés
2002	.Net 1.0	Version initiale
2003	.Net 1.1	Support IPv6, exécution side by side, contrôle mobile ASP.Net
2005	.Net 2.0	Génériques, ClickOnce, ASP.Net 2.2, support 64 bits
2006	.Net 3.0	WPF, WCF, WWF, CardSpace
2007	.Net 3.5	Linq, collections, réseau p2p
2010	.Net 4.0	Parallélisme, nouveautés sur le réseau, WWF, WCF, réseau, déploiement
2012	.Net 4.5	Démarrage plus rapide, tableaux + 2 Go, JIT en arrière-plan, UTF-16, amélioration ZIP, IDNA,
2015-2016	.Net 4.6.x .Net Core 1.x	JIT 64 bits, diverses améliorations, types SIMD, certificat X509 avec algo ECDSA
2017-2018	.Net 4.7.x .Net Core 2.x	Améliorations sur l'ensemble des couches
2019	.Net 4.8 .Net Core 3.x	Nouveautés dans WCF, WPF, CLR
2020	.Net 5.0	Version finale

DevCon#9

Conférence développeur 100 % .Net+Azure+Window



19 NOVEMBRE 2020

A PARTIR DE 13H30

10 sessions & quickies

.Net 5 - C++20 - Grafana+Azure - Projet Orleans
Blazor - .Net Core - etc.

Agenda complet et inscription : **https://www.programmez.com**

Evenement virtuel

Conférence du magazine Programmez!

En partenariat avec SoftFluent, ZDNet et CACD2

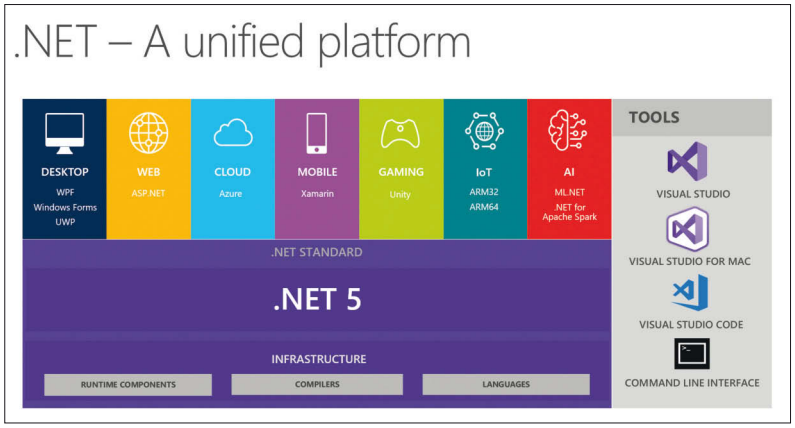

Anthony Giretti
MVP, MCSD
Developer, Blogger, Speaker
anthony.giretti@gmail.com
http://anthonygiretti.com


Miguel Bernard
MVP, Blogger, Speaker
http://blog.miguelbernard.com
m@miguelbernard88


Christophe PICHAUD
Leader de Domaine NET chez Infeeny
christophep@cpixxi.com
www.windowscpp.com

.NET 5 arrive (enfin) !

.NET 5 arrive à grands pas. Depuis son annonce le 6 mai 2019, beaucoup de ses nouveautés ont été présentées et sont déjà quasiment prêtes. A l'heure où nous écrivons ces lignes .NET 5 se trouve en preview 8 et pas mal de fonctionnalités sont déjà plutôt stables, notamment ASP.NET Core 5 et C# 9 que nous avons eu l'occasion de tester.



Depuis la création du projet .NET Core 1 en novembre 2016, Microsoft a ajouté environ cinquante mille API .NET Framework à la plateforme. .NET Core 3 comble une grande partie de l'écart entre .NET Core et .NET Framework 4.8 en termes de capacité. .NET 5 continue à combler ce retard et même le rattraper en s'appuyant sur les efforts réalisés avec .NET Core. Il reprend également le meilleur de Mono pour créer une plateforme unique que vous pourrez utiliser pour tous vos programmes .NET.

.NET 5 une nouvelle étape après .NET Core

.NET 5 est la prochaine grande révolution de .NET après .NET Core. Il sera en effet possible de créer des applications diverses avec le même runtime, donc une uniformité dans les comportements d'exécution de vos applications .NET ainsi qu'une expérience de développement homogène (base de code unique). Cela signifie donc : le code de vos applications et vos fichiers de projets vont très fortement se ressembler; peu importe le type de projet que vous développerez. Pour que tout cela soit possible .NET 5 va reprendre comme mentionné précédemment le meilleur de .NET Core, Mono, .NET Framework, mais aussi Xamarin.

En plus des habitudes appréciées prises avec .NET Core telles que l'implémentation d'application cross-platform, la haute performance, un CLI efficace, l'aspect open source et communautaire sur GitHub et les outils tels que Visual Studio et Visual Studio Code, il sera possible de profiter :

- De l'interopérabilité avec Java sur toutes les plateformes ;
- Idem pour Objective-C et Swift ;
- CoreFX sera amélioré pour prendre en charge la compilation statique de .NET (à l'avance – ahead of time), des empreintes plus petites et la prise en charge d'un plus grand nombre de systèmes d'exploitation.

La sortie de .NET 5 est planifiée pour Novembre 2020. A noter que Microsoft a volontairement nommé la nouvelle version de .NET « .NET.5 » pour éviter toute confusion avec .NET Framework 4.x. En passant, la dernière version de .NET Framework est et sera la version 4.8. Microsoft a également mentionné que .NET 5 ne sera pas supporté à long terme et que seulement les versions numérotées paires seront supportées à long terme. Enfin, Microsoft prévoit de ne pas sortir (ou peu) de versions mineures, l'objectif étant de sortir une fois par an une version majeure de .NET et de s'y tenir. Voici un diagramme des versions actuelles et planifiées de .NET : 1

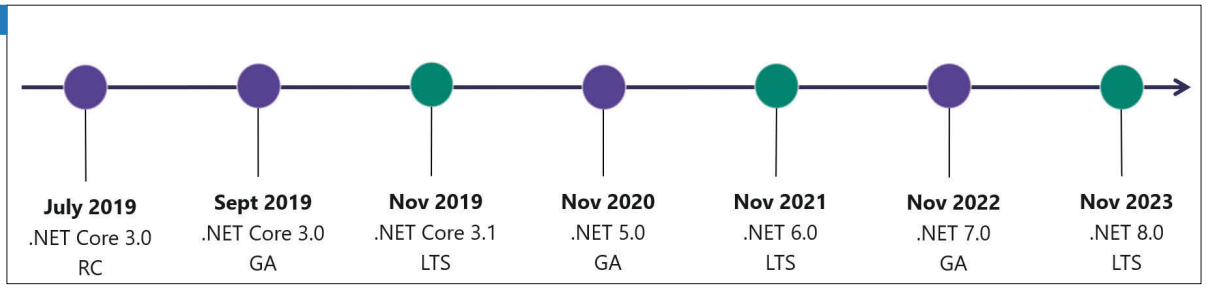
Mono et CoreCLR

Différences et points communs

Mono est l'implémentation interplateforme de .NET. Il a commencé par être une alternative open-source à .NET Framework et a fait la transition vers le ciblage des appareils mobiles tels que les appareils iOS et Android. Mono est le runtime utilisé pour exécuter Xamarin.

CoreCLR est le runtime utilisé dans le cadre de .NET Core. Il a été principalement destiné à la prise en charge des applications cloud, il est utilisé pour les applications de bureau, l'IoT et le machine learning. .NET Core et Mono ont beaucoup de similitudes mais aussi des différences qui les caractérisent. Il est donc logique de permettre au développeur de choisir l'expérience qu'il souhaite en faisant en sorte que le passage de l'un à l'autre se fasse de la manière la plus simple possible.

1



JIT (just in time compiler)

Dès le début, .NET s'est appuyé sur un compilateur juste à temps (JIT) pour traduire le code de langage intermédiaire (IL) en code optimisé. Depuis ce temps, Microsoft a construit un runtime géré basé sur un JIT efficace, performant et permettant une expérience de développement rendant la programmation rapide et facile.
L'expérience par défaut pour la plupart des application .NET 5 utilisera le runtime CoreCLR basé sur JIT. Les deux exceptions notables seront iOS et Blazor (WebAssembly), puisque les deux nécessitent une compilation native à l'avance (AOT).

AOT (Ahead of time)

Le compilateur Mono quant à lui, pour .NET, est un compilateur AOT qui permet de compiler du code en code natif qui peut s'exécuter sur une machine, tout comme le code C++. Le projet Blazor, comme mentionné précédemment utilise déjà Mono AOT et est le premier projet à passer à .NET 5. La compilation AOT restera nécessaire également pour iOS et certaines consoles de jeux car Microsoft fera de la compilation AOT une option pour les applications qui nécessitent démarrage rapide et / ou faible empreinte.

Conclusion

Microsoft investira dans le débit et la fiabilité dans CoreCLR tout en travaillant davantage pour améliorer le démarrage et la réduction de consommation mémoire avec le compilateur Mono AOT.
L'investissement en termes d'effort n'étant pas identique sur ces aspects, cela ne veut pas dire que l'investissement sur d'autres sera différent également. Par exemple les capacités de diagnostic doivent être les mêmes sur .NET 5, tant pour les diagnostics fonctionnels que pour les diagnostics de performance.
Toutes les applications .NET 5 seront également constructibles avec le .NET CLI, en assurant aux développeurs de pouvoir disposer des mêmes outils en ligne de commande. Enfin C# évoluera sur tous les projets .NET 5 en se présentant sous la neuvième version majeure : C# 9.

AMÉLIORATIONS CONCRÈTES DANS .NET 5

A l'heure où nous écrivons ces lignes .NET 5 en est à sa preview 8. Nous sommes tout proche de la version Release candidate puisque la preview 8 contient toutes les nouveautés de .NET 5. Il est possible de télécharger le SDK ici : https://dotnet.microsoft.com/download/dotnet/5.0
Pour migrer les projets existants, il suffit de mettre à jour le target framework comme suit :

```
<TargetFramework>netcoreapp5.0</TargetFramework>
```

Concrètement l'unification de l'expérience de développement avec le .NET SDK va permettre d'utiliser des librairies de classes de base (BCL) dans Xamarin ; actuellement Xamarin utilise la librairie des classes de base de mono (Mono BCL). .NET 5 va également supporter le développement mobile car son SDK va permettre de créer des projets Xamarin Forms, et le CLI pour également en être capable **(dotnet new XamarinForms)**. Les projets d'applications natives seront compatibles sur différents supports tels que Windows Desktop, Microsoft Duo (Android), et iOS. Un projet Blazor pourra également fonctionner sur des mobiles et des applications de bureau et pas seulement dans un navigateur. Les applications Cloud natives seront plus performantes et prendront moins de place, enfin .NET 5 va supporter HTTP3. Voyons désormais un peu plus en détail les améliorations de .NET 5.

Réorganisation des repository Github

Avec ASP.NET, .NET Core, Entity Framework Core, il y avait environ une centaine de repository à gérer, c'était devenu difficile, Microsoft a nettement simplifié ceci en proposant trois repository consolidés :
https://github.com/dotnet/runtime
https://github.com/dotnet/aspnetcore
https://github.com/dotnet/sdk

Amélioration des performances du code généré par le JIT

Le code machine généré par le JIT est beaucoup plus performant, les améliorations sont tellement nombreuses qu'il est impossible de toutes les décrire ici, mais voici le repository Github si vous voulez en savoir plus : https://github.com/dotnet/runtime

Les diagnostics du chargement des assembly avec Event Pipe

Microsoft propose un outil nommé dotnet-trace qui a été introduit avec le SDK .NET 3. Il est toujours disponible avec .NET 5 et ce dernier propose en plus de ce qui existait déjà les diagnostics de chargement des assembly. Le tutoriel complet peut être trouvé ici : https://github.com/richlander/testapps/tree/master/trace-assembly-loading

Améliorations du Garbage Collector

Dans cette mouture de .NET 5, le Garbage collector bénéficie aussi de quelques améliorations et notamment :

- Les pauses du GC éphémères sont plus courtes pour les scénarios où certains threads GC ont pris beaucoup plus de temps à marquer que d'autres ;
- Introduction du Pinned Object Heap, qui permettra au GC de gérer les objets épinglés (Pinned Objects) séparément, et par conséquent d'éviter les effets négatifs des objets épinglés sur les heap générationnels ;
- Autorisation de l'allocation d'objets volumineux à partir de la liste libre pendant le balayage de l'arrière-plan SOH (Start of Heading) ;
- Des corrections pour réduire le temps de suspension des threads Garbage Collector en tâches de fond et utilisateurs. Cela réduit le temps total qu'il faut pour suspendre les threads gérés avant qu'un Garbage Collector puisse se produire.

Améliorations de l'assembly System.Text.Json

Microsoft a introduit dans .NET 3.0 une nouvelle assembly, destinée à remplacer (essentiellement pour des raisons de performance) NewtonSoft.Json. Si l'arrivée de cette nouvelle assembly a largement été bien accueillie, elle a en revanche souffert de quelques manques que la version de .NET 5 vient combler :

- Préservation des références d'objet : par défaut, **Newtonsoft.Json**

sérialise par valeur. Par exemple, si un objet contient deux propriétés qui contiennent une référence à un même objet, les valeurs des propriétés de cet objet sont dupliquées dans le JSON. **Newtonsoft.Json** a un paramètre **PreserveReferencesHandling** sur **JsonSerializerSettings** qui vous permet de sérialiser par référence. **System.Text.Json** quant à lui prend en charge uniquement la sérialisation par valeur et lance une exception pour les références circulaires.

- Création d'un nouveau namespace **System.Net.Http.Json** contenant des méthodes d'extension pour HttpClient (serialisation / désérialisation) :
  - **HttpClientJsonExtensions** : contient les méthodes d'extension pour envoyer/recevoir du contenu HTTP en tant que JSON, exemple :

```
var result = await Http.GetJsonAsync<Profile[]>("api/users/profiles")
```

  - **HttpContentJsonExtensions** : contient les méthodes d'extension pour lire, puis analyser le HttpContent de JSON, exemple :

```
var response = await _httpClient.GetAsync("api/users/profiles");
response.EnsureSuccessStatusCode();
return await response.Content.ReadFromJsonAsync<Profile[]>();
```

  - **JsonContent** : fournit du contenu HTTP basé sur JSON. Exemple :

```
var result = await httpContent.ReadAsByteArrayAsync();
```

Amélioration des performances des expressions régulières

Le namespace **System.Text.RegularExpressions** existe dans le framework .NET depuis des années, depuis .NET Framework 1.1. Il est utilisé par des milliers d'applications. Dans tout cela, il représente une source importante de consommation de temps processeur.

En termes de performance les **Regex** n'ont pas reçu beaucoup d'attention depuis, excepté depuis l'arrivée de .NET Core 2 avec l'introduction d'une option de compilation pour les Regex nommée **RegexOptions.Compiled**. Cette option permet de spécifier que l'expression régulière est compilée dans une assembly et que cela permet une exécution plus rapide mais augmente le temps de démarrage. Quant à .NET 3, il a bénéficié de quelques optimisations grâce à l'introduction de **Span<T>** pour améliorer l'utilisation de la mémoire dans certains scénarios.

Dans .NET 5 le moteur de Regex a bénéficié d'améliorations qui ont, dans la plupart des cas, permis d'améliorer les performances par 3 voir 6.

Voici ci-dessous grâce à l'outil de Visual Studio .NET Object Allocation Tracking, on peut voir l'allocation mémoire des Regex comparé entre .NET Core 3.1 à gauche et .NET 5 à droite. Je vous laisse faire vos propres conclusions : 2

Les applications à fichier unique

Les applications de fichier unique sont publiées et déployées en tant que fichier unique. L'application et ses dépendances sont toutes incluses dans ce fichier. Lorsque l'application est en cours d'exécution, les dépendances sont chargées directement à partir de ce fichier en mémoire. Il n'y a pas de pénalité de rendement avec cette approche. Lorsqu'elles sont combinées avec la coupe d'assemblage et la compilation à l'avance (AOT), les applications de fichiers uniques sont plus petites et rapidement en démarrage.

Dans .NET 5.0, les applications de fichiers uniques sont principalement axées sur Linux.

Pour déployer une application à fichier unique, voici les commandes à exécuter :

- Application dépendante du framework:

```
dotnet publish -r linux-x64 --self-contained false /p:PublishSingleFile=true
```

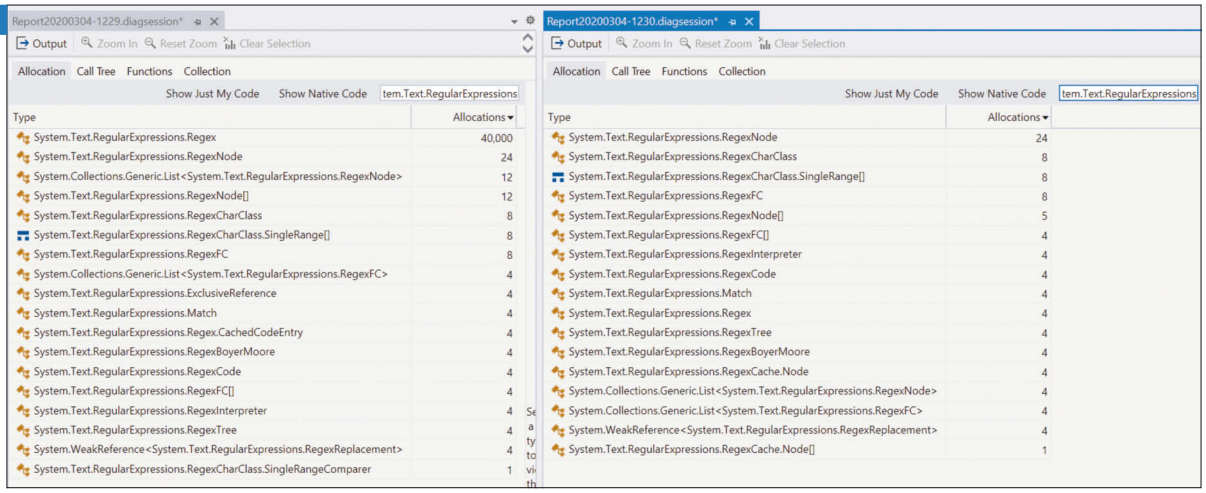
- Application à fichier unique prête à être activée :

```
dotnet publish -r linux-x64 --self-contained true /p:PublishSingleFile=true /p:PublishTrimmed=true /p:PublishReadyToRun=true
```

Il est possible d'effectuer la même configuration d'application à fichier unique à partir d'un fichier de projet :

```
<Project Sdk="Microsoft.NET.Sdk">

 <PropertyGroup>
  <OutputType>Exe</OutputType>
  <TargetFramework>net5.0</TargetFramework>
```

2



```
  <PublishSingleFile>true</PublishSingleFile>
  <SelfContained>true</SelfContained>
  <RuntimeIdentifier>linux-x64</RuntimeIdentifier>
  <PublishTrimmed>true</PublishTrimmed>
  <PublishReadyToRun>true</PublishReadyToRun>
 </PropertyGroup>

</Project>
```

Autres améliorations

- Amélioration des performances HTTP/1.1 ;
- Amélioration des performances HTTP/2 ;
- Amélioration des performances des extensions sur les strings, comme string.ToLowerInvariant etc. ;
- Amélioration des performances pour les processeurs de type ARM64 ;
- Réduction de la taille des images des containers tels que Docker ;
- Les annotations nullables () ;
- Et bien d'autres encore.

C# 9

L'an dernier, nous avons eu droit à une nouvelle version majeure de C# contenant beaucoup de nouvelles fonctionnalités et d'améliorations du langage. À peine quelques mois plus tard, nous avons déjà droit à C# 9.0 qui risque d'être encore plus prometteur. Microsoft met les bouchées doubles pour faire de C# un des langages les plus agréables à utiliser, en enlevant tout de qui est source d'irritation, et en simplifiant la syntaxe afin que les développeurs puissent gagner en vitesse.
Dans cet article, nous découvrirons quelques-unes des nouvelles fonctionnalités.

Attention! Les fonctionnalités et exemples décrits dans cet article sont sujets à changement tant que la prochaine version de C# n'est pas livrée officiellement. Il est aussi possible que certaines fonctionnalités soient remises à plus tard.

Les propriétés immuables

Les propriétés immuables sont déclarées à l'aide du nouveau mot-clé **init**. Ce mot-clé remplace **set** et permet d'indiquer que l'on souhaite pouvoir assigner une valeur à la propriété, mais seulement lors de l'initialisation de l'objet et ne plus le permettre par la suite. Ainsi, la propriété devient immuable.

```
public class Order
{
  public int Number { get; init; }
}
```

Cette nouvelle fonctionnalité est très pratique lorsqu'on veut utiliser les initialiseurs d'objet sans pour autant permettre la mutation de la propriété une fois l'objet construit.

```
var o = new Order
{
  Number = 1,
};

o.Number = 2; // Erreur de compilation
```

Constructeurs primaires

Actuellement, une des choses les plus irritantes avec les constructeurs est le fait qu'ils sont souvent très redondants et ne contiennent pas d'informations très utiles, et ce, surtout pour les classes simples. En effet, la plupart du temps vous aurez un paramètre qui aura le même nom qu'une propriété et le constructeur fera simplement l'assignation de cette valeur du paramètre vers la propriété. Beaucoup de lignes de code qui ont peu de valeur, mais qui sont obligatoires afin d'obtenir une bonne encapsulation.
Les constructeurs primaires tentent d'adresser et de simplifier cette problématique. L'idée est de laisser le compilateur émettre tout ce code par défaut. Pour ce faire, vous devrez ajouter des parenthèses à la déclaration de votre classe en indiquant les paramètres voulus. Derrière, le compilateur se chargera d'ajouter une propriété à votre objet et d'en faire l'assignation lors de la construction. Ainsi, vous pourrez grandement réduire le nombre de lignes requises pour déclarer un type, parfois même jusqu'à une seule ligne!
Vous trouverez un exemple regroupant les records et un constructeur primaire dans le prochain exemple.

Les records

Avec l'arrivée des records, C# a maintenant une façon intégrée dans le langage pour créer des types immuables. Ne vous méprenez pas, il est déjà possible aujourd'hui d'atteindre le même résultat, cependant ceci vous nécessitera beaucoup de travail.
Afin de garder une syntaxe simple et légère, Microsoft a dû introduire un nouveau mot-clé, **record**. Celui-ci doit être utilisé en remplacement du mot-clé **class** ou **struct** lorsque vous définissez un type. C'est en combinant avec les constructeurs primaires que vous irez chercher le maximum de valeur avec cette fonctionnalité.

```
public class Person
{
  public Person(string firstName, string lastName)
  {
    this.FirstName = firstName;
    this.LastName = lastName;
  }

  public string FirstName { get; }
  public string LastName { get; }
}

// Équivalent avec un enregistrement et un constructeur primaire
public record Person(string firstName, string lastName);
```

Vous avec bien vu, une seule ligne pourra remplacer une déclaration qui autrefois en prenait plus de onze. D'ailleurs, plus la classe est complexe et longue, plus le bénéfice en nombre de lignes sauvées sera accentué.
L'immutabilité offre bien des avantages, mais peut être inconvénient dans certains cas. Par exemple, si vous avez un objet plutôt

complexe et que vous voulez modifier une seule propriété, eh bien ce n'est pas permis, car l'objet est immuable. Vous devez construire un nouvel objet en spécifiant toutes les propriétés, dont celle qui aura une valeur différente. Vous comprendrez que si votre objet est très gros ou complexe, ceci devient vite encombrant.
Heureusement, Microsoft a pensé à vous en vous offrant le nouveau mot-clé **with** pour gérer ce cas. Ce mot-clé permet avec une syntaxe toute simple de cloner un objet tout en remplaçant les valeurs des propriétés spécifiées.

```
var p = new Person("Miguel", "Bernard");
var p2 = p with { lastName = "Lapierre" };
```

p2 est maintenant un nouvel objet distinct, qui possède une référence mémoire différente de **p1**. Ce nouvel objet contient « Miguel » comme prénom et « Lapierre » comme nom de famille. Les records ont aussi un autre comportement qui diffère grandement de ce auquel vous êtes habitué. Lors de comparaison entre deux objets, les enregistrements utilisent l'égalité structurelle au lieu de l'égalité référentielle par défaut. Qu'est-ce que cela veut dire ? Eh bien, au lieu de simplement comparer le pointeur en mémoire, les enregistrements vont comparer toutes les valeurs de toutes leurs propriétés afin de déterminer si une égalité existe. Par exemple, ce comportement peut être très utile lorsque vous avez à manipuler des objets de transfert de données (DTO/POCO). Petit piège auquel il faut quand même faire attention. Actuellement, ce comportement s'applique seulement à la méthode **Equals** et pas à l'opérateur **==**.

```
var p = new Person("Miguel", "Bernard");
var p2 = new Person("Miguel", "Bernard");

p.Equals(p2); // True
p == p2;    // False
```

Les programmes de niveau supérieur

Plusieurs langages offrent une forme très simple pour exécuter un programme, souvent en une seule ligne pour un programme de type « Hello world ». L'équivalent en C# n'en prendra qu'une dizaine de lignes.

```
// Avant
namespace csharp
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Hello World!");
    }
  }
}
```

Avec C# 9.0, le compilateur est capable d'émettre automatique la grande majorité de ces lignes, il n'est donc plus requis de les spécifier. Tous les cas spéciaux sont supportés, que ce soient les appels asynchrones, les arguments d'entrées ou les valeurs de retour. Voici un programme qui combine un peu de tout :
Voir le fichier program.cs

```
var s = new AsyncService();
var result = await s.MyAsyncMethod();
Console.WriteLine($"Hello {args[0]}! result: {result}");
return 1;
```

Il est bon de prendre note ici que cet exemple est le contenu complet du fichier **Program.cs**. Plus besoin de déclarer un **namespace** ni de méthode **Main**. Pour ce qui est des arguments, on peut y accéder en appelant la variable **args** qui existe presque par magie dans le contexte de ce fichier. Ce contexte d'exécution est d'ailleurs asynchrone, ce qui veut dire que vous pouvez faire des appels asynchrones avec **await** sans aucun problème.

Amélioration des expressions logiques

C# 9.0 supporte désormais les expressions logiques directement dans les filtres par patron. Cette fonctionnalité apporte aussi une foule de nouveaux mots-clés tels que : **and**, **or** et **not**.

```
string ÉtatDeSanté(decimal poid) => poid switch
{
  >= 18.5m and < 25m => "normal",
  < 15m or > 30m => "en santé",
  _ => "inconnu"
};
```

Dans certains cas, ces mots-clés rendent la lecture beaucoup plus facile que leur équivalent en symbole : **&&**, **||** et **!**.

```
if (s is not string)
// est plus facile à lire que
if (!(s is string))
```

Détection implicite de type

Le compilateur étant maintenant plus intelligent, il est désormais capable de déterminer le type implicitement à partir du contexte d'exécution. Ainsi, lors de la création d'un objet avec **new** vous n'aurez plus toujours besoin de répéter le type.

```
Cat c = new Cat("Chester"); // Avant
Cat c = new("Chester"); // C# 9.0
```

Validation des valeurs nulles simplifiée

Vous pouvez maintenant utiliser une syntaxe simplifiée pour valider les valeurs nulles directement dans la signature d'une méthode à l'aide de l'opérateur **!** Vous devez placer celui-ci en suffixe au paramètre qui doit être non nul et le tour est joué.

```
public void Before(string name)
    {
      if (name is null)
        throw new ArgumentNullException();
    }

    public void Now(string name!) // C# 9.0
    {
    }
```

Le reste… on verra bien

Il y a encore quelques autres fonctionnalités qui sont proposées, mais comme elles sont encore en ébauche, elles risquent fortement

d'être reportées à une version ultérieure. Comme .NET est maintenant open source, si vous êtes curieux vous pouvez suivre le progrès et toutes les discussions autour de C# 9.0 sur GitHub.
https://github.com/dotnet/roslyn/blob/master/docs/Language%20Feature%20Status.md

Pour conclure

Un des plus grands reproches que la communauté exprime face à C# est le fait qu'il est très verbeux. Comme vous pouvez le constater, cette version met un accent important sur la simplification du langage afin d'aborder ce problème. Avec tous ces nouveaux outils, les développeurs C# n'ont plus rien à envier aux autres langages.

Exemples supplémentaires

Vous pouvez retrouver sur le dépôt GitHub ci-dessous le code source des exemples cités, ainsi que des cas d'utilisation plus avancés.
https://github.com/mbernard/codeSamples/tree/master/csharp9

Windows Form, WPF, WinUI et Xamarin avec .NET Core

Windows Forms pour .NET Core

.NET Core propose des outils graphiques dans les éditions preview de Visual Studio.
Windows Forms ou WinForms est une librairie graphique évènementielle qui encapsule le GDI Windows. C'est robuste, rapide et fiable. Cela existe depuis la première version de .NET en 2002. Pourquoi WinForms pour .NET Core ? .NET Core est open-source et multiplateformes (Windows, Linux, Mac). Le support de WinForms pour passer de .NET Framework à .NET Core nécessite des ajustements. WinForms ne fonctionne que sur Windows.

WinForms Designer pour Visual Studio en Preview

Le Designer est en preview dans Visual Studio depuis le mois de mai 2020 dans la Preview v 16.6. Le Designer a complètement été réécrit. Ce module semble être complexe car il met du temps à passer en RTM.

WPF et .NET Core

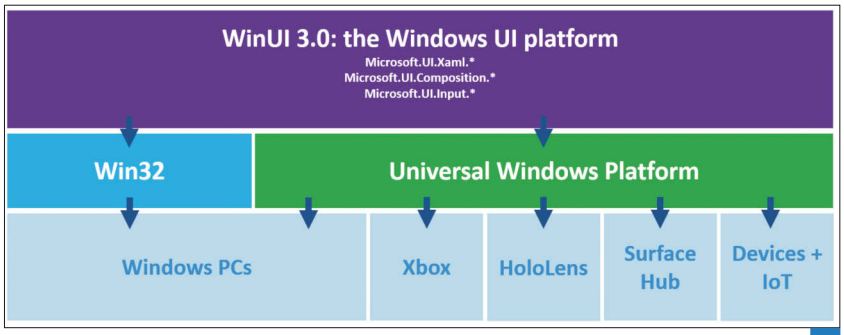
WPF fournit aussi un Designer réécrit from scratch. Comme pour WinForms, ce module est complexe et prend du temps. A l'heure de l'écriture de cet article, il n'est toujours pas disponible. WPF n'est disponible que pour Windows Desktop. Pas pour Linux ni Mac.

Que choisir ?

WPF et ses composants XAML ont de nouveau le vent en poupe car les composants XAML de Windows 10 sont maintenant en open-source (WinUI XAML sur GitHub). WinUI c'est l'avenir car WinUI permet l'inclusion de contrôles XAML en WinForms, WPF et Win32. Microsoft ne fait pas ses produits avec WPF. Le modèle MVVM est difficile à debugger. Préférez WinForms ! Plus rapide, plus léger, plus facile à comprendre. De plus, c'est du Windows à l'état brut.

WinUI 3

WinUI 3 élargira considérablement la portée de WinUI pour inclure la plateforme d'interface utilisateur native complète de Windows



3

10, qui sera désormais entièrement découplée du SDK UWP.
Nous nous concentrons sur l'activation de trois cas d'utilisation principaux :

- Moderniser les applications Win32 existantes : cela vous permet d'étendre les applications Win32 (WPF, WinForms, MFC ..) existantes avec l'interface utilisateur moderne de Windows 10 à votre propre rythme en utilisant la dernière version à venir de Xaml Islands ;
- Créer de nouvelles applications Windows : cela vous permet de créer facilement de nouvelles applications Windows modernes "à la carte" avec votre choix de modèle d'application (Win32 ou UWP) et de langage (.NET ou C ++) ;
- Activation d'autres frameworks : cela fournit l'implémentation native d'autres frameworks tels que React Native lors de l'exécution sous Windows

L'équipe a récemment publié WinUI 3 Preview 1, qui est une préversion précoce qui inclut la prise en charge des applications Win32 et UWP. L'aperçu 1 est disponible pour tout le monde, mais notez qu'il présente des limitations et des problèmes connus, il n'est donc pas équipé pour les applications de production. Commencez ou apprenez-en plus sur l'aperçu 1 ici. La roadmap WinUI ressemble à ceci : 3
Les API UWP Xaml existantes qui sont livrées dans le cadre du système d'exploitation ne recevront plus de nouvelles mises à jour de fonctionnalités. Elles recevront toujours des mises à jour de sécurité et des correctifs critiques selon le cycle de vie du support Windows 10.
La plateforme Windows universelle contient plus que le cadre Xaml

(par exemple, modèle d'application et de sécurité, pipeline multimédia, intégrations de shell Xbox et Windows 10, prise en charge étendue des appareils) et continuera d'évoluer. Toutes les nouvelles fonctionnalités Xaml seront simplement développées et livrées dans le cadre de WinUI à la place.

Avantages de WinUI 3

WinUI 3 offrira un certain nombre d'avantages par rapport au framework UWP Xaml actuel, WPF, WinForms et MFC, ce qui fera de WinUI le meilleur moyen de créer des interfaces utilisateur pour les applications Windows :

1° La plate-forme d'interface utilisateur native de Windows

WinUI est la plateforme d'interface utilisateur native hautement optimisée utilisée pour créer Windows lui-même. Elle est désormais plus largement disponible en utilisation pour tous les développeurs qui veulent atteindre des applications pleinement Windows. Il s'agit d'une plateforme d'interface utilisateur entièrement testée et éprouvée qui alimente l'environnement du système d'exploitation et les expériences essentielles de plus de 800 millions de PC Windows 10, XBox One, HoloLens, Surface Hub et d'autres appareils.

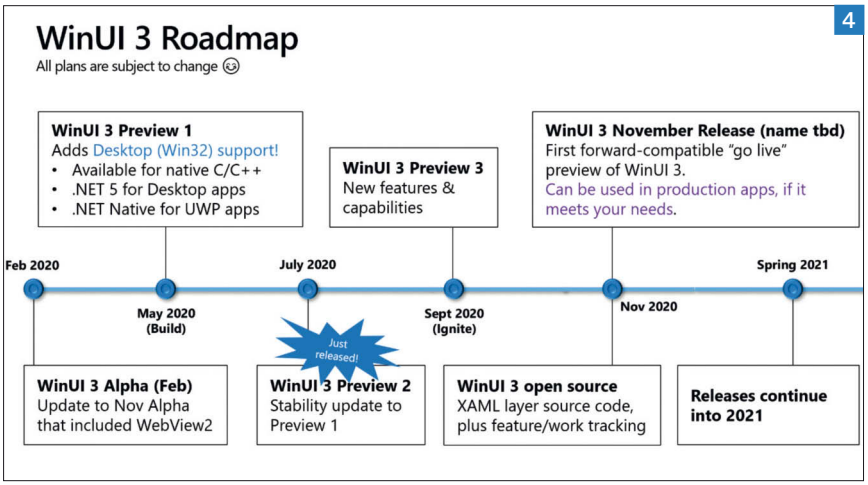
2° Le dernier design Fluent

WinUI est le principal objectif de Microsoft pour l'interface utilisateur et les contrôles Windows natifs et accessibles. Il est la source définitive du système de conception Fluent sur Windows.
Il prendra également en charge les dernières innovations de composition et de rendu de niveau inférieur telles que les animations vectorielles, les effets, les ombres et l'éclairage.

3° Développement de bureau "à la carte" plus facile

WinUI 3 vous permettra de mélanger et assortir plus facilement la bonne combinaison de :

- Langage: C # (.NET), C++ ;
- Modèle d'application: UWP, Win32 ;
- Emballage: MSIX, AppX pour le Microsoft Store, non emballé ;
- Interopérabilité: utilisez WinUI 3 pour étendre les applications WPF, WinForms et MFC existantes avec l'interface utilisateur Fluent moderne.



4

4° Rétrocompatibilité pour les nouvelles fonctionnalités

Les nouvelles fonctionnalités de WinUI continueront d'être rétrocompatibles avec un large éventail de versions de Windows 10 : vous pouvez commencer à créer et à publier des applications avec de nouvelles fonctionnalités dès leur sortie, sans avoir à attendre que vos utilisateurs mettent à jour Windows.

5° Prise en charge du développement natif

WinUI peut être utilisé avec .NET, mais ne dépend pas de .NET : WinUI est 100% C++ et peut être utilisé dans des applications Windows non gérées, par exemple en utilisant C++ 17 standard via C++ / WinRT.

6° Mises à jour plus fréquentes

WinUI continuera à livrer de nouvelles versions stables 3x par an, avec des versions mensuelles en avant-première.

7° Développement open source et engagement communautaire

WinUI continuera à être développé en tant que projet open source sur GitHub. WinUI 2 est déjà open source dans ce dépôt, et il est prévu d'ajouter le framework WinUI 3 Xaml complet dans le dépôt dans les mois à venir.
Vous pouvez vous engager directement avec l'équipe d'ingénierie principale de Microsoft et contribuer aux rapports de bogue, aux idées de fonctionnalités et même au code : consultez le Guide de contribution pour plus d'informations.
Vous pouvez également essayer les versions mensuelles des versions préliminaires pour voir les nouvelles fonctionnalités en développement et aider à façonner leur forme finale.

8° Une cible Windows native pour les frameworks web et multiplateformes

WinUI 3 est mieux optimisé pour les bibliothèques et les frameworks sur lesquels s'appuyer.
Par exemple, la nouvelle implémentation Windows native C++ React haute performance travaille actuellement sur l'utilisation de WinUI 3 comme base. 4
Pour le moment, nous sommes en WinUI 3 Preview 2.

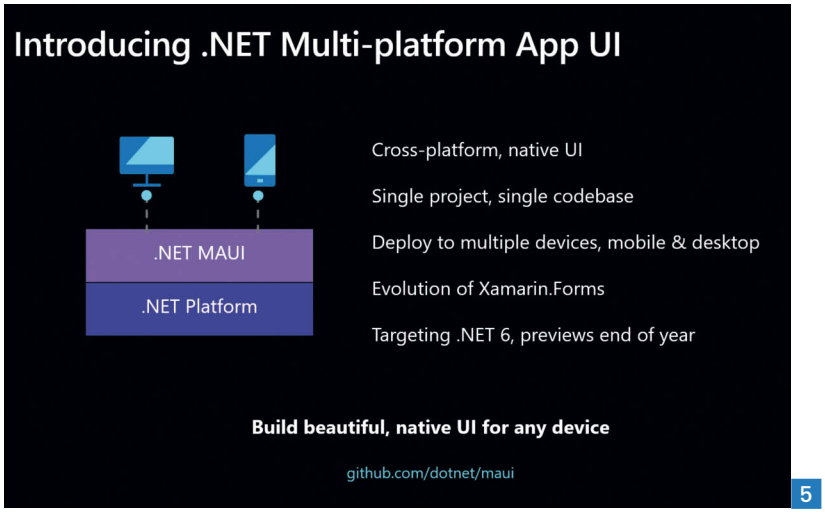
WinUI c'est du C++

Comme indiqué dans le point n°5 précédent : « WinUI peut être utilisé avec .NET, mais ne dépend pas de .NET : WinUI est 100% C++ et peut être utilisé dans des Application Windows » , n'importe quelle application Windows.
L'envers du décor, c'est que les équipes Windows ne veulent pas de .NET dans Windows. Vous vous rappelez le traumatisme de Windows Vista ? Relisez ceci : https://arstechnica.com/information-technology/2011/06/windows-8-for-software-developers-the-longhorn-dream-reborn/ , ils expliquent le trauma de Windows Longhorn et Microsoft est passé à deux doigts de la catastrophe industrielle. Les équipes Windows ont une liste exhaustive de problèmes et de questions sur .NET auxquels les équipes DevDiv (qui font .NET) n'arrivent pas à répondre ni résoudre. Donc, les équipes Windows ignorent .NET. Le XAML de Windows 10 (depuis Windows 8) est écrit en C++. Vous pourriez me dire : et pourquoi pas WPF ? Parce que c'est du .NET…
Les équipes Windows fournissent Windows XML qui se nomme

maintenant WinUI : la bataille est gagnée (une fois de plus) pour C++. Le système d'exploitation Windows fournit une extension de COM qui se nomme Windows Runtime (WinRT) et c'est une combinaison de nouvelles fonctionnalités MIDL et le support du C++ Moderne dans Windows. Le code Windows ne change pas d'une année sur l'autre et il est fait pour rester très longtemps. Windows est (very) LTS (Long Time Support). Il est important de savoir qu'il existe des divergences en interne chez Microsoft sur l'adoption de .NET et Windows. Cela permet de comprendre beaucoup de choses. WinUI est le meilleur exemple ! Avec .NET Core, on a le support WinForms et WPF mais le vrai hot stuff NEW, c'est WinUI et Microsoft ne fait pas la promotion de WPF. Vous avez des applications WPF ? C'est certainement couplé avec du WCF, donc non supporté par .NET Core… Laissez-les en .NET Framework, c'est du legacy.

Le mobile avec Xamarin et le projet MAUI

Les produits Xamarin pour iOS et Xamarin pour Android existent toujours. La nouveauté concerne Xamarin.Forms qui passe en version 4.8 depuis Août 2020. La communauté Xamarin est en ébullition car le projet MAUI (Multi Application User Interface) arrive. C'est l'unification de Xamarin.Forms avec .NET Core sous .NET 5 ou .NET 6. Il n'y a pas beaucoup d'informations existantes sur MAUI. Le projet n'est pas en preview publique pour le moment mais nous avons un slide : 5



ASP.NET CORE 5

Et voilà la nouvelle version tant attendue d'ASP.NET Core !

Tout comme comme .NET 5, la version est encore en preview au moment où nous écrivons ces lignes, et comme pour .NET 5 nous en sommes à la preview 8 qui contient toute les caractéristiques de la sortie finale en novembre.

La plupart des évolutions concernent Blazor, mais pas que, heureusement ! Mais tout d'abord regardons comment migrer d'ASP.NET Core 3.1 à ASP.NET Core 5.0 ainsi que les améliorations les plus marquantes.

Pour pouvoir migrer, il va bien entendu falloir télécharger le dernier SDK de .NET 5 ici : https://dotnet.microsoft.com/download/dotnet/5.0 mais aussi télécharger Visual Studio 2019, il faut savoir que la version minimale requise est la version 16.8 que vous pouvez télécharger ici : https://visualstudio.microsoft.com/fr/downloads/?utm_medium=microsoft&utm_source=docs.microsoft.com&utm_campaign=inline+link&utm_content=download+vs2019

D'ASP.NET Core 3.1 à ASP.NET Core 5.0

Migrer d'une version à l'autre est assez simple, cependant il y a quelques breaking changes que nous verrons un peu plus bas.

Premièrement il faut mettre à jour le target framework, comme suit :

```
<Project Sdk="Microsoft.NET.Sdk.Web">

 <PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
 </PropertyGroup>

</Project>
```

Pour ce qui concerne Blazor c'est un peu différent. Deux nouvelles propriétés s'ajoutent :

```
<Project Sdk="Microsoft.NET.Sdk.Web">

 <PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
  <RuntimeIdentifier>browser-wasm</RuntimeIdentifier>
  <UseBlazorWebAssembly>true</UseBlazorWebAssembly>
 </PropertyGroup>

</Project>
```

Concernant Blazor encore, la référence au package Microsoft.AspNetCore.Components.WebAssembly.Build n'est tout simplement plus nécessaire.

Enfin pour tout ce qui est relatif à ASP.NET Core en général, les packages suivants doivent être mis à jour :

- Microsoft.AspNetCore.*
- Microsoft.Extensions.*
- System.Net.Http.Json

Exemple :

```
<ItemGroup>
  <PackageReference Include="Microsoft.AspNetCore.JsonPatch" Version="5.0.0-preview.8.*" />
  <PackageReference Include="Microsoft.Extensions.Caching.Abstractions" Version="5.0.0-preview.8.*" />
  <PackageReference Include="System.Net.Http.Json" Version="5.0.0-preview.8.*" />
</ItemGroup>
```

Pour finir, les images docker subissent elles aussi un tout petit changement :

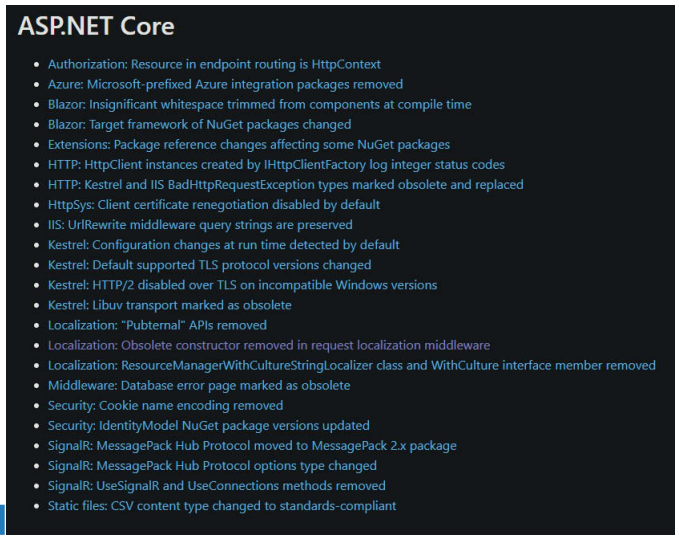
- Avant :

```
docker pull mcr.microsoft.com/dotnet/core/aspnet:3.1
```

- Après :

```
docker pull mcr.microsoft.com/dotnet/aspnet:5.0
```

Concernant maintenant les breaking changes, il ne sont pas si nombreux que cela, Blazor, Kestrel, SignalR et les assembly de localisation subissent la plupart des modifications de rupture. Voici l'Url décrivant les modifications à effectuer :



6

https://docs.microsoft.com/en-us/dotnet/core/compatibility/3.1-5.0.
Ci-dessous le résumé des breaking changes : 6

Les extensions JSON pour HttpRequest et HttpResponse

Vous pouvez maintenant lire et écrire facilement les données JSON à partir d'un **HttpRequest** et **HttpResponse** à l'aide des nouvelles méthodes d'extension **ReadFromJsonAsync** et **WriteAsJsonAsync**. Ces méthodes d'extension utilisent l'assembly **System.Text.Json. Exemple** d'un endpoint lightweight :

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/countries", async context =>
    {
        var countryService = context.Request.HttpContext.RequestServices.GetRequiredService<
CountryService>();

        var countries = await countryService.Get();

        await context.Response.WriteAsJsonAsync(countries);
    }).RequireAuthorization();
});
```

Méthode d'extension autorisant l'accès anonyme

Sur l'exemple précédent, nous avions ajouté une méthode d'extension permettant de forcer une authentification. Avec ASP.NET Core 5, il est désormais possible, et ce manière explicite, d'autoriser l'accès à un endpoint de façon anonyme. Si nous reprenons l'exemple précédent, cela donne ceci :

```
app.UseEndpoints(endpoints =>
{
    endpoints.MapGet("/countries", async context =>
    {
        var countryService = context.Request.HttpContext.RequestServices.GetRequiredService<
CountryService>();

        var countries = await countryService.Get();

        await context.Response.WriteAsJsonAsync(countries);
    }).AllowAnonymous();
});
```

La gestion customisée des échecs d'autorisations

La gestion personnalisée des échecs d'autorisation est désormais plus facile avec la nouvelle interface **IAuthorizationMiddleware ResultHandler** qui est invoquée par **AuthorizationMiddleware**. L'implémentation par défaut reste la même, mais un gestionnaire personnalisé peut être enregistré dans le système d'injection de dépendance qui permet des choses comme les réponses HTTP personnalisées en fonction des raisons de l'échec de l'autorisation. Un exemple qui démontre l'utilisation de **l'IAuthorizationMiddleware ResultHandler** :

```
using System;
namespace CustomAuthorizationFailureResponse.Authorization
{
  public class SampleAuthorizationMiddlewareResultHandler : IAuthorizationMiddlewareResultHandler
  {
    private readonly IAuthorizationMiddlewareResultHandler _handler;

    public SampleAuthorizationMiddlewareResultHandler(IAuthorizationMiddlewareResultHandler
 handler)
    {
      _handler = handler ?? throw new ArgumentNullException(nameof(handler));
    }

    public async Task HandleAsync(
      RequestDelegate requestDelegate,
      HttpContext httpContext,
      AuthorizationPolicy authorizationPolicy,
      PolicyAuthorizationResult policyAuthorizationResult)
    {
      // if the authorization was forbidden, let's use custom logic to handle that.
      if (policyAuthorizationResult.Forbidden && policyAuthorizationResult.AuthorizationFailure !
= null)
      {
        // as an example, let's return 404 if specific requirement has failed
        if (policyAuthorizationResult.AuthorizationFailure.FailedRequirements.Any(requirement
=> requirement is SampleRequirement))
        {
          httpContext.Response.StatusCode = (int)HttpStatusCode.NotFound;
          await httpContext.Response.WriteAsync(Startup.CustomForbiddenMessage);

          // return right away as the default implementation would overwrite the status code
          return;
        }
        else if (policyAuthorizationResult.AuthorizationFailure.FailedRequirements.Any(requirement
=> requirement is SampleWithCustomMessageRequirement))
        {
          // if other requirements failed, let's just use a custom message
          // but we have to use OnStarting callback because the default handlers will want to
modify i.e. status code of the response
```

```
            // and modifications of the response are not allowed once the writing has started
            var message = Startup.CustomForbiddenMessage;

            httpContext.Response.OnStarting(() => httpContext.Response.BodyWriter.WriteAsync
(Encoding.UTF8.GetBytes(message)).AsTask());
          }
        }

        await _handler.HandleAsync(requestDelegate, httpContext, authorizationPolicy, policy
AuthorizationResult);
      }
    }
}
```

Les filtres Hub de SignalR

Les filtres Hub, appelés pipelines Hub dans ASP.NET SignalR, est une fonctionnalité qui vous permet d'exécuter du code avant et après l'appel des méthodes Hub, similaire à la façon dont vous pouvez exécuter du code avant et après une demande http (ou l'entrée dans un contrôleur comme les filtres d'action dans MVC ou une WebAPI). Les utilisations courantes incluent la journalisation, la gestion des erreurs et la validation des arguments. Exemple :

```
public class CustomFilter : IHubFilter
{
  public async ValueTask<object> InvokeMethodAsync(
    HubInvocationContext invocationContext, Func<HubInvocationContext, ValueTask<object>> next)
  {
    Console.WriteLine($"Calling hub method '{invocationContext.HubMethodName}'");
    try
    {
      return await next(invocationContext);
    }
    catch (Exception ex)
    {
      Console.WriteLine($"Exception calling '{invocationContext.HubMethodName}'");
      throw ex;
    }
  }
}
```

La configuration est ensuite simple. Exemple :

```
public void ConfigureServices(IServiceCollection services)
{
  services.AddSignalR(options =>
  {
    options.AddFilter<CustomFilter>();
  });

  services.AddSingleton<CustomFilter>();
}
```

L'envoi de frames PING HTTP/2

HTTP/2 dispose d'un mécanisme permettant d'envoyer des frames PING afin de s'assurer qu'une connexion inactive est toujours fonctionnelle. Ceci est particulièrement utile lorsque vous travaillez avec des flux à longue durée de vie qui sont souvent inactifs (par exemple, les streams gRPC). Dans ASP.NET Core 5, Microsoft ajouté la possibilité d'envoyer des frames PING périodiques dans **Kestrel** en fixant des limites sur **KestrelServerOptions**. Exemple :

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
      Host.CreateDefaultBuilder(args)
.ConfigureWebHostDefaults(webBuilder =>
{
  webBuilder.ConfigureKestrel(options =>
  {
    options.Limits.Http2.KeepAlivePingInterval = TimeSpan.FromSeconds(10);
    options.Limits.Http2.KeepAlivePingTimeout = TimeSpan.FromSeconds(1);
  });
  webBuilder.UseStartup<Startup>();
});
```

Le décodage de header personnalisé

Microsoft a ajouté la possibilité de spécifier le codage **System.Text.Encoding** à utiliser pour interpréter les headers entrants en fonction du nom du header. Ceci au lieu d'UTF-8 par défaut. Nous pouvons désormais définir la propriété **RequestHeaderEncodingSelector** sur **KestrelServerOptions** pour spécifier l'encodage à utiliser.

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
Host.CreateDefaultBuilder(args)
.ConfigureWebHostDefaults(webBuilder =>
{
  webBuilder.ConfigureKestrel(options =>
  {
    options.RequestHeaderEncodingSelector = encoding =>
    {
      switch (encoding)
      {
        case "Host":
          return System.Text.Encoding.Latin1;
        default:
          return System.Text.Encoding.UTF8;
      }
    };
  });
  webBuilder.UseStartup<Startup>();
});
```

La console de logging JSON

Microsoft a apporté des améliorations au provider de logging **ILogger (Microsoft.Extensions.Logging)**. Les développeurs peuvent désormais implémenter un **ConsoleFormatter** pour exercer un contrôle complet sur la mise en forme et la colorisation de la sortie de la console.

Parallèlement à cela, Microsoft a également ajouté un formateur JSON intégré qui émet des logs JSON structurés à la console. Pour remplacer le logger par défaut par le logger JSON, voici comment procéder :

```
public static IHostBuilder CreateHostBuilder(string[] args) =>
Host.CreateDefaultBuilder(args)
.ConfigureLogging(logging =>
```

```
{
  logging.AddJsonConsole(options =>
  {
    options.JsonWriterOptions = new JsonWriterOptions() { Indented = true };
  });
})
.ConfigureWebHostDefaults(webBuilder =>
{
  webBuilder.UseStartup<Startup>();
});
```

Exemple d'un log structuré produit :

```
{
 "EventId": 0,
 "LogLevel": "Information",
 "Category": "Microsoft.Hosting.Lifetime",
 "Message": "Now listening on: https://localhost:5001",
 "State": {
  "Message": "Now listening on: https://localhost:5001",
  "address": "https://localhost:5001",
  "{OriginalFormat}": "Now listening on: {address}"
 }
}
```

Le rafraîchissement automatique avec dotnet watch

Dans .NET 5, l'exécution de **dotnet watch** sur un projet ASP.NET Core lancera désormais le navigateur par défaut et actualisera automatiquement le navigateur au fur et à mesure que le développeur ajoutera des modifications à son code. Cela signifie que vous pourrez ouvrir un projet ASP.NET Core dans votre éditeur de texte préféré, exécuter ensuite une fois **dotnet watch**, puis l'outillage associé à ce dernier gèrera la reconstruction, le redémarrage et le rechargement de votre application. Microsoft apportera la fonctionnalité de rafraîchissement automatique à Visual Studio à l'avenir, car cette fonction est absente de Visual Studio pour le moment.

L'isolation CSS des composants Blazor

Blazor prend maintenant en charge la définition des styles CSS qui sont étendus à un composant donné. Les styles CSS spécifiques aux composants facilitent l'intégration des styles de votre application et évitent les effets secondaires involontaires des styles globaux. Les développeurs pourront utiliser des styles spécifiques aux composants dans un fichier .razor.css qui correspond au nom du fichier .razor pour le composant. Par exemple, prenons un composant nommé **MyComponent.razor** qui ressemble à ceci :

```
<h1>My Component</h1>

<ul class="cool-list">
  <li>Item1</li>
  <li>Item2</li>
</ul>
```

Il est possible de définir un fichier MyComponent.razor.css avec les styles de MyComponent :

```
h1 {
  font-family: 'Comic Sans MS'
}

.cool-list li {
  color: red;
}
```

Le lazy loading dans Blazor WebAssembly

Le lazy loading vous permet d'améliorer le temps de chargement de votre application Blazor WebAssembly en reportant le téléchargement de dépendances d'applications spécifiques jusqu'à ce qu'elles soient requises. Le lazy loading peut être utile si votre application Blazor WebAssembly présente de grandes dépendances qui ne sont utilisées que pour des parties spécifiques de l'application. Pour retarder le chargement d'une assembly, il suffit d'ajouter **BlazorWebAssemblyLazyLoad** au groupe d'éléments dans votre fichier de projet : les assemblys marqués pour le lazy loading doivent être explicitement chargés par l'application avant d'être utilisés. Pour charger les assembly en période d'exécution, il faut utiliser le service **LazyAssemblyLoader** comme suit :

```
@inject LazyAssemblyLoader LazyAssemblyLoader

@code {
  var assemblies = await LazyAssemblyLoader.LoadAssembliesAsync(new string[] { "Lib1.dll" });
}
```

L'exemple ci-dessous illustre le lazy loading d'une assembly lors de la navigation vers la page 1 :

```
@using System.Reflection
@using Microsoft.AspNetCore.Components.Routing
@using Microsoft.AspNetCore.Components.WebAssembly.Services
@inject LazyAssemblyLoader LazyAssemblyLoader

<Router AppAssembly="typeof(Program).Assembly" AdditionalAssemblies="lazyLoadedAssemblies"
 OnNavigateAsync="@OnNavigateAsync">
  <Navigating>
    <div>
      <p>Loading the requested page...</p>
    </div>
  </Navigating>
  <Found Context="routeData">
    <RouteView RouteData="@routeData" DefaultLayout="typeof(MainLayout)" />
  </Found>
  <NotFound>
    <LayoutView Layout="typeof(MainLayout)">
      <p>Sorry, there is nothing at this address.</p>
    </LayoutView>
  </NotFound>
</Router>

@code {
  private List<Assembly> lazyLoadedAssemblies =
    new List<Assembly>();
```

```
private async Task OnNavigateAsync(NavigationContext args)
{
  if (args.Path.EndsWith("/page1"))
  {
    var assemblies = await LazyAssemblyLoader.LoadAssembliesAsync(new string[] { "Lib1.dll" });
    lazyLoadedAssemblies.AddRange(assemblies);
  }
}
```

L'ajout du composant InputRadio dans Blazor

Blazor dans .NET 5 inclut désormais les composants **InputRadio** et **InputRadioGroup** intégrés. Ces composants simplifient la liaison de données aux groupes de boutons radio avec validation intégrée aux côtés des autres composants d'entrée de formulaire Blazor.

```
Opinion about blazor:
<InputRadioGroup @bind-Value="survey.OpinionAboutBlazor">
  @foreach (var opinion in opinions)
  {
    <div class="form-check">
      <InputRadio class="form-check-input" id="@opinion.id" Value="@opinion.id" />
      <label class="form-check-label" for="@opinion.id">@opinion.label</label>
    </div>
  }
</InputRadioGroup>
```

L'ajout de l'autofocus sur les inputs dans Blazor

Blazor dispose maintenant d'une méthode de commodité **FocusAsync** sur **ElementReference** pour permettre à l'utilisateur courant de se focaliser automatiquement sur l'input HTML désiré.

```
<button @onclick="() => textInput.FocusAsync()">Set focus</button>
<input @ref="textInput" />
```

Influencer les balises HTML head dans Blazor

L'ajout des nouveaux composants **Title**, **Link** et **Meta** permettent définir par programmation le titre d'une page et d'ajouter dynamiquement des balises de lien et de méta HTML d'une application Blazor. Pour utiliser les nouveaux composants **Title**, **Link** et **Meta** :

- Ajouter une référence de package au package Microsoft.AspNetCore.Components.Web.Extensions ;
- Inclure une référence de script à _content/Microsoft.AspNetCore.Components.Web.Extensions/headManager.js ;
- Ajouter une directive @using pour **Microsoft.AspNetCore.Components.Web.Extensions.Head**.

Exemple :

```
@if (unreadNotificationsCount > 0)
{
  var title = $"Notifications ({unreadNotificationsCount})";
  <Title Value="title"></Title>
  <Link rel="icon" href="icon-unread.ico" />
}
```

Protection du sessionStorage et localStorage dans Blazor

Dans les applications Blazor Server, il est possible de persister l'état de l'application dans le stockage local ou de session afin que l'application puisse la réhydrater ultérieurement si nécessaire. Lorsque le développeur stocke l'état de l'application dans le navigateur de l'utilisateur, il faut également s'assurer qu'elle n'a pas été altérée. Blazor dans .NET 5 aide à résoudre ce problème en fournissant deux nouveaux services : **ProtectedLocalStorage** et **ProtectedSessionStorage**. Ces services aident à stocker l'état dans le stockage local et de session respectivement. Ils prennent aussi soin de protéger les données stockées à l'aide des API de protection des données ASP.NET Core. Pour utiliser les nouveaux services de stockage de navigateur protégés il faut :

- Ajouter une référence de package à **Microsoft.AspNetCore.Components.Web.Extensions** ;-
- Configurer les services en appelant **services.AddProtectedBrowserStorage()** (configuré dans le **Startup.cs**) ;
- Injecter **ProtectedLocalStorage** et **ProtectedSessionStorage** dans de le / les composants désirés :

```
@inject ProtectedLocalStorage ProtectedLocalStorage
@inject ProtectedSessionStorage ProtectedSessionStorage
```

Enfin utiliser le service souhaité pour obtenir, définir et supprimer l'état de façon asynchrone :

```
private async Task IncrementCount()
{
  await ProtectedLocalStorage.SetAsync("count", ++currentCount);
}
```

ENTITY FRAMEWORK CORE 5

Entity Framework Core 5 est actuellement en développement au moment où nous écrivons ces lignes. Par conséquent cet article contient seulement un aperçu des modifications les plus pertinentes de cette mouture numéro 5 d'Entity Framework 5.

Le mapping table par type (TPT mapping)

Par défaut, Entity Framework Core 5 mappe une hiérarchie d'héritage des types .NET à une seule table de base de données. C'est ce qu'on appelle le mapping de table par hiérarchie (TPH). Entity Framework Core 5 permet également de mapper chaque type .NET dans une hiérarchie d'héritage à une table de base de données différente, appelée mapping de table par type (TPT). Exemple :

```
public class Animal
{
  public int Id { get; set; }
  public string Species { get; set; }
}

public class Pet : Animal
{
  public string Name { get; set; }
}
```

```
public class Cat : Pet
{
  public string EducationLevel { get; set; }
}

public class Dog : Pet
{
  public string FavoriteToy { get; set; }
}
```

Par défaut la table suivante est générée si le modèle précédent est utilisé pour générer la base de données :

```
CREATE TABLE [Animals] (
  [Id] INT NOT NULL IDENTITY,
  [Species] NVARCHAR(MAX) NULL,
  [Discriminator] NVARCHAR(MAX) NOT NULL,
  [Name] NVARCHAR(MAX) NULL,
  [EducationLevel] NVARCHAR(MAX) NULL,
  [FavoriteToy] NVARCHAR(MAX) NULL,
  CONSTRAINT [PK_Animals] PRIMARY KEY ([Id]));
```

A la place, Entity Framework Core 5 va générer les 4 tables suivantes :

```
CREATE TABLE [Animals] (
  [Id] int NOT NULL IDENTITY,
  [Species] NVARCHAR(MAX) NULL,
  CONSTRAINT [PK_Animals] PRIMARY KEY ([Id])
);

CREATE TABLE [Pets] (
  [Id] int NOT NULL,
  [Name] NVARCHAR(MAX) NULL,
  CONSTRAINT [PK_Pets] PRIMARY KEY ([Id]),
  CONSTRAINT [FK_Pets_Animals_Id] FOREIGN KEY ([Id]) REFERENCES [Animals] ([Id]) ON DELETE
 NO ACTION
);

CREATE TABLE [Cats] (
  [Id] int NOT NULL,
  [EducationLevel] NVARCHAR(MAX) NULL,
  CONSTRAINT [PK_Cats] PRIMARY KEY ([Id]),
  CONSTRAINT [FK_Cats_Animals_Id] FOREIGN KEY ([Id]) REFERENCES [Animals] ([Id]) ON DELETE
 NO ACTION,
  CONSTRAINT [FK_Cats_Pets_Id] FOREIGN KEY ([Id]) REFERENCES [Pets] ([Id]) ON DELETE NO ACTION
);

CREATE TABLE [Dogs] (
  [Id] int NOT NULL,
  [FavoriteToy] NVARCHAR(MAX) NULL,
  CONSTRAINT [PK_Dogs] PRIMARY KEY ([Id]),
  CONSTRAINT [FK_Dogs_Animals_Id] FOREIGN KEY ([Id]) REFERENCES [Animals] ([Id]) ON DELETE
 NO ACTION,
  CONSTRAINT [FK_Dogs_Pets_Id] FOREIGN KEY ([Id]) REFERENCES [Pets] ([Id]) ON DELETE NO
 ACTION
);
```

Pour que cela se produise, il faut utiliser annotations habituelles et créer une classe par table comme suit :

```
[Table("Animals")]
public class Animal
{
  public int Id { get; set; }
  public string Species { get; set; }
}

[Table("Pets")]
public class Pet : Animal
{
  public string Name { get; set; }
}

[Table("Cats")]
public class Cat : Pet
{
  public string EducationLevel { get; set; }
}

[Table("Dogs")]
public class Dog : Pet
{
  public string FavoriteToy { get; set; }
}
```

Ou utiliser le **ModelBuilder** comme suit :

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
  modelBuilder.Entity<Animal>().ToTable("Animals");
  modelBuilder.Entity<Pet>().ToTable("Pets");
  modelBuilder.Entity<Cat>().ToTable("Cats");
  modelBuilder.Entity<Dog>().ToTable("Dogs");
}
```

Reconstruction des tables SQLite

Par rapport à d'autres bases de données, SQLite est relativement limitée dans ses capacités de manipulation de schéma. Par exemple, la suppression d'une colonne d'une table existante nécessite que la table entière soit supprimée et recréée. Entity Framework Core 5 Migrations prend désormais en charge la reconstruction automatique de la table pour les modifications de schéma qui l'exigent. Voici les étapes réalisées par Entity Framework Core 5 :

- Une table temporaire est créée avec le schéma souhaité pour la nouvelle table ;
- Les données sont copiées à partir de la table actuelle dans la table temporaire ;
- L'application des clés étrangères est désactivée ;
- La table actuelle est supprimée ;
- La table temporaire est renommée pour être la nouvelle table.

Les fonctions table évaluées (Table-Valued functions)

Entity Framework Core 5 inclut la prise en charge des fonctions table évaluées. Ces fonctions peuvent ensuite être utilisées dans les

requêtes LINQ où la composition supplémentaire sur les résultats de la fonction sera également traduite en SQL. Exemple :

```
CREATE FUNCTION GetReports(@employeeId INT)
RETURNS @reports TABLE
(
    Name NVARCHAR (50) NOT NULL,
    IsDeveloper BIT NOT NULL
)
AS
BEGIN
    WITH cteEmployees AS
    (
        SELECT Id, Name, ManagerId, IsDeveloper
        FROM Employees
        WHERE Id = @employeeId
        UNION ALL
        SELECT e.Id, e.Name, e.ManagerId, e.IsDeveloper
        FROM Employees e
        INNER JOIN cteEmployees cteEmp ON cteEmp.Id = e.ManagerId
    )
    INSERT INTO @reports
    SELECT Name, IsDeveloper
    FROM cteEmployees
    WHERE Id != @employeeId

    RETURN
END
```

Ensuite, il faut créer les modèles associés pour pouvoir utiliser cette fonction table évaluée et les enregistrer dans le **ModelBuilder** :

```
public class Employee
{
  public int Id { get; set; }
  public string Name { get; set; }
  public bool IsDeveloper { get; set; }

  public int? ManagerId { get; set; }
  public virtual Employee Manager { get; set; }
}

public class Report
{
  public string Name { get; set; }
  public bool IsDeveloper { get; set; }
}
```

```
modelBuilder.Entity<Employee>();
modelBuilder.Entity(typeof(Report)).HasNoKey();
```

Enfin en déclarant la fonction comme suit dans le **ModelBuilder** :

```
    public IQueryable<Report> GetReports(int managerId)
    => FromExpression(() => GetReports(managerId));
```

```
modelBuilder.HasDbFunction(() => GetReports(default));
```

On peut ensuite l'utiliser dans une Requête Linq, Exemple :

```
var query = from employee in context.Employees
from report in context.GetReports(e.Id)
where report.IsDeveloper == true
select new
{
  ManagerName = employee.Name,
  EmployeeName = report.Name,
});
```

Le DbContextFactory

Vous souvenez vous des services Singleton qui vous empêchaient d'utiliser des Repository dont la durée de vie était prévue pour ne durer que le temps d'une requête Http (**Scoped** dans ASP.NET Core) ? Eh bien il fallait créer une factory (ou utiliser **DbContextFactory qui existait déja**)pour contextualiser votre repository (et donc sa connexion à la base de données). Eh bien Entity Framework Core 5 introduit le **AddDbContextFactory** et **AddPooledDbContextFactory** qui va permettre de manière native d'enregistrer une DbContextFactory dans le système d'injection de dépendance, exemple :

```
services.AddDbContextFactory<MyDbContext>(b =>
b.UseSqlServer(@"Server=(localdb)\mssqllocaldb;Database=MyDatabase"));
```

Exemple à l'utilisation :

```
public class MyRepository
{
  private readonly IDbContextFactory<MyDbContext> _contextFactory;

  public MyRepository(IDbContextFactory<MyDbContext> contextFactory)
  {
    _contextFactory = contextFactory;
  }
}
```

Conclusion

.NET 5 est une nouvelle direction importante et passionnante pour .NET..NET va devenir plus simple, mais va également proposer une capacité plus large et utilitaire. Toutes les nouvelles fonctionnalités de développement feront partie de .NET 5, y compris les nouvelles versions C# et F#. Microsoft va nous proposer un avenir prometteur dans lequel les développeurs pourront utiliser les mêmes API .NET et langages pour cibler un large éventail de types d'applications, de systèmes d'exploitation et d'architectures de puces. Il sera facile d'apporter des modifications aux configurations de build pour créer des applications différemment, dans Visual Studio, Visual Studio pour Mac, Visual Studio Code, Azure DevOps ou à la ligne de commande.

Cet article vous a proposé de découvrir dans les grandes lignes les nouvelles fonctionnalités de .NET 5 mais .NET 5 est bien plus large que cela, ASP.NET Core 5 ou Entity Framework Core 5 , pour ne citer qu'eux, proposent des tonnes de nouveautés que nous n'avons pas pu décrire ici. Soyez cependant assurés que le passage de .NET Core 3.1 à 5 sera le plus simple possible, Microsoft l'a promis !

Jérémy Jeanson
Développeur
MVP

Initier un projet d'application Windows avec.net 5 sans se louper

Ces dernières années, Microsoft a multiplié les solutions pour créer des applications pour Windows : .Net MAUI, WinUi 3, Projet Reunion, .Net core 3.1, UWP, WinRT, XAML Island, etc. Mais aujourd'hui avec l'arrivée de .Net 5, où en est-on ? Quelles solutions avons-nous à disposition et comment se lancer dans de bonnes conditions ? Un développeur .Net peut choisir entre trois solutions stables : WinForm, WPF et UWP.

Si vous êtes déjà habitué à .Net Framework et .net core, vous vous demandez peut-être où se cachent les nouveautés pour le desktop ? Ne cherchez pas du côté des modèles de projets, il n'y a rien de nouveau de ce côté. Tout se cache sous le capot. **.Net 5 est une version axée sur l'unification des API, la stabilité, l'amélioration des performances et de l'accessibilité.**

Faire un choix parmi les technologies actuelles

Au moment de faire son choix, il faut se faire une raison. Toutes les stacks techniques annoncées par Microsoft exploitent XAML ou une solution à base de Markup. Même si WinForm a de nombreux avantages, il ne trouve pas sa place dans ce futur. On réservera donc WinForm à la migration d'application .net vers .net 5. Les projets ainsi migrés pourront utiliser progressivement WPF.

Concernant UWP, le discours de Microsoft n'est pas forcément très limpide. Son terrain s'est réduit. Sa mort a été annoncée. Cette incertitude a eu un impact négatif sur l'engouement des développeurs. Pour compliquer la chose, on parle beaucoup de WinRT (Windows Runtime) en parallèle d'UWP ou à la place de UWP. À la sortie de Windows 8, la dénomination WinRT a été mise en opposition à WinJS. D'où aujourd'hui une plus grande confusion. Pour être clair : non, Microsoft ne ressuscite pas Metro. Microsoft désigne par WinRT, l'ensemble des API modernes utilisables par une application desktop.

La technologie a évolué. Les nouveaux contrôles ont été extraits d'UWP pour devenir WinUI et permettre aux développeurs d'utiliser plus rapidement de nouvelles fonctionnalités avec Windows 10 sans attendre après les SDK ou Windows Update. Ceci a facilité l'adoption du Fluent Design. Aujourd'hui, **il faut entrevoir UWP comme un paradigme pour la création d'application moderne**. Ce choix reste donc viable.

La version de WPF fournie avec .net 5 inclut l'ensemble des possibilités initialement offertes par le .net Framework. Les développeurs déjà coutumiers de WPF ne seront donc pas perdus. En comparaison avec .net Framework, .net 5 utilise davantage d'API fournies par l'OS. L'expérience utilisateur n'en est que meilleure (boîtes de dialogue, écran HDPI…). S'ajoute à cela un gain de performance non négligeable. Pour autant, on reste sur un socle Win32 et non WinRT. Pour utiliser WinRT, il faudra utiliser le package NuGet C#/WinRT.

Anticiper WinUI 3 et Project Reunion

UWP permet déjà d'utiliser WinUI 2. Si vous commencez dès aujourd'hui avec WinUI et UWP, il n'y a donc rien à faire pour anticiper WinUI 3.

Côté WPF, le développeur qui souhaite adapter une interface moderne avec WPF devra passer par XAML Island. Ceci lui permettra d'embarquer les éléments graphiques d'UWP dans son application (dont WinUI 2). WPF est donc une option intéressante si l'on souhaite adopter le Fluent Design et se préparer à WinUI 3 tout en restant sur Win32. Cela fait de WPF la solution la plus malléable et la plus progressive.

Il reste à traiter la grande inconnue : le Project Reunion. Heureusement, depuis son annonce Microsoft a clarifié la chose. Reunion n'est pas un nouveau modèle d'application. Il faut l'entrevoir comme un ensemble de composants qui faciliteront l'accès aux API modernes. Une application UWP ou WPF pourra en profiter. Pour WPF, l'adoption de C#/WinRT est un premier pas vers Reunion. Aujourd'hui, UWP ne profite pas de nouveauté liée à Reunion.

La problématique du déploiement

Si votre choix entre WPF et UWP n'est pas encore défini, celui-ci se fera certainement au moment d'établir les contraintes de déploiement :

- Une application WPF peut être déployée de toutes les manières possibles et imaginables (xCopy, MSI, MSIX, Microsoft Store, …) ;
- Une application UWP ne peut être déployée que via MSIX ou le Microsoft Store.
- Pour une application Desktop, WPF est évidemment le choix le plus souple.

Comment organiser sa logique applicative ?

Le choix d'utiliser XAML pousse à s'intéresser au pattern MVVM. Vous pouvez bien évidemment coder vos propres outils pour exploiter MVVM, .net 5 n'impose rien. Le plus simple étant d'utiliser une solution du marché telle que MVVM Light ou Prism. Je ne cite que ces deux exemples, car ils reflètent bien les tendances, complications et avantages liés à MVVM.

D'un côté, MVVM Light est à entrevoir comme une boîte à outils. On peut y piocher ce que l'on souhaite. Notre développement n'est pas conditionné par cette librairie.

En ce qui concerne Prism, la situation est totalement différente. Il s'agit d'un véritable framework qui va cadrer notre manière de coder, structurer, et naviguer dans l'application.
Attention donc à bien choisir vos librairies en fonction de votre situation :

- Si vous n'avez pas encore fait de choix techniques ou devez partir de zéro, Prism est une bonne solution ;
- Si vous avez un projet existant, ou avez déjà cadré votre architecture, MVVM Light vous facilitera la tâche.

Concernant MVVM, je ne vais pas faire une énième redite de ses principes, mais en extraire quelques règles qui vous éviteront les écueils les plus courants :

- View First : progressivement, toutes les technologies qui exploitent XAML ont évolué vers un modèle de navigation qui impose l'instanciation de la View (Page, Window). Ne cherchez donc pas à instancier vos ViewModel avant les Views. De plus, c'est à la View de référencer le ViewModel (ceci vous évitera les références circulaires et les problèmes de libération de la mémoire associés).
- Navigation : créez une classe qui se chargera de la centralisation de toutes les méthodes de navigation (ouverture de Window ou navigation vers une Page). Ceci facilitera la migration si un jour il vous faut quitter WPF ou UWP pour un autre type de projet. Cette approche facilite aussi l'utilisation du pattern d'IOC.
- View Code : contrairement à ce que l'on peut souvent entendre, il n'est pas interdit d'avoir du code derrière une View. Il n'est pas rare que cela soit l'approche la plus performante et facile pour manipuler l'interface graphique.
- Attached properties : codez des Attached Properties dès que vous constatez qu'un comportement doit être reproduit sur plusieurs Views. Ceci est préférable à l'héritage de Views.
- Views : ne jamais instancier un élément graphique via le ViewModel. Laisser cette tâche à votre View principale. Si l'instanciation dépend de l'état du ViewModel, il suffit d'utiliser un Binding (sur la ou les propriétés à observer).
- Communication entre ViewModels : si vos ViewModels doivent communiquer entre eux (échange de données, évènements), il ne faut pas céder à la tentation d'utiliser des Events (pour éviter le couplage trop fort entre les classes et les références circulaires). Il faut impérativement passer par le pattern Publisher – Subscriber (Pub-Sub). Celui-ci vous permet de faciliter les tests, la gestion de votre mémoire et l'évolutivité de l'application. Prism et MVVM Light ont des classes pour cela (respectivement : EventAggregator et Messenger)
- Utilisation du Pub-Sub : quand un ViewModel n'est plus utile, pensez à le désabonner de votre EventAggregator ou Messenger (afin de vous assurer de ne pas bloquer inutilement des ressources en mémoire).
- Faciliter l'usage du designer XAML : souvent, le test d'une interface XAML passe par d'interminables phases de compilation et exécution. Pour éviter cela, il suffit de créer un VieModel dédié au design. Celui-ci doit respecter la même interface que le ViewModel réel. Il est associé à la View via l'attribut "d:DataContext". À noter qu'il est aussi possible de simulateur l'état des contrôles en mode design en utilisant "d:" devant l'attribut que l'on veut remplacer en mode design.
- VieModel fonctionnel : afin de respecter au mieux les exigences fonctionnelles de vos utilisateurs, donnez des noms à vos ViewModel qui respectent le vocabulaire des utilisateurs. Considérez le ViewModel comme étant le garant de la logique fonctionnelle. Cela vous évitera d'éparpiller les règles fonctionnelles entre des domaines qui n'ont rien à voir. Un ViewModel = un domaine fonctionnel.

Comment garantir la stabilité et l'évolutivité ?

La stabilité d'une application dans le temps n'est possible que si l'on évite les deux problèmes les plus courants :

- Le code spaghetti : tout est tellement lié que l'on ne peut plus prendre de recul. Si les classes ne sont pas directement liées, il y a souvent plusieurs classes ou méthodes statiques qui sont appelées d'un peu partout. Une modification d'une portion de code peut avoir un impact non prévisible sur une autre. Ce qui peut aboutir à des effets de bords et des régressions.
- L'application non testable : il n'y a pas de tests unitaires codés, ou l'application n'est testable que manuellement et dans un environnement bien défini. Reproduire un cas qui produit une erreur est difficile.

Pour éviter cela, les solutions sont simples :

- Utiliser l'Inversion Of Control (IOC) ;
- Coder systématiquement des tests unitaires (en profitant du découpage offert par l'IOC).

L'IOC va avoir un impact très positif sur votre application :

- Diminution du couplage entre les différents modules fonctionnels d'une application. Ceci facilite les évolutions futures (changement d'API système par exemple).
- Empêche la création de classes à tout faire.
- Simplification du travail d'équipe (une personne peut travailler sur une classe qui dépend d'une autre qui est codée par un collègue avant que celui-ci n'ait fini son implémentation).
- Pour faciliter les tests (utilisation d'un Framework de Mocks pour ne pas avoir besoin de coder plusieurs implémentations uniquement pour les tests). Les Mocks vont aussi permettre de simuler des comportements qui n'ont lieu que dans des situations de problèmes système (perte d'accès réseau, serveur non accessible, refus de droits).
- Documentation et définition des attentes (un développeur qui lit les tests unitaires aura une compréhension plus rapide du rôle de chaque classe).
- Une grande modularité qui permet d'accélérer les migrations. Vous pouvez identifier rapidement les points de blocage d'une migration et adopter rapidement les nouveautés en ne touchant qu'aux classes qui contiennent l'implémentation concrète du code. Les classes qui en dépendent ne sont pas impactées directement (ce qui réduit les effets de bords et régressions).

Un projet .net 5 est en mesure d'utiliser nombre de solutions d'IOC. Si vous avez opté pour l'utilisation de Prism pour votre MVVM, un container est déjà disponible pour gérer l'IOC.
Dans le cas contraire, la plus simple consiste à utiliser "Microsoft.Extensions.Hosting". Les développeurs utilisant déjà ASP .net core ne seront pas dépaysés. Il s'agit de la librairie qui est utilisée pour la construction du WebHost.
De la même manière que pour un projet web, vous aurez à produire

une classe qui fournira un Host. Pour un projet à ASP.net core, la détermination des Controller à instancier est établie par des Routes. Le WebHost sait quel Controller il doit instancier. Vous n'avez donc jamais besoin de manipuler votre Host.
Pour une application desktop (UWP ou WPF), vous aurez besoin de passer par votre Host pour demander l'instanciation de vos ViewModels. Cet Host doit donc être unique et accessible à tous. L'implémentation d'un Singleton s'impose d'elle-même.
Avant de coder votre Host, il faudra installer le package nuget "Microsoft.Extensions.Hosting". Ce package n'est utile que pour le projet qui doit accueillir votre Host. Si vous avez prévu de découper votre projet entre plusieurs assemblies, il faudra ajouter le package Nuget "Microsoft.Extensions.Hosting.Abstractions" à chaque projet qui doit accéder à votre Host ou qui doit lui fournir des services (il n'est pas rare qu'un assembly externe fasse office de plug-in qu'il faudra injecter dynamiquement). Ce package contient les interfaces et classes génériques utiles à l'injection et la configuration des services.
Attaquons maintenant le code d'une classe Host type :

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Threading.Tasks;

namespace MyDemo
{
 public sealed class Host
 {
  #region Singleton

  /// <summary>
  /// Static constructor
  /// </summary>
  static Host()
  {
   Current = new Host();
  }

  /// <summary>
  /// current instance
  /// </summary>
  public static Host Current { get; }

  #endregion

  #region Declarations

  private readonly IHost _host;

  #endregion

  #region Constructor / Services

  /// <summary>
  /// Constructor
  /// </summary>
  private Host()
  {
   _host = new HostBuilder()
    .ConfigureServices(ConfigureServices)
    .Build();
  }

  /// <summary>
  /// Configure services
  /// </summary>
  /// <param name="context"></param>
  /// <param name="services"></param>
  private void ConfigureServices(HostBuilderContext context, IServiceCollection services)
  {
   // Add ViewsModels, Views, services
   services.AddTransient<IMainViewModel, MainViewModel>();
   services.AddTransient<IMainView, MainView>();
   services.AddTransient<ISettingsViewModel, SettingsViewModel>();
   services.AddTransient<ISettingsView, SettingsView>();
   services.AddTransient<INavigationService, NavigationService>();
  }

  #endregion

  #region Methodes

  /// <summary>
  /// Start the host
  /// </summary>
  /// <returns></returns>
  public async Task StartAsync()
  {
   await _host.StartAsync();
  }

  /// <summary>
  /// Stop the host
  /// </summary>
  /// <returns></returns>
  public async Task StopAsync()
  {
   await _host.StopAsync();
   _host.Dispose();
  }

  /// <summary>
  /// Get an instance of service of type T
  /// </summary>
  /// <typeparam name="T"></typeparam>
  /// <returns></returns>
  public T GetService<T>()
  {
   return _host.Services.GetService<T>();
  }

  #endregion
```

```
}
}
```

On a les bases d'un Singleton. Le constructeur fait un appel à sa méthode interne ConfigureServices pour enregistrer les classes qui seront à instancier. Les méthodes StarAsync et StopAsync sont à appeler respectivement lors du lancement et de la fermeture de l'application. Pour UWP, StarAsync sera à appeler via la classe App sur sa méthode OnLaunched.
Exemple :

Code complet sur programmez.com & github

Le StopAsync ne peut pas être utilisé avec UWP.
Pour une application WPF, il faudra modifier le fichier App.xaml et supprimer l'attribut MainWindow. On profite ensuite des méthodes Application_Startup et Application_Exit pour définir la fenêtre principale de l'application et manipuler la classe Host.
Exemple :

Code complet sur programmez.com & github

Par la suite, vos Views n'auront qu'à appeler la méthode GetService pour obtenir une instance de la classe désirée.
Exemple, le MainView peut demander son ViewModel comme ceci :

```
var vm = Host.Current.GetService<IMainViewModel>();
```

La classe Host présentée ici n'est fournie qu'à titre d'exemple, mais elle prend en compte les principaux usages. Elle doit être adaptée dans le cas où votre application doit prendre en compte des plug-ins. Ceux-ci devront être chargés au moment de l'appel à la méthode ConfigureServices.
Concernant l'usage de ce Singleton pour obtenir des instances de View, ViewModel, Service. La mauvaise pratique consiste à appeler la méthode GetService d'un peu partout. La meilleure solution pour l'éviter consiste à coder une classe dédiée à la navigation (comme déjà évoqué avec MVVM). Cette classe devra être la seule à faire appel à Host. Ceci reproduit le comportement que l'on peut déjà avoir avec MVC (chaque Controller définit les Services utiles via son constructeur et n'a pas conscience d'utiliser l'IOC). La classe de navigation sera alors responsable de demander la création des Views. L'ensemble des classes utiles aux Views étant fourni via IOC, les Views n'auront pas besoin d'appeler Host. La classe de navigation pourra aussi être introduite dans les View et ViewModel via leurs constructeurs et l'IOC. La boucle vertueuse est bouclée. L'IOC est présente, mais très discrète.

Comment garantir la qualité de son UI ?

Construire une interface avec XAML n'a rien de véritablement sorcier. La modularité et la simplicité de XAML peuvent cependant aboutir à de mauvaises pratiques qui peuvent rapidement mettre en péril votre application.
Le site Docs de Microsoft fournit déjà nombre de conseils (https://docs.microsoft.com/en-us/windows/apps/desktop/). La plupart sont axés performances et gestion des ressources. Voici une liste complémentaire qui est davantage orientée sur la maintenabilité du code :

- Éviter de créer des contrôles personnalisés. Il est préférable d'utiliser les contrôles de base en leur associant des Templates personnalisés. Via les Attached Properties, on peut très bien ajouter un comportement nouveau à un contrôle existant (de plus, cela est testable). En plus de garantir de meilleures performances, cela permet d'inclure plus rapidement de nouveaux développeurs en cours de projet. Ils n'ont pas à apprendre la palette de contrôles que vous avez créée avant de se lancer. Leur intégration est bien plus rapide. S'ils oublient un élément, il suffit de l'ajouter. Leur code n'est pas à supprimer ou à réécrire entièrement.
- Éviter de dupliquer les Styles similaires. Un Style peut être basé sur un Style existant et ne comprendre que les propriétés qui changent. De plus un Style peut affecter un Template à un Control et lui changer quelques propriétés. Un Style peut donc aussi vous éviter de dupliquer les Templates. Ceci facilitera les futures évolutions et réduit le temps passé à chercher les codes XAML concernés. De plus cela pallie l'incapacité de Visual Studio à identifier les codes XAML dupliqués.
- Ne pas mettre tous les Templates et Styles dans un seul dictionnaire de ressources. Le fait de charger très tôt des Templates qui ne sont pas utiles au lancement de l'application va ralentir celui-ci. Ces éléments doivent être au plus près des contrôles qui les utilisent. S'ils ne sont utiles que sur une seule View, ils doivent être dans le nœud Resources de la Window ou Page. S'ils sont utiles à plusieurs Views, ils doivent être dans un dictionnaire qui n'est référencé que par les Views qui en ont l'usage. Outre le gain de performance non négligeable, cela permet au développeur d'avoir moins de code à parcourir avant d'identifier une portion de XAML problématique ou devant évoluer. Cette pratique rend aussi votre contrôle de sources plus pratique. Que vous utilisiez Git, TFVC ou autre, vous serez en mesure d'identifier rapidement les changements survenus dans le temps sur une interface. Chose quasiment impossible si toutes les modifications de toutes les interfaces ont lieu sur un seul et unique fichier pour toute l'application.
- Vérifier régulièrement la complexité des Views. Il n'est pas rare que l'on imbrique des contrôles de même type les uns dans des autres, sans pour autant que ce soit utile. Je pense entre autres aux Grid qui finissent souvent dans des Grids qui elles-mêmes sont dans d'autres Grids. L'imbrication de trop de contrôles à un impact très négatif sur les performances et la compréhension de l'interface par les développeurs. Vous-même, vous ne pouvez pas être certain que vous comprendrez ce même code un an plus tard. Il ne faut pas hésiter à inclure des commentaires dans le XAML plutôt que de regrouper les contrôles.
- Dans les listes, tableaux et grilles : résistez à l'envie de redéfinir les conteneurs qui servent à la réplication des éléments (que ce soit pour la disposition des Items, la création de marges ou bordures). De base ceux-ci sont optimisés pour garantir de bonnes performances, et une cohérence entre l'expérience de l'utilisateur qui utilise la souris et le clavier ou l'écran tactile. Ce type de modification implique une bonne synchronisation avec son équipe. Si l'on est seul, une bonne organisation est nécessaire afin de ne pas oublier de l'appliquer partout.

Un mot au sujet des Bindings compilés : dans un contexte du desktop, ceci est une fonctionnalité propre à UWP. La même fonctionnalité existe pour mobiles avec Xamarin et le tout nouveau MAUI. Contrairement à UWP, elle ne nécessite pas l'utilisation de la syntaxe x:Bind. Quand les Bindings compilés seront supportés par

WPF et WinUI, il est fort probable que leur syntaxe suive la même logique que MAUI et reste telle qu'elle est actuellement.
Si vous entrevoyez le portage d'une application UWP vers WPF, évitez donc x:Bind. Si vous restez sur UWP, x:Bind est une solution très puissante qui permet de détecter nombre de problèmes à la compilation. Il est donc conseillé de l'utiliser.

Paramètres et préférences de l'utilisateur

Sur UWP, la gestion des préférences de l'utilisateur n'est pas un sujet à problématique. La classe ApplicationData.LocalSettings fournit tout ce dont on peut avoir besoin. Concernant les paramètres, gérés par l'IT lors du déploiement ou par les développeurs, il n'existe pas de solution intégrée. Il ne s'agit clairement pas d'un scénario ciblé par UWP. Rien ne vous interdit cependant d'ajouter à votre application des écrans dédiés à l'administrateur et d'utiliser ApplicationData.LocalSettings pour cela.
Comme bien souvent, WPF est plus souple. On sent bien l'héritage de Win32. On peut accéder à l'ensemble du stockage du PC, et au réseau. On peut donc envisager tout type de solution à base de fichiers, de base de données locale ou distante. Contrairement à UWP, on peut y accéder directement.
Il y a cependant deux options intégrées à WPF et .net 5 qui sont fort pratiques :

- Properties.Settings.Default : il s'agit de la même solution qui existe avec .net Framework depuis .net 2. Le designer a été conservé et n'a pas changé. On peut toujours y définir des paramètres utilisateur et applicatifs (pour l'IT). Le fichier XML résultant peut être modifié lors du déploiement par l'IT.
- ConfigurationBuilder : on retrouve là les mêmes possibilités que celles qui sont offertes par ASP.net core. La souplesse de la solution n'est plus à démontrer. On peut utiliser des fichiers, des variables d'environnement, et mixer le tout de manière transparente.

Properties.Settings.Default ne nécessite pas la mise en place d'une logique compliquée, car il s'agit d'un objet intrinsèque accessible de tous. Cependant, cela va à contrario de l'IOC et des tests unitaires. Pour garantir un usage propre et testable, il est donc conseillé de créer une classe qui va encapsuler tous les appels à Properties.Settings.Default. Cette classe sera utilisée par l'application et substituée par des Mock qui exploiteront la même Interface pour les tests unitaires. Si un jour vous souhaitez ne plus utiliser Properties.Settings.Default, le changement pourra se faire en douceur sans avoir d'impact sur le reste de l'application.
L'utilisation de ConfigurationBuilder peut se faire avec une approche très simple. Il suffit de modifier sa classe Host en lui ajoutant une méthode générique telle que celle-ci :

Code complet sur programmez.com & github

Cette méthode peut être utilisée dans la méthode ConfigureServices pour définir un Singleton qui représentera vos paramètres. Elle va lire le contenu d'un fichier appsettings.json se trouvant à la racine de votre application, tenter de le désérialiser et retourner votre objet. Si certaines propriétés du fichier json ne sont pas présentes sur votre classe, cela ne pose pas de problèmes. Il est même possible d'utiliser plusieurs classes différentes et utiliser la même méthode GetSettings pour les alimenter à partir d'un seul fichier.

Thread et UI

Après avoir suivi l'ensemble des conseils évoqués jusqu'ici, votre application ne court plus qu'un seul et unique danger : le gel de l'interface. Il s'agit d'un problème assez fréquent avec les applications .net Framework qui tend à disparaître avec .net 5. La raison est très simple : .net 5 et son prédécesseur .net core 3.1 nous contraignent de plus en plus à utiliser async / await pour toutes les opérations qui potentiellement peuvent conduire à un gel de l'interface si on les exécute sur son thread principal (lecture de fichiers, accès réseau, API…). Ceci permet de garder une interface réactive.
De manière générale, à partir du moment où votre application a besoin d'accéder à une ressource externe, utiliser async / await. Si vous utilisez un code ou une librairie synchrone que vous avez pris l'habitude de réutiliser d'un projet à l'autre, prenez le temps de les convertir vers async/await. Vos utilisateurs vous remercieront.
Dans le même ordre d'idée, il vous faut résister à la tentation de lancer des threads manuellement. Que ce soit via la classe Task ou toute autre solution. N'utiliser ces méthodes que pour lancer des tâches véritablement lourdes et/ou parallélisables. Et même dans ce cas, il faut privilégier l'async/await pour attendre la fin d'exécution de ces méthodes.
Une gestion manuelle des threads implique l'usage du Dispatcher pour mettre à jour l'interface. Ceci peut compliquer le débogage.

Localisation

Que l'on soit avec UWP ou WPF, la localisation de ressources est possible et peut être unifiée. Dans les deux cas, il est possible d'utiliser les fichiers RESX. Ces fichiers sont acceptés par WPF et .net Standard. UWP ne supportant pas ces fichiers, la mort du RESX a été annoncée à tort. Il s'agit du seul format qui peut être partagé avec le .net Framework. Il ne faut donc pas s'en priver. Pour l'utiliser avec UWP, il suffit de créer un projet .net Standard, d'y ajouter les fichiers RESX, et de rendre publiques les classes générées. Il reste alors à référencer ce projet dans l'application UWP.
Pour faciliter les traductions, il est conseillé d'utiliser le Multilingual App Toolkit. Cette extension pour Visual Studio va se charger de suivre l'évolution des fichiers RESX de votre langue à défaut (même si vous avez plusieurs fichiers). Elle va permettre la création de fichiers XLIFF à destination des traducteurs (un par langue à supporter, quel que soit le nombre de fichiers RESX). Quand ces fichiers sont traduits, l'extension va générer les fichiers RESX des langues supportées (ceci se fait automatiquement lors de la compilation).
Le format XLIFF est un format couramment utilisé par les traducteurs professionnels. Microsoft fournit un outil gratuit via lequel il est possible d'identifier rapidement les textes en attente de traduction et de les traduire. Celui-ci facilite grandement le travail.
Même une application qui n'est disponible que dans une langue peut tirer profit de la localisation. Le fait de centraliser les textes a de nombreux avantages. Notamment pour corriger rapidement une faute d'orthographe ou pour s'adapter si le vocabulaire change.

Conclusion

.Net 5 offre un contexte de développement agréable et performant dans lequel on peut rapidement se perdre. J'espère que les conseils évoqués ici vous permettront de vous lancer en toute tranquillité tout en restant prêts pour les évolutions à venir.

Christophe PICHAUD
Leader de Domaine NET chez Infeeny
christophep@cpixxi.com | www.windowscpp.com

Pourquoi C++ en 2020 ?

Le C++ est omniprésent dans l'industrie, les jeux vidéo, le médical, l'internet, les applications de bureau, le mobile, l'IA et le Cloud. L'avantage de C++ est qu'il bénéficie de toute la panoplie du C et de ses librairies. Ainsi, toutes les libs C/C++ de Linux sont facilement réutilisables. Dans d'autres langages, il faut des bindings (des ponts) sinon il est impossible de faire appel à ces libs. L'avantage de C/C++ est son côté multiplateforme supérieur à tous ses concurrents. Le C++ existe depuis 40 ans et il a été bâti après de mures réflexions au fil des ans. Il surpasse les langages comme Java et C#, Python et Rust de par sa STL, ses templates, sa capacité d'abstraction et son niveau de génération de code optimisé. C'est pour cela que tous les grands logiciels sont faits en C/C++.

Pourquoi utiliser C++ en 2020 ? La question est posée… C++ est un descendant de C, regardez : 1
Sur l'index TIOBE, C est repassé devant Java. Java est à 14%. Avec C et C++ on est à 24%. C++ est 4ème. Comme vous pouvez le constater, on est loin de ces sondages foireux qui classent JS en premier. Les vrais logiciels sont faits en C/C++, pas en JS dans des browsers. Il faut dire les choses.

C++, un meilleur C

C++ est un langage riche qui a été mis au point au fur et à mesure du temps, calmement et avec soin. Les rumeurs prétendent que C++ est difficile d'accès, un peu élitiste. Ce n'est pas vrai. 2
C++ apporte :

- La notion de classes ;
- Les hiérarchies de classes ou classes dérivées ;
- Les fonctions virtuelles ;
- La surcharge des opérateurs ;
- Les pointeurs intelligents ;
- Les templates ;
- La librairie standard STL.

```
class CDocManager : public CObject
{
    DECLARE_DYNAMIC(CDocManager)
public:

// Constructor
    CDocManager();

    //Document functions
    virtual void AddDocTemplate(CDocTemplate* pTemplate);
    virtual POSITION GetFirstDocTemplatePosition() const;
    virtual CDocTemplate* GetNextDocTemplate(POSITION& pos) const;
    virtual void RegisterShellFileTypes(BOOL bCompat);
    void UnregisterShellFileTypes();
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName); // open named file
    virtual CDocument* OpenDocumentFile(LPCTSTR lpszFileName, BOOL bAddToMRU); // open named file
    virtual BOOL SaveAllModified(); // save before exit
    virtual void CloseAllDocuments(BOOL bEndSession); // close documents before exiting
    virtual int GetOpenDocumentCount();
    virtual CDocTemplate* GetBestTemplate(LPCTSTR lpszFileName); // return best template named file

    // helper for standard commdlg dialogs
    virtual BOOL DoPromptFileName(CString& fileName, UINT nIDSTitle,
            DWORD lFlags, BOOL bOpenFileDialog, CDocTemplate* pTemplate);

//Commands
    // Advanced: process async DDE request
    virtual BOOL OnDDECommand(_In_z_ LPTSTR lpszCommand);
    virtual void OnFileNew();
    virtual void OnFileOpen();

// Implementation
protected:
    CPtrList m_templateList;
    int GetDocumentCount(); // helper to count number of total documents

public:
    static CPtrList* pStaticList;       // for static CDocTemplate objects
    static BOOL bStaticInit;            // TRUE during static initialization
    static CDocManager* pStaticDocManager;  // for static CDocTemplate objects

public:
    virtual ~CDocManager();
#ifdef _DEBUG
    virtual void AssertValid() const;
    virtual void Dump(CDumpContext& dc) const;
#endif
};
```

3

Les classes

Les classes regroupent des membres (fonctions, variables) qui forment un groupe cohérent et unique. On identifie une classe comme un élément autonome qui propose des méthodes (autre nom pour méthode) et des variables pour répondre à un besoin. Exemple de classe : 3
Cette classe est autonome. Elle hérite de CObject, possède des membres variables et fonctions simples et virtuelles. Pour l'utiliser, il suffit de déclarer un objet et de l'utiliser directement.
L'exemple de base est celui-ci :

```
A a ;
If( a.membre == value)
  …
a.fonction() ;
```

Comment définir une classe ?

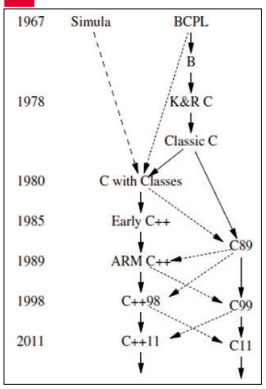
Ce n'est pas très compliqué. Prenons l'exemple d'un logiciel de dessin. On va modéliser les concepts d'un logiciel de dessin. Au départ, de quoi a-t-on besoin ?

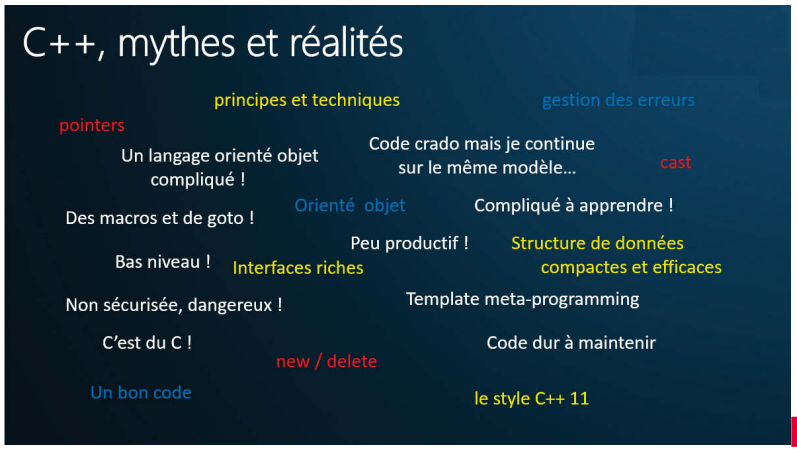
- D'une fenêtre de dessin ;
- De gabarits ;
- De propriétés externes ajustables ;
- Des options pour charger et sauvegarder les dessins.

Prenons un papier et un crayon et notons les classes au fur et à mesure : C'est la première phase !
Voici les différents éléments identifiés :

- CWindow : la fenêtre

1





2

• CShape : l'objet de dessin, le gabarit
• CProperties : les propriétés
• CFileManager : le gestionnaire de fichiers

Ensuite, on essaie de « remplir les carrés ». Chaque classe doit avoir des membres qui lui sont propres. Au fur et à mesure de la réflexion, on s'aperçoit que l'on réinvente peut-être des choses existantes : exemple, la fenêtre qui existe déjà sous Gnome ou Windows… Il faut donc affiner le design et le rendre appliqué au système d'exploitation.

C'est là que les classes peuvent avoir des membres purement techniques, issues de l'OS, qui sont un peu du domaine de la plomberie. On fera attention de bien isoler la plomberie du code « métier ». Ce n'est pas évident. Si le code de plomberie est mêlé au code métier, la classe va vite devenir un fourre-tout difficile à gérer. Il ne faut pas hésiter à isoler certaines choses.

La classe CShape

Cette classe représente le gabarit à dessiner. L'utilisateur utilise la souris et décide de créer un gabarit d'une taille particulière à une position particulière. On peut donc en déduire certaines propriétés comme position et taille.

```
int x ;
int y ;
CRect size ;
```

CRect est une classe qui représente un rectangle avec les membres left, top, right, bottom, TopLeft, BottomRight, etc.

Lorsque l'utilisateur va dessiner un gabarit, il va utiliser sa souris pour dimensionner l'objet, il faut donc gérer les évènements :

• LeftMouseDown
• LeftMouseUp
• MouseMove

C'est qu'il ne faut pas lâcher en se disant c'est trop compliqué. Dans la construction logicielle, on détaille tout à l'unité de fabrication la plus petite. Chaque opération d'un logiciel est codée, rien n'est le fruit du hasard ou du « ben, ça marche tout seul ! », non.

Les évènements LeftMouseDown/Up et MouseMove

Les évènements doivent être gérés dans la classe CWindow. Ça va ressembler à cela :

```
class CModeler1View : public CScrollView
{
…
public:
    afx_msg void OnLButtonDown(UINT nFlags, CPoint point);
    afx_msg void OnLButtonUp(UINT nFlags, CPoint point);
    afx_msg void OnMouseMove(UINT nFlags, CPoint point);
…
}
```

Comme vous pouvez le constater, les méthodes utilisent un objet CPoint.

Un peu d'algorithmes

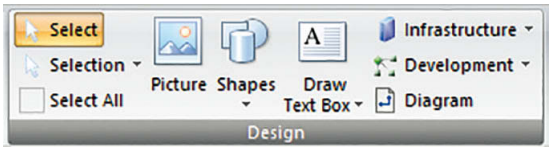
Pour avoir une mécanique de fonctionnement d'un logiciel de dessin, il va falloir un peu d'algorithmes. En effet, la souris peut faire double emploi. Je m'explique. Quand l'utilisateur fait un clic gauche, cela peut vouloir signifier :

• Dessin d'un gabarit ;
• Redimensionnement d'un gabarit ;
• Sélection d'un gabarit.

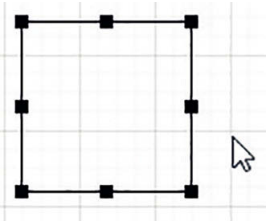
On s'aperçoit qu'il n'y a pas que l'action de la souris qui rentre en ligne de compte, mais aussi des états. Cela peut se manifester par une sélection dans une barre d'outils :

• Dessiner ;
• Sélectionner.

Voici la barre de sélection :

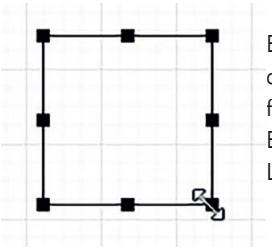


La sélection d'un objet peut aussi être orientée. En effet, si un objet est sélectionné, son contour se voit ajouter des carrés de sélection (un tracker) qui ressemble à cela :



Lorsque la souris bouge (évènement MouseMove) et passe au-dessus d'un petit carré, le curseur de la souris se voit modifié pour indiquer que le redimensionnement est possible. Il y a donc deux états :

• L'état du type d'opération (select, draw) sur LeftMouseClick ;
• L'état du mode de l'opération (move, sizing) sur MouseMove.



Et maintenant que l'on a cerné la cinématique, il ne reste plus qu'à faire l'algorithme.

Examinons le code de la fonction LeftMouseDown :

```
void CElementManager::OnLButtonDown(CModeler1View* pView, UINT nFlags, const
CPoint& cpoint)
{
  if( m_type == ElementType::type_unknown )
  {
    //FIXME : do we need to handle not implemented objects ?
    return;
  }

  CPoint point = cpoint;
  ViewToManager(pView, point);
  m_clickPoint = point;
  m_lastPoint = point;
  m_selectPoint = point;

  if( m_type == ElementType::type_select )
  {
```

```
    m_selectMode = SelectMode::none;

    // Check for resizing (only allowed on single selections)
    if( HasSelection() &&  m_selection.GetCount() == 1)
    {
      std::shared_ptr<CElement> pElement = m_selection.GetHead();
      //Change cursor look because mouse click is over an object for sizing
      m_nDragHandle = pElement->HitTest(point, pView, TRUE);
      if (m_nDragHandle != 0)
      {
        m_selectMode = SelectMode::size;

        if (m_nDragHandle == 2)
        {
          m_connectorInUse = ConnectorType::connector2;
        }
        else
        {
          m_connectorInUse = ConnectorType::connector1;
        }
      }
      else
      {
        m_connectorInUse = ConnectorType::connector2;
      }
    }

    if( m_selectMode == SelectMode::none )
    {
      // See if the click was on an object
      std::shared_ptr<CElement> pElement = m_objects.ObjectAt(point, m_selectType);
      if( pElement != NULL )
      {
        if( IsSelected(pElement) == false )
        {
          if( (nFlags & MK_SHIFT) || (nFlags & MK_CONTROL))
          {
          }
          else
            SelectNone();

          if (pElement->m_bGrouping == false)
          {
            m_objectId = pElement->m_objectId;
            Select(pElement);
          }
          else
          {
            for (vector<std::shared_ptr<CElement>>::const_iterator itSel = pElement->
m_pElementGroup->m_Groups.begin(); itSel != pElement->m_pElementGroup->m_Groups.
end(); itSel++)
            {
              std::shared_ptr<CElement> pObj = *itSel;
              Select(pObj);
            }
          }
          pElement->m_bMoving = true;
        }
        else
        {
          // if shift or control i spressed, unselect the element
          if ((nFlags & MK_SHIFT) || (nFlags & MK_CONTROL))
          {
            Deselect(pElement);
          }
        }

        m_selectMode = SelectMode::move;

        // Update UI
        UpdateUI(pView, pElement);
        // Redraw
        Invalidate(pView, pElement);
      }
      else
      {
        // See if the click was on an object
        // TRUE -> select and start move if so
        // FALSE -> Click on background, start a net-selection
        // m_selectMode = netSelect;

        if( HasSelection() )
        {
          SelectNone();
          Invalidate(pView, pElement);
        }

        m_selectMode = SelectMode::netselect;
      }
    }
  }
  // We are not in a select operation
  // -> this is a drawing operation
  // We have to create...
  // Create a Drawable Object...
  else
  {

#ifdef VERSION_COMMUNITY
    if (CFactory::g_counter > MAX_SHAPES && IsMyLocalDev() == false)
    {
      AfxMessageBox(_T("Maximum number or shapes reached !\nFor more, please buy the A
rchitect Edition."));
      return;
    }
#endif

    SelectNone();

    std::shared_ptr<CElement> pNewElement = CFactory::CreateElementOfType(m_type,
m_shapeType);
```

```
    if( m_type == ElementType::type_unknown )
    {
      return;
    }

    if (m_shapeType == ShapeType::selection)
    {
      m_bSelectionHasStarted = true;
      pSelectionElement = pNewElement;
    }

    pNewElement->m_point = point;
    // For plumbing purpose...
    pNewElement->m_pManager = this;
    pNewElement->m_pView = pView;

    // Add an object
    m_objects.AddTail(pNewElement);

    // Store last created object
    m_objectId = pNewElement->m_objectId;

    // Select the new element
    Select(pNewElement);

    m_selectMode = SelectMode::size;

    m_nDragHandle = 1;
    m_connectorInUse = ConnectorType::connector2;
    FindAConnectionFor(pNewElement, point, pView, ConnectorType::connector1);

    pView->GetDocument()->SetModifiedFlag();

    // Update ClassView & FileView
    UpdateClassView();//pNewElement);
    UpdateFileView();//pNewElement);

    // Update UI
    UpdateUI(pView, pNewElement);
  }

}
```

Examinons maintenant la fonction OnMouseMove :

```
void CElementManager::OnMouseMove(CModeler1View* pView, UINT nFlags, const CPoint& cpoint)
{
  // a SELECT operation is started
  if( m_selectMode == SelectMode::none )
  {
  }

  CPoint point = cpoint;
  CPoint m_movePoint = point;
  ViewToManager(pView, point);
  CPoint lastPoint = point;

  std::shared_ptr<CElement> pElement = m_selection.GetHead(); //m_objects.FindElement(m
_objectId);
  if( pElement == NULL )
  {
    return;
  }

  if( m_type == ElementType::type_select )
  {
    if( HasSelection() )
    {
      // Change cursor look temporary just because mouse could be over a shape
      int nHandle = pElement->HitTest(point, pView, true);
      if (nHandle != 0)
      {
        SetCursor(pElement->GetHandleCursor(nHandle));
      }

      if( m_selectMode == SelectMode::move )
      {

       for( vector<std::shared_ptr<CElement>>::const_iterator itSel = m_selection.m_objects
.begin() ; itSel!=m_selection.m_objects.end() ; itSel++ )
        {
          std::shared_ptr<CElement> pObj = *itSel;

          CPointdelta = (CPoint)(point - m_lastPoint);
          InvalObj(pView, pObj);
          pObj->m_rect += delta;
          pObj->m_point = pObj->m_rect.TopLeft();
          InvalObj(pView, pObj);
        }

        pView->GetDocument()->SetModifiedFlag();
      }

      if( m_selectMode == SelectMode::size )
      {
        if( m_nDragHandle != 0)
        {

          std::shared_ptr<CElement> pObj = m_selection.GetHead();
          pObj->MoveHandleTo(m_nDragHandle, point, pView);
          FindAConnectionFor(pElement, point, pView, m_connectorInUse);
          InvalObj(pView, pObj);

          pView->GetDocument()->SetModifiedFlag();
        }
      }

    }
  }
  else
  {
    if( m_selectMode == SelectMode::size )
    {
      pElement->m_last = point;
```

```
      pElement->InvalidateObj();
      FindAConnectionFor(pElement, point, pView, m_connectorInUse);
      InvalObj(pView, pElement);

      pView->GetDocument()->SetModifiedFlag();
    }
  }

  m_lastPoint = point;

  // Check for mouse cursor -> sizing/moving
  if( m_selectMode == SelectMode::size )
  {
    //if (m_nDragHandle != 0)
    {
      SetCursor(m_selection.GetHead()->GetHandleCursor(m_nDragHandle));
    }
  }

  if( m_type == ElementType::type_select && m_selection.GetCount() == 0)
  {
    SetCursor(AfxGetApp()->LoadStandardCursor(IDC_ARROW));
  }
}
```

Examinons maintenant la fonction LeftMouseUp :

```
void CElementManager::OnLButtonUp(CModeler1View* pView, UINT nFlags, const CPoint& cpoint)
{
  CPoint point = cpoint;
  ViewToManager(pView, point);


  //if( m_selectMode == SelectMode::move || m_selectMode == SelectMode::size )
  {
    // Selection Moving or Sizing is finished.
    // Nothing to do.
    m_selectMode = SelectMode::none;
  }

  //m_bDrawing = FALSE;

  std::shared_ptr<CElement> pElement = m_objects.FindElement(m_objectId);
  if (pElement == NULL)
    return;

  if (m_type == ElementType::type_select)
  {
    if (HasSelection() && m_selection.GetCount() == 1)
    {
      // Nothing to do...
  }
  else
  {
    // Finish a drawing operation
    pElement->m_last = point;
    pElement->InvalidateObj();
    pElement->CheckForKeepingAMinimumSize();
    // Switch the view in Select Mode
    m_type = ElementType::type_select;
  }

  //
  // We are gonna end a slection rect...
  //

  if (m_bSelectionHasStarted == true)
  {
    CRect rect = pSelectionElement->m_rect;

    // remove the selection element from the objects list
    vector<std::shared_ptr<CElement>>::iterator pos;
    pos = find(m_objects.m_objects.begin(), m_objects.m_objects.end(), pSelectionElement);
    if (pos != m_objects.m_objects.end())
    {
      m_objects.m_objects.erase(pos);
    }

    // remove the selection element from the selection list
    vector<std::shared_ptr<CElement>>::iterator pos2;
    pos2 = find(m_selection.m_objects.begin(), m_selection.m_objects.end(), pSelectionElement);
    if (pos2 != m_selection.m_objects.end())
    {
      m_selection.m_objects.erase(pos2);
    }

    //ViewToManager(pView, rect);
    SelectNone();

    shared_ptr<CElement> pLastSelected = nullptr;
    vector<std::shared_ptr<CElement>> v = m_objects.ObjectsInRectEx(rect, m_selectType);
// version Ex : do not select lines with full connector
    if (v.size() != 0)
    {
      for (std::shared_ptr<CElement> pElement : v)
      {
        if (IsSelected(pElement) == false)
        {
          if (pElement->m_bGrouping == false)
          {
            Select(pElement);
            pLastSelected = pElement;
          }
        }
      }
    }

    if (pLastSelected != nullptr)
    {
      // Update UI
      UpdateUI(pView, pLastSelected);
    }
```

```
    pSelectionElement = nullptr;
    m_bSelectionHasStarted = false;
  }

  m_bSizingALine = false;

  // Set selectType to default
  m_selectType = SelectType::intuitive;
  m_connectorInUse = ConnectorType::connector2;
  m_bDrawRectForConnectionPoint = false;

  pElement->m_bMoving = FALSE;
  // Redraw
  InvalObj(pView, pElement);

  m_selectMode = SelectMode::none;
  pView->GetDocument()->SetModifiedFlag();
  pView->GetDocument()->UpdateAllViews(pView);
}
```

Maintenant que les évènements sont codés, il faut coder les commandes de la barre d'outils.

Création d'un gabarit

Pour créer un gabarit, le Ribbon met à disposition un ensemble de gabarits. Chaque gabarit possède un index et il existe une enum qui contient le mapping entre l'index et le nom de l'élément :

Code complet sur programmez.com & github

Voici le Ribbon : 4

Examinons l'évènement associé à cet élément du Ribbon :

```
void CModeler1View::OnModelingShapes()
{
  // TODO: Add your command handler code here
  GetManager()->m_type = ElementType::type_shapes_simple;
  int shapeId = CMFCRibbonGallery::GetLastSelectedItem(ID_DESIGN_SHAPES);
  GetManager()->m_shapeType = CShapeType::ToShapeType(OffsetShapes_Simple + shapeId);
}
```

Le positionnement des états permet de passer dans le mode suivant sur l'évènement LeftMouseBtnDown :

```
std::shared_ptr<CElement> pNewElement =
  CFactory::CreateElementOfType(m_type, m_shapeType);
pNewElement->m_point = point;
// For plumbing purpose...
pNewElement->m_pManager = this;
pNewElement->m_pView = pView;
// Add an object
m_objects.AddTail(pNewElement);
// Store last created object
m_objectId = pNewElement->m_objectId;
// Select the new element
Select(pNewElement);
m_selectMode = SelectMode::size;
```

Voici le déroulé des opérations :

- Création de l'élément ;
- Association des éléments technique (manager, view) ;
- Ajout de l'élément à liste des objets à afficher ;
- Sélection de l'élément ;
- Passage en état mode sizing.

Accès aux données

Dessiner des gabarits, c'est bien, mais il faut aussi penser aux opérations Load & Save et là il faut un support. Via le Framework des MFC je peux faire un support fichiers très facilement via la méthode Serialize du Document. Cela est très facile. Moi ce que je veux, c'est un support de bases de données comme SQLite par exemple. C'est portable et très rapide et surtout très pratique. Procédons par étape :

- Download de SQLite ;
- Recompilation ;
- Création du wrapper C++ ;
- Création de la couche d'accès aux données.

Download de SQLite

Il faut aller sur https://sqlite.org/index.html et télécharger le fichier https://sqlite.org/2020/sqlite-src-3330000.zip

Pour ceux qui veulent une solution simple, avec un seul fichier (sqlite3.c) on préférera https://sqlite.org/2020/sqlite-amalgamation-3330000.zip

Une fois que l'on a les sources, on les extrait dans un répertoire sqlite et on crée un projet Visual C++ de type DLL.

Dans le header sqlite3.h, on ajoute les extensions SQLITE_API devant les fonctions que l'on veut exporter. Vous allez me dire, lesquelles ? Toutes ! Par défaut, sqlite est une librairie statique donc il faut exporter les fonctions si l'on veut utiliser une DLL.

Contenu du fichier sqlite.h :

```
#ifdef SQLITE_EXPORTS
#define SQLITE_API __declspec(dllexport)
#else
#define SQLITE_API __declspec(dllimport)
#endif
```

Contenu du fichier sqlite3.h :

```
../..
SQLITE_API int sqlite3_open(
  const char *filename,  /* Database filename (UTF-8) */
  sqlite3 **ppDb         /* OUT: SQLite db handle */
);
SQLITE_API int sqlite3_open16(
  const void *filename,  /* Database filename (UTF-16) */
  sqlite3 **ppDb         /* OUT: SQLite db handle */
);
SQLITE_API int sqlite3_open_v2(
  const char *filename,  /* Database filename (UTF-8) */
  sqlite3 **ppDb,        /* OUT: SQLite db handle */
  int flags,             /* Flags */
  const char *zVfs       /* Name of VFS module to use */
);
../…
```

Gestion des utilisateurs et mot de passe

Pour avoir le support des user/pwd, il faut ajouter un bout de code à sqlite3.c. Nous allons piocher dans les extensions de sqlite et

récupérer un bout de code depuis sqlite-src-3330000.zip\sqlite-src-3330000\ext\userauth\.
Ajoutez le bout de code au bout du fichier sqlite3.c.
Cela permet d'avoir une fonction qui prend user/pwd :

```
int sqlite3_user_authenticate(
  sqlite3 *db,        /* The database connection */
  const char *zUsername, /* Username */
  const char *aPW,     /* Password or credentials */
  int nPW             /* Number of bytes in aPW[] */
);
```

Compilation de la DLL

Voici le projet de DLL C++ dans Visual Studio : 5

Création du wrapper

Pour créer le wrapper, on va faire des classes simples, mais pratiques à utiliser :

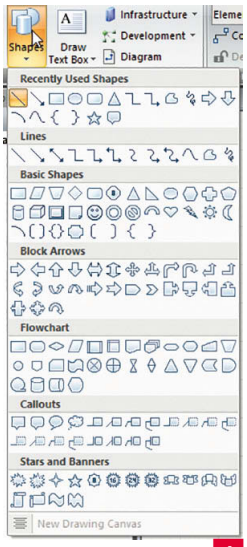
- Database ;
- DatabaseException ;
- ColumnType ;
- PreparedStmt ;
- Query. 6

Contenu du fichier Database.h :

```
#ifndef __Database_H__
#define __Database_H__

#include <sqlite/sqlite3.h>

namespace SQLite
{

  class Query;
  class PreparedStmt;

  class SQLITEWRAPPER_API Database
  {
  public:
    Database(void);
    Database(std::string filename);
    virtual ~Database(void);

  public:
    void SetDatabaseName(std::string fileName);
    void AddUser(std::string user, std::string password);
    bool AuthenticateUser(std::string user, std::string password);
    bool IsOpen();
    sqlite3_stmt* PrepareQuery(std::string query);
    void SetFileName(std::string filename);
    bool CreateAdminUser();
    bool Open();
    bool OpenEx(std::string user, std::string password);
    bool Close();
    void ReportError(std::string context);
    int ExecuteSQL(std::string sql, int& rows);
    int ExecuteCommand(std::string command, int& rows);
    Query ExecuteQuery(std::string query, int& rows);
    PreparedStmt CreatePreparedStmt(std::string statement);
    int ExecuteScalar(std::string query);
    sqlite_int64 GetLastRowID();
    void SetBusyTimeout(int timeout);
    sqlite_int64 InsertAndReturnID(std::string sqlinsert);

  protected:
    std::string m_DatabaseFileName;
    bool m_bOpen;
    sqlite3* m_instance;
  };

}
#endif //__Database_H__
```

Contenu de PreparedStmt.h :

```
#ifndef __PreparedStmt_H__
#define __PreparedStmt_H__

#include <sqlite/sqlite3.h>
#include <string>
#include "Query.h"

namespace SQLite
{

  class SQLITEWRAPPER_API PreparedStmt
  {
  public:
    PreparedStmt(void);
    PreparedStmt(sqlite3* instance, sqlite3_stmt* stmt);
    virtual ~PreparedStmt(void);

  public:
    PreparedStmt& operator=(const PreparedStmt& right);
    void SetStringParameter(int index, std::string value);
    void SetIntegerParameter(int index, int value);
    void SetDoubleParameter(int index, double value);
    void SetInt64Parameter(int index, sqlite_int64 value);
    void SetNullParameter(int index);
    void Reset();
    void Close();
    int Execute(int& rozs);
    sqlite_int64 ExecuteAndGetID();
    Query ExecuteAndGetQuery();

  protected:
    sqlite3_stmt* m_statement;
    sqlite3* m_instance;
    int m_NumberOfParameters;

  public:
    inline bool IsOpen() { return m_instance != NULL; }
```



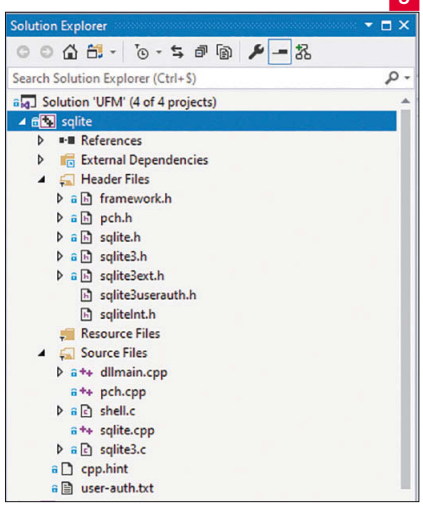
4

```
    inline bool IsStmtOpen() { return m_statement != NULL; }
    inline void SetNull() { m_statement = NULL; }
    inline bool CheckParameter(int index) { return (index <= m_NumberOfParameters); };
    inline int  GetNumberOfParams() { return m_NumberOfParameters; }
  };

}

#endif //__PreparedStmt_H__
```

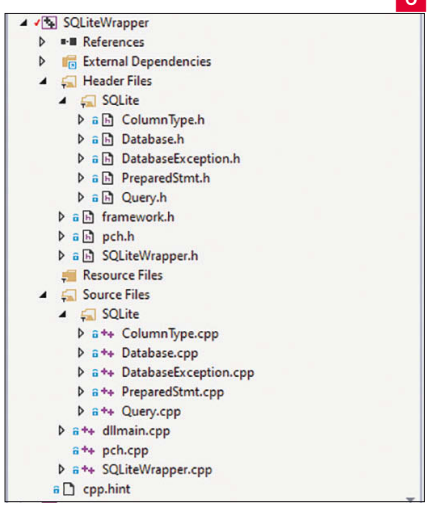
Contenu de Query.h :

```
#ifndef __Query_H__
#define __Query_H__

#include <string>
#include <sqlite/sqlite3.h>
#include "common/ColumnType.h"
#include "common/DatabaseException.h"

namespace SQLite
{

  class SQLITEWRAPPER_API Query
  {
  public:
    Query(void);
    virtual ~Query(void);
    Query(sqlite3_stmt* stmt, sqlite3* instance, bool norows, bool ownstatement = true);
    Query(const Query& query);
    Query& operator=(const Query& query);

  protected:
    sqlite3_stmt* m_statement;
    sqlite3* m_instance;
    int m_NbCols;
    bool m_bEOF;
    bool m_bDestroyStatement;
```

5



6




```
  protected:
    inline bool IsOpen() { return m_instance != NULL; }
    inline bool IsStmtOpen() { return m_statement != NULL; }
    inline void SetNull() { m_statement = NULL; }

  public:
    inline bool IsEOF() { return m_bEOF; }
    inline bool IsColumnIndexValid(int index) { return (((index >= 0) && (index <= (m_Nb
Cols - 1)))); }
    inline int GetNumberOfColumns() { return m_NbCols; }
    void Close();
    std::string GetColumnName(int index);
    ColumnType GetColumnType(int index);
    double GetDoubleValue(int index);
    int GetIntValue(int index);
    long GetLongValue(int index);
    std::string GetStringValue(int index);
    void MoveNext();
  };

}

#endif //__Query_H__
```

Ces classes encapsulent l'accès à la librairie C SQLite en C++. Pourquoi ? Le code C++ est bien plus lisible quand on utilise des objets plutôt que des fonction C en vrac. Voici le contenu de la fonction Database ::OpenEx qui prend un user/pwd :

```
  bool Database::OpenEx(std::string user, std::string password)
  {
      int retValue = sqlite3_open_v2(m_DatabaseFileName.c_str(), &m_instance, SQLITE_
OPEN_READWRITE | SQLITE_OPEN_CREATE, NULL);
      m_bOpen = (retValue == SQLITE_OK);
      if (!m_bOpen)
      {
          ReportError(" opening database file " + m_DatabaseFileName + " ");
      }

      retValue = sqlite3_user_authenticate(m_instance, user.c_str(), password.c_str(),
 password.size());
      if (retValue == SQLITE_OK)
          return true;

      return false;
  }
```

Son utilisation est très simple :

```
  // open database
  SQLite::Database db;
  string dbName = UFM_SQLITE_DATABASE;
  db.SetDatabaseName(dbName);
  if (!db.OpenEx(UFM_SQLITE_USER, UFM_SQLITE_PASSWORD))
      return;
```

Pour la gestion des requêtes, le principe est le même. Exemple de stockage de diagramme :

```
    // store to db
    SQLiteDiagramEntity diagramEntity(&db);
    diagramEntity.FileName = diagramName;
    diagramEntity.Json = strJson;
    diagramEntity.InsertOrUpdate(m_diagramId);
    db.Close();
```

Voici le code de la méthode InsertOrUpdate :

```
bool SQLiteDiagramEntity::InsertOrUpdate(int& id)
{
  string sql;

  if (id == 0)
  {
    // Insert
    // "INSERT INTO Diagram VALUES (NULL, datetime('now'), ?, ?);"
    sql = SQLiteQueryString::GetSQLQuery(SQLiteQueryString::Insert_Diagram);

    SQLite::PreparedStmt stmt = m_pDatabase->CreatePreparedStmt(sql);
    stmt.SetStringParameter(1, this->FileName);
    stmt.SetStringParameter(2, this->Json);
    //stmt.SetNullParameter(6);

    int rows;
    if (!stmt.Execute(rows))
      return false;

    stmt.Close();

    id = m_pDatabase->GetLastRowID();
  }
  else
  {
    // Update
    // "UPDATE Diagram SET Json=? WHERE DiagramPK=?;"
    sql = SQLiteQueryString::GetSQLQuery(SQLiteQueryString::Update_Diagram);

    SQLite::PreparedStmt stmt = m_pDatabase->CreatePreparedStmt(sql);
    stmt.SetStringParameter(1, this->Json);
    stmt.SetIntegerParameter(2, id);
    //stmt.SetNullParameter(6);

    int rows;
    if (!stmt.Execute(rows))
      return false;

    stmt.Close();
  }

  return true;
}
```

Le mécanisme est simple : on déclare un statement avec une requête SQL, on lui affecte les différents paramètres, et on exécute le tout. Cela c'est pour l'insertion ou la modification de données. Pour la récupération, le code est différent. Voici la méthode qui charge tous les diagrammes de la base :

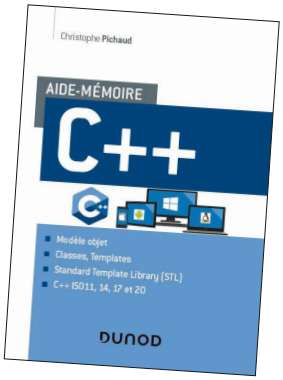
```
vector<shared_ptr<SQLiteDiagramEntity>> SQLiteDiagramEntity::SelectAll()
{

  vector<shared_ptr<SQLiteDiagramEntity>> vDiagrams;
  string sql;

  try
  {
    // "SELECT DiagramPK, LastUpdate, FileName, Json FROM Diagram;"
    sql = SQLiteQueryString::GetSQLQuery(SQLiteQueryString::SelectAll_Diagram);
    int rows = 0;

    //SQLite::Query q;
    //q = m_pDatabase->ExecuteQuery("select count(*) from Diagram", rows);
    //int count = q.GetLongValue(0);
    //q.Close();

    rows = 0;
    SQLite::Query query;
    query = m_pDatabase->ExecuteQuery(sql, rows);

    while (query.IsEOF() == false)
    {
      shared_ptr<SQLiteDiagramEntity> sde = make_shared<SQLiteDiagramEntity>();

      sde->DiagramPK = query.GetLongValue(0);
      sde->LastUpdate = query.GetStringValue(1);
      sde->FileName = query.GetStringValue(2);
      sde->Json = query.GetStringValue(3);

      vDiagrams.push_back(sde);

      query.MoveNext();
    }

    query.Close();
  }
  catch (SQLite::DatabaseException ex)
  {
    std::cerr << ex.ToString() << std::endl;
  }

  return vDiagrams;
}
```

Conclusion

Le C++ Moderne, c'est l'utilisation accrue de la STL, des smart pointers, des containers et des applications sophistiquées. On tire parti du système d'exploitation et de ses librairies et aussi des librairies open source. L'accès aux données avec SQLite est très pratique. Le code présenté est disponible sur :

https://github.com/ChristophePichaud/UltraFluidModelerNET

Avec une syntaxe proche à 95% de Java et C#, C++ est facile à apprendre. Pour vous initier, n'oubliez pas mon livre « Aide-Mémoire C++ » chez DUNOD.



Alain Zanchetta
Principal Software Developer dans l'équipe Windows Mixed Reality de Microsoft.

Introduction à C++/WinRT

Windows 8 avait introduit un nouveau modèle d'application – Universal Windows Application – qui était destiné à remplacer le modèle existant depuis toujours. Ce basculement ne s'est pas effectué et les applications classiques – souvent appelées Applications Win32 – continuent à dominer sur nos bureaux.

Néanmoins, ce nouveau modèle a de nombreux avantages techniques et est en particulier associé à une redéfinition complète des API de Windows, totalement orientée objet et cohérente, ce qui rend accessible aux développeurs et développeuses l'ensemble des domaines couverts par ces API.

Les adeptes du C++ et de Win32 font généralement face à des collections de fonctions sans aucune cohérence d'un domaine à un autre : parfois elles retournent des entiers pour indiquer leur succès ou la raison de leur échec, parfois elles retournent des booléens et un appel à GetLastError() est alors nécessaire pour savoir ce qui s'est réellement passé. Les API UWP apportent à ces développeurs et développeuses le confort auquel sont habitué·e·s depuis de nombreuses années leurs homologues adeptes du C# et de .NET.

Malheureusement, migrer une application en bloc d'un modèle de programmation et son ensemble d'API à un autre n'est en pratique que rarement faisable, car cela prend énormément de temps et le retour sur investissement est très long, comparé au simple ajout de nouvelles fonctions au code existant.

Est-ce à dire que les développeurs et développeuses C++ sont condamné·e·s à écrire du code obsolète ? Non, car lors de la Build 2020, Microsoft a enterré définitivement la séparation entre les applications modernes et les applications classiques en formalisant son initiative appelée « Project Reunion » : **un des aspects majeurs de cette initiative est de rendre les nouvelles API de Windows accessibles à toutes les applications**.

Cette possibilité n'est pas totalement nouvelle – un bon nombre des nouvelles classes étaient déjà accessibles aux applications Win32 – mais elle devient une priorité et se généralise. Par ailleurs, pour les développements en C++, ce rapprochement entre les applications classiques et les APIs UWP est facilité par l'adoption d'une autre technologie – ***C++/WinRT*** – qui donne accès à ces API via du code C++ standard, alors que le développement UWP s'appuyait auparavant sur un dialecte appelé C++/CX qui rebutait les puristes. La suite de cet article explique pourquoi ce C++/WinRT était nécessaire et comment le mettre en œuvre.

Pourquoi une nouvelle technologie C++ ?

Pour être universelle, l'interface de programmation d'un système d'exploitation doit être indépendante du langage de programmation, mais aussi supporter les évolutions des classes qu'elle contient et celles des outils de développement (il est généralement déconseillé de fusionner dans le même binaire des composants écrits avec différentes versions d'un compilateur). Pour cela, elle s'appuie sur une ***ABI – Application Binary Interface*** – qui définit comment s'effectuent au plus bas niveau les instanciations d'objets et les appels de méthodes. Chaque langage de programmation – C++, C#, JavaScript, etc. – utilise ensuite une projection spécifique de ces API pour exposer au développeur ou à la développeuse une intégration naturelle et consistante avec reste du langage.

Au plus bas niveau, l'ABI du Windows Runtime emprunte les éléments fondamentaux de COM comme la notion d'interface dérivant de IUnknown ainsi que l'utilisation de HRESULT pour signaler les erreurs, l'immuabilité des interfaces une fois définies. Les principales différences avec le COM « standard » résident dans les modes d'activation, les modèles de threading, les métadonnées (remplacement des .TLB par des .WINMD) ainsi que dans l'utilisation d'un certain nombre d'interfaces prédéfinies pour exposer de manière standard quelques mécanismes comme les événements, les itérations ou encore les appels asynchrones.

Un langage comme C# masque toute cette complexité en exposant au niveau de la classe elle-même l'union des méthodes appartenant aux différentes interfaces implémentées par cette classe (et en transformant au passage les HRESULT en exceptions).

Dans le cas du C++, la correspondance entre classe C++ et classe WinRT est tout simplement impossible à réaliser directement : par exemple, les opérateurs new et delete donnent toute la charge de l'allocation et de la libération de la mémoire au code client d'un objet, alors que ces opérations sont de la responsabilité de l'implémentation de cet objet.

La projection de l'ABI en C++ peut alors suivre plusieurs voies, qui ont été explorées et utilisées par Microsoft :

Exposer ou implémenter directement les interfaces COM de l'ABI	***WRL (Windows Runtime Libraries)***
Étendre le langage C++ pour se rapprocher du confort de C#	***C++/CX***
S'appuyer sur des classes C++ générées par les outils.	***C++/WinRT***

Avant de plonger dans le vif du sujet, c'est-à-dire la mise en œuvre C++/WinRT, comparons ces différentes approches dans un petit exemple affichant l'identifiant, l'intervalle de publication et la latence de l'accéléromètre par défaut.

WRL = efficace, mais lourd

À chaque fois qu'une classe évolue (ajouts de méthodes dans une nouvelle version), on doit définir une nouvelle interface, voire deux si certaines méthodes ajoutées sont statiques. Prenons l'exemple de Windows.Devices.Sensors.Accelerometer qui masque… 9 interfaces COM différentes !

Membre	Interface
MinimumReportInterval (Propriété)	IAccelerometer
ReportInterval (Propriété)	IAccelerometer
GetCurrentReading (Méthode)	IAccelerometer
ReadingChanged (Evénement)	IAccelerometer
Shaken (Evénement)	IAccelerometer
ReadingTransform (Propriété)	IAccelerometer2
MaxBatchSize (Propriété)	IAccelerometer3
ReportLatency (Propriété)	IAccelerometer3
ReadingType (Propriété)	IAccelerometer4
ReportThreshold (Propriété)	IAccelerometer5
DeviceId (Propriété)	IAccelerometerDeviceId
GetDefault() (Méthode statique)	IAccelerometerStatics
GetDefault(AccelerometerReadingType) (Méthode statique)	IAccelerometerStatics2
FromIdAsync (Méthode statique)	IAccelerometerStatics3
GetDeviceSelector (Méthode statique)	IAccelerometerStatics3

```
ComPtr<IAccelerometerStatics> spAccelerometerStatics;

RETURN_IF_FAILED(GetActivationFactory(HStringReference(RuntimeClass_Windows_Devices
_Sensors_Accelerometer).Get(), &spAccelerometerStatics));
ComPtr<IAccelerometer> spAccelerometer;
RETURN_IF_FAILED(spAccelerometerStatics->GetDefault(&spAccelerometer));
ComPtr<IAccelerometerDeviceId> spAccelerometerDeviceId;
RETURN_IF_FAILED(spAccelerometer.As(&spAccelerometerDeviceId));
HString deviceId;
RETURN_IF_FAILED(spAccelerometerDeviceId->get_DeviceId(deviceId.GetAddressOf()));
UINT32 reportInterval;
RETURN_IF_FAILED(spAccelerometer->get_ReportInterval(&reportInterval));
ComPtr<IAccelerometer3> spAccelerometer3;
RETURN_IF_FAILED(spAccelerometer.As(&spAccelerometer3));
unsigned int latency;
RETURN_IF_FAILED(spAccelerometer3->get_ReportLatency(&latency));
cout << "Device #" << deviceId.GetRawBuffer(nullptr) << ":\n Report Interval: "
   << reportInterval << "ms\n Latency: "<< latency << "ms";
```

C++/CX = facile… mais pas du « vrai » C++

L'exemple précédent s'écrit en C++/CX aussi simplement qu'en C# :

```
Accelerometer^ defaultAccelerometer = Accelerometer::GetDefault();
String^ deviceId = defaultAccelerometer->DeviceId;
unsigned int reportInterval = defaultAccelerometer->ReportInterval;
unsigned int latency = defaultAccelerometer->ReportLatency;

cout << "Device #" << deviceId->Data() << ":\n Report Interval: "
   << reportInterval << "ms\n Latency: " << latency << "ms";
```

Mais malheureusement, ce n'est pas du C++ standard et c'est totalement rédhibitoire pour un langage qui devient aujourd'hui de plus en plus portable avec sa standardisation respectée par quasiment tous les compilateurs.

Au passage, l'utilisation d'une syntaxe proche du C++ CLI de .NET avec son ^ pour les références a souvent semé la confusion parmi les développeurs et développeuses en leur faisant penser que le code généré n'était pas du code natif, mais un code intermédiaire.

C++/WinRT = standard… et facile ?

Le C++/WinRT est du pur C++ (version 17) et s'appuie sur la génération de classes :

- Les classes sont générées par le compilateur à partir des fichiers de métadonnées .WINMD. Ces classes sont simples à utiliser et s'intègrent parfaitement dans un code C++ totalement standard.
- L'implémentation de composants WinRT passe par l'utilisation de fichiers IDL et l'héritage des classes générées. Ceci est moins aisé que la consommation des API, mais reste néanmoins lui aussi basé sur du code C++ standard.

Notre exemple devient :

```
Accelerometer defaultAccelerometer = Accelerometer::GetDefault();

hstring deviceId = defaultAccelerometer.DeviceId();
unsigned int reportInterval = defaultAccelerometer.ReportInterval();
unsigned int latency = defaultAccelerometer.ReportLatency();

wcout << L"Device #" << deviceId.c_str() << L":\n Report Interval: "
   << reportInterval << L"ms\n Latency: " << latency << L"ms";
```

Nous allons maintenant détailler un certain nombre de mécanismes de mise en œuvre de C++/WinRT.

Intégrer C++/WinRT dans une application

Même les applications les plus archaïques avec leur Winmain et leur boucle GetMessage() / TranslateMessage() / DispatchMessage() peuvent utiliser C++/WinRT sans devoir être restructurées. Imaginons une telle application qu'on aimerait moderniser un petit peu en adaptant ses couleurs à la luminosité ambiante.

Les API du Windows Runtime contiennent la classe Windows.Devices.Sensors.LightSensor qui permet d'accéder au capteur de luminosité éventuellement présent sur la machine, ce qui est assez fréquent sur les portables récents.

Moderniser notre vénérable application ne demande que quelques étapes :

- Activer le support de C++/WinRT dans le projet ;
- Inclure les en-têtes nécessaires à l'utilisation de la classe LightSensor ;
- Accéder à cette classe comme à une classe C++ normale : construction puis appels de méthodes.

Pour activer le support de C++/WinRT, le plus simple est d'ajouter au projet le package Nuget correspondant :

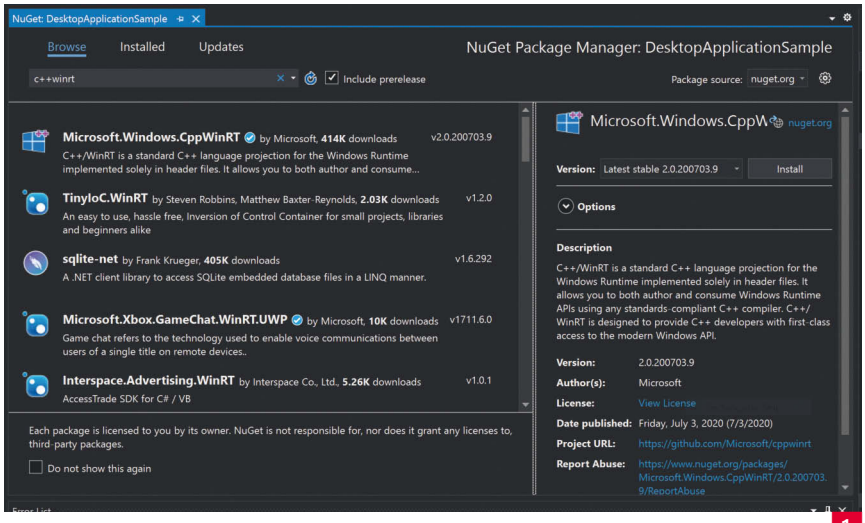
- Cliquer avec le bouton droit sur le projet et sélectionner de « Manage Nuget Packages… ». Chercher « C++ winrt » sur nuget.org et installer la version la plus récente de Microsoft.Windows.CppWinRT. 1
- L'affichage des propriétés du projet permet de vérifier que le langage de programmation est C++ 17.
- La même fenêtre permet de préciser quelle version du SDK de Windows 10 utiliser : le MSDN décrit quand les différents membres de chaque classe ont été ajoutés : si une méthode ou une propriété semble manquer dans Intellisense ou lors de la compilation, c'est généralement dû au référencement d'une version trop ancienne du SDK de Windows 10 ; on peut commencer par laisser l'option par défaut – « 10.0 (latest installed version) » –

et ne préciser un SDK particulier que si cela s'avère nécessaire (généralement, quand le projet est partagé entre différentes machines). 2

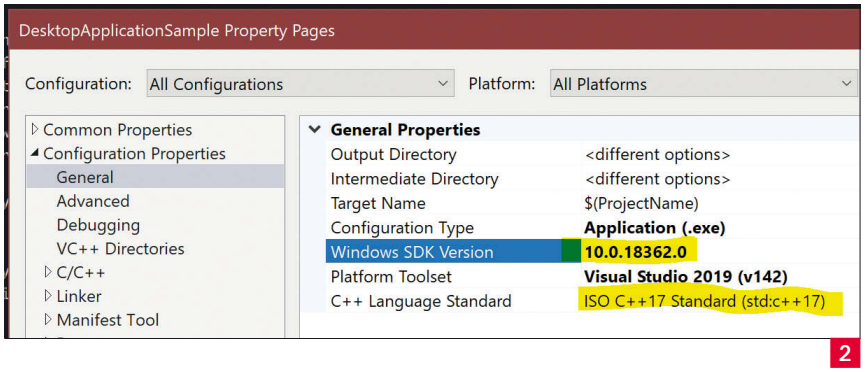
- En termes d'API Windows, LightSensor appartient à l'espace de noms « Windows.Devices.Sensors » : le fichier d'en-tête correspondant sera donc « winrt/Windows.Devices.Sensors.h ». Ces en-têtes sont générés à partir des fichiers .WINMD et respectent strictement cette correspondance entre l'espace de nom et le nom de l'en-tête.

On peut inclure ce fichier dans le .H des headers précompilés du projet – framework.h dans cet exemple – sans avoir besoin d'inclure des dépendances additionnelles. Néanmoins, si on utilise des opérations asynchrones, il faudra aussi inclure winrt/Windows.Foundation.h :

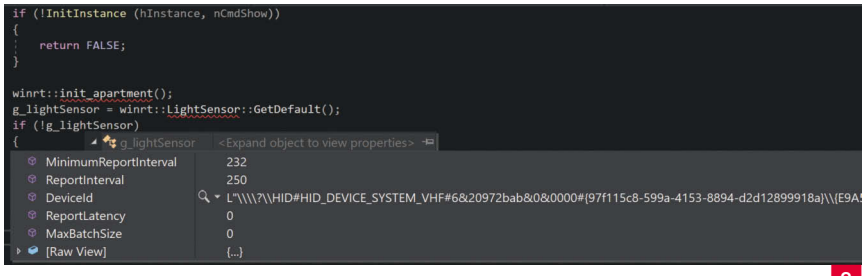
```
#pragma once
#include "targetver.h"
#define WIN32_LEAN_AND_MEAN // Exclude rarely-used stuff from Windows headers
```



1



2



3

```
// Windows Header Files
#include <windows.h>
// C RunTime Header Files
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>

#include <winrt/Windows.Devices.Sensors.h>
```

On peut maintenant déclarer un objet de type LightSensor. Les classes générées dans winrt/Windows.Device.Sensors.h appartiennent à l'espace de nom C++ winrt::Windows::Devices::Sensors : on peut utiliser le nom complet de la classe ou utiliser l'instruction « using namespace ». Dans cet article, nous utiliserons une option intermédiaire pour mettre en évidence les objets et fonctions de C++/WinRT :

```
namespace winrt
{
  using namespace Windows::Devices::Sensors;
}

winrt::LightSensor g_lightSensor{ nullptr };
```

À présent, on peut initialiser le support de WinRT et obtenir le capteur par défaut de la machine dans la fonction wWinmain. On notera au passage que l'absence de capteur de luminosité n'étant pas une erreur de programmation, LightSensor::GetDefault() ne déclenche pas d'exception et se contente de retourner une référence nulle.

```
winrt::init_apartment();

g_lightSensor = winrt::LightSensor::GetDefault();
if (!g_lightSensor)
{
  MessageBox(nullptr,
        L"Impossible d'adapter les couleurs à la luminosité ambiante",
        L"Pas de capteur de luminosité."
        MB_OK | MB_ICONERROR);
}
```

Exécuter ce code sous le débogueur permet de voir que celui-ci a été mis à jour pour être en mesure de découvrir et de lire les propriétés des références C++/WinRT : 3

Pour finir, il reste à appeler la méthode GetCurrentReading :

```
auto lightReading = g_lightSensor.GetCurrentReading();
if (lightReading != nullptr)
{
  float illuminance = lightReading.IlluminanceInLux();
  UpdateColorTheme(illuminance);
}
```

Et voilà une application classique qui a maintenant accès aux dernières API de Windows !

Super-smart pointers

Les classes C++ WinRT sont des « super-smart » pointers : un smart pointer habituel ne sait en réalité pas grand-chose de l'objet qu'il gère. Il ne s'intéresse qu'à la durée de vie de celui-ci (et c'est déjà bien pratique) et se contente de transmettre sans valeur ajoutée les appels effectués via l'opérateur ->. Les classes C++ WinRT sont générées à partir des métadonnées des classes qu'elles représentent. Elles peuvent de ce fait offrir bien plus de services : rassembler les méthodes de toutes les interfaces implémentées par l'objet WinRT, en gérer l'instanciation, etc. Examinons comment les différents types de membres des classes WinRT – ce qui est visible dans la documentation en ligne – se traduisent en C++.

Constructeur

Si la classe WinRT possède un constructeur, celui-ci se traduit automatiquement par un constructeur de même signature pour la classe C++ correspondante. Par exemple le Windows.UI.Popups. MessageDialog qui en possède deux (message ou message + titre) se construit naturellement dans le code :

```
MessageDialog dialog(L"Hello world !");
MessageDialog dialog2(L"Hello world !", L"C++/WinRT Rocks !");
```

Dans le cas où il y a un constructeur par défaut, il faut bien comprendre la différence entre ces classes et de simples smart pointeurs : ici, la simple déclaration d'une variable peut suffire à déclencher une instanciation :

```
Microsoft::WRL::ComPtr<IFoo> spFoo; // Pointeur nul vers une interface IFoo
std::shared_ptr<Bar> bar ;          // Pointeur nul vers une instance de Bar
winrt::Windows::Data::Json::JsonArray js; // Instancie une JsonArray
winrt::Windows::Data::Json::JsonArray js{nullptr}; // Référence nulle vers JsonArray
```

Le dernier exemple – initialisation à nullptr en utilisant un constructeur acceptant un unique paramètre de type std::nullptr_t – rappelle que la classe C++ WinRT reste une référence même si elle est suffisamment intelligente pour créer des instances. Les constructeurs et opérateurs de copie sont définis de manière à partager les références à la manière des smart pointers, ce qui autorise par exemple le stockage de ces objets dans les conteneurs C++ sans duplication coûteuse d'objet :

```
namespace winrt
{
   using namespace Windows::Foundation;
}

…
std::vector<winrt::Uri> trustedAddresses;
trustedAddresses.push_back(winrt::Uri(L"http://www.microsoft.com"));
trustedAddresses.push_back(winrt::Uri(L"http://www.programmez.com"));
```

Méthodes

Elles sont traduites naturellement en méthodes C++, y compris les méthodes statiques. Si une méthode retourne une autre classe WinRT, sa projection retournera la classe C++ correspondante. Si celle-ci appartient à un autre espace de noms, il faut penser à inclure manuellement le fichier d'en-tête correspondant sous peine de voir apparaître une erreur cryptique à la compilation :

```
auto buffer = winrt::CryptographicBuffer::DecodeFromHexString(L"30310AFF"); // OK
auto length = buffer.Length(); // Erreur

1>… foo.cpp(52,29): error C3779: 'winrt::impl::consume_Windows_Storage_Streams_
IBuffer<winrt::Windows::Storage::Streams::IBuffer>::Length': a function that returns 'auto'
 cannot be used before it is defined
1>… \Generated Files\winrt\impl\Windows.Storage.Streams.0.h(392): message : see declaration
of 'winrt::impl::consume_Windows_Storage_Streams_IBuffer<winrt::Windows::Storage:
:Streams::IBuffer>::Length'
```

Propriétés

Contrairement à l'API WinRT ou C++/CX, le C++ standard n'a pas la notion de propriété. Les propriétés des objets WinRT se traduisent donc par des méthodes en C++. C++ WinRT ne s'encombre pas avec des préfixes get/set bavards et exploite la surcharge des méthodes C++:

```
• Type NomDeLaPropriete() pour lire
• void NomDeLaPropriete(Type nouvelleValeur) pour écrire
```

Par exemple :

```
// Ecrit la propriété DefaultCommandIndex
dialog.DefaultCommandIndex(0);

// Lit la propriété DefaultCommandIndex
uint32_t index = dialog.DefaultCommandIndex();
```

Événements

Ici encore, C++ WinRT va à l'essentiel et définit les constructeurs et opérateurs de conversion permettant un code à la fois lisible et efficace ; les méthodes pour enregistrer et désenregistrer un handler d'événement portent le nom de l'événement et sont différenciées par leurs paramètres, par exemple pour traiter l'événement Added du DeviceWatcher à l'aide d'une expression lambda :

```
auto watcher = winrt::DeviceInformation::CreateWatcher(
                    winrt::DisplayMonitor::GetDeviceSelector());

// Enregistrement du handler
auto addedToken = watcher.Added(
    [](const winrt::DeviceWatcher&, const winrt::DeviceInformation& device) {
        …
});

// Désenregistrement
Watcher.Added(addedToken)
```

L'enregistrement de méthodes de classes pour traiter les événements est lui aussi très concis grâce à la définition des constructeurs appropriés pour les objets de type EventHandler :

```
void MainPage::Initialize()
{
  auto watcher = winrt::DeviceInformation::CreateWatcher(
                    winrt::DisplayMonitor::GetDeviceSelector());

  auto addedToken = watcher.Added({ this, &MainPage::OnDisplay
Added });
  …
}

void MainPage::OnDisplayAdded(const winrt::DeviceWatcher&,
  const winrt::DeviceInformation& device)
{

}
```

Enfin, il existe une troisième version de la méthode correspondant à un événement, qui prend comme premier paramètre winrt::auto_revoke et retourne un objet dont le destructeur effectuera le désenregistrement automatiquement :

```
// Mainpage.h
class MainPage {
…

winrt::Windows::Devices::Enumeration::DeviceWatcher::Added_revoker m_addedRevoker;

void OnDisplayAdded(
  const winrt::Windows::Devices::Enumeration::DeviceWatcher&,
  const winrt::Windows::Devices::Enumeration::DeviceInformation& device);
};
```

```
// MainPage.cpp
void MainPage::Initialize()
{
  m_addedRevoker = watcher.Added(
                winrt::auto_revoke, { this, &MainPage::OnDisplayAdded });
  …
}
```

Appels asynchrones

L'API WinRT contient de nombreuses méthodes asynchrones auxquelles il est difficile d'échapper (la norme était que toutes les méthodes effectuant un traitement potentiellement long comme un accès disque ou accès réseau devaient être définies ainsi afin d'éviter de bloquer l'interface graphique). Ces méthodes retournent des instances de IAsyncAction, IASyncOperation<T>, etc.
La méthode la plus simple pour attendre le résultat d'une opération asynchrone consiste à appeler get() sur celle-ci :

```
winrt::Uri image(L"ms-appx:///Assets/sample_image.jpg");
auto file = winrt::StorageFile::GetFileFromApplicationUriAsync(image).get();
```

Cependant, le code ci-dessus lève une exception s'il est exécuté sur le thread de l'IHM à cause du contrôle ci-dessous (dans Windows.Foundation.h) :

```
inline void check_sta_blocking_wait() noexcept
{
// Note: A blocking wait on the UI thread for an asynchronous operation can cause a deadlock.
//  See  https://docs.microsoft.com/windows/uwp/cpp-and-winrt-apis/concurrency#block-the-calling-thread
  WINRT_ASSERT(!is_sta_thread());
}
```

L'appel de get() est valide sur un thread d'arrière-plan. Pour attendre le résultat d'un appel asynchrone déclenché sur le thread de l'interface graphique, la solution recommandée consiste à s'appuyer sur les coroutines de C++ qui permettent d'interrompre le déroulement d'une méthode puis de le reprendre sur un autre thread, à la manière des async / await de C# :

- Déclarer la méthode comme retournant une winrt::Windows::Foundation::IAsyncAction (ou une interface similaire comme IAsyncOperation<T>) ;
- Utiliser co_await pour récupérer les résultats des appels asynchrones ;
- Si on doit mettre à jour l'interface utilisateur après un ou plusieurs appels asynchrones, il est nécessaire de revenir sur le thread de l'interface. Ceci s'effectue ainsi :
  - capturer le contexte d'exécution :

```
winrt::apartment_context ui_thread;
```

revenir sur ce thread :

```
co_await ui_thread;
```

Par exemple :

```
winrt::IAsyncAction MainPage::LoadImage()
{
  // Capturer le contexte du thread de l'IHM
  winrt::apartment_context ui_thread;

  // Charger l'image de manière asynchrone
  winrt::Uri image(L"ms-appx:///Assets/sample_image.jpg");
  winrt::StorageFile file =
      co_await winrt::StorageFile::GetFileFromApplicationUriAsync(image);

  // Revenir sur le thread de l'IHM
  co_await ui_thread;

  // Faire quelque chose avec le fichier obtenu
  MyTB().Text(file.Path());
}
```

Implémenter un composant WinRT

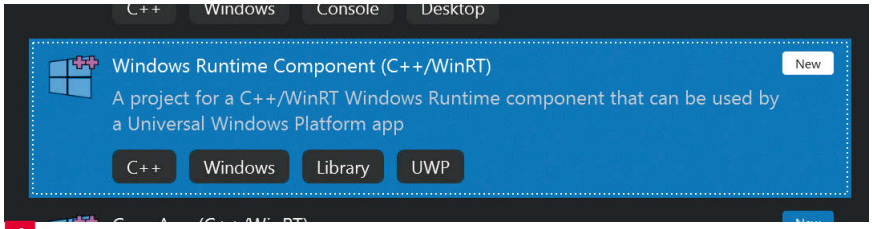
Après avoir regardé les principaux mécanismes de consommation de WinRT en C++, intéressons-nous maintenant à l'autre côté : l'écriture de tels composants. Pourquoi en avons-nous besoin ? La première utilisation sera l'implémentation d'un modèle MVVM pour une application XAML : à condition d'éviter les délires des chargements dynamiques et inversions de dépendances, le MVVM est une architecture saine qui permet réellement de finaliser une interface sans avoir à retoucher sans cesse au code métier. L'article « XAML

et C++/WinRT » montrera aussi que les pages et contrôles composés XAML sont aussi de tels composants. Une deuxième raison d'implémenter des composants WinRT est la réutilisabilité via des packages Nuget, grâce à leur interface « ABI-stable » indépendante du langage et de la version du compilateur utilisé.
L'équipe de C++ WinRT a donné la priorité à la consommation des composants plutôt qu'à leur implémentation : celle-ci demande quelques étapes supplémentaires, et en particulier l'utilisation du langage IDL pour la définition de l'interface des composants.
Pour faire une DLL réutilisable, le type de projet Visual Studio à choisir est « Windows Runtime Component » : 4
Le projet ainsi créé contient une classe dont la définition est basée sur trois fichiers source (ainsi que plusieurs autres fichiers générés) : class.idl contient la définition de l'interface publique du composant ;
• class.h et class.cpp en contiennent l'implémentation.
• L'IDL utilisé ici correspond à une nouvelle version spécifique à C++/WinRT. Il n'est plus nécessaire de définir séparément les interfaces et les classes comme avec le COM classique : la définition de la classe est suffisante. Après un renommage de bon aloi, on peut ajouter quelques propriétés et méthodes à notre composant dans le fichier .IDL :

```
namespace BTCube
{
  enum LedState { Off, Blinking, On };

  [default_interface]
  runtimeclass ConnectedCube
  {

    ConnectedCube(String bluetoothAddress);

    Windows.Foundation.IAsyncOperation<Boolean> TryConnectAsync();
    void Disconnect();

    Boolean IsSolved{ get; };
    LedState Leds{ get; set; };

    static String[] FindCubes();
  }
}
```

À partir de ces déclarations, Visual C++ génère plusieurs fichiers. La grosse différence entre C++ WinRT et l'implémentation classique d'un composant COM avec ATL ou WinRT avec WRL, c'est que la classe C++ que l'on écrit n'est pas celle qui est directement accédée depuis l'extérieur de la DLL (oubliées les méthodes retournant des HRESULTS) : C++ WinRT utilise abondamment les templates et les optimisations du compilateur pour autoriser une implémentation (presque) totalement libérée des tracasseries de bas niveau. Voici le .H correspondant à notre classe de Cube connecté :

```
#pragma once

#include "ConnectedCube.g.h"
```



4

```
namespace winrt::BTCube::implementation
{
  struct ConnectedCube : ConnectedCubeT<ConnectedCube>
  {
    ConnectedCube(const winrt::hstring& bluetoothAddress);


    winrt::Windows::Foundation::IAsyncOperation<bool> TryConnectAsync();
    void Disconnect();

    bool IsSolved();
    LedState Leds();
    void Leds(LedState value);

    static winrt::com_array<winrt::hstring> FindCubes()

  private:
…
  };
}

namespace winrt::BTCube::factory_implementation
{
  struct ConnectedCube : ConnectedCubeT<ConnectedCube, implementation::ConnectedCube>
  {
  };
}
```

Notre classe appartient à l'espace de noms "winrt::BTCube::implementation" alors que les types publics appartiennent simplement à "winrt::BTCube". On retrouve les correspondances entre les concepts de l'API et leur implémentation présentée précédemment (par exemple propriété = une ou deux méthodes surchargées).
Il faut bien saisir la différence entre l'interface publique du composant – définie dans le fichier .IDL – et la partie publique de cette classe C++, celle-ci correspond un peu au concept "Internal" de C# : elle peut contenir tout ce dont la DLL – qui contient généralement plusieurs classes – a besoin pour réaliser ses tâches. Par exemple, on a vu que certains objets de l'API WinRT n'avaient pas de constructeur public : on peut implémenter de tels composants en supprimant tout constructeur du fichier .IDL tout en conservant des constructeurs dans la **classe d'implémentation**.
Bien que ce soit du C++ standard, on voit apparaître dans la déclaration de ConnectedCube certains types de données qui sont propres à C++ WinRT comme les classes com_array ou hstring. C'est en fait totalement inévitable, car c'est l'ABI évoquée en début de cet article qui garantit la compatibilité entre des composants écrits dans différents langages (ou avec le même langage, mais dif-

Boutique Programmez!

Les anciens numéros de PROGRAMMEZ!

Le magazine des développeurs

226

231

236

238

239

240

241

HS 01 été 2020

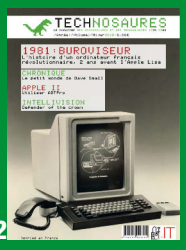
242

Tarif unitaire 6,5 € (frais postaux inclus)

Technosaures

Le magazine à remonter le temps !

N°1



N°2

N°3

N°4 Standard 10 €

N°4 Deluxe 15 €

Prix unitaire : **7,66 €** (frais postaux inclus)

Histoire de la micro-informatique 1973-2007

12,99 € (frais postaux inclus)

☐ 226 : |__| ex ☐ 240 : |__| ex
☐ 236 : |__| ex ☐ 241 : |__| ex
☐ 238 : |__| ex ☐ HS 01 : |__| ex
☐ 239 : |__| ex ☐ 242 : |__| ex

soit |__|__| exemplaires x 6,50 € = |__|__|__|__| €

Technosaures ☐ N°1 ☐ N°2 ☐ N°3
soit |__|__| exemplaires x 7,66 € = |__|__|__|__| €
☐ N°4 Deluxe ..... 15 €
☐ N°4 Standard ..... 10 €
☐ Histoire de la Micro-informatique ..... 12,99 €

..... |__|__|__|__| €

Commande à envoyer à :
Programmez!
57 rue de Gisors
95300 Pontoise

soit au **TOTAL** = |__|__|__|__| €

☐ M. ☐ Mme ☐ Mlle Entreprise : |__|__|__|__|__|__|__|__| Fonction : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Prénom : |__|__|__|__|__|__|__|__|__|__|__|__| Nom : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Adresse : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Code postal : |__|__|__|__|__| Ville : |__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|__|

Règlement par chèque à l'ordre de Programmez ! I Disponible sur www.programmez.com

Offres pouvant s'arrêter à tout moment, sans préavis | 1 ca 2Gésp R

férents compilateurs). La bonne nouvelle est que les classes intermédiaires de C++ WinRT ont été conçues pour être à la fois efficaces et totalement compatibles avec la STL : l'implémentation de ConnectedCube:: FindCubes reste donc naturelle pour un •e habitué•e du C++.

```
winrt::com_array<winrt::hstring> ConnectedCube::FindCubes()
{
  std::vector<winrt::hstring> cubes;

  cubes.push_back(L"cube #1");    // ou du vrai code ;-)
  …
  return winrt::com_array<winrt::hstring>(cubes);
}
```

L'article « XAML et C++ WinRT » contient d'autres exemples similaires, en particulier afin de permettre l'utilisation directe du composant WinRT dans un contexte de liaison de données (databinding).

Pour utiliser ce composant dans un autre projet de la même solution (le packaging Nuget est en dehors du périmètre de cet article), il suffit de l'ajouter aux références du projet consommateur. Dans celui-ci, le composant est vu comme n'importe quel autre composant C++ WinRT :

- Il faut inclure <winrt/BTCube.h> (si le client est une application XAML, il faut l'inclure dans le header précompilé),
- Puis le composant s'utilise comme décrits dans la première partie de cet article. Seule son interface publique, définie dans le fichier .IDL, est accessible.

Si on examine le package généré pour une application XAML, on y trouve la DLL du composant, son fichier winmd de métadonnées, et la déclaration des classes activables dans le manifeste du package :

```
<Extensions>
 <Extension Category="windows.activatableClass.inProcessServer">
  <InProcessServer>
   <Path>BTCube.dll</Path>
   <ActivatableClass ActivatableClassId="BTCube.ConnectedCube"
            ThreadingModel="both"/>
  </InProcessServer>
 </Extension>
</Extensions>
```

En conclusion

Grâce à la combinaison d'une simplicité du code comparable à celle du C# et d'un respect absolu des standards, le C++/WinRT rend le développement UWP accessible, à la fois dans les scénarios purement UWP ou dans les scénarios de modernisation d'applications existantes. Les performances n'ont pas été oubliées. C++/WinRT constitue un élément essentiel de la stratégie de Microsoft en termes d'API : les équipes internes l'ont déjà adopté depuis quelque temps (auparavant, les API était systématiquement implémentées via WRL et généralement consommées en C++/CX).

L'article « XAML et C++/WinRT » va capitaliser sur les bases présentées ici pour montrer comment implémenter un pattern classique – le MVVM – avec cette nouvelle technologie.

XAML et C++/WinRT

Après avoir exploré les bases de C++ WinRT, creusons le sujet en examinant comment implémenter les éléments courants d'une application XAML comme le modèle MVVM.

Pour créer un projet XAML en C++/WinRT, il faut commencer par télécharger les modèles de projets correspondants à l'adresse : https://docs.microsoft.com/en-us/windows/apps/desktop/visual-studio-templates, puis créer un projet en choisissant le modèle "Blank App (C++/WinRT)". La page principale du projet ainsi généré permet d'ores et déjà de noter quelques points importants :

- Toutes les pages XAML sont implémentées sous forme de composants C++/WinRT avec les trois fichiers vus précédemment: .IDL, .H et .CPP ;
- Les propriétés qu'on désire rendre accessibles au databinding doivent être déclarées dans le fichier .IDL ;
- Les éléments XAML nommés comme les boutons ne sont pas déclarés dans fichier .IDL, mais seront accessibles comme des propriétés dans le XAML et le code C++ : ::winrt::Windows::UI::Xaml::Controls::Button myButton() correspond à <Button x:Name="myButton"/>
- Les handlers d'événements ne sont pas déclarés dans le fichier .IDL, mais juste dans les .H et .CPP.

Liaison de données (databinding)

Les composants C++/WinRT sont compatibles avec {binding} et {x:bind}, cette deuxième forme étant recommandée pour des raisons de performance.

Notification de mise à jour

Un des avantages du databinding de XAML est la mise à jour automatique de l'interface lorsque les données changent, grâce à l'implémentation de INotifyPropertyChanged: lorsqu'un composant C++ WinRT implémente une interface standard, on déclare cette implémentation sans répéter les membres correspondants dans le fichier IDL, mais on doit évidemment les implémenter dans les fichiers .H et éventuellement .CPP :

CubeViewModel.idl

```
runtimeclass CubeViewModel : Windows.UI.Xaml.Data.INotifyPropertyChanged {…
```

CubeViewModel.h

```
struct CubeViewModel : CubeViewModelT<CubeViewModel>
{
…

 winrt::event_token PropertyChanged(
        Windows::UI::Xaml::Data::PropertyChangedEventHandler const& handler)
```

```
  {
    return m_propertyChanged.add(handler);
  }

  void PropertyChanged(winrt::event_token const& token) noexcept
  {
    m_propertyChanged.remove(token);
  }
…
private:
  winrt::event<Windows::UI::Xaml::Data::PropertyChangedEventHandler>
                                        m_propertyChanged;
```

CubeViewModel.cpp

```
// déclencher l'événement PropertyChanged
m_propertyChanged(*this,
      winrt::Windows::UI::Xaml::Data::PropertyChangedEventArgs(L"IsConnected"));
```

Propriétés de type liste

Pour publier une propriété de type liste, il faut (1) utiliser une des interfaces de Windows.Foundation.Collections et (2) s'appuyer sur une des classes génériques comprises dans C++/WinRT comme le winrt::single_threaded_observable_vector qui peut être initialisé à partir d'un vecteur standard. Voici un exemple de chargement d'une telle liste à partir d'un fichier .JSON :

CubeViewModel.idl

```
import "CubeSequence.idl";

..
runtimeclass CubeViewModel : Windows.UI.Xaml.Data.INotifyPropertyChanged
{
…

  Windows.Foundation.Collections.IObservableVector<CubeSequence> Sequences {get;};
  Windows.Foundation.IAsyncAction
            LoadSequencesAsync(Windows.Storage.StorageFile file);
…
}
```

CubeViewModel.h

```
struct CubeViewModel : CubeViewModelT<CubeViewModel>
{
…
  winrt::Windows::Foundation::Collections::IObservableVector
                                <winrt::CubeStudio::CubeSequence
    Sequences() const
  {
    return m_sequences;
  }

  winrt::Windows::Foundation::IAsyncAction
          LoadSequencesAsync(winrt::Windows::Storage::StorageFile const& file)
private:
  winrt::Windows::Foundation::Collections::IObservableVector
                    <winrt::CubeStudio::CubeSequence> m_sequences;
};
```

On notera au passage l'utilisation du namespace complet winrt::CubeStudio::CubeSequence, car CubeViewModel appartient au namespace winrt::CubeStudio::**implementation** qui possède aussi une clase CubeSequence.

CubeViewModel.cpp

Comme la méthode LoadSequencesAsync s'appuie sur le mécanisme des coroutines pour effectuer ses appels asynchrones, on mémorise le contexte d'exécution initial qui correspond au thread de l'interface afin de pouvoir rebasculer sur celui-ci au moment du déclenchement de l'événement PropertyChanged: en effet, le data-binding va mettre l'interface à jour immédiatement et ceci doit se passer sur ce thread.

```
winrt::IAsyncAction CubeViewModel::LoadSequencesAsync(winrt::StorageFile const& file)
{
  // Sauver le contexte d'exécution
  winrt::apartment_context ui_thread;

  winrt::hstring fileContent{ co_await winrt::FileIO::ReadTextAsync(file) };
  auto jsonRoot = winrt::JsonObject::Parse(fileContent);

  std::vector<winrt::CubeStudio::CubeSequence> sequences;

  auto jsonSequences = jsonRoot.GetNamedArray(L"Sequences");
  uint32_t const size = jsonSequences.Size();
  for (uint32_t i = 0; i < size; ++i)
  {
    auto jsonSequence = jsonSequences.GetObjectAt(i);
    auto description = jsonSequence.GetNamedString(L"Description");
    auto source = jsonSequence.GetNamedString(L"Source");
    auto target = jsonSequence.GetNamedString(L"Target");
    auto moves = jsonSequence.GetNamedString(L"Moves");
    sequences.push_back(winrt::make<CubeSequence>(description, source, target, moves));
  }

  m_sequences = winrt::single_threaded_observable_vector<winrt::CubeStudio::CubeSequence
>(std::move(sequences));
  // Revenir sur le thread UI pour déclencher l'événement
  co_await ui_thread;
  m_propertyChanged(*this, winrt::Windows::UI::Xaml::Data::PropertyChangedEventArgs(L
"Sequences"));
}
```

On remarquera au passage le std::move lors de l'initialisation du winrt::single_threaded_observable_vector: cette économie d'une copie des données illustre le constant objectif d'efficacité de l'équipe de C++/WinRT.

MainPage.xaml

```
<ListView Grid.Row="2" ItemsSource="{x:Bind VM.Sequences, Mode=OneWay}">
  <ListView.ItemTemplate>
    <DataTemplate x:DataType="local:CubeSequence">
      <StackPanel>
        <TextBlock FontWeight="Bold" Text="{x:Bind Description}"/>
        <TextBlock Margin="16,0,0,0" Text="{x:Bind Moves}"/>
      </StackPanel>
    </DataTemplate>
  </ListView.ItemTemplate>
</ListView>
```

Implémentation d'un convertisseur

Lier par exemple la visibilité d'un élément graphique à une valeur booléenne nécessite l'utilisation d'une classe implémentant l'interface IValueConverter :

```
<Page.Resources>
  <local:BoolToVisibilityConverter x:Key="b2v"/>
</Page.Resources>
…

<TextBlock Text="Connected"
        Visibility="{x:Bind VM.IsConnected,
                    Mode=OneWay, Converter={StaticResource b2v}}"/>
```

BoolToVisibilityConverter.idl

```
runtimeclass BoolToVisibilityConverter : Windows.UI.Xaml.Data.IValueConverter
{
  BoolToVisibilityConverter();
}
```

BoolToVisibilityConverter.h

```
struct BoolToVisibilityConverter:BoolToVisibilityConverterT<BoolToVisibilityConverter>
{
  BoolToVisibilityConverter() = default;

  winrt::Windows::Foundation::IInspectable Convert(
    winrt::Windows::Foundation::IInspectable const& object,
    winrt::Windows::UI::Xaml::Interop::TypeName const& targetType,
    winrt::Windows::Foundation::IInspectable const& parameter,
    const winrt::hstring& language);

  winrt::Windows::Foundation::IInspectable ConvertBack(
    winrt::Windows::Foundation::IInspectable const&,
    winrt::Windows::UI::Xaml::Interop::TypeName const&,
    winrt::Windows::Foundation::IInspectable const&,
    const winrt::hstring&)
  {
    throw_hresult(E_NOTIMPL);
  }
};
```

La méthode Convert prend en paramètre et retourne des objets de type winrt::Windows::Foundation::IInspectable, mais ce convertisseur ne manipule que des types simples : bool en entrée et Windows::UI::Xaml::Visibility en sortie. La conversion se fait via les fonctions génériques box_value et unbox_value :

BoolToVisibilityConverter.cpp

```
winrt::IInspectable BoolToVisibilityConverter::Convert(
    winrt::IInspectable const& object,
    winrt::TypeName const&,
    winrt::IInspectable const&,
    const winrt::hstring&)
{
 auto value = unbox_value<bool>(object);
 auto visibility = value ? winrt::Visibility::Visible : winrt::Visibility::Collapsed;
 return box_value(visibility);
}
```

Commandes

À titre personnel, je ne suis pas un grand fan de l'utilisation des commandes dans le modèle MVVM, car elles nécessitent souvent des mécanismes compliqués pour gérer les messages d'erreur ou les simples boîtes de dialogue de sélection de fichiers, mais ce tour d'horizon de l'implémentation d'un MVVM en C++/WinRT ne serait pas complet sans les mentionner. Là encore, il "suffit" d'implémenter une interface définie par XAML – Windows.UI.Xaml.Input.ICommand – ce qui ne pose pas de difficulté particulière.

CubeCommand.idl

```
runtimeclass CubeCommand : Windows.UI.Xaml.Input.ICommand
{
}
```

CubeCommand.h

```
struct CubeCommand : CubeCommandT<CubeCommand>
{
  // Interface privée : par exemple
  CubeCommand(std::function<void()> executeCallback, bool isEnabled);
  bool IsEnabled() const { return m_isEnabled; }
  void IsEnabled(bool newValue);

  // Implémentation de ICommand
  bool CanExecute(winrt::Windows::Foundation::IInspectable const&) const
  {
    return m_isEnabled;
  }
  void Execute(winrt::Windows::Foundation::IInspectable const&);

  winrt::event_token CanExecuteChanged(winrt::Windows::Foundation::EventHandler<winrt::
Windows::Foundation::IInspectable> const& handler);
  void CanExecuteChanged(winrt::event_token const& token) noexcept;

private:
…
};
```

Puis dans le ViewModel :

CubeViewModel.idl

```
runtimeclass CubeViewModel : Windows.UI.Xaml.Data.INotifyPropertyChanged
{
…
  Windows.UI.Xaml.Input.ICommand ConnectCommand{ get; };
…
```

CubeViewModel.h

```
struct CubeViewModel : CubeViewModelT<CubeViewModel>
…
 winrt::Windows::UI::Xaml::Input::ICommand ConnectCommand() const
   { return m_connectCommand; }
…
 void Connect();
```

CubeViewModelcpp

```
CubeViewModel::CubeViewModel()
  : m_connectCommand([this]() { Connect(); }, true)
{
}
```

Traitement des événements

Les méthodes de traitement d'événements générés par Visual Studio ont la signature correspondant au type Windows.UI.Xaml.RoutedEventHandler, par exemple :

```
<Button Content="Load Sequences" Click="LoadSequencesButton_Click"/>
```

Déclenche l'exécution de la méthode suivante :

```
Void MainPage::LoadSequencesButton_Click(
    winrt::Windows::Foundation::IInspectable const& sender,
    winrt::Windows::UI::Xaml::RoutedEventArgs const& e);
```

Si son implémentation nécessite d'appeler des méthodes asynchrones, il est possible de la transformer en coroutine C++ et de bénéficier de la simplicité apportée par co_await en changeant manuellement son type de retour dans les fichiers .H et .cpp correspondants. Par exemple :

winrt::Windows::Foundation::IAsyncAction

```
MainPage::LoadSequencesButton_Click(IInspectable const& , RoutedEventArgs const& )
{
 try
 {
  winrt::FileOpenPicker jsonOpenPicker;
  jsonOpenPicker.ViewMode(PickerViewMode::List);
  jsonOpenPicker.SuggestedStartLocation(PickerLocationId::DocumentsLibrary);
  jsonOpenPicker.FileTypeFilter().Append(L".json");

  StorageFile file{ co_await jsonOpenPicker.PickSingleFileAsync() };
  if (!file) co_return;

  co_await VM().LoadSequencesAsync(file);
 }
 catch (winrt::hresult_error& e)
 {
  winrt::MessageDialog dlg(e.message(), L"Impossible de charger les séquences.");
  dlg.ShowAsync();
 }
}
```

Accéder au type caché derrière IInspectable

Si le code de traitement d'un événement a besoin d'accéder au type réel de l'objet – typiquement Windows::UI::Xaml::Controls ::Button – il n'est pas possible de changer ce type dans la déclaration de la méthode comme on vient de le faire pour le type de retour, par exemple :

```
winrt::Windows::Foundation::IAsyncAction LoadSequencesButton_Click(
          winrt::Windows::UI::Xaml::Controls::Button const& sender,
        winrt::Windows::UI::Xaml::RoutedEventArgs const& e);
```

va déclencher une erreur de compilation avec le message "cannot convert from 'const winrt::Windows::Foundation::IInspectable' to 'const winrt::Windows::UI::Xaml::Controls::Button'".

De la même manière, l'accès à un dictionnaire de ressources XAML va aussi retourner un objet de type IInspectable :

```
auto key = box_value(L"MainVM");
winrt::IInspectable value = Application::Current().Resources().Lookup(key);
```

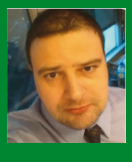
Pour accéder au type désiré – Button ou CubeViewModel dans ces exemples, les objets C++/WinRT disposent de la méthode générique as<> qui va déclencher une exception si l'objet n'a pas le type attendu, ou try_as<> qui retournera nullptr dans ce cas :

```
CubeViewModel mainvm = value.as<CubeViewModel>();
winrt::Button vm2 = value.try_as<winrt::Button>();
```

Conclusion

XAML UWP est disponible depuis plusieurs années, et ses avantages et inconvénients sont a priori bien connus. Les dernières évolutions des technologies de développement Windows lui donnent encore plus de portée avec les XAML Islands qui permettent d'intégrer dans des applications classiques des éléments d'interfaces XAML ou WinUI qui décuple les avancées en termes d'IHM et les versions de Windows. C++/WinRT complète ces avancées en rendant le développement d'applications XAML en C++ plus accessible, via les mécanismes du langage – par exemple les coroutines – mais aussi (et surtout?) par le respect du standard C++ qui facilite l'intégration de code portable.

John Thiriet
Lead Engineer at Edenred
Mobile enthusiast



Paulin Laroche
Mobile Tech Lead at Edenred
Xamarin Lover

Les nouveautés de Xamarin

Xamarin est bien vivant. Et si une année nous l'a prouvé, c'est bien 2020. Bien évidemment la situation sanitaire n'a pas facilité notre vie et parfois nous avons dû faire des choix qui nous ont peut-être amené à moins suivre l'actualité Xamarin. Heureusement, les équipes sont restées mobilisées. Nous vous proposons une petite session de rattrapage de Xamarin en l'an 2020. Avant de parler de Xamarin.Forms concentrons-nous un peu sur l'écosystème Xamarin qui lui sert de socle.

Android

Depuis presque deux ans, l'écosystème natif Android vit une petite révolution visant à « modulariser » d'avantage la plateforme, d'augmenter les performances tout en diminuant la taille des applications. Après un travail commencé en 2019, 2020 a été l'année où Xamarin a vraiment pu profiter complètement de toutes ces nouveautés.

En plus de nombreuses modifications permettant des gains sur les temps de compilation ou de déploiements et sur lesquelles nous n'allons pas rentrer dans les détails ici, voici une liste des points importants à retenir.

App Bundle

Android App Bundle est un nouveau format d'empaquetage d'application pour Android qui réduit considérablement la taille d'installation pour les utilisateurs finaux. Traditionnellement, lors de la construction d'une application Android, on crée un fichier APK qui est identique pour tous les utilisateurs téléchargeant l'application. Avec Android App Bundle, les applications sont livrées dynamiquement aux utilisateurs en fonction des spécificités de leur appareil. Cela signifie que l'application installée peut être réduite jusqu'à 50% sans aucun changement de code ! Après quelques déboires initiaux, le support de ce mode de publication par Xamarin a été finalisé en tout début d'année. BundleTool est l'outil permettant la génération de AppBundle. Il est inclus dans Xamarin et a été mis à jour en version 0.14.0 apportant ainsi des améliorations et des corrections de bogues.

Compression LZ4

Les assemblies managées sont maintenant compressées par défaut pour les builds de release, ce qui réduit considérablement les tailles des APK et Android App Bundle. Les assemblies sont compressées avec l'algorithme LZ4 pendant les builds, puis décompressées sur l'appareil pendant le démarrage de l'application.

Mono.Android et types de références nullables

L'assembly Mono.Android a été annotée pour être compatible avec les vérifications de type de référence nullable de C# 8.0. Les projets peuvent désormais utiliser du code provenant de Mono.Android dans un contexte nullable et tirer parti de vérifications de nullabilité supplémentaires du compilateur.

Amélioration de l'IntelliSense

Java.Lang.Object contient plusieurs propriétés et méthodes qui doivent être publiques pour prendre en charge les liaisons entre Java et Xamarin.Android mais qui ne sont pas destinées à être utilisées dans le code écrit à la main. Ces membres sont maintenant cachés d'IntelliSense, ce qui facilite la recherche des membres utiles pour les développeurs.

Support de Android 11

Au moment où ces lignes sont écrites Android 11 est disponible en version finale. Les bindings Xamarin Android sont prêts en bêta et devraient être finalisés dans peu de temps. Note intéressante, ces nouveaux bindings utilisent avantageusement le support des membres d'interface par défaut de C# 8.0 résultant en certaines classes marquées comme obsolètes.

Projets de bindings

Les projets de bindings (ou processus de liaison) font le pont entre le monde .NET et le monde natif. Ce sont donc eux qui nous permettent d'appeler des SDK natifs depuis un code C#.

Parmi les nouveautés intéressantes on note qu'il est désormais possible d'exposer des méthodes contenant plus de 14 paramètres.

Une petite amélioration permet également l'utilisation des éléments « mapping » et « field » sans préciser de type Java associé. Cela permet plus de flexibilité pour ajouter des énumérations ne correspondant à aucun type Java.

Finalement, et pour les plus anciens, on notera la dépréciation du parseur « jar2xml » et de la cible de génération de code « XamarinAndroid ».

Pour le premier il faut maintenant utiliser le parser « class-parse » :

```
<PropertyGroup>
 <AndroidClassParser>class-parse</AndroidClassParser>
</PropertyGroup>
```

Et pour le second il faut utiliser la cible « XAJavaInterop1 » :

```
<PropertyGroup>
 <AndroidCodegenTarget>XAJavaInterop1</AndroidCodegenTarget>
</PropertyGroup>
```

Kotlin bindings

L'usage de Kotlin gagne des parts de marché et remplacera peut-être même à terme Java. Il y a un certain nombre de SDK qui ont déjà été migrés de Java à Kotlin. La capacité de réutiliser les composants construits avec Kotlin est devenue de plus en plus impor-

tante pour les développeurs Xamarin à mesure que leur popularité continue de croître.
Depuis peu, il est donc possible de créer des projets de liaison de composants développés en Kotlin. Et même si le processus de liaison Kotlin est similaire à Java, il nécessite des étapes supplémentaires et des paramètres de configuration pour être construits et exécutés dans le cadre d'une application Xamarin.Android.

D8 et R8

Afin de suivre l'évolution des outils natifs, D8 et R8 ont été mis à jour en version 1.6.82. Ces deux outils fonctionnent de pair, le premier convertissant le byte code Java en code DEX, le second produisant un code DEX optimisé.
Google a déprécié le compilateur DX DEX en faveur du compilateur D8 DEX. Après le 1er février 2021, DX ne fera plus partie du SDK Android ou d'Android Studio. Les auteurs d'applications sont encouragés à migrer leurs projets vers D8 au plus tôt pour se préparer à ce changement.

```
<PropertyGroup>
 <AndroidDexTool>d8</AndroidDexTool>
</PropertyGroup>
```

R8 est le remplaçant de Proguard. Même si ce dernier a été mis à jour en version 6.2.2, les développeurs d'applications devraient considérer la migration vers R8 dès que possible.

Manifest merger

Votre projet ne contient peut-être qu'un seul fichier manifest mais vos dépendances peuvent en avoir également et parfois certains conflits surviennent, notamment au niveau des permissions. L'outil de fusion manifeste combine tous les éléments XML de chaque fichier en suivant certains heuristiques de fusion et en obéissant aux préférences de fusion que vous avez définies avec des attributs XML spéciaux. Il est dorénavant possible d'utiliser le « manifest merger » de Google pour mieux contrôler la fusion des différents manifest. Dans la dernière version de Xamarin Android l'outil est inclus dans sa version 27.0.0.

Startup tracing

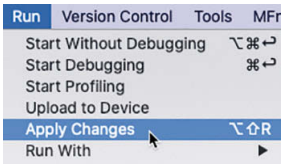
L'année dernière, Mono a introduit une fonctionnalité qui permet d'utiliser un profileur AOT intégré sur une application pour générer un profil AOT. Le profileur effectue le profilage de la mémoire, du temps d'exécution et même un échantillonnage statistique. Cela génère un profil AOT qui peut être utilisé pour optimiser votre application lors de l'utilisation de la fonctionnalité AOT de Mono avec un profil.
Il est maintenant possible de générer des profils personnalisés pour vos applications Android. Cela offre une expérience de démarrage optimale à vos utilisateurs car parfaitement adaptée à votre application, en contrepartie d'un impact minimal sur le poids de l'application.

AAPT2

AAPT2, ou Android Asset Packaging Tool, est un outil de génération utilisé pour compiler et emballer les ressources de votre application. AAPT2 analyse, indexe et compile les ressources dans un format binaire optimisé pour la plateforme Android.
Il n'est pas une nouveauté en soit et est disponible en Xamarin depuis très longtemps. Il a été défini comme valeur par défaut des nouveaux projets l'année dernière mais a été mis à jour en version 4.0.0 très récemment.
Pour l'activer il suffit d'écrire les lignes suivantes dans votre projet Android :

```
<PropertyGroup>
 <AndroidUseAapt2>d8</AndroidUseAapt2>
</PropertyGroup>
```

Android Apply Changes



Apply Changes vous permet de pousser des modifications de ressources sur votre application en cours d'exécution sans redémarrer votre application et, dans certains cas, sans redémarrer l'activité actuelle. C'est particulièrement utile pour les applications Xamarin Android utilisant des fichiers de layout XML ou AXML. Il ne faut pas confondre cette fonctionnalité avec le Hot Reload du XAML de Xamarin.Forms, ici c'est une fonctionnalité 100% native exposée maintenant aux applications Xamarin Android.

AndroidX et API 29

Parlons d'AndroidX. Ce sujet seul pouvant faire l'objet d'un article complet :

- Ce nouvel ensemble de bibliothèques remplace les bibliothèques de support Android avec lesquelles nous sommes tous familiarisés tout en étant iso fonctionnelles avec ces dernières ;
- Les intégrer demande pas mal de travail mais c'est indispensable au futur de votre application puisque les nouvelles fonctions de support ne sont fournies que dans AndroidX ;
- A partir de novembre 2020, toute application existante devra cibler les API 29 pour être publiées dans le Google Play Store. Certaines fonctionnalités requièrent AndroidX et c'est donc le moment idéal pour considérer une migration.

iOS

Comparativement à Android, peu de nouveautés importantes à signaler côté iOS cette année.

iOS 14

Au moment où ces lignes sont écrites, iOS 14 vient tout juste de sortir en version finale. Le support de iOS14 est donc en preview côté Xamarin. On peut s'attendre à un support rapide après la mise à disposition en version finale de Xcode 12.

UIWebView

L'API UIWebView est obsolète, il est encouragé à mettre à jour vers WKWebView pour une sécurité et une fiabilité améliorée. WKWebView garantit que le contenu Web compromis n'affecte pas le reste d'une application en limitant le traitement Web à la vue

Web de l'application. Et il est pris en charge sous iOS et macOS, et par mac Catalyst.
Si votre application est concernée et que vous utilisez correctement le linker à la compilation, il est possible d'ajouter cette option aux paramètres additionnels de mtouch.

```
--optimize=force-rejected-types-removal
```

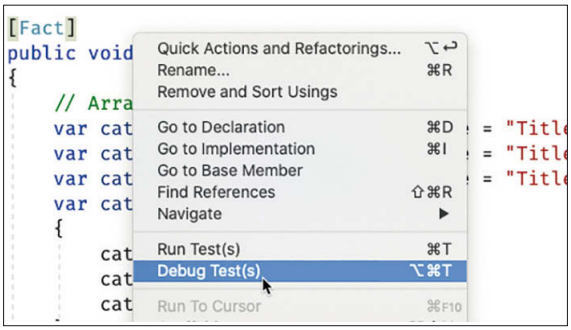
L'App Store n'accepte plus les nouvelles applications utilisant UIWebView depuis avril 2020 et les mises à jour d'applications utilisant UIWebView dès décembre 2020.

Swift bindings

La plateforme iOS est en constante évolution et l'usage de Swift désormais très important. De nombreux SDK très utilisés sont passés d'Objective-C à Swift. A partir du moment où une bibliothèque Swift propose des fichiers d'en-têtes Objective-C, il est possible de créer des projets de liaison et de les utiliser avec Xamarin iOS. Il ne faudra juste pas oublier de référencer le package Nuget « Xamarin.iOS.SwiftRuntimeSupport » mis à jour cette année.

Visual Studio

De nombreuses nouveautés ont été intégrées afin de supporter les fonctionnalités de Xamarin ou bien même de tout simplement améliorer notre productivité. Pour cet article nous avons choisi de nous concentrer sur les fonctionnalités ayant un impact direct pour le développeur Xamarin.



Visual Studio 2019 for Mac

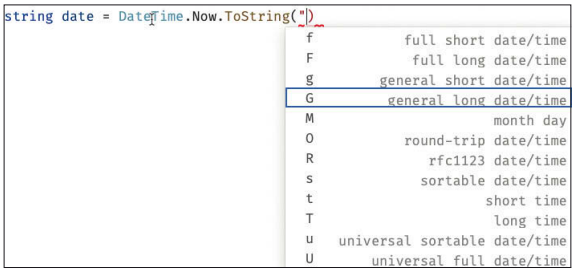
Travaillant sur Mac, Visual Studio 2019 for Mac est mon IDE principal et la moindre petite amélioration peux s'avérer très utile.

Tests unitaires

La possibilité de faire un clic droit sur un test unitaire pour le lancer et le déboguer est de cette veine. Enfin !

IntelliSense

Il est possible de générer automatiquement des types IComparable et IEquatable ou bien d'initialiser automatiquement des propriétés ou des champs pour les paramètres de constructeurs inutilisés. De même il est maintenant possible de générer des propriétés durant la génération d'un constructeur.
Finalement, on voit une amélioration de l'auto-complétion pour les types DateTime et TimeSpan.



Visual Studio 2019

Sans surprise, Visual Studio 2019 a reçu énormément de nouveautés et de corrections cette année.

Android

Pour simplifier la migration vers AndroidX, Visual Studio dispose maintenant d'un assistant à activer dans les options de l'éditeur.
Le support des fichiers XML Android s'est également amélioré. L'édition des XML Android est plus rapide et plus stable et permet l'auto-complétion d'attributs, de tags ou d'autres éléments de ces fichiers. Il est aussi maintenant possible de préciser un fichier de configuration pour le diagnostic de fichiers layout Android.

Xamarin.Forms

L'expérience de sélection du template Xamarin.Forms a été modifiée pour être plus visuelle. Les templates Flyout et Tabbed utilisent maintenant le Shell. Pour ceux ne souhaitant pas utiliser Shell, il faut maintenant utiliser le template Blank.
Parmi toutes les nouveautés de l'éditeur XAML. On note qu'il dispose maintenant d'une prévisualisation des couleurs et permet de visualiser la hiérarchie des contrôles Xamarin.Forms dans la fenêtre « document outline ».
Le Hot Reload s'est également trouvé amélioré. Il est possible de définir les propriétés de l'émulateur telles que les modes light/dark ou bien les tailles de polices en utilisant la fenêtre « Environment Settings » durant une session de XAML Hot Reload. Une nouvelle option permet de recharger automatiquement toutes les surfaces de design quand une modification est faite dans le code raccourcissant ainsi la boucle de feedback.

Xamarin.Forms

Xamarin.Forms 4.5+

De nombreux contrôles de base ont reçu des améliorations et optimisations. Nous verrons dans cet article une sélection des points que nous pensons importants.

AndroidX

Xamarin.Forms utilise désormais les dernières bibliothèques AndroidX de Google. Ceux-ci remplacent les bibliothèques de support Android qui ne sont plus mises à jour. Cela ne nécessite aucune modification de votre code.

Embedded Fonts

Désormais, pour utiliser des polices personnalisées, il vous suffit de les ajouter dans votre projet de code partagé Xamarin.Forms en tant que ressources incorporées et de les référencer, plutôt que de les ajouter à chacune des plateformes.

```
using Xamarin.Forms;
[assembly: ExportFont("CuteFont-Regular.ttf")]

<Label Text = "Hello Embedded Fonts" FontFamily = "CuteFont-Regular" />
```

VisualStateManager Target

Jusqu'à présent, le VisualStateManager interagissait uniquement avec les contrôles sur lesquels il était déclaré. L'ajout de l'attribut Target aux setters permet maintenant de modifier les propriétés de n'importe quel contrôle dans l'arborescence visuelle.

```
<BoxView x:Name="Screen" BackgroundColor="Black" Opacity="0.6"/>
<ImageButton x:Name="PlayPauseToggle"
    Source="{StaticResource PauseIcon}"
    Clicked="PlayPauseToggle_Clicked">
  <VisualStateManager.VisualStateGroups>
    <VisualStateGroup Name="PlaybackStates">
      <VisualState Name="paused">
        <VisualState.Setters>
          <Setter Property="Source" Value="{StaticResource PlayIcon}"/>
          <Setter TargetName="Screen" Property="BoxView.Opacity" Value="0.9"/>
        </VisualState.Setters>
      </VisualState>
      <VisualState Name="playing">
        <VisualState.Setters>
          <Setter Property="Source" Value="{StaticResource PauseIcon}"/>
          <Setter TargetName="Screen" Property="BoxView.Opacity" Value="0.6"/>
        </VisualState.Setters>
      </VisualState>
    </VisualStateGroup>
  </VisualStateManager.VisualStateGroups>
</ImageButton>


VisualStateManager.GoToState(PlayPauseToggle,
  (e.State == MediaElementState.Playing)
  ? "playing"
  : "paused");
```

Shell Modal

Que vous utilisiez des onglets ou un menu déroulant, Shell est le moyen le plus simple de créer votre application mobile multiplateforme. Une des fonctionnalité les plus intéressante de l'utilisation du Shell est la navigation basée sur URI. Désormais, il est possible d'afficher des pages modales en utilisant cette même méthode. La propriété attachée Shell.PresentationMode permet de définir si vous souhaitez présenter la page en tant que modale.

```
<ContentPage x:Class="RegistrationModal" Shell.PresentationMode="ModalAnimated">
  // your content
</ContentPage>
```

Étant donné que cette page ne fait pas partie de votre navigation principale, aucune route n'est définie dans AppShell.xaml. Pour accéder à cette page, enregistrez-lui un itinéraire qui pourra ensuite référencer votre appel GotoAsync.

```
Routing.RegisterRoutes("registration", typeof(RegistrationModal));
Shell.Current.GoToAsync("//login/registration");
```

PlatformSpecifics

Xamarin.Forms utilise les composants natifs de chaque plateforme. Un des avantages majeurs de l'utilisation de Xamarin.Forms est de pouvoir accéder facilement aux propriétés et méthodes de ces composants. Il est donc possible de s'adresser directement à un composant natif et agir sur celui-ci avec, par exemple, l'application d'un effet de flou sur iOS.

```
<StackLayout
  xmlns:iOSSpecific="clr-namespace:Xamarin.Forms.PlatformConfiguration.iOSSpecific;
assembly=Xamarin.Forms.Core">
  <Entry Text="HelloWorld" iOSSpecific:VisualElement.BlurEffect="Dark" />
</StackLayout>
```

CarouselView

C'est une vue permettant de présenter des données dans une mise en page déroulante, dans laquelle les utilisateurs peuvent glisser pour se déplacer dans une collection d'éléments. Par défaut, CarouselView affiche son élément horizontalement.
Tout d'abord, un seul élément sera affiché à l'écran. Avec des gestes de balayage, nous pouvons naviguer de l'avant et de l'arrière dans la collection d'éléments. L'apparence de chaque élément peut être définie dans la propriété ItemTemplate (de type DataTemplate).

IndicatorView

Avec IndicatorView, vous pouvez facilement montrer à l'utilisateur quelle partie du CarouselView est présente à l'écran.

```
<ContentPage ...>
  <AbsoluteLayout>
    <CarouselView ...>
      <!-- CarourselView Content -->
    </CarouselView>
    <IndicatorView
      x:Name="indicatorview"
      AbsoluteLayout.LayoutBounds="0.5,0.95,100,100"
      AbsoluteLayout.LayoutFlags="PositionProportional"
      IndicatorColor="LightGray"
      IndicatorSize="10"
      SelectedIndicatorColor="Black" />
  </AbsoluteLayout>
</ContentPage>
```

SwipeView

Révélez le contenu contextuel d'un élément lors du balayage dans des directions spécifiques en l'enveloppant avec un SwipeView.
SwipeView offre des propriétés permettant d'afficher certains éléments suite à un balayage dans un direction donnée : LeftItems, RightItems, TopItems et BottomItems.

```
<SwipeView>
  <SwipeView.LeftItems>
```

```
    <SwipeItems>
      <SwipeItem Text="Favorite"
            IconImageSource="favorite.png"
            BackgroundColor="LightGreen"
            Invoked="OnFavoriteSwipeItemInvoked" />
      <SwipeItem Text="Delete"
            IconImageSource="delete.png"
            BackgroundColor="LightPink"
            Invoked="OnDeleteSwipeItemInvoked" />
    </SwipeItems>
  </SwipeView.LeftItems>
  <!-- Content -->
</SwipeView>
```

StateTriggers

Les déclencheurs d'état sont un groupe spécialisé de déclencheurs qui définissent les conditions dans lesquelles un VisualState doit être appliqué.

Les déclencheurs d'état sont ajoutés à la collection StateTriggers d'un VisualState. Cette collection peut contenir un ou plusieurs déclencheurs. Un VisualState est appliqué lorsque des déclencheurs d'état de la collection sont actifs.

Les déclencheurs suivants sont disponibles :

- Déclencheur adaptatif - réagit aux changements de largeur et de hauteur d'une fenêtre d'application ;
- Compare Trigger - se produit lorsque deux valeurs sont comparées ;
- Device Trigger - se produit lors de l'exécution sur le périphérique spécifié ;
- Déclenchement d'orientation - se produit lorsque l'orientation de l'appareil change.

MediaElement

Le contrôle MediaElement fait également partie des nouveautés de cette version. C'est un contrôle de base pour la lecture audio et vidéo. Ce contrôle a été présenté dans l'application XamarinTV, lors de l'événement en direct Microsoft Surface Dual-Screen de Redmond. Le contrôle est extrêmement facile à utiliser. Il suffit de définir la source sur n'importe quel URI de média sécurisé.

Appareils double écrans

De nombreux appareils à 2 écrans font leur apparition sur le marché. Afin de gérer les 2 écrans, deux classes, TwoPaneView et DualScreenInfo, sont disponibles.

La classe TwoPaneView représente un conteneur avec deux vues qui dimensionnent et positionnent le contenu dans l'espace disponible, côte à côte ou de haut en bas.

```
<ContentPage
  xmlns:dualScreen="clr-namespace:Xamarin.Forms.DualScreen;assembly=Xamarin.Forms.DualScreen">
  <dualScreen:TwoPaneView>
    <dualScreen:TwoPaneView.Pane1>
      <!-- Content 1 -->
    </dualScreen:TwoPaneView.Pane1>
    <dualScreen:TwoPaneView.Pane2>
      <!-- Content 2 -->
    </dualScreen:TwoPaneView.Pane2>
  </dualScreen:TwoPaneView>
</ContentPage>
```

La classe DualScreenInfo vous permet de déterminer le volet où se trouve votre vue, sa taille, la position de l'appareil, l'angle de la charnière, etc.

Des propriétés sont disponibles afin de connaître notamment la position de la charnière sur l'écran, si l'appareil est en mode paysage ou non (les API natives ne signalent pas correctement l'orientation) ou bien si l'application est en mode Tall, Wide ou SinglePane.

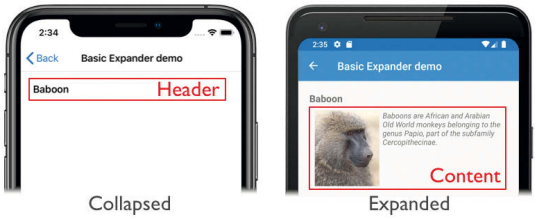
Sur Android et UWP, il est possible de récupérer l'angle actuel de la charnière de l'écran grâce à la méthode GetHingeAngleAsync.

```
var hingeAngle = await DualScreenInfo.Current.GetHingeAngleAsync();
```

Expander

Le contrôle Expander fait également partie des nouveautés de cette version. Il possède une en-tête et un contenu. Le contenu est affiché ou masqué en appuyant sur l'en-tête de l'Expander.

Les captures d'écrans montrent un Expander dans ses états réduit et développé, avec des zones rouges indiquant l'en-tête et le contenu.



Grid

Il n'est plus besoin de présenter la fameuse Grid de Xamarin.Forms. Historiquement, nous déclarions nos lignes et colonnes à l'aide des propriétés Grid.RowDefinitions ou Grid.ColumnDefinitions. Depuis Xamarin.Forms v4.7, les développeurs peuvent déclarer lignes et colonnes à l'aide d'une liste de valeurs séparées par des virgules.

```
<Grid VerticalOptions = "FillAndExpand"
   RowDefinitions = "1 *, Auto, 25, 14, 20"
   ColumnDefinitions = "*, 2 *, Auto, 300">
</Grid>
```

En utilisant le nouveau format, nous avons supprimé une quantité importante de XAML et l'ajout d'une autre ligne ou colonne sera également simplifié.

Shapes et Paths

La classe Path dérive de la classe Shape et peut être utilisée pour dessiner des courbes et des formes complexes décrites à l'aide d'objets de type Geometry.

Path définit les propriétés suivantes :

- Data, de type Geometry, qui spécifie la forme à dessiner ;
- RenderTransform, de type Transform, qui représente la transformation appliquée à la géométrie d'un tracé avant son dessin.

```
<Path Data="M 10,100 L 100,100 100,50Z"
   Stroke="Black"
   StrokeThickness="1"
   Aspect="Uniform"
   HorizontalOptions="Start" />
```

GestureRecognizers

Un « drag and drop » permet de faire glisser des éléments graphiques d'un emplacement de l'écran vers un autre emplacement à l'aide d'un geste continu.

Dans Xamarin.Forms, la reconnaissance des mouvements de glissement est fournie par la classe DragGestureRecognizer.

```
<Image Source="monkeyface.png">
  <Image.GestureRecognizers>
    <DragGestureRecognizer CanDrag="True" />
  </Image.GestureRecognizers>
</Image>
```

XAML Hot reload

Xamarin.Forms XAML Hot Reload nous permet d'apporter des modifications XAML pendant le déboggage et de voir ces modifications immédiatement reflétées dans notre application en cours d'exécution. XAML Hot Reload fonctionne sur les plateformes Android et iOS, mais n'est actuellement pas disponible pour les applications de la plateforme Windows universelle (UWP) ou les applications macOS.

Toutes les modifications apportées à XAML affecteront l'interface utilisateur, mais tous les autres états de l'application seront conservés. Cela inclut l'état de navigation et l'état en mémoire. En d'autres termes : nous pouvons itérer plus rapidement le développement des écrans sans avoir à relancer l'application à chaque modification !

Et la suite ?

Xamarin.Forms 5.0 est la dernière version majeure de Xamarin.Forms. Les nouvelles fonctionnalités majeures et le développement seront désormais disponibles dans .NET MAUI et la boîte à outils de la communauté Xamarin.

Bien évidemment, à nouvelle version de Xamarin, nouvelles fonctionnalités, parmi lesquelles on trouvera une bonne partie de ce que qui est aujourd'hui disponible en « preview ». Malheureusement parfois, des « breaking changes » sont aussi nécessaire pour garantir que Xamarin.Forms est dans un état maintenable et stable.

IDE supportés

Afin de tirer parti des nouvelles architectures processeurs et des correctifs de sécurité critiques dans les plateformes natives, l'équipe de Xamarin a pris la décision de ne plus supporter Visual Studio 2017. Dorénavant les développeurs devront utiliser Visual Studio 2019.

Composants abandonnés

Les DataPages ont été publiée en « préversion » il y a plusieurs années. De par leur faible utilisation, le choix a été fait de les retirer de Xamarin.Forms.

De la même manière les tâches XFCorePostProcessor permettaient de maintenir une compatibilité avec Xamarin.Forms 2.5. Il est bien évident qu'ils ne sont maintenant plus nécessaires et auront donc le même destin.

Modifications

L'équipe Xamarin a évalué l'usage et l'intérêt d'avoir certains contrôles intégrés à Xamarin.Forms. Suite à cela, plusieurs d'entre eux vont être renommés tels que MasterDetailPage ou bien déplacés vers le Xamarin Community Toolkit tels que MediaElement, MarkupExtensions et l'Expander.

Références

Pour plus d'informations sur toutes ces nouveautés et avoir la liste des références de cet article nous avons créé un dépôt GitHub. Pour y accéder il vous suffit de scanner le QR Code.

Julien Chomarat
Senior Software Engineer chez Microsoft. Développeur dans l'âme, j'aide les entreprises et les partenaires à s'approprier les services Azure à travers des projets innovants, en mode co-développement. **https://github.com/jchomarat**

Développer une application desktop pour Windows et macOS avec Xamarin.Forms

Aujourd'hui, le développement applicatif est à l'heure du web : une application, quelle qu'elle soit, est web car elle doit être disponible sur tous les périphériques avec un minimum d'effort, c'est-à-dire un minimum de code spécifique pour une plateforme et un maximum de code mutualisé.

Cependant, il reste des cas d'usages où une application « desktop », ou client lourd, est toujours utile. Mais dans ce cas, veut-on que cette application tourne aussi sur téléphone et ordinateur Windows/Mac ?

Il existe plusieurs façons de faire du développement d'application desktop multi-plateforme, comme *Electron* (et équivalent via « webview »), *Python* et ses librairies graphiques (*tkinter*, *Qt*, etc.). Je vais vous présenter ici l'approche .NET : *Xamarin*. *Xamarin* est « une plateforme open source qui permet de générer des applications modernes et performantes pour *iOS*, *Android* et *Windows* avec .NET. *Xamarin* est une couche d'abstraction qui gère la communication entre le code partagé et le code de plateforme sous-jacent. » (1). En revanche, il nous manque deux choses dans cette définition : le support de *macOS* et le support Linux !

Je vais vous présenter dans cet article toutes les étapes pour développer une application « desktop » qui tournera sous *Windows 10* et *macOS* avec environ 95% de code mutualisé. *Xamarin.Forms* permet aussi de cibler Linux via GTK# (en Preview) via une approche communautaire.

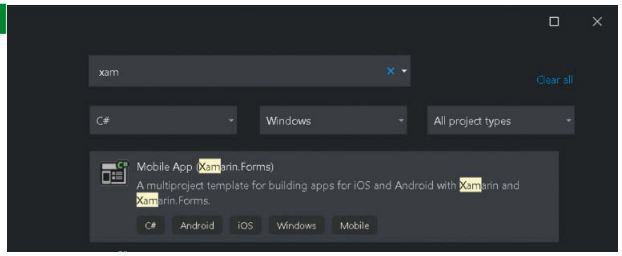
Notre cas d'usage : CrossPad

Restons simple : développons un « notepad » multi-plateforme (hors mobile) que nous appellerons CrossPad qui sera en *WPF* et *UWP* pour *Windows* et en natif sous *macOS*. Je laisse volontairement de côté les déclinaisons mobiles pour que nous nous concentrions sur les approches desktop.

Avant de débuter, arrêtons-nous sur les deux remarques suivantes :

- Cette application est disponible sur mon Github (2), mais les choses évoluent très rapidement en ce moment chez Microsoft. Il se peut donc que des éléments écrits ici ne soient plus entièrement corrects. Je tâcherai de tenir le projet Github à jour au gré des annonces et autres améliorations ;
- Nous ne pouvons pas générer notre application CrossPad pour *macOS* depuis une machine *Windows*. Nous devons par conséquent « avoir sous la main » un *Mac* pour le faire.

1



Prérequis

Avant toute chose, assurons nous d'avoir tous les prérequis sur nos deux machines Windows et Mac.

- Windows 10 : Nous allons utiliser Visual Studio 2019 (version 16.8.0 Preview 2.0 à date, mais la version stable marche aussi). Lors de l'installation (ou de la mise à jour) assurez-vous d'avoir les composants suivants :

- *Développement mobile avec .NET (Mobile development with .NET) ;*
- *Développement Universal Windows Platform.*

En ce qui concerne le runtime .NET, je suis aujourd'hui sur la version 5.0 Preview 8.

- Mac : Sur mon Mac (Catalina 10.15.5) j'utilise Visual Studio 2019 version 8.8 Preview ainsi que Xamarin.Mac 6.18.0.23.

Démarrons les développements sous Windows

Nous allons écrire l'essentiel de notre application sous *Windows* – une fois notre application fonctionnelle en *WPF* et en *UWP*, nous basculerons sur le *Mac* pour apporter les « petites » spécificités, et ainsi avoir aussi notre CrossPad sous *Mac*.

Pourquoi allons-nous faire deux versions pour Windows ? La première raison est bien entendu dans un but éducatif, pour voir les deux façons. La deuxième raison est que WPF tourne sur les versions antérieures de Windows, alors que *UWP* ne tourne que sous Windows 10. Et, enfin, l'approche *UWP* apporte une fraîcheur dans l'UI que nous verrons, même avec une simple application telle que CrossPad.

Création de la solution Visual Studio et des différents projets

Depuis Visual Studio, créons une nouvelle solution de type « Mobile App (Xamarin.Forms) » : 1

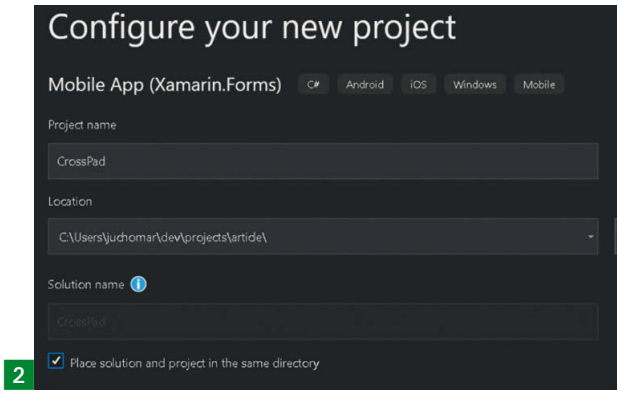
Donnons-lui le nom CrossPad, spécifions un emplacement (et assurons-nous que le fichier solution sera mis dans le même répertoire) : 2

Pour finir, dans le dernier écran, décochons les options iOS & Android car nous voulons seulement CrossPad sur Windows (et Mac un peu plus tard). 3

Assurons-nous que les paquets *NuGet* sont à jour (version 4.8.0.1269 pour Xamarin.Forms à date d'écriture) : clic-droit sur solution > Gestion des paquets *NuGet* > Mise à jour.

Nous nous retrouvons donc avec

- Un projet nommé « CrossPad » qui contiendra notre code de



2

base, mutualisé à toutes nos déclinaisons. Dans ce projet, *MainPage.xaml* sera bien notre interface principale – et c'est ce fichier que nous éditerons qui sera « injecté » dans les différentes « builds » *WPF*, *UWP* ou *macOS* (et bien entendu *iOS* & *Android* si notre application est mobile) ;

- Un projet nommé « CrossPad.UWP » qui représente la version UWP de notre application.

Vous l'aurez compris, l'idée est de toucher le moins possible aux déclinaisons et de mettre un maximum de code dans le projet principal CrossPad.

Avant d'aller plus loin, je vous propose un petit rappel sur les frameworks .NET (car oui, il faut suivre ...) 4

- .NET Framework c'est le framework historique, tournant sous Windows et dont la dernière version stable est la 4.8 ;
- .NET Core est le « nouveau » framework, multi-plateforme (sauf la partie WPF et Windows Forms qui ne tournent que sous Windows) ;
- Xamarin (mono) pour le développement d'application Windows & Mac et mobile (iOS & Android) ;
- .NET Standard : une libraire , type interface, pour homogénéifier tout cela et dont la dernière version est 2.1.

A la fin de l'année 2020, Microsoft va sortir le .NET 5, qui est en fait la fusion du .NET Core et du .NET Framework pour qu'il n'y en ait plus qu'un ! Vous pouvez à date utiliser la Preview 8 de .NET 5.

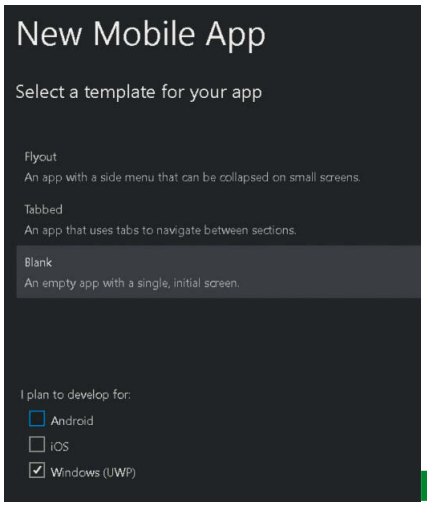
Revenons maintenant à CrossPad et inspectons la configuration du projet principal (CrossPad). Vous verrez qu'il cible « netstandard2.0 ». Cela est bien pour notre application *UWP* (qui à date ne supporte pas encore netstandard2.1, la dernière version). En revanche, pour notre version WPF, nous devons cibler la dernière version de .NET Core (version 3.1 ou *.Net 5.0 Preview 8* pour pouvoir utiliser la librairie *Xamarin.Forms.Platform.WPF)*. Editons donc le fichier « .csproj » du projet CrossPad (en mode code) et modifions le « TargetFramework » pour arriver à ça :

```
<PropertyGroup>
  <TargetFrameworks>netstandard2.0;net5.0</TargetFrameworks>
  <ProduceReferenceAssembly>true</ProduceReferenceAssembly>
</PropertyGroup>
```

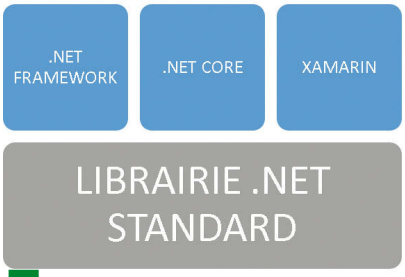
Notez bien que *TargetFramework* devient *TargetFrameworks* (avec le « s ») pour en mettre plusieurs.

Ajoutons un nouveau projet à notre solution de type « WPF (.NET Core) » nommé « CrossPad.WPF », dans le répertoire de celle-ci et référençons :

- Le paquet NuGet « Xamarin.Forms.Platform.WPF » dans sa dernière version (et qui doit être la même que Xamarin.Forms du projet CrossPad) ;
- Le projet CrossPad lui-même, car, c'est celui-ci qui aura le code mutualisé.



3



4

A la date d'écriture, *Xamarin.Forms 4.8.0.1269* comporte un petit bug. Ajoutons donc dans le fichier « .csproj » du projet CrossPad.WPF » le bloc suivant :

```
<ItemGroup>
  <EmbeddedResource Remove="**\*.xaml" />
</ItemGroup>
```

Pour finir, nous devons nous assurer que ce projet WPF recevra bien l'UI mutualisée. Apportons donc les modifications suivantes :

- Dans le fichier « *MainWindow.xaml* », Ajoutons le *namespace* :

```
xmlns:wpf="clr-namespace:Xamarin.Forms.Platform.WPF;assembly=Xamarin.Forms.Platform.WPF"
```

Changeons le tag global de « *Window* » vers « *wpf:FormsApplicationPage* »

- Dans le fichier « *MainWindow.xaml.cs* », Ajoutons les références vers *Xamarin.Forms* :

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.WPF;
```

Changeons l'héritage de la classe vers « *FormsApplicationPage* » Ajoutons les 2 lignes de code suivante dans le constructeur, après l'appel à « *InitializeComponent*() » :

```
Forms.Init();
LoadApplication(new CrossPad.App());
```

Lancement de nos deux version Windows

A ce stade, nous pouvons lancer les versions *UWP* et *WPF* de CrossPad avec l'UI du template Visual Studio. Si vous inspectez le code, vous vous rendrez compte que les deux fichiers xaml des deux projets WPF et UWP ne contiennent rien – hormis une référence vers le projet CrossPad. L'interface de notre application se trouve bien définie dans le fichier « *MainPage.xaml* » du projet CrossPad. D'ailleurs, ajoutez un label dans ce fichier et vous verrez en recompilant que les deux déclinaisons *WPF* et *UWP* sont mises à jour.

La structure finale de notre solution devient plus claire :

- Toute l'UI sera dans CrossPad/MainWindow.xaml (et autre views si nécessaire) ;
- Tout le code « métier » sera aussi dans CrossPad/MainWindow.xaml (ce que j'entends par code métier c'est bien la gestion des fichiers txt : ouverture, sauvegarde, gestion des évènements de l'UI (click sur les boutons, etc.)

Cependant, et c'est là que cette application est intéressante, nous allons rencontrer un problème !

Lorsqu'une application permet à un utilisateur de choisir un fichier sur son disque, ou permet de sélectionner un emplacement de sauvegarde, la fenêtre qui s'ouvre est propre au système d'exploitation et/ou à la technologie utilisée. Même sous Windows, je ne fais pas appel à cette fenêtre de la même façon en *WPF* ou *UWP*. Il est donc là le code spécifique que je vais devoir écrire pour chacune de mes déclinaisons. Voyons donc comment, grâce à une interface (au sens OO du terme), nous allons pouvoir isoler le code le plus spécifique depuis notre projet principal, et l'implémenter dans chacune de nos cibles.

Développement d'une interface de sélection de fichier

Pour définir le code le plus spécifique dont nous avons besoin, prenons le cas d'usage suivant :

En tant qu'utilisateur je veux pouvoir sélectionner un fichier sur mon disque dur pour l'ouvrir dans CrossPad.

Les actions qui découlent sont :

- Je clique sur un bouton dans l'interface pour sélectionner un fichier ;
- Je sélectionne un fichier sur mon disque dur depuis la fenêtre de sélection ;
- Le fichier est ouvert et lu via .NET ;
- Le texte du fichier apparaît dans mon application, dans un champ texte multilignes.

La seule action dépendante de la plateforme est la 2. En effet, ajouter un bouton sur l'UI avec une action sur le clic, on est bien sur le projet principal. Lire le contenu d'un fichier et l'afficher, idem. Nous devons donc écrire une interface qui renvoie le chemin vers le fichier sélectionné. Cette interface sera appelée depuis le code commun et implémentée dans chacune des déclinaisons, dans les contraintes définies. Si ce n'est pas clair, pas de soucis, le code le sera plus !

Interface IFileService

Dans le projet principal « CrossPad », ajoutons des dossiers Core/Services et ajoutons une nouvelle interface dans ce dernier dossier appelée « *IFileService* » :

```
using System.Threading.Tasks;
namespace CrossPad.Core.Services
{
  public interface IFileService
  {
    Task<string> GetFilePath(Tuple<string, string>[] AllowedTypes = null);

    Task<string> SetFilePath(Tuple<string, string>[] AllowedTypes = null);
  }
}
```

Cette interface, propose deux méthodes : une pour récupérer un chemin de fichier (GetFilePath) et une pour définir un chemin lors d'une sauvegarde. Ces deux méthodes prennent optionnellement un tableau de tuple pour filtrer les types de fichiers dans la fenêtre de sélection de fichier (description en item1 et extension en item2).

Implémentation de IFileService dans la version UWP

Dans le projet CrossPad.UWP, ajoutons un dossier « Custom » et ajoutons une nouvelle classe nommée « FilePickerService.cs » qui implémentera notre interface précédemment créée (pour des soucis de lisibilité, je ne vais montrer que l'implémentation de GetFilePath, tout le code étant bien entendu disponible sur GitHub (2)) :

```
../..
[assembly: Dependency(typeof(CrossPad.UWP.Custom.FileService))]
namespace CrossPad.UWP.Custom
{
  public class FileService : IFileService
  {
    public async Task<string> GetFilePath(Tuple<string, string>[] AllowedTypes = null)
    {
      var tcs = new TaskCompletionSource<string>();

      var picker = new Windows.Storage.Pickers.FileOpenPicker();
      picker.ViewMode = Windows.Storage.Pickers.PickerViewMode.List;
      picker.SuggestedStartLocation = Windows.Storage.Pickers.PickerLocationId.DocumentsLibrary;

      if (AllowedTypes != null && AllowedTypes.Length > 0)
      {
        AllowedTypes.ForEach(t =>
        {
          picker.FileTypeFilter.Add(t.Item2);
        });
      }

      Windows.Storage.StorageFile file = await picker.PickSingleFileAsync();

      if (file != null)
      {
        return file.Path;
      }
      else
      {
        return string.Empty;
      }
    }
../..
```

Dans cette implémentation nous ouvrons un FileOpenPicker, et si l'utilisateur sélectionne bien un fichier, son chemin d'accès est renvoyé (en passant par une Task pour que cela ne « freeze » pas l'UI pendant l'opération). Ce composant est bien propre à *UWP*.

Implémentation de IFileService dans la version WPF

Gardons le même principe que précédemment : ajoutons une nouvelle classe dans le dossier « Custom » avec l'implémentation suivante :

```
../..
[assembly: Dependency(typeof(CrossPad.WPF.Custom.FileService))]
namespace CrossPad.WPF.Custom
{
  public class FileService : IFileService
  {
    public Task<string> GetFilePath(Tuple<string, string>[] AllowedTypes = null)
    {
      var tcs = new TaskCompletionSource<string>();

      OpenFileDialog pickDialog = new OpenFileDialog();
      if (AllowedTypes != null && AllowedTypes.Length > 0)
      {
        pickDialog.Filter = string.Join("|", AllowedTypes.Select(t => $"{t.Item1}|*{t.Item2}")
.ToArray());
      }

      if (pickDialog.ShowDialog() == true)
      {
        tcs.SetResult(pickDialog.FileName);
      }
      else
      {
        tcs.SetResult(string.Empty);
      }

      return tcs.Task;
    }
../..
```

Nous avons grosso modo le même algorithme, seul l'appel à la fenêtre de sélection est différent (OpenFileDialog() pour *WPF*). Notons aussi au passage que la gestion des filtres de type de fichier à afficher n'est pas la même… pensons toujours à prévoir une interface **g-é-n-é-r-i-q-u-e** .

Appelons cette interface depuis le code commun

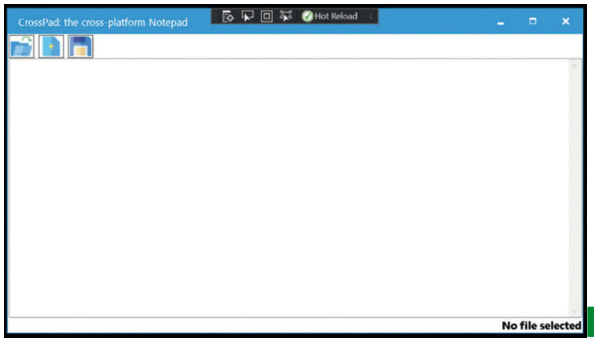
Il ne nous reste plus qu'à implémenter dans notre code commun l'UI de notre application et, sur le clic du bouton « ouvrir un fichier », faire appel à ce code spécifique.

Je passe également tous les détails de l'interface, vous trouverez tout sur le GitHub (2). Voici notre application en WPF : 5

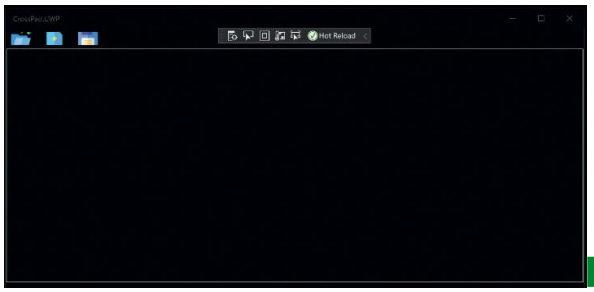
Et celle en UWP : 6

Pour gérer convenablement l'ouverture d'un fichier, le gestionnaire d'évènement du bouton « open » se trouve dans le fichier « *MainWindow.xaml.cs* » du project CrossPad ; et cette méthode est :

```
../..
private async void Open_Clicked(object sender, EventArgs e)
{
  // Call the Picker from the current platform implementation of the service
  string selectedFileName = await DependencyService.Get<IFileService>().GetFilePath(new
[] { new Tuple <string, string>("Text file", ".txt") });
  if (!string.IsNullOrEmpty(selectedFileName))
  {
    // Store the file name and path to save it later
```



5



6

```
    fileName = selectedFileName;

    // Open the file
    string fileContent = FilesHelper.ReadAll(fileName);

    // Update the UI
../..
```

La magie commence : « DependencyService.Get<IFileService> … » va aller chercher l'implémentation de l'interface IFileService pour le type d'application courante, sans que l'on fasse quoi que ce soit, et ça, c'est bien cool ! Nous n'avons pas besoin de faire des conditions pour savoir si nous sommes en *WPF*, *UWP* ou *macOS*. Lorsque l'application est générée par Visual Studio, toutes les « bonnes implémentations » sont injectées.

Et le Mac dans tout ça

Implémentons maintenant la version *macOS* de notre application. Comme je le signalais en introduction, nous sommes obligés de passer « sous mac ».

Partageons le code

Notre première étape est de transférer la solution Visual Studio sur notre mac… et que je ne vous voie pas prendre une clé USB ! Utilisons Github pour cela :

- Connectez-vous à Github (ou créez un compte, c'est gratuit) ;
- Ajouter un nouveau repo (privé ou public) ;
- Depuis votre machine *Windows*, et depuis le terminal tapez les commandes suivantes pour ajouter votre projet à un repo et le pousser.

```
~CrossPad> git init # initialise un repo git local
~CrossPad> git add . # ajoute tous les fichiers – pensez au .gitignore pour ne pas tout mettre
~CrossPad> git commit -m "Version initiale" # commiter les fichiers
~CrossPad> git remote add origin http://github.com/UserName/Repo # lier le repo local a
```

```
votre repo Github, pensez a mettre la bonne URL
~CrossPad> git push origin master # Pousser tout votre code local sur Github
```

Création du projet macOS

Nous pouvons maintenant basculer sur le Mac, récupérer notre solution (git clone…) et l'ouvrir dans Visual Studio for Mac (attention, si vous avez la double authentification sur Github, pensez à créer une clé d'application dans un premier temps, et à l'utiliser sur votre Mac). Tout comme nous l'avions fait pour la version *WPF*, nous devons ajouter un nouveau projet pour la version *Mac* de notre application. Pour cela, faisons un clic-droit sur notre solution et ajouter nouveau projet : 7

Appelons ce nouveau projet « CrossPad.MacOS », dans le répertoire de la solution globale. 8

Ajoutons le paquet NuGet *Xamarin.Forms* (dans la même version que pour les autres projets, version 4.8.0.1269 à la date d'écriture). Notre solution est maintenant prête. La dernière chose à faire, c'est finaliser la magie : faire en sorte que lorsque notre application *macOS* se compile, le code commun de CrossPad, contenant entre autres l'UI, soit injecté. Nous avons un peu de code à ajouter :

• Dans le fichier « *main.cs* », la méthode *main* doit être

```
static void Main(string[] args)
{
  NSApplication.Init();
  NSApplication.SharedApplication.Delegate = new AppDelegate();
  NSApplication.Main(args);
}
```

• Le fichier « *appdelegate.cs* » doit contenir les deux références suivantes :

```
using Xamarin.Forms;
using Xamarin.Forms.Platform.MacOS;
```

Et le code devient du coup :

```
public class AppDelegate : FormsApplicationDelegate
{
  NSWindow window;

  public AppDelegate()
  {
    var style = NSWindowStyle.Closable | NSWindowStyle.Resizable | NSWindowStyle.Titled;
    var rect = new CoreGraphics.CGRect(200, 1000, 1024, 768);
    window = new NSWindow(rect, style, NSBackingStore.Buffered, false);
    window.Title = "CrossPad: the cross-platform Notepad";

    window.TitleVisibility = NSWindowTitleVisibility.Visible;
  }

  public override NSWindow MainWindow
  {
    get { return window; }
  }

  public override void DidFinishLaunching(NSNotification notification)
  {
    Forms.Init();
    LoadApplication(new CrossPad.App());
    base.DidFinishLaunching(notification);
  }

  public override void WillTerminate(NSNotification notification)
  {
    // Insert code here to tear down your application
  }
}
```

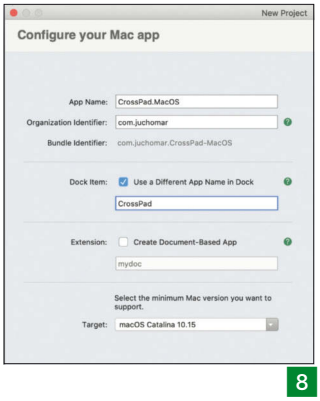
Le fichier « *main.storyboard* » quant à lui peut être effacé. Il ne nous servira pas.

Implémentation de IFileService

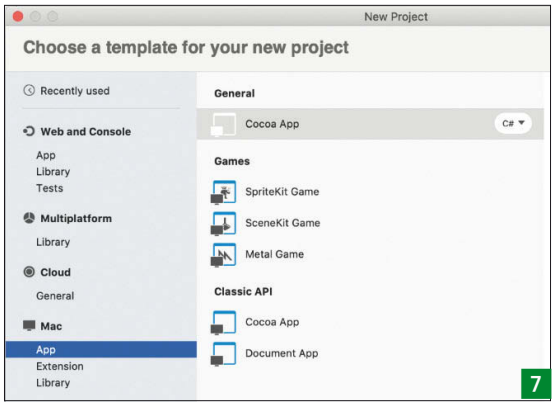
La dernière étape consiste à implémenter notre interface pour le code spécifique *macOS*.

Une fois de plus, je ne vais pas tout montrer ici, mais l'approche est identique à ce que nous avons fait sous *Windows*. Il faut juste savoir comment ouvrir un sélectionneur de fichiers sous *macOS* en .NET. Le reste du code permet de passer notre méthode en asynchrone pour ne pas geler l'UI et de manipuler les « AllowedTypes ».

```
../..
[assembly: Dependency(typeof(CrossPad.MacOS.Custom.FileService))]
namespace CrossPad.MacOS.Custom
{
  public class FileService : IFileService
  {
    public Task<string> GetFilePath(Tuple<string, string>[] AllowedTypes = null)
    {
      var tcs = new TaskCompletionSource<string>();

      var dlg = NSOpenPanel.OpenPanel;
      dlg.CanChooseFiles = true;
      dlg.CanChooseDirectories = false;
      dlg.AllowedFileTypes = AllowedTypes.Select(t => (t.Item2.StartsWith(".") ? t.Item2.
Substring(1) : t.Item2)).ToArray<string>();

      if (dlg.RunModal() == 1)
      {
        // Nab the first file
```



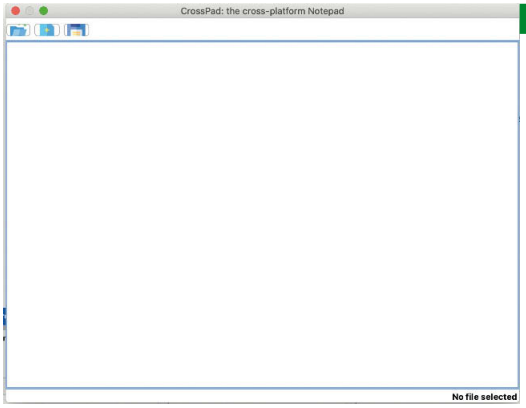
8



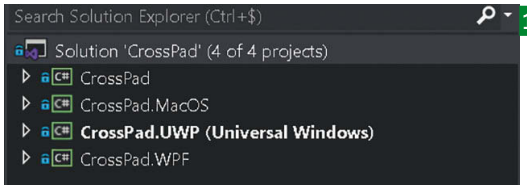
7

```
        var url = dlg.Urls[0];

        if (url != null)
        {
            tcs.SetResult(url.Path);
        }
      }
      else tcs.SetResult(string.Empty);

      return tcs.Task;
    }

../..
```

Tout est prêt ! Lançons maintenant CrossPad sous macOS. 9



9



10

En conclusion

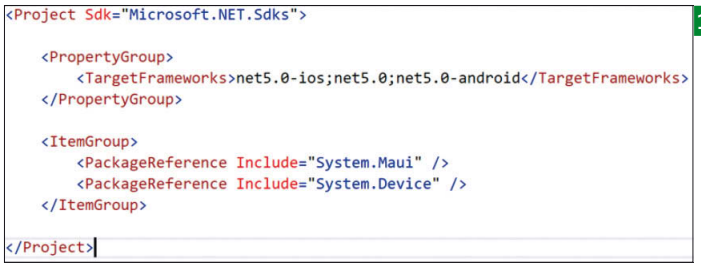
Ce que nous avons vu dans cet article c'est qu'il est possible de développer une application desktop tournant sous *Windows* (en *WPF* et/ou *UWP*) et sous *macOS* en .NET. 10

Nous trouvons dans notre solution Visual Studio un projet principal avec le code commun, le code métier et les interfaces que les déclinaisons devront implémenter et des projets par cible de plateforme. Xamarin nous permet de mettre en place simplement cette mécanique de gestion de code spécifique. L'idée étant bien entendu de minimiser cela pour simplifier le support ou les évolutions fonctionnelles de notre application.

Ok, mais c'est tout ?

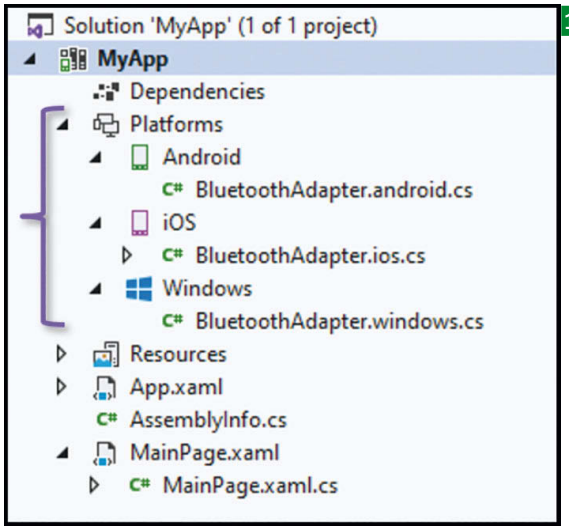
Je ne peux vous quitter sans vous parler du futur ! Durant la conférence **Build 2020** qui s'est tenue en Mai dernier, Microsoft a annoncé un nouveau projet : *.NET Maui*. *.NET Maui* rentre dans la stratégie du *One .NET*, initiative qui cherche à fusionner tout l'écosystème *.NET* en un seul et unique runtime (se référer plus haut dans l'article pour les différences entre ces frameworks).

Pour en revenir à *Maui*, qu'est-ce que c'est ? *.NET Maui* est l'évolution de *Xamarin.Forms* (qui vient de fêter ses 6 ans d'existence soit dit en passant) permettant de cibler depuis la même application pléthore de plateformes. La capture ci-contre, tirée du blog officiel .NET de Microsoft (4) résume bien cela : 11



11

On peut voir que le projet Visual Studio cible plusieurs plateformes lors de la compilation. La gestion du code spécifique (car ne nous leurrons pas, il y'en aura, mais comme précédemment lors du développement de CrossPad, l'idée est de le minimiser) sera beaucoup plus simple : 12



12

Comme nous pouvons le voir, nous avons un seul projet Visual Studio et le code spécifique (*Bluetooth adapter* dans cet exemple) directement dans le projet, par plateforme – pas besoin comme nous l'avons fait d'avoir un projet par plateforme dans notre solution Visual Studio.

A ce jour, Microsoft prévoit de supporter les mêmes plateformes, ce qui inclut donc Windows, Mac et Linux sur la partie Desktop – iOS & Android sur la partie mobile.

Tout ceci est attendu pour .NET 6 (5) – autour de Novembre 2021. Mais d'ici là, soyons assuré que Microsoft communiquera intensivement pour :

- Commencer à préparer les développeurs à migrer leur application Xamarin.Forms vers .NET Maui ;
- Nous permettre à nous développeurs de tester tout ça via des versions Preview/Beta.

Happy coding !

(1) https://docs.microsoft.com/fr-fr/xamarin/get-started/what-is-xamarin
(2) https://github.com/jchomarat/CrossPad
(3) https://docs.microsoft.com/fr-fr/dotnet/standard/net-standard
(4) https://devblogs.microsoft.com/dotnet/introducing-net-multi-platform-app-ui/
(5) https://github.com/dotnet/maui/wiki/Roadmap

Makram Jandar
Consultant Data AI
makram.jandar@infeeny.com

No Ops ?! Yes, Back2Bash…

Il y a de cela 10 ans, avec l'avènement des systèmes PaaS (Platform-as-a-Service), on pensait déjà que l'exploitation ne devrait plus être un souci. Mais malheureusement pour les louangeurs de ce néologisme qu'est le NoOps, les opérations subsistent, moins comme des opérations d'infrastructure, mais plus comme des opérations d'application. Quant à l'idée souvent fantasmée qu'on puisse tout orchestrer de façon scalable, agnostique, voire idempotente tout le cycle d'un projet, de l'approvisionnement de l'infra, jusqu'à la mise en prod, test, recettage et s'ensuit, pour peu qu'on sache la rationnaliser, le NoOps comme levier d'innovation peut s'avérer extrêmement puissant ! Peu y parviennent véritablement, non sans une remise à plat périlleuse de tout un pan technologique comme Snowflake, ou tel Microsoft par une profonde remise en question en s'ouvrant à l'univers linuxien (WSL, AZ Cli, AKS, etc.), ou encore Google en démocratisant l'IA…

Ne s'en tenir qu'à l'Infra, certes aujourd'hui les développeurs peuvent la décrire en full code, néanmoins ils doivent vraiment la superviser. Cela signifie qu'ils doivent s'occuper des questions de sécurité au niveau de la plateforme… S'occuper des trousseaux de clés que se trimballent les micro-services, leurs droits et privilèges, une bérézina consensuelle tant les protocoles d'authentification se rivalisent d'exotisme les uns les autres. Ceux-ci peuvent, en cas de mauvaise manip par *cascadologie* - puisqu'ils s'y prêtent - faire sauter le plus agile des backlogs. Ceci donc m'amène en toute logique à lui consacrer l'intégralité de ce premier article, en prélude à un deuxième qui suivra. Ceci posant ainsi les jalons d'une démarche NoOps allant de la conception à l'industrialisation de bout en bout d'une solution MLOps(1).

Mais avant tout, consolidons l'idée souvent galvaudée derrière ce concept qu'est donc l'IaC. En effet, derrière ce terme « Infrastructure en tant que code » se conçoit tout un éventail de techniques visant à ce que tout le déploiement de l'infrastructure soit automatisé, itératif, et managé par un "code" qui doit être stocké dans un contrôle de version. Voilà, c'est quick, très Cartésien !

On souhaite ainsi éviter que les process manuels sujets aux bugs aient une quelconque implication dans le déploiement des applis, avec la garantie qu'en cas d'imprévus, si toute notre infra était à reconstruire, nous pourrions d'un click — « *très important* ça » — recréer tout le schmilblick à l'identique. Et de deux, l'IaC où qu'il se trouve n'obéit à aucun format précis de fichier dans lequel notre infrastructure doit être définie. Certes, JSON & YAML, en sont les Fast & Furious, mais ça reste du déclaratif, ou pour le moins du conventionnel. De plus, il n'est aucune raison manifeste pour discréditer PowerShell ou Bash en tant q'IaC à part entière.

De ce point de vue, il n'y pas de différence, si ce n'est *the Same Old Song*, comme quoi les procédés déclaratifs pour la définition de l'infrastructure sont meilleurs que les impératifs.

Nous avons d'un côté, JSON qui ne contient aucune logique - il exprime sans façon toutes les "ressources" qui forment l'infrastructure et leur configuration. Et puis de l'autre, si nous choisissons d'écrire un script à l'aide de la CLI Azure, alors il est intrinsèquement impératif - il décrit les étapes indispensables pour provisionner l'infrastructure.

Mais est-ce un frein pour autant ?! Absolument pas, et même que les qualité d'un script Bash l'emportent largement. D'autant que combiné avec des Templates YAML (2) qui nous permettent de définir le contenu, la logique et les boucles itératives, il n'est qu'un pas à franchir pour se hisser au niveau des outils experts et même les dépasser. D'ailleurs, s'il est une interface qui crée le consensus parmi la majorité des acteurs du cloud, c'est donc bien le CLI.

On peut citer parmi ses avantages et de loin exhaustivement :

- Meilleure maintenabilité ;
- Facile à lire ;
- Facile et beaucoup plus rapide à créer ;
- Plus de contrôle ;
- Une littérature monstrueuse sur le sujet « tous les barbus de l'âge d'or de la scriptologie shamanique s'y sont frottés »

Rentrons dans le vif…

PREREQUIS

- Un compte AZURE gratuit : https://azure.microsoft.com/fr-fr/free/ ;
- Un compte Azure DevOps gratuit : https://azure.microsoft.com/fr-fr/services/devops/ ;
- Des connaissances en programmation, d'un langage de scripts, GIT ou similaire, et d'une super bonne dose de patience suffisent à mon avis !!

Nous utiliserons Azure DevOps pour faire fonctionner le pipeline en plusieurs étapes (stages) : build, model training, service de scoring, etc. Si vous n'avez pas encore d'organisation Azure DevOps, créez-en une en suivant les instructions du Quickstart (3).

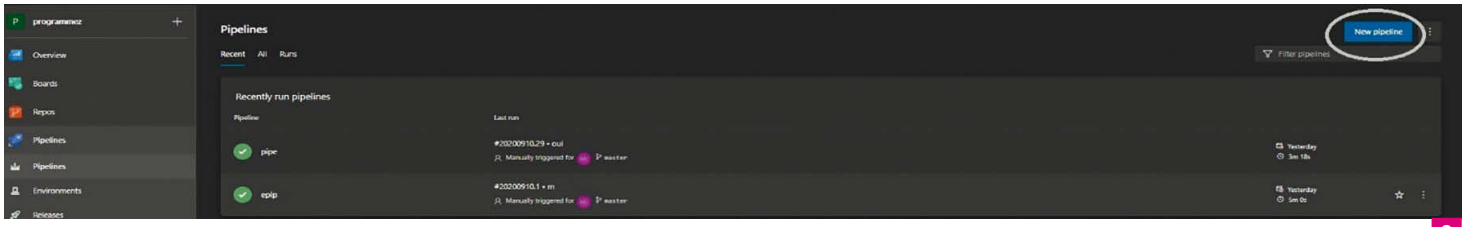
Si vous avez déjà une organisation Azure DevOps, créez un nouveau projet en utilisant le guide (4).

(1) Un ensemble de techniques qui permet aux scientifiques et aux développeurs d'applications de contribuer à la mise en production des modèles de Machine Learning, et où il est question de suivre, versionner, auditer, certifier et réutiliser chaque actif du cycle de vie ML tout en fournissant des services d'orchestration pour rationaliser sa gestion.

(2) https://docs.microsoft.com/fr-fr/azure/devops/pipelines/process/templates

(3) https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/create-organization

(4) https://docs.microsoft.com/en-us/azure/devops/organizations/projects/create-project



Création du PAT. 2

Création du service de connexion

Le pipeline d'approvisionnement d'IaC nécessite une connexion au service Azure Resource Manager 1

Nommez-le ARM, si autre nom, dans ce cas modifier la valeur de la clé SVCS :AZURE_RM_SVC_CONNECTION : **ARM** de la template jobs.yml.

Laissez le champ Groupe de ressources vide.

La création du périmètre du service de connexion de l'Azure Resource Manager nécessite des autorisations « Owner » ou « User Access Administrator » sur l'abonnement. Vous aurez également besoin d'autorisations suffisantes pour enregistrer une demande auprès de votre Azure AD tenant, ou vous pouvez obtenir l'ID et le secret du Service Principal auprès de votre administrateur Azure AD. Ce Principal doit avoir les autorisations "Contributeur" sur l'abonnement.

IaC PIPELINING

A la racine de notre projet nous créerons 3 fichiers jobs.yml, pipe.yml et epip.yml (code fourni en fin d'article).

Dans Azure DevOps nous créerons deux pipelines le premier en attachant le fichier **pipe.yml** pour la création des ressources et le deuxième un **epip.yml** pour leur destruction. 2

Ensuite le jeton d'accès personnel (PAT) (5) que nous utiliserons comme mot de passe alternatif pour s'authentifier dans Azure DevOps.

Après création du PAT, copier le contenu :

- Allez à la page Pipelines de votre service web Azure DevOps, sélectionnez le pipeline approprié (pipe.yaml dans notre cas après copie des codes), puis sélectionnez Modifier ;
- Créer une variable nommée MYPAT, puis collez-y votre PAT ;
- Sélectionnez l'icône Verrouillage secret pour stocker la variable de manière cryptée.
- Sauvegardez le pipeline.

Important : pensez à décommenter la valeur **#jobs.yml** du nœud **paths.include** du fichier **pipe.yml** pour enclencher le build automatique lors de chaque commit sur la branche master.

Amusez-vous aussi à rajouter des listes que ce soit de ressources, d'envs, vers, etc., à condition de bien respecter les conventions de nommage, et, pourquoi pas, créer les vôtres.

Et voilà, la seule limite c'est votre imagination…

REVERSE ENGINEERING

Notre unique template **jobs.yml** est constitué de deux blocs :

Les parameters :

Ils sont le cerveau du Pipeline, le catalogue des objets à industrialiser ainsi que la définition des variables paramétriques d'instanciation : Organisations (ORGS), Environnements (ENVS), Régions (LOCS), Versions (VERS) et en dernier, les ressources à provisionner (AZRS), à mettre à jour ou à supprimer.

Création du service de connexion. 1

Notons que le rajout d'une liste de définition sous l'un objets racines multipliera automatiquement les jobs de provisionnement des ressources, que ce soit dans leurs environnements respectifs (Dev, Prod, et s'ensuit), ou pour gérer le versionning en mode Spin-off, etc.

Prenons à titre d'exemple le tableau associatif des environnements (ENVS) dont la composition est la suivante :

ENVS : Tableau associatif des environnements contenant 2 envs : Dev et Prod

- TYPE: ENV « Convention de nommage - 3 alphanumériques »
UNIC: dev1 « Code environnement unique - 4 alphanumériques »
DON: " " « dependsOn: indexation de la dépendance dérivée de l'objet supérieur. Et c'est ce qui nous permet aussi de chaîner l'enclenchement les Jobs les uns à la suite des autres. Sinon, il est laissé vide en cas de non-dépendance et on peut alors opter pour la parallélisation pour l'ordonnancement de la création des ressources. »
NAME: DEV Et enfin le nom de l'environnement (choix libre)
- TYPE: ENV
UNIC: prd2
DON: dev1 « Dépend de dev1 »
NAME: PRD

Poursuivons avec le tableau associatif des ressources Azure :

AZRS: Il contient, comme son nom le laisse deviner, les listes des ressources Azure à créer/màj…

- ID: '2' « Numéro d'ordre de la ressource relativement aux différents environnements, versions et s'ensuit »
TYPE: STORAGE_ACCOUNT « Type de la ressource »

(5) https://docs.microsoft.com/en-us/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate

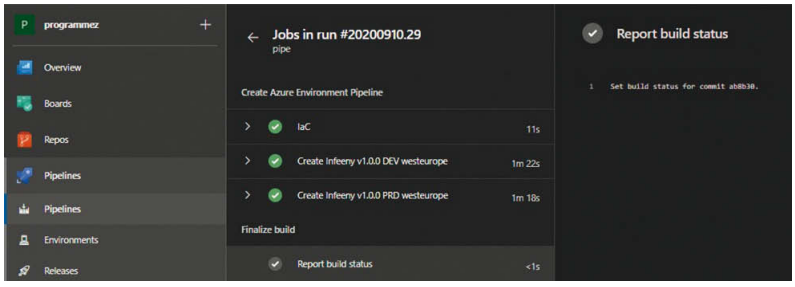
```
  UNIC: sa1 « Codification de la ressource avec une convention de nommage sur deux lettres plus numéro d'ordre »
  CREATE: >- « Commande de création AZ CLI »
    "az storage account create -n $AZR_NAME -g $RESOURCE_GROUP1 -l $AZR_LOC
    --sku Standard_LRS --kind StorageV2 --encryption-services blob file
    --https-only true --tags ${AZR_TAGS}"
  UPDATE: >- « Commande de mise à jour AZ CLI »
    "az storage account update -n $AZR_NAME -g $RESOURCE_GROUP1
    --encryption-key-source Microsoft.Storage"
```

Les Jobs :
Ils sont les orchestrateurs de la logique algorithmique en charge de l'ordonnancement de la création de ressources en bouclant « ${{ each … }} » sur la liste des paramètres les définissant. Le tout avec une gestion des dépendances imbriquées « dependsOn », d'abord le versioning, puis les environnements et ainsi de suite…

```
jobs:
- job: IaC
 steps:
 - task: AzureCLI@2 [6]
   displayName: Conf. & Setup
   inputs:
    azureSubscription: ${{ parameters.SVCS.AZURE_RM_SVC_CONNECTION }} [7]
    scriptType: "bash"
    scriptLocation: inlineScript
    inlineScript: |
     # Providers Registration : Machine Learning & App Insights [8]
     echo Microsoft.MachineLearning microsoft.insights | xargs -n 1 az provider register -n
   env:
    PAT: $(MYPAT)
- ${{ each ORG in parameters.ORGS }}:
 - ${{ each VER in parameters.VERS }}:
  - ${{ each ENV in parameters.ENVS }}:
   - ${{ each LOC in parameters.LOCS }}:
    - job: ${{ format('{0}{1}{2}{3}', ORG.UNIC, VER.UNIC, ENV.UNIC, LOC.UNIC) }}
      displayName: ${{ format('{0} {1} {2} {3} {4}', parameters.PIPE, ORG.NAME, VER.NAME, ENV
.NAME, LOC.NAME) }}
      dependsOn:
      - IaC
      - ${{ if ne(ENV.DON, '') }}:
       - ${{ format('{0}{1}{2}{3}', ORG.UNIC, VER.UNIC, ENV.DON, LOC.UNIC) }}
      - ${{ if ne(VER.DON, '') }}:
       - ${{ format('{0}{1}{2}{3}', ORG.UNIC, VER.DON, ENV.UNIC, LOC.UNIC) }}
```

3

Pipeline de déploiement des Jobs Azure



3

La tâche Azure CLI

Et enfin le cœur de l'engin, commenté, extensible, perfectible, prêt à bouffer de l'asphalte à cœur joie ;)

>> Déclaration des variables paramétriques :

```
AZURE_RM_SVC_CONNECTION=${{ parameters.SVCS.AZURE_RM_SVC_CONNECTION }}
AZR_LOC=${{ LOC.NAME }}
AZR_TYPE=${{ AZR.TYPE }}
AZR_TAGS="${{ format('ORG={0} VER={1} ENV={2} LOC={3} PID={4} NUM={5}', ORG.NAME,
 VER.NAME, ENV.NAME, LOC.NAME, AZR.ID, AZR.UNIC) }}"
AZR_BASE=${{ ORG.UNIC }}${{ VER.UNIC }}${{ ENV.UNIC }}${{ LOC.UNIC }}az${{ AZR.ID }}
AZR_UPDATE=${{ AZR.UPDATE }}
```

>> Authentification et création de groupe de variables :(9)
Le groupe de variables sert à stocker les valeurs que vous souhaitez contrôler et les rendre disponibles sur plusieurs pipelines. Le cas présent, nous y stockerons les variables nécessaires pour déployer le Build et la Release de notre IA. L'authentification exige un Personal Access Token (PAT) que nous avons créé (10) au préalable sur le portail DevOps de votre organisation https://dev.azure.com/**votreOrganisation**/

```
if [[ $AZR_TYPE == VARIABLE_GROUP ]]; then
 # Export Personal Access Token for Azure DevOps Services Signing
 export AZURE_DEVOPS_EXT_PAT=$MYSECRETPAT
 # Configure the Azure DevOps CLI
 az devops configure -d organization=${{ parameters.MAIN.ORG_URL }} project=${{ parameters.
MAIN.PROJECT }}
Fi
```

>> Création/MàJ de la ressource Azure :
Ici nous créerons notre ressource et stockerons le résultat de l'appel dans un fichier json

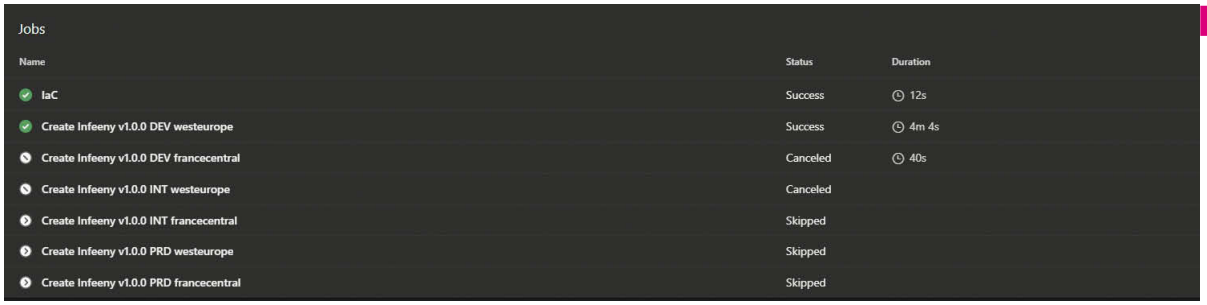
```
AZR_NAME=${AZR_BASE}${{ AZR.UNIC }}
$(echo ${{ AZR.CREATE }}) > ${AZR_NAME}Info.json
[[ ! -z $AZR_UPDATE ]] && $(echo ${{ AZR.UPDATE }}) > ${AZR_NAME}Info.json
```

>> Nettoyage :

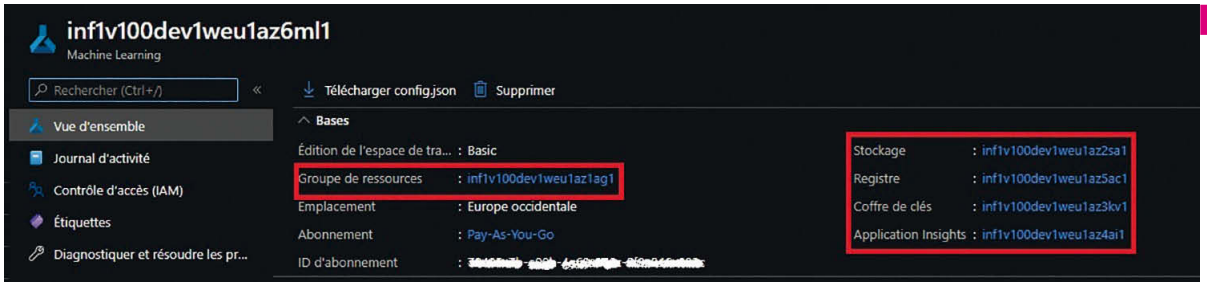
```
Ensuite nous le nettoyons pour n'en garder que l'ID de la ressource
 grep -v "with name" ${AZR_NAME}Info.json > ${AZR_NAME}.json
echo "##vso[task.setvariable variable=${{ AZR.TYPE }}${{ AZR.ID }}]${AZR_NAME}"
```

>> Stockage Id de la ressource et état de l'appel (Succès/Échec) :
Et pour conclure, nous exporterons la variable Id de la ressource servant notamment pour la création des ressources qui en dépen-

(6) https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/deploy/azure-cli
(7) https://docs.microsoft.com/en-us/azure/devops/pipelines/library/service-endpoints
(8) https://docs.microsoft.com/en-us/rest/api/resources/providers
(9) https://docs.microsoft.com/fr-fr/azure/devops/pipelines/library/variable-groups
(10) https://docs.microsoft.com/fr-fr/azure/devops/organizations/accounts/use-personal-access-tokens-to-authenticate



4 *Liste des Jobs.*



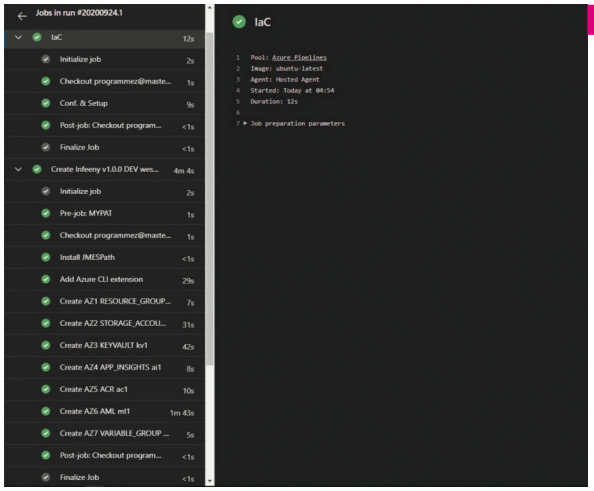
6 *Ressource Machine Learning*

dent. La lib JMESPath (11) que nous avons installée nous sera d'un grand secours pour parser le json afin d'extraire l'Id. Une seule passe suffit ! Zéro redoublement de passes Azure CLI (Show, List, tutti-quanti), pas de grigri, c'est direct au but.
Cependant, il est des ressources qui mettent des lustres à pointer du nez. Ce cas-ci, je vous conseille de placer les mises en attente stratégiques genre *Sleep* ou *Wait* avant de réenclencher le turbo…

```
AZR_STATE=$(/home/vsts/work/jp -f ${AZR_NAME}.json -u properties.provisioningState)
AZR_ID=$(/home/vsts/work/jp -f ${AZR_NAME}.json -u id)
echo "##vso[task.setvariable variable=${{ AZR.TYPE }}${{ AZR.ID }}_ID]${AZR_ID}"…
```

Vers l'infini, et au-delà !

Nous pourrons ainsi instancier autant de Jobs/Instances que le « *produit* » des versions, environnements, locations, et même de différentes organisations/tenants moyennant une légère modification du script, afin de basculer d'une souscription à une autre.
Notons également qu'un job est une série d'étapes qui se déroulent de manière séquentielle comme une unité. En d'autres termes, un job est la plus petite unité de travail qui peut être programmée.
Prenons l'exemple suivant d'un pipeline comprenant :
1 Organisation : Infeeny ;
1 Version : 1.0.0 ;
3 Environnements : DEV, INT et PROD ;
2 Locations : westeurope et francecentral.
Nous créerons ainsi 1 x 1 x 3 x 2 = 6 Jobs/Instances parfaitement identiques se lançant les uns après les autres en fonction de nos critères de dépendance. 4
Quant aux ressources du cloud Azure ils seront déployés également et de manière séquentielle puisque nous nous sommes assurés de définir pour chaque ressource une relation en la marquant comme étant dépendante ou pas d'une ou de plusieurs ressources. 5

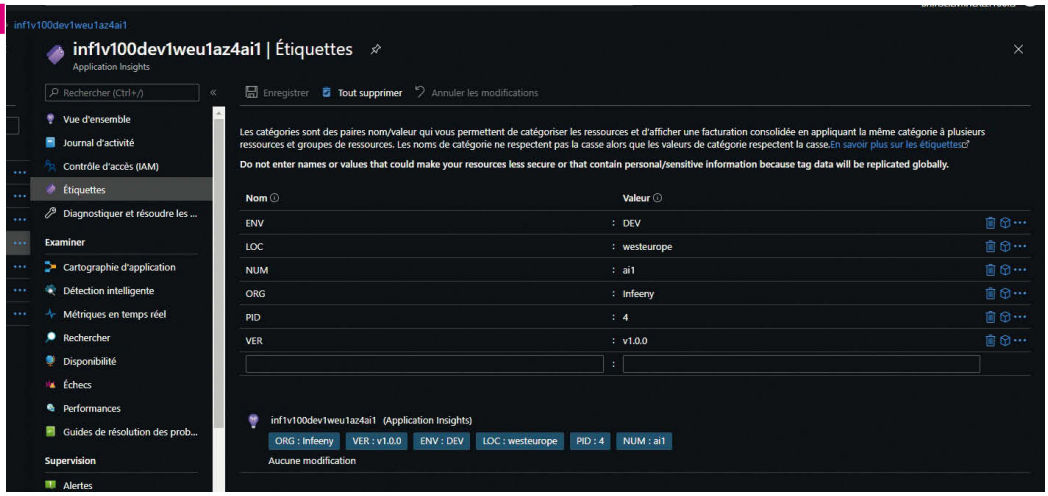


5 *Liste des Ressources par Job/Instance.*

Par exemple la resource Machine Learning dépend d'un « Groupe de ressources/RESOURCE_GROUP », d'un « Compte de stockage/STORAGE_ACCOUNT », d'un « Coffre de clés/KEY_VAULT », d'une « Application Insights/APP_INSIGHTS » et enfin d'un « Registre de conteneurs/ACR ». 6
Notons aussi que les noms de nos ressources sont préfixés de manière dynamique comme dans notre exemple ci-dessus avec « inf1v100dev1weu1 » qu'est la concaténation des codes Organisation, Version, Environnement et Location.
Les Tags aussi sont gérés de manière dynamique, comme dans l'exemple suivant ou la ressource Application Insights « inf1v100dev1weu1az4ai1 » est taguée cette fois avec les noms complets des variables d'environnements qui la définissent. 7
D'un point de vue technique, on peut créer/mettre à jour autant de ressources ou d'instances IaC sans qu'on ait à modifier quoi que ce soit ou alors très peu dans le cœur de l'engin. On se satisfera du déclaratif, à formuler les listes de nos objets dans les paramètres tout en observant les conventions de nommage, le chaînage, les dépendances, et bien sûr la bonne commande Azure CLI servant à les manipuler. Néanmoins, ceci demeure un PoC, qu'on peut nette-

(11) https://jmespath.org/

Tags 7



ment améliorer en blindant les tests, voire le refactoriser en full Bash afin de générer dynamiquement nos templates… Et plus encore, pour peu que vous soyez du genre « *même pas peur* » carrément le reWrapper en API. Ça a toujours commencé comme ça, les Ansible, Terraform et Puppets Show !!

Faut dire que notre approche NoOps en matière d'IAC répond parfaitement aux impératifs métiers ainsi qu'aux « meilleures pratiques » quant à la personnalisation des modèles des déploiements en mode Agile/DevOps. De plus, étant peu bavarde grâce au templating yaml sa maintenance en est grandement facilitée, ainsi :

- Nos infrastructures peuvent être déployées de manière progressive et avec plus de confiance à mesure que nos systèmes deviennent plus complexes ;
- Les ressources existantes peuvent être réutilisées par notre modèle. Il suffit de pointer ces ressources dans le template ;
- Lorsque l'on déploie des ressources infrastructurelles supplémentaires, il devient utile d'échelonner les changements dans les différents environnements. Vous pourriez envisager de réaliser une série de tests d'intégration ou de composants avant de passer au PRD ;
- C'est une bonne pratique que de protéger le lancement des modifications de la PRD contre les changements provenant d'autres branches que le master. Les conditions dans notre pipeline d'Azure peuvent vous aider à mettre en place des contrôles de ce type ;
- Pour les besoins de notre MLOps (dans la suite), nous faisons une séparation des préoccupations entre le déploiement de l'infrastructure et le déploiement des artefacts ML. Ainsi, les deux types sont déployés à des moments différents et avec des conduites d'automatisation différentes.
- De multiples contrôles de sécurité supplémentaires (règles de réseau virtuel, contrôle d'accès basé sur les rôles et identités personnalisées) peuvent être appliqués aux ressources Azure. Les contrôles peuvent être ajoutés directement à partir de notre template jobs.

Selon moi, c'est ça le NoOps : du full code de bout en bout, ne rien lâcher à la GUI, car ça fait un peu mélange de genres et ce que l'on gagne par simplisme d'un point de vue assistanat s'avère, malheureusement dans la plupart des cas, un frein pour notre compréhension du modèle architectural ainsi que de ses interactions structurellement bas-niveau.

Je ne sais pas s'il faut trouver un compromis. Me concernant, l'abstraction GUI est la parfaite antithèse de ce qu'est l'essence même du NoOps !! Et souvent, quand on est dans une impasse, alors c'est l'une des deux, soit on n'utilise pas la bonne librairie, dans ce cas cherchez, vous trouverez ; ou bien alors, on ne maîtrise pas toutes les subtilités conceptuelles, lesquelles et au risque de faire mon enquiquineur, se résument en deux mots « AS CODE », et rien que, *grand bien vous fasse* !!!

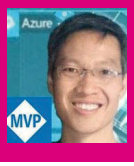
Le code est disponible sur :

https://github.com/makramjandar/NoOps-Yes-Back2Bash

In Code

We Trust

Louis Charavner
Développeur IoT, spécialisé en Smart Factory



Vincent Thavonekham
Head of Smart Factory VISEO, MVP/RD

Maxime Billemaz
Développeur IoT, spécialisé en Smart Factory

Azure FarmBeats

L'AGRICULTURE DU FUTUR, CONDUITE PAR LES DONNÉES

Les techniques d'agriculture fondées sur les données contribuent à stimuler la productivité agricole en augmentant les rendements et en réduisant les pertes liées aux coûts des ressources en eau. Toutefois, ces techniques ont connu une adoption rare en raison du coût élevé de la collecte de données et les solutions de connectivité limitées. Dans cet article, nous présentons FarmBeats, une plateforme IoT conçue pour l'agriculture qui permet une collecte de données sécurisée de divers capteurs, caméras et drones. La solution FarmBeats arrive à pallier les problèmes de batteries des drones, de connectivité de la ferme et les problèmes de connexion liés au climat.

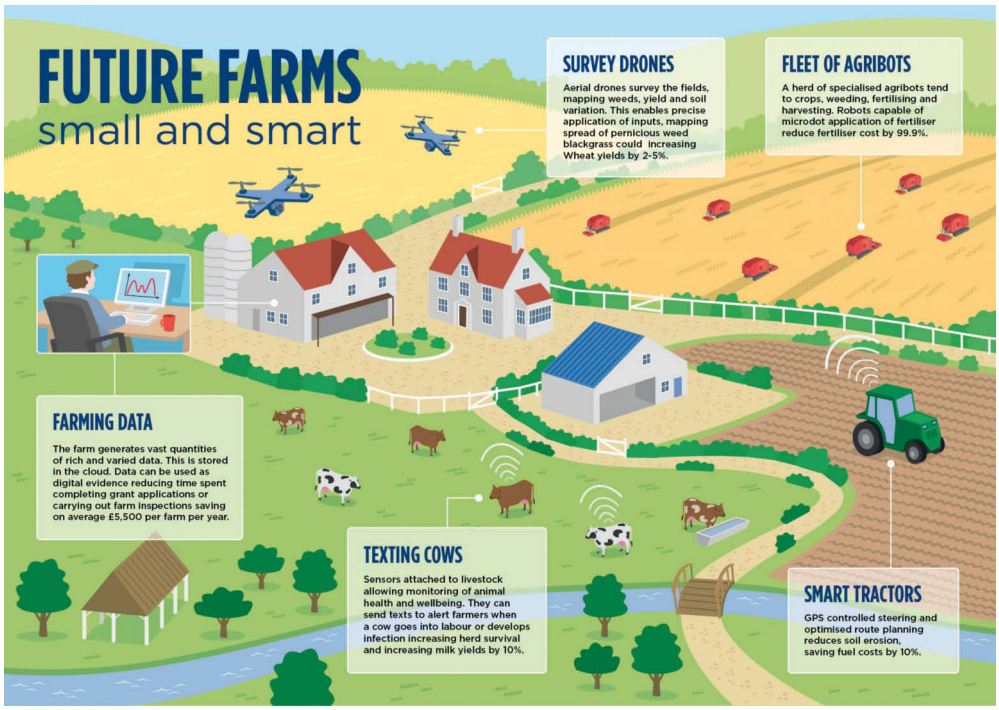
Après cette brève introduction, nous enchaînerons immédiatement sur les aspects techniques, avec l'hypothèse que vous disposez de quelques connaissances d'Azure. Enfin, nous présenterons un cas d'usage. Et si vous avez des difficultés sur la partie technique, pas de panique, nous avons créé des vidéos et le code est sous Github du MUG Lyon (C#, Python). 1

La ferme du futur aura des **smart sensors (capteurs intelligents)** pour remonter des données valorisables (insights) au fermier.

Un smart sensor est un dispositif qui **prend des données de l'environnement** physique et utilise des **ressources de calcul intégrées** pour exécuter des fonctions prédéfinies lors de la détection d'une entrée spécifique. Puis **traite les données** avant de les transmettre à une passerelle qui va se charger de les transmettre à un environnement cloud.

Dans Azure FarmBeats on a trois capteurs principaux :

- Les capteurs de télémétrie : température, humidité du sol, etc.
- Les capteurs météorologiques : pluie, soleil, neige, etc.
- Les capteurs d'imagerie : drone, caméras, satellites, etc. 2



1 *Le futur de l'agriculture : une énorme quantité de capteurs disposés dans les champs et sur le bétail collecteront des informations utiles. Ces capteurs doivent être capables de fonctionner avec une très faible consommation électrique et d'être protégés contre les rigueurs climatiques.. Image : (cc-by-sa 4.0) nesta*

2 *Capteur de moisissure compatible Azure FarmBeats par Teralytic.*

Mise en pratique

Créer un simulateur pour des données de température.

Pour créer le simulateur, vous devez avoir une **inscription Azure**. Puis, rendez-vous dans le **marketplace** Azure pour créer une nouvelle ressource Azure FarmBeats. 3

Une fois tous les champs renseignés cliquez sur « create » et attendez entre **30-40 minutes,** le temps qu'Azure provisionne les différents services nécessaires..

Dans la figure 4, le ressource group « rg-azfarmbeat-tuto, avec les 26 ressources Azure créés par FarmBeat. Nous allons voir à la fin de l'article pourquoi il y en a autant. 4

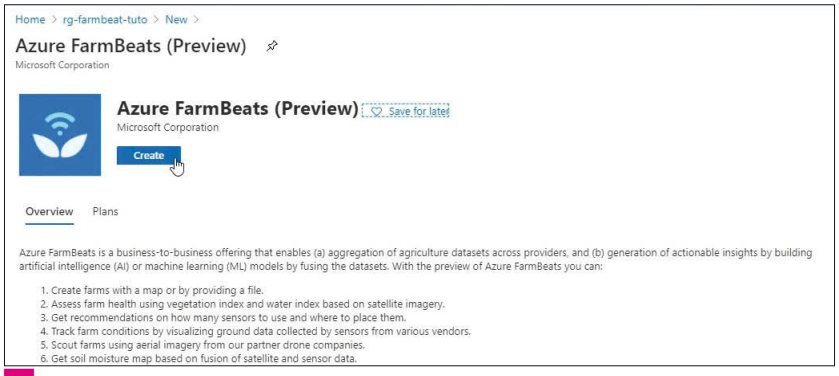
On peut remarquer que FarmBeats crée des ressources Azure déjà connues des développeurs, et que l'on utilise en production.

Une des ressources est un **App Service** « *web-farmebe-we-dev* »; ouvrez le lien du site Web, et allez ensuite dans « Create Farm ». Vous devriez avoir cette interface, où vous allez délimiter la zone de votre champ agricole. 5

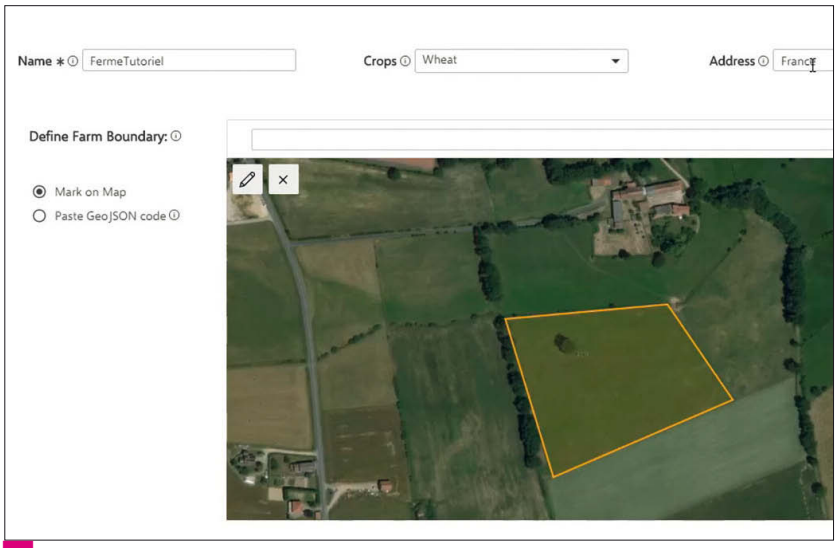
Maintenant que nous avons notre ferme, nous allons mettre en place **un capteur simulé** pour envoyer des données dans notre interface.

Pour cela vous devez d'abord créer un **« partenaire »**, avec le script de génération de partenaire (https://aka.ms/farmbeatspartnerscriptv3).

Avec ce script vous devriez avoir :



3 *Création dans Azure de FarmBeats.*

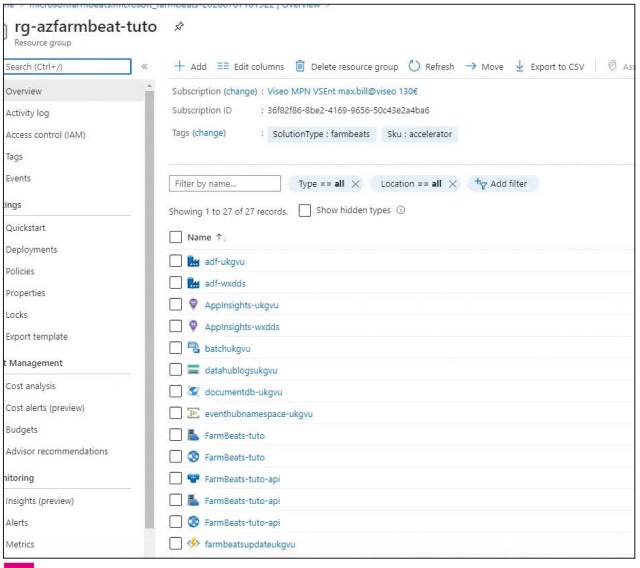


5 *Formulaire de création d'un champ dans Azure FarmBeats.*

- Api endpoint ;
- Tenant ID ;
- Client ID ;
- Client Secret ;
- EventHub connection string.

Pour commencer à créer nos ressources via l'API, il va nous falloir un jeton JWT. Voici un script en Python qui va vous permettre de l'obtenir.

```
import requests
import json
import msal

# Your service principal App ID
CLIENT_ID = ""
# Your service principal password
CLIENT_SECRET = ""
# Tenant ID for your Azure subscription
TENANT_ID = ""

AUTHORITY_HOST = 'https://login.microsoftonline.com'
AUTHORITY = AUTHORITY_HOST + '/' + TENANT_ID
```



4 *Liste d'une partie des 26 ressources Azure.*

```
ENDPOINT = ""
SCOPE = ENDPOINT + "/.default"

context = msal.ConfidentialClientApplication(CLIENT_ID, authority=AUTHORITY, client_credential
=CLIENT_SECRET)
token_response = context.acquire_token_for_client(SCOPE)
# We should get an access token here
access_token = token_response.get('access_token')
print("ACESS TOKEN : " + str(access_token))
```

Une fois le jeton obtenu, nous allons créer quatre ressources :

- Un deviceModel ;
- Un device ;
- Un sensorModel ;
- Un sensor.

Chaque appel est détaillé dans le swagger de farmBeats (voir le lien Swagger à la fin).

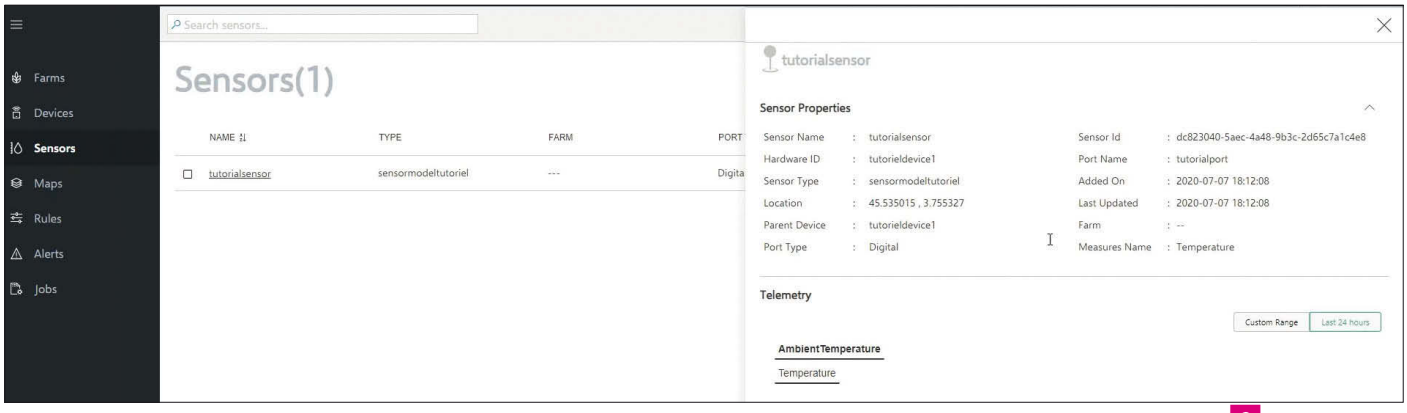
Voici des exemples de requêtes qu'il faut envoyer à l'API.

DeviceModel

```
{
  "type": "Node",
  "manufacturer" : "tutorielmanu",
  "productCode": "EnviroMonitor#6800",
  "ports": [
    {
      "name": "tutorielport",
      "type": "Digital"
    }
  ],
  "name": "devicemodeltutoriel",
  "description": "Je suis un mock d'un device model."
}
```

Device

```
{
  "deviceModelId": "<VOTRE DEVICE MODEL ID>",
```



6 *Un nouveau capteur listé, ainsi que ses propriétés.*

```
  "hardwareId": "tutorieldevice1",
  "reportingInterval": 5,
  "location": {
    "latitude": 45.535015,
    "longitude": 3.755327
  },
  "name": "tutorieldevice1",
  "description": "Je suis le device du tutoriel."
}
```

SensorModel

```
{
  "type": "Digital",
  "manufacturer": "tutorielmanu",
  "productCode": "RS-CO2-N01",
  "sensorMeasures": [
    {
      "name": "Temperature",
      "dataType": "Double",
      "type": "AmbientTemperature",
      "unit": "Celsius",
      "aggregationType": "None",
      "description": "Je suis un capteur de temperature"
    }
  ],
  "name": "sensormodeltutoriel",
  "description": "Je suis un sensor model pour le tutoriel."
}
```

Sensor

```
{
  "hardwareId": "tutorieldevice1",
  "sensorModelId": "<VOTRE SENSOR MODEL ID>",
  "location": {
    "latitude": 45.535015,
    "longitude": 3.755327
  },
  "port": {
    "name": "tutorialport",
    "type": "Digital"
  },
  "deviceId": "<VOTRE DEVICE ID>",
  "name": "tutorialsensor",da sdasdas
  "description": "Je suis un sensor de tutoriel"
}
```

Dans votre interface FarmBeats vous devriez avoir un nouveau sensor et un nouveau device. 6

Nous allons envoyer des données sur ce capteur. Voici un script Python inspiré de la documentation de FarmBeats pour l'envoi de données. Il envoie une température aléatoire toutes les cinq secondes.

```
import json
import azure
from azure.eventhub import EventHubClient, Sender, EventData, Receiver, Offset
import threading
import datetime
import random

def send_data():
 EVENTHUBCONNECTIONSTRING = "<VOTRE EVENTHUB CONNECTION STRING>"
 # Votre nom de eventHub devrait être celui-ci.
 EVENTHUBNAME = "sensor-partner-eh-00"

 telemetry_message = {
  "deviceid": "<VOTRE DEVICE ID>",
  "timestamp": datetime.datetime.utcnow().isoformat(),
  "version" : "1",
  "sensors": [
   {
    "id": "<VOTRE SENSOR ID>",
    "sensordata": [{
     "timestamp": datetime.datetime.utcnow().isoformat(),
     "Temperature": random.randint(150, 170)
    }]
   }
  ]
 }

 write_client = EventHubClient.from_connection_string(EVENTHUBCONNECTIONSTRING, eventhub
=EVENTHUBNAME, debug=True)
 sender = write_client.add_sender(partition="0")
 write_client.run()
```

```
print("Sending telemetry: " + json.dumps(telemetry_message))
sender.send(EventData(json.dumps(telemetry_message)))
write_client.stop()

def setInterval(func,time):
  e = threading.Event()
  while not e.wait(time):
    func()

setInterval(send_data, 5)
```

Ce script va envoyer toutes les cinq secondes une température aléatoire comprise entre 150 et 170 à notre capteur de température simulé. Pour finir, ajoutez le device à votre ferme et vous devriez voir vos données. 7

Vous pourrez retrouver une vidéo complète et explicative en fin d'article.

Cas d'utilisation

Microsoft a déployé FarmBeats dans deux fermes. Une dans l'état de Washington et une au nord de la ville de New York, sur une superficie qui varie entre 2 et 40 hectares.

La ferme située dans le nord de l'État de New York suit le modèle de l'agriculture soutenue par la communauté (CSA) et cultive des légumes, des fruits, des céréales, ainsi que des produits laitiers, de la volaille et de la viande. Ces déploiements comprennent : des capteurs, des caméras, des drones, une station IoT, un PC passerelle (gateway) et le cloud. 8

Microsoft a utilisé trois drones professionnels. Soit vous savez piloter des drones, et Microsoft permet d'utiliser le DJI Mobile SDK pour interfacer les drones avec FarmBeats, soit vous souhaitez un mode « Drone as a Service » alors, vous utiliserez des drones AzurDrones également compatibles Azure… Cela dit, la plateforme de ces drones autonomes prennent de la place ! (9)

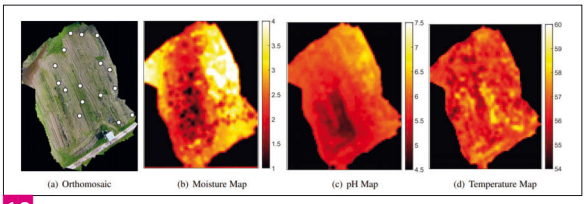
L'application FarmBeats planifie ensuite une trajectoire de vol à l'aide de l'algorithme FarmBeats qui optimise la batterie des drones en fonction de la puissance et la direction du vent. Une fois sa mission terminée, le drone retourne automatiquement à sa position d'origine et transfère l'enregistrement vidéo pendant le vol vers la gateway. 10

Comme le montre la figure 10, en se basant sur les valeurs des capteurs dans le reste de la ferme, le pipeline de prévision de l'humidité peut estimer que la partie supérieure gauche de la ferme a un taux d'humidité élevé même si cette partie n'a pas de capteur à cet endroit.

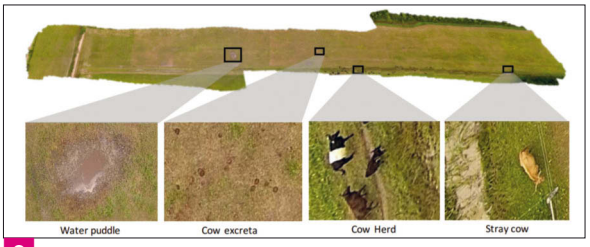
De même, la carte du pH génère une donnée intéressante en ce sens que le bas gauche et le centre de la ferme ont un pH très faible et sont très acides. Grâce à cette carte, l'agriculteur a appliqué de la chaux pour améliorer le pH et rendre le sol plus neutre 11

Sur la figure 11, FarmBeats surpasse les techniques d'interpolation existantes. En particulier, FarmBeats peut estimer avec précision les variations des différentes valeurs des capteurs sur le terrain. Alors que les méthodes classiques basées sur les capteurs ne reflètent pas les variations et ont donc une corrélation presque nulle avec les valeurs des capteurs. Les estimations de FarmBeats ont une corré-

9 *Taille de la plateforme d'accueil des drones autonomes AzurDrone.*

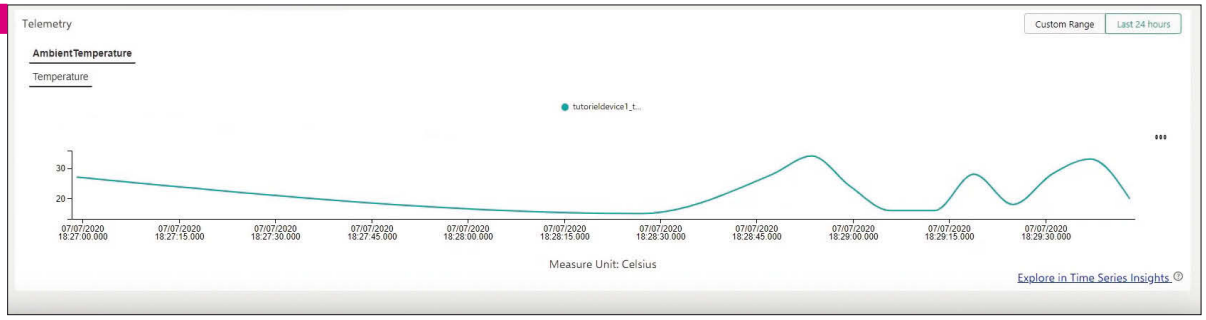


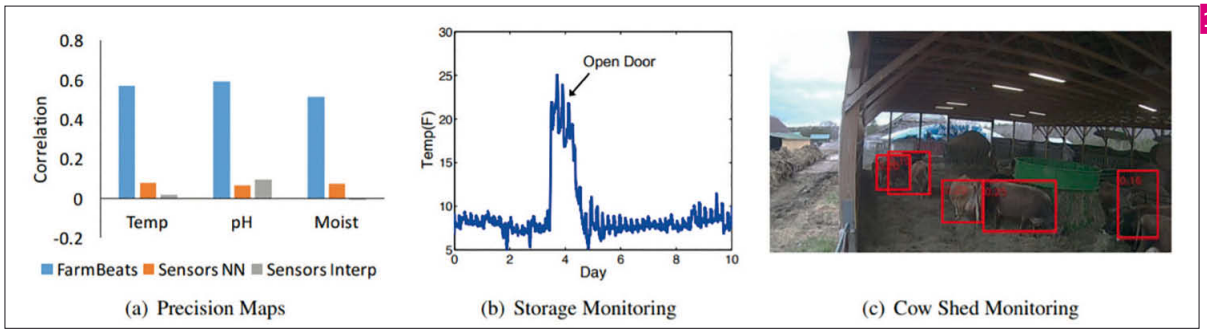
10 *Génération de cartes de chaleur de moisissure, d'acidité et de température via des photos orthomosaiques.*



8 *Collecte d'informations de la ferme avec des photos orthomosaiques (via des drones).*

7 *Remonté des données du capteur.*





11 *Exemples d'insights remontés par les capteurs à Azure FarmBeats.*

lation positive élevée avec les valeurs réelles des capteurs, indiquant ainsi l'utilité d'employer la vidéo du drone en conjonction avec les estimations de FarmBeats. Enfin, les cartes de précision générées par FarmBeats ont une taille moyenne inférieure de 3 ordres de grandeur à celle de la vidéo et peuvent être facilement envoyées dans le cloud pendant les périodes de connectivité.

La figure 11 met en évidence deux autres applications pour lesquelles les agriculteurs ont utilisé FarmBeats. Tout d'abord, l'agriculteur de New York a utilisé les capteurs FarmBeats pour surveiller ses congélateurs de stockage. La température dans ces congélateurs est soigneusement régulée en dessous de 10 F pour éviter que les produits ne se détériorent. Comme le montre la figure 11 (b), un employé laissant la porte ouverte pourrait faire monter cette température et causer des pertes à l'agriculteur. Ce problème est résolu par FarmBeats en permettant des notifications automatiques basées sur ces relevés de capteurs dans l'application téléphonique de FarmBeats. Ensuite, les agriculteurs ont branché des caméras à différents endroits comme les étables et les ont connectées à la station de base FarmBeats la plus proche. Une image de la caméra est présentée à la figure 11 (c). Bien que l'objectif de l'application actuelle soit de surveiller de visu les vaches, il est possible de créer une application capable de détecter des anomalies dans le comportement des vaches ou d'utiliser les mouvements des vaches pour suivre les déplacements des animaux.

Outils à disposition dans Azure FarmBeats

À ce jour, bien qu'en préversion, la plateforme Azure Farmbeats met à disposition déjà de nombreux services. C'est pour cela qu'elle crée parfois de 26 à 31 ressources de services, selon la version de FarmBearts.

- Évaluation de la santé des exploitations agricoles à l'aide de l'indice de végétation et de l'indice d'humidité basés sur l'imagerie satellite ;
- Recommandations sur le nombre de capteurs d'humidité à utiliser et recommandations sur leurs emplacements ;
- Suivi de l'état des fermes en visualisant les données au sol recueillies par les capteurs de différents fournisseurs ;
- Obtention d'une carte de l'humidité du sol basée sur la fusion des données des satellites et des capteurs ;
- Obtention des informations exploitables en construisant des modèles AI/ML à partir d'ensembles de données agrégées ;
- Développement ou amélioration de votre solution numérique pour l'agriculture en fournissant des conseils sur la santé des exploitations agricoles.

Ressources

Le code est disponible sur Github :
https://bit.ly/35v8nwT

La vidéo très complète de notre article :
https://youtu.be/Cd5sPuQJvIY

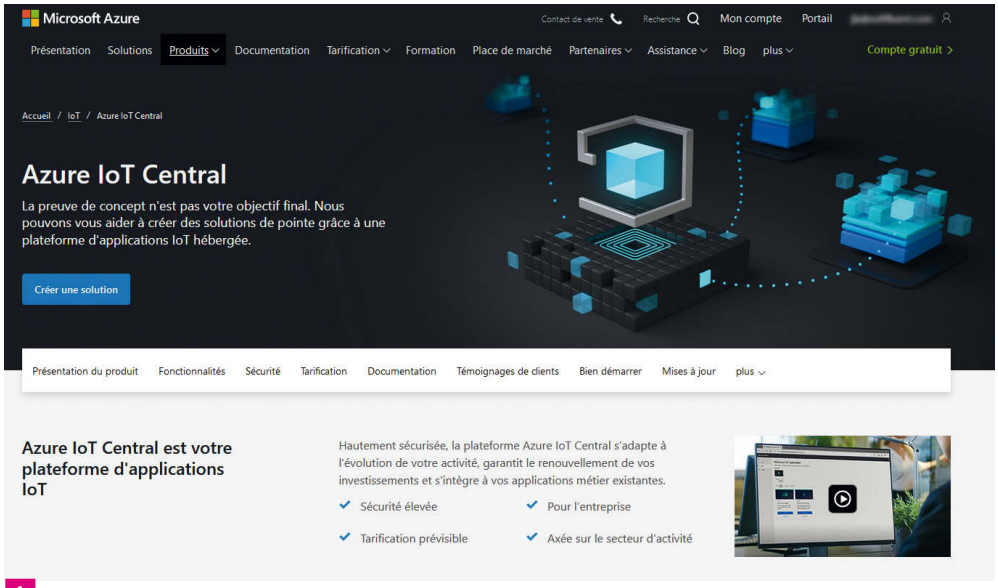
Swagger de FarmBeats :
https://farmbeatsproductionbits.blob.core.windows.net/farmbeatsguide/AzureFarmBeats-API-1.2.11.html

Référence :
http://www.nesta.org.uk/blog/precision-agriculture-almost-20-increase-income-possible-smart-farming

Jérémy Larrieu
Titulaire d'une licence Systèmes Informatiques et Logiciels, Jérémy a d'abord travaillé en tant que développeur Web Freelance puis développeur .NET. Il rejoint SoftFluent en 2018 en tant que consultant .NET et intervient actuellement sur un projet lui permettant d'exploiter les technologies Angular et du Framework .NET. Jérémy développe des solutions réseaux et IoT sur mesure.

Azure IoT Central : l'IoT en toute simplicité

Avec Azure IoT, Microsoft propose un ensemble d'outils permettant de connecter, d'exploiter et de sécuriser les données en provenance de périphériques IoT. Le principal avantage avancé, est de disposer d'une plateforme complète permettant d'utiliser des composants déjà codés et managés afin de se concentrer sur la partie métier de son application ainsi que sa mise à l'échelle. Nous allons faire un tour d'horizon des fonctionnalités proposées dans Azure IoT Central ainsi qu'un test de celui-ci afin d'une part, de voir comment le prendre en main et d'autre part d'évaluer les fonctionnalités proposées sur un cas concret.



1

Azure IoT Central 1

Azure IoT Central vise à accélérer la création de solutions IoT tout en réduisant leur coût. En effet, il n'est pas nécessaire de maintenir une infrastructure **serveur**, cela étant assuré par Azure IoT. Les applications ainsi créées peuvent alors aussi s'intégrer à des applications métiers existantes.

Concrètement, il s'agit de créer une application à partir d'un modèle existant ou de créer soi-même un modèle.

- Une application consiste principalement en un tableau de bord regroupant, en fonction du besoin métier :
- Des graphiques et des indicateurs basés sur les données de télémétrie collectées par les appareils ;

Des contrôles permettant le suivi et l'exécution d'actions sur les appareils ou les groupes d'appareils.

Azure IoT Central met déjà à disposition des modèles d'applications, triés dans 4 catégories « métier » :

- Retail :
  - Logistique connectée : il s'agit du suivi en temps réel des expéditions ainsi que de leurs conditions de transport au moyen de capteurs de température, d'humidité, de choc, …
  - Centre de distribution numérique : amélioration de la gestion des centres de distribution au moyen de caméras connectées faisant le tri entre les paquets identifiés et ceux qui ne le sont pas.
  - Analytique dans le magasin : surveillance des conditions dans un magasin comme la température et l'humidité afin de pouvoir piloter le chauffage ou la climatisation par zone pour améliorer l'expérience client.
  - Analytique dans le magasin : paiement : suivi de l'affluence dans un magasin et la manière dont elle est gérée afin d'améliorer la fluidité de celle-ci notamment lors du passage en caisse. Les capteurs doivent permettre d'évaluer la longueur d'une file d'attente pour une caisse, ainsi que le temps d'attente.
  - Gestion intelligente des stocks : réductions des ruptures de stocks ainsi que détermination du niveau de stock adapté à la demande au moyen de tags RFID ou de caméras connectées.
  - Centre de micro-traitements : amélioration de la gestion des centres de traitement entièrement automatisés permettant de stocker des produits au plus près des clients finaux et des magasins. Cela comprend la supervision des conditions ambiantes au moyen de capteurs (température, humidité, …) ainsi que des robots d'expédition.
- Energie :
  - Suivi de compteurs intelligents : suivi de la consommation ainsi que l'état d'un réseau électrique au moyen de compteurs connectés.
  - Suivi des panneaux solaires : supervision de la production d'électricité au moyen de panneaux solaires ainsi connaissance de l'état du réseau en temps réel.
- Administration :
  - Gestion connectée des déchets : optimisation de la collecte des déchets par le suivi du niveau de remplissage, d'odeur ou le poids de bennes à ordures.
  - Suivi de la consommation d'eau : surveillance de la consommation d'eau et gestion le débit d'eau afin d'en réduire la consommation. Cela s'adresse aux distributeurs d'eau.
  - Suivi de la qualité de l'eau : suivi en temps réel de la qualité de l'eau pour les distributeurs d'eau.

- Santé :
  - Suivi continu des patients : suivi en temps réel et à distance de l'état de santé des patients et priorisation des interventions pour le personnel soignant.

Une application ***Azure IoT Central*** se décompose en plusieurs parties :

- Les modèles d'appareils ;
- Les appareils ;
- Les groupes d'appareils ;
- Les travaux ;
- Les tableaux de bord ;
- Les analytiques ;
- Les règles ;
- Les tableaux de bords ;
- Les exportations de données.

Les modèles d'appareils

Les modèles d'appareils contiennent toutes les informations concernant un appareil : les données envoyées, les propriétés et les commandes disponibles ainsi que des pages d'affichage permettant de visualiser les données. Il est possible d'utiliser des modèles prédéfinis provenant d'un catalogue d'appareils certifiés ou bien de définir soi-même un modèle.

Un modèle d'appareil contient :

- Un « modèle de capacité » contenant la définition des données de télémétrie envoyées, des propriétés pouvant être lues et/ou envoyées à l'appareil ainsi que des commandes exécutables par l'appareil. Ces capacités peuvent être définies dans une ou plusieurs interfaces.
- Des « propriétés de cloud » permettant de stocker des informations concernant l'appareil. Elles ne seront jamais partagées avec l'appareil. Par exemple, cela peut être utilisé pour stocker la date de la dernière alerte remontée par un appareil.
- Des « personnalisations » permettant de modifier certaines définitions du modèle de capacité de l'appareil, par exemple en remplaçant le nom d'une propriété.
- Des « affichages » sous la forme de tableaux de bord et de formulaires permettant de surveiller et de gérer les appareils individuellement.

Attention : un modèle d'appareil personnalisé est lié à une application ***Azure IoT Central*** mais il peut être exporté au format JSON : seules les interfaces du modèle de capacité sont exportables.

Attention : pour être associé à un appareil, le brouillon de modèle doit être publié. A partir de ce moment, il n'est plus possible de modifier les interfaces du modèle de capacité. Pour mettre à jour un modèle, il faut créer une nouvelle version ainsi qu'une nouvelle interface : les versions précédentes ne sont pas modifiables.

Les appareils

La page des appareils contient tous les appareils connectés à l'application, regroupés par modèle.

Lors de la création d'un appareil, il est demandé un modèle, un nom, un identifiant ainsi que d'indiquer si l'appareil est simulé. Une fois validé, la page de l'appareil affiche les tableaux de bord et les formulaires définis dans le modèle ainsi qu'une vue affichant les données brutes de l'appareil.

Dans la partie supérieure droite se trouvent différentes commandes permettant :

- D'afficher et de définir les paramètres de connexions de l'appareil : identifiant et scope de l'appareil, méthode de connexion ;
- De bloquer un appareil ;
- D'attacher un appareil à une passerelle ;
- De renommer ou de supprimer l'appareil ;

Les groupes d'appareils

Les groupes d'appareils permettent de grouper les appareils du même modèle au moyen de critères se basant sur les informations basiques (nom, simulé, bloqué, …) ainsi que sur les propriétés cloud ou autres de l'appareil.

Il s'agit de regrouper les appareils de manière logique par bâtiment, étage ou zone par exemple, lorsque l'on souhaite superviser un nombre important de capteurs répartis sur plusieurs sites.

Les travaux

Les travaux permettent d'effectuer une tâche sur un groupe d'appareils. Il est possible de modifier la valeur d'une propriété ou d'une propriété cloud ou d'exécuter une commande.

Cela peut s'avérer pratique lorsque l'on souhaite exécuter une commande ou mettre à jour une propriété sur un grand nombre d'appareils ou sur un groupe d'appareil en particulier : par exemple mettre à jour un seuil d'alerte ou la sensibilité d'un capteur. Pour chaque tâche, il est possible de voir l'état d'avancement, le statut pour chaque appareil ainsi qu'un historique des exécutions sur les 30 derniers jours.

Les tableaux de bord

Les tableaux de bord constituent la page d'accueil d'une application ***Azure IoT Central***. Il est donc important que cela reflète des informations utiles au besoin métier. Pour cela, il est possible de créer des graphiques, des indicateurs ou des actions. Après avoir sélectionné un groupe d'appareils ou des appareils individuellement, il faut choisir les données de télémétrie, propriété, propriété cloud ou commande que l'on souhaite utiliser.

Une fois ajoutées au tableau de bord, il est possible de personnaliser la vignette :

- Changer le type de visualisation : histogramme, courbes, secteurs, … ;
- Modifier la taille ;
- Configurer le titre, les axes, la plage de temps, …

Il est possible de créer un tableau de bord uniquement pour soi ou pour tous les utilisateurs de l'application. Attention, ce choix est disponible uniquement à la création.

Les analytiques

La page Analytique permet de lancer une analyse sur les données à partir des données de télémétrie ou des agrégats sur ces données, tout en définissant la dimension permettant leur regroupement.

La sélection se fait par groupe d'appareils et permet de visualiser les données sous la forme de courbe avec un sélecteur pour affiner la période de temps sélectionnée : les dernières 24h par défaut.

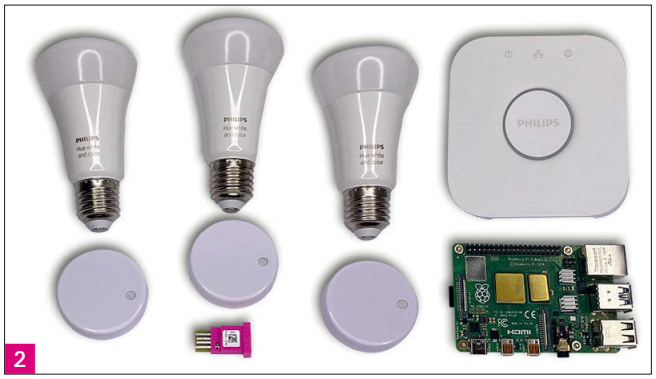
Le graphique permet de voir ces données sous la forme d'un tableau ainsi que de les exporter au format CSV.

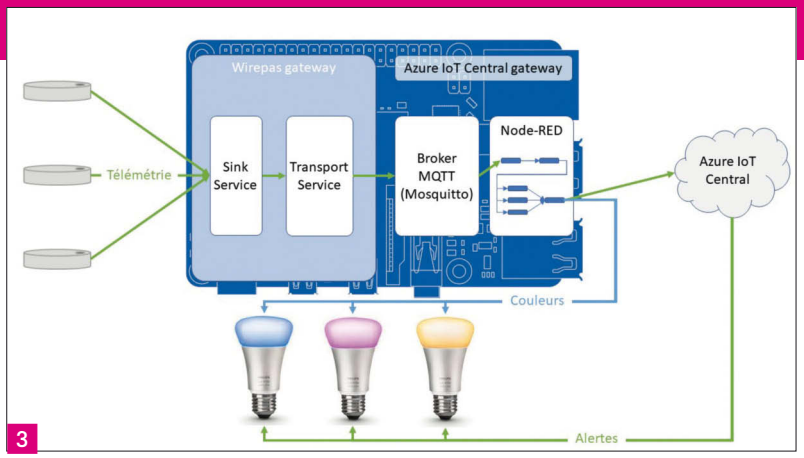
Les règles

Les règles permettent d'effectuer des actions en fonction de critères fixés sur les données de télémétrie pour un modèle d'appareil particulier.

Il est possible de filtrer les appareils d'un même modèle au moyen des propriétés ou des propriétés cloud disponibles. Ensuite, il faut définir les conditions en utilisant les données de télémétrie et, au besoin, une agrégation de temps permettant de sélectionner l'agrégation utile pour chaque donnée de télémétrie utilisée.

Enfin, il reste à créer les actions qui seront





exécutées par la règle. Il existe 5 actions possibles :

- Email : permet l'envoi d'un email à partir d'un modèle prédéfini contenant les informations utiles concernant la règle qui a déclenché l'envoi.
- Groupe d'actions Azure Monitor : permet de déclencher un groupe d'actions dans Azure Monitor comme l'envoi de SMS, passer un appel vocal, …
- Webhook : permet de notifier un système externe via un webhook. Lorsque la règle s'exécute, une requête POST est envoyée à l'URL définie. Un lien est disponible permettant de voir le schéma des données envoyées au webhook.
- Microsoft Power Automate : permet de déclencher des workflows définis dans Power Automate.
- Microsoft Azure Logic Apps : permet de déclencher une Logic App au moyen du connecteur **Azure IoT Central** v3.

Ces actions permettent donc de compléter la supervision ou l'analyse des données via les tableaux de bord ou les tableaux analytiques fournis, par la levée d'alertes métiers (mails simples ou actions Azure Monitor) mais aussi de s'intégrer à des solutions applicatives tierces de façon plus ou moins avancées directement (webhook) ou via des services applicatifs Azure (Power Automate, Logic App).

Les exportations de données

Les exportations de données sont conçues pour permettre l'envoi de données vers différentes destinations Azure, ce qui permet aussi l'intégration à d'autres applications métiers : Blob Storage, Event Hubs, Service Bus ou webhook. Au moment de la rédaction de cet article, Il existe 2 moyens pour créer une exportation de données : la première est considérée « legacy » et la seconde est en « préversion ».

Dans la version dite « legacy », une exportation se définit comme suit :

- Un nom ;
- La chaîne de connexion à la destination ;
- D'éventuels paramètres supplémentaires en fonction du type de destination choisi ;
- Les données à exporter. Cela consiste à sélectionner jusqu'à 3 types de données : télémétrie, appareils et modèles d'appareils.

Cette version des exportations de données est assez peu flexible en matière de sélection de données ou de destinations. En effet, il n'est pas possible de créer un filtre permettant d'exporter une portion des données ou de pouvoir paramétrer plusieurs destinations.

C'est dans cette optique que la « préversion » a été mise en place.

Ici, les exportations sont divisées en 2 parties : les destinations et les exportations de données à proprement parler.

Les destinations deviennent réutilisables : les duplications et les mises à jour de paramétrage sont grandement simplifiées. Elles se paramètrent de la même manière qu'auparavant au moyen d'une chaîne de connexion, et d'éventuels paramètres supplémentaires suivant le type sélectionné, ainsi que d'un nom.

Les exportations de données offrent une plus grande souplesse dans leur paramétrage. Il est maintenant possible de sélectionner les données à exporter tout en pouvant ajouter des filtres, enrichir les données et sélectionner une ou plusieurs destinations.

Le test

Afin d'évaluer les fonctionnalités proposées par ***Azure IoT Central***, j'ai choisi un cas simple qui permet de voir :

- Comment remonter des données depuis des capteurs vers ***Azure IoT Central utilisant le protocole Wirepas(1) ;***
- Comment mettre en place un tableau de bord ;
- Comment définir des alertes en cas de dépassement de seuil.

Pour cela, j'ai décidé d'utiliser des tags ***Ruuvi***, qui embarquent un capteur de température, d'humidité, de pression atmosphérique et un détecteur de choc, pour l'envoi de données vers ***Azure IoT Central*** et des ampoules connectées Philips Hue pour visualiser les alertes.

Le principe est simple : chaque ampoule est associée à un tag ***Ruuvi*** et prend une couleur correspondant à la température mesurée par le capteur. Si la température dépasse un certain seuil, la lampe clignote.

Pour la mise en place du prototype de la passerelle, j'ai utilisé le matériel suivant :

- 1 Raspberry Pi4 4GB avec Raspbian Lite pour jouer le rôle de la gateway Wirepas/Azure iOT Central
- 1 dongle USB u-blox NINA-B1 avec Wirepas 5
- 3 tags Ruuvi avec Wirepas 5
- 1 kit de démarrage Philips Hue avec un pont et 3 ampoules couleur 2

Du côté logiciel, j'ai mis en place :

- Le SinkService Wirepas en charge de collecter les mesures des capteurs.
- Le TransportService Wirepas servant à envoyer ces messages vers mon broker MQTT(2).
- ***Mosquitto(3)*** faisant office de broker MQTT.
- ***Node-RED(4)*** pour gérer les flux d'extraction, de transformation et d'envoi des données vers **Azure IoT Central** ainsi que la récupération des alertes.

Voici un schéma de la solution mise en place 3

Vous pouvez retrouver tous les détails techniques de la mise en place de la passerelle ainsi que les étapes suivantes détaillées sur mon Github à cette adresse :

https://github.com/adrenalinedj/wirepas-azure-iot-central

Une fois le matériel assemblé et les logiciels installés, il est nécessaire de créer un modèle d'appareil correspondant aux tags **Ruuvi** que l'on utilisera dans **Azure IoT Central**. Pour cela, il faut définir une interface de données ainsi que des affichages permettant de visualiser les données de chaque appareil. Pour que le modèle d'appareil puisse être utilisé, il faut le publier en cliquant sur le lien en haut de la page. 4

A présent, il est possible de déclarer les capteurs que l'on va utiliser. Pour faire simple, j'ai utilisé le même formalisme pour définir l'identifiant de chaque appareil : « RuuviTag » suivi du numéro présent sur l'étiquette au dos du capteur. Ce numéro correspond à l'adresse du capteur (source endpoint) dans le protocole Wirepas. 5

Maintenant que les capteurs sont déclarés dans **Azure IoT Central**, il faut créer les flux dans ***Node-RED***. Pour cela, il faut définir un flux par capteur et ajouter des nœuds permettant l'extraction des données et la transformation des messages reçus afin que cela corresponde à l'interface que l'on a créée dans le modèle d'appareil. L'envoi des données se fait au moyen de nœuds disponibles dans la bibliothèque de plugins de ***Node-RED***. Il faut spécifier l'identifiant du capteur ainsi que sa clé dans le message puis lui fournir le nom de la machine du IoT Hub de l'application. Ce nom de machine n'est pas directement accessible dans l'interface **d'Azure IoT Central**, il est possible d'utiliser l'outil **dps-keygen** présent sur le Github d'Azure. Cette méthode est détaillée dans la documentation technique du test sur mon projet Github.

Pour chaque flux, j'ai aussi ajouté l'envoi d'une couleur aux ampoules correspondant à la température relevée par le capteur associé en envoyant une requête HTTP à l'API REST présente dans le bridge Philips Hue. 6

Une fois les flux déployés, les données devraient commencer à arriver dans ***Azure IoT Central***. Vous pouvez les voir en allant sur la page d'un des capteurs et en ouvrant l'onglet « Données brutes ». 7

Nous pouvons passer à la création d'un tableau de bord reflétant les données que l'on souhaite suivre. Pour ma part, j'ai simplement ajouté un diagramme de suivi de la température et de l'humidité pour mes trois capteurs. 8

Ensuite, on peut configurer les alertes en allant sur la page des Règles. Il suffit de sélectionner le modèle d'appareil concerné par l'alerte et de définir les conditions : une température supérieure à 20°C dans mon cas. Pour l'envoi de l'alerte, j'ai choisi comme action un webhook et défini une URL pointant vers le Node-RED installé sur la passerelle : ouverture et redirection de ports seront certainement nécessaires pour l'accès depuis Azure à la passerelle. 9

Dès lors, il ne reste plus qu'à créer un flux, dans Node-RED, pour router l'alerte vers l'ampoule associée au tag concerné par l'alerte. 10

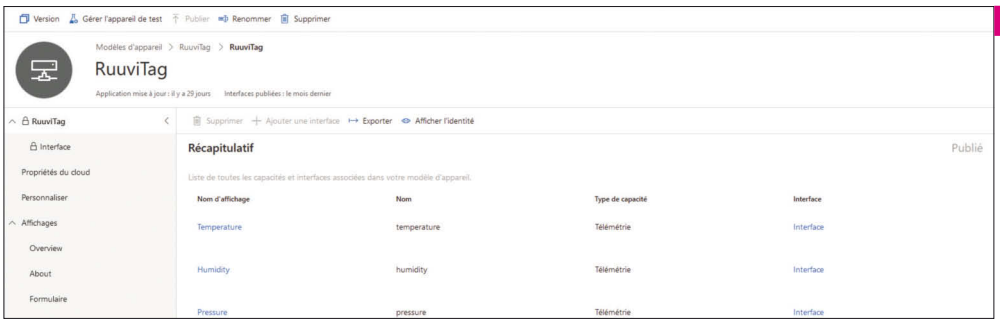
Maintenant, chaque ampoule devrait prendre la couleur correspondant à la température relevée par le capteur associé et se mettre à clignoter pendant 15 secondes lorsqu'une alerte est envoyée par ***Azure IoT Central***.

Bilan

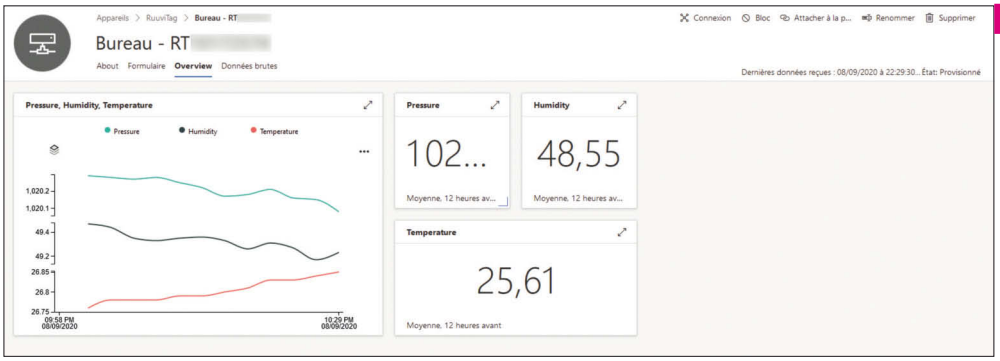
La proposition de Microsoft de mettre en place une application IoT avec des tableaux de bord, des alertes et une connectivité simplifiée pour les appareils est intéressante et fonctionne plutôt bien. Cependant, il manque quelques fonctionnalités qui en feraient un outil vraiment puissant.

La connectivité est plutôt facile à mettre en place même si dans l'exemple utilisé pour le test, il a fallu déclarer les appareils un par un. ***Azure IoT Central*** propose une fonctionnalité de provisionnement et d'approbation automatique permettant d'éviter cette étape d'appairage.

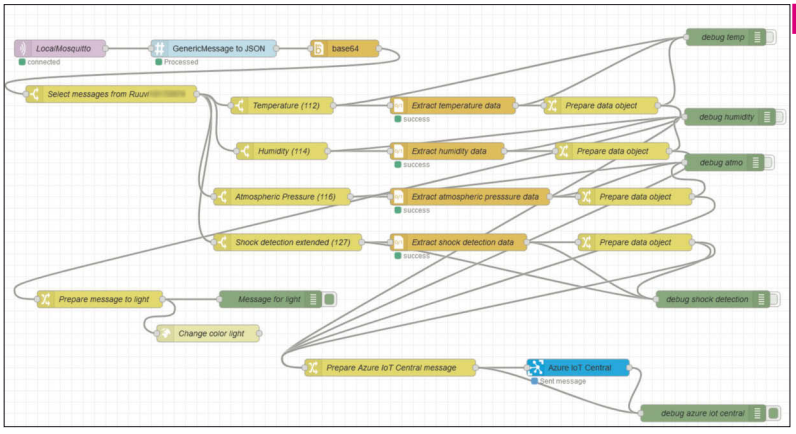
Les tableaux de bords sont rapides à construire et les graphiques sont plutôt simples à configurer. Mais le nombre de graphiques type et leur personnalisation sont plutôt limités. Par exemple, les gra-

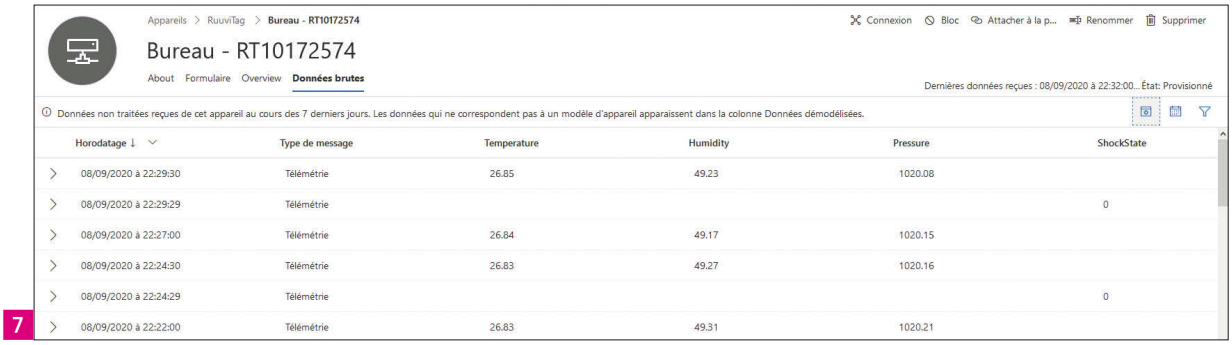


4

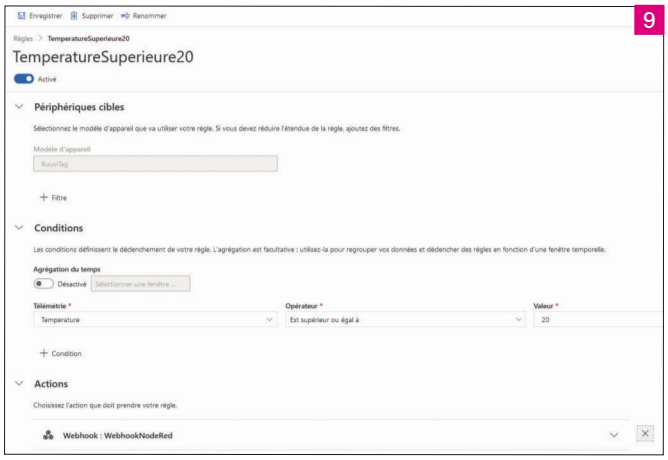


5

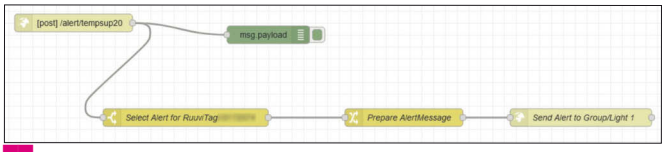

6



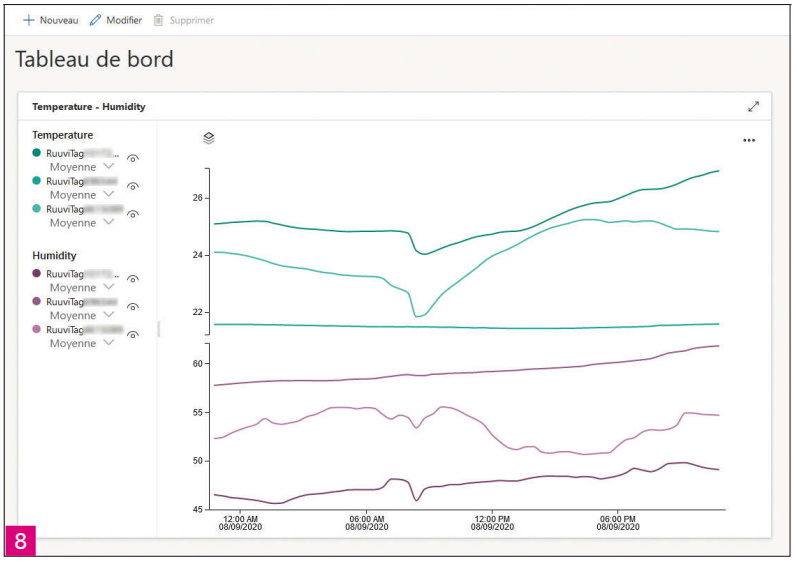
7



9



10



8

phiques affichent l'identifiant de chaque appareil mais il n'est pas possible d'afficher le nom de l'appareil : il devient assez rapidement difficile de s'y retrouver.

En ce qui concerne les règles, cela s'avère pratique pour programmer des alertes vers différents types de destinations (cloud, webhook ou email). Mais il manque un peu de finesse dans la sélection des appareils : il n'est possible de sélectionner que des modèles d'appareils ; pouvoir sélectionner des groupes ou des appareils individuellement serait plus souple.

Conclusion

Dans une situation réelle avec un grand nombre de capteurs, il sera préférable d'ajuster à la fois le matériel ainsi que certaines parties logicielles de la passerelle.

En effet, le ***Raspberry Pi4*** peut rendre beaucoup de services mais s'avère vite limité en situation réelle de montée en charge quand le nombre de messages à transformer et à envoyer commence à augmenter. Il vaut mieux se tourner alors vers des cartes mères embarquées de niveau industriel comme les **NoahV2** de Broachlink par exemple. Cela permettra, aussi, de transformer la gateway en périphérique compatible Azure IoT Edge pour du calcul et de l'analyse déportée

Du côté logiciel de la passerelle, ***Node-RED*** permet de mettre en place et de modifier rapidement des flux en mode **'low-code'**, pouvant cependant vite devenir complexes. Il faut en effet renouveler l'opération à chaque ajout de capteur, ce qui entraîne une duplication des flux : la maintenance en devient plus fastidieuse car il faudra appliquer les modifications à tous les flux manuellement. Cependant, Microsoft a mis au point un SDK, disponible pour plusieurs langages, dont C#, Python ou Javascript (NodeJS), permettant de développer soi-même des applications IoT : **Microsoft Azure IoT SDK**[5]. Il devient donc possible de développer soi-même cette étape de transformation et d'envoi des données tout en bénéficiant du provisionnement automatique disponible dans ***Azure IoT Central***. Il est aussi possible d'utiliser comme socle, Windows IoT Enterprise afin de bénéficier d'un système qui s'intègrera parfaitement dans un écosystème Microsoft existant.

Malgré des limitations vite rencontrées, Microsoft offre une solution simple et bien fournie pour démarrer une application IoT sans avoir à créer tout de zéro.

Vous l'aurez certainement remarqué et compris, ***Azure IoT Central*** est une application simple à utiliser et rapide à mettre en place s'adressant à des besoins simples : plus les besoins se complexifieront, plus il conviendra de se tourner vers le développement d'une application sur mesure répondant à votre besoin, utilisant d'autres services Azure IoT comme **Azure IoT Hub** pour la collecte en masse des données, **Time Series Insight** ou **Stream Analytics** pour leur exploitation avancée.

Références

https://wirepas.com/
https://mqtt.org/
https://mosquitto.org/
https://nodered.org/
https://github.com/Azure/azure-iot-sdks

Julien FOURRE
Consultant BI & devopps
jumar84@hotmail.fr / *chaine youtube : Passion développement*
(https://www.youtube.com/user/jumar84)

Piloter votre chaîne YOUTUBE avec vos propres KPI avec C#, AZURE et POWER BI

Si vous venez comme moi de créer votre chaîne YouTube (ou que vous souhaitez la créer), il faut pouvoir la superviser pour vérifier que le contenu intéresse une population de plus en plus nombreuse pour qu'elle puisse continuer à vivre dans le temps.

Si vous êtes une entreprise, beaucoup utilisent YouTube pour faire la promotion des produits ou des compétences de leurs collaborateurs. Il faut dans ce cas-là identifier les vidéos de promotion que vous produisez afin de différencier celles qui marchent et celles qui ne marchent pas. YouTube est en effet une formidable vitrine avec 2 milliards d'utilisateurs chaque mois et 80 000 vidéos vues par seconde. YouTube propose évidemment des choses pour superviser l'ensemble, avec des KPI spécifiques. Mais on se retrouve rapidement limité par rapport aux analyses que l'on souhaite faire. On peut, avec Azure (un déclencheur, un traitement, du stockage), du code (C#) et l'outil de dataviz power bi réaliser une architecture à moindre coût avec un processus 100 % automatisé. Ensuite, vous aurez simplement à vous concentrer sur la publication de vos vidéos et leur contenu, le pilotage des statistiques de votre chaîne tournera tout seul.

Voilà tous les outils/services managés/langages utilisés

Côté dev :

- Projet de type « ASP.NET Web Application (.net framework) » et le langage C# pour l'API REST ;
- Librairies google : Google.Apis.YouTube.v3 (via le gestionnaire de package Nugets) ;
- L'IDE est Visual Studio 2019 en édition community.

Côté YouTube :

- Création d'un compte sur https://console.developers.google.com pour attaquer l'api rest de YouTube ;
- Lecture du contrat d'interface des fonctions proposées : https://developers.google.com/youtube/v3/docs .

Côté Azure :

- Service « base de données SQL » (mode PASS) pour le stockage des données et la construction du modèle en étoile ;
- Service « App service » web (mode PASS) pour l'hébergement de l'API REST ;
- Service « Azure fonction » (mode PASS) pour la synchronisation des données avec YouTube en mode journalier.

Côté power bi :

- Création du rapport avec « Power BI Desktop » et ses KPI ;
- Mode direct query avec la base de données ;
- Publication du rapport en ligne et partage du rapport.

1

En prérequis :

- Un poste de dev/travail classique suffira (j'ai utilisé Windows 10 entreprise 64 bits, 16 Go de RAM ; CPU 4 cœurs) ; aucun programme ne tournera en local au final ; simplement en DEBUG au début ;
- Une connexion internet depuis ce même poste ;
- Visual Studio 2019 community : https://visualstudio.microsoft.com/fr/vs/community/
- Un compte Azure gratuit : https://azure.microsoft.com/fr-fr/free/ ou bien un compte Azure que vous avez avec votre entreprise. Pour ma part, j'ai un compte Azure avec un peu plus de 100 dollars par mois par le biais de mon employeur Infeeny, ce qui est largement suffisant ;
- De bonnes connaissances en C# (programmation objet), entity framework (librairies de connexion vers la base de données) et API REST C# (si vous voulez apprendre, ne pas avoir peur, il existe beaucoup de tutoriaux sur le net) ;
- Des connaissances la Business intelligence (modèle en étoile, language DAX pour Power BI)
- Des connaissances en dataviz avec Power BI
- Un gestionnaire de configuration de type Git ou Azure devops pour backup de l'ensemble et faire du versionning ; surtout utile si vous travaillez à plusieurs ;
- Pour faire du code et de l'architecture, je dirai qu'il m'a fallu 7 ou 8 jours pour réaliser le tout (montée en compétence + la réalisation).

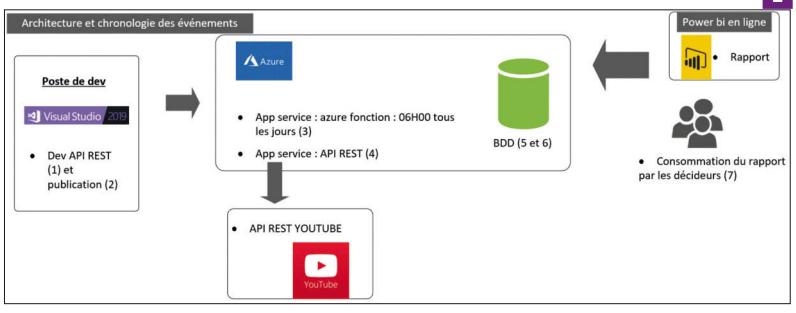
Processus de réalisation :

Dev api rest :

La première étape, une fois Visual Studio 2019 installé, est de créer le WEBSERVICE de type « ASP. NET Web application (net application) » ; le projet aura la propriété « Web api », puisqu'on aura uniquement des traitements back office.

Le rôle du WS sera de se connecter à l'api YouTube, d'en récupérer les statistiques (sur votre chaîne YouTube) et d'enregistrer les datas

1



dans la base de données dans azure avec un stockage brut et un modèle en étoile construit préalablement.
Avant d'attaquer le code, il faut créer un compte sur https://console.developers.google.com/. Celui-ci permettra à l'api rest de s'authentifier sur l'api YouTube.
J'ai utilisé mon compte Hotmail, celui que j'ai utilisé sur YouTube. Vous avez besoin d'un nom d'application qui pointera sur votre chaîne YouTube (« Passion programmation » pour moi) et d'un API_KEY, qui sera référencé dans le code appelant.
La documentation de l'api YouTube et de ses nombreuses fonctions est ici : https://developers.google.com/youtube/v3/docs/?apix=true. On peut lancer les fonctions en ligne, ce qui est vraiment très pratique.
Pour le projet dans Visual Studio : il faut ajouter des librairies via le gestionnaire de package :

- Entityframework by Microsoft ;
- Google.Apis.Youtube.v3 by Google Inc ;
- Swashbuckle by Richard Morris si vous voulez partager votre WS avec d'autres systèmes appelants avec une interface swagger.

Créer d'abord une classe qui va manipuler une classe Google : YouTubeService

```
using Google.Apis.YouTube.v3;

public class AttaqueYoutubeApi
  {
    // id de votre chaîne youtube
    private static string channelId= "UCjufGYFITwn0ZwqiA";

    //object qui permet d'attaquer l'api YouTube et ses nombreuses fonctions
    private YouTubeService youtubeService = null;


    public AttaqueYoutubeApi()
    {
       youtubeService = new YouTubeService(new BaseClientService.Initializer()
       {
//paramétres créés sur https://console.developers.google.com
         ApiKey = ConfigurationManager.AppSettings["YoutubeApiKey"],
         ApplicationName = ConfigurationManager.AppSettings["YoutubeApplicationName"]
       });
    }
```

A partir de cet objet, on peut attaquer toutes les fonctions de l'interface YouTube.
Voici la liste complète de tout ce que l'on peut appeler : https://developers.google.com/youtube/v3/docs.
Sur chaque fonction de l'interface, il existe une fenêtre « try this api » sur la droite de la page. On peut passer les paramètres d'entrée et voir la sortie avec un exemple concret en Json. Très pratique.
Plusieurs choses sont à noter. On ne peut pas récupérer tout ce que l'on souhaite sur un seul appel ni sur une seule fonction. On est limité sur plusieurs aspects : le nombre de résultats retournés et le nombre d'appels que l'on fait sur une période. La cible est de consolider l'ensemble en base de données.
On utilise 4 fonctions de l'api YouTube : une pour récupérer l'ensemble des vidéos d'une chaîne, une pour les statistiques de la chaîne, une pour les statistiques de chaque vidéo et une pour les catégories. Voici leur contrat d'interface simplifié avec uniquement les champs que l'on récupère :

Fonction liste des vidéos de la chaîne :
GET https://www.googleapis.com/youtube/v3/search

Nom param IN	Format param In
channelId	string
dateDebut	DateTime
dateFin	DateFin
MaxResults	Integer

Nom param OUT (json format)	Format param Out (json format)
channelTitle	string
Description (channel)	string
totalResults	integer
resultsPerPage	integer
videoId (liste d'items)	string
kind (type de l'item)	string
PublishedAt	DateTime (date de publication de la video)

Code http 200 (ok) ou 400 (bad request) pour chaque fonction.

Fonction statististiques des vidéos de la chaine :
GET https://www.googleapis.com/youtube/v3/videos

Nom param IN	Format param In
VideoId	string
MaxResults	Integer

Nom param OUT (json format)	Format param Out (json format)
ViewCount	integer
LikeCount	integer
DislikeCount	integer
CommentCount	integer
CategoryId	String (catégorie video)

Fonction récupération de la catégorie de chaque vidéo de la chaine :
GET https://www.googleapis.com/youtube/v3/videoCategories

Nom param IN	Format param In
Id	String (id de la catégorie)

Nom param OUT (json format)	Format param Out (json format)
Title	String (libellé de la catégorie)

Fonction récupération de la catégorie de chaque vidéo de la chaine :
GET https://www.googleapis.com/youtube/v3/videoCategories

Nom param IN	Format param In
Id	string

Nom param OUT (json format)	Format param Out (json format)
Title	String (libellé de la catégorie)

Idem pour les codes http.

Fonction récupération des statistiques de la chaine :
GET https://www.googleapis.com/youtube/v3/channels

Nom param IN	Format param In
Id	String (id de la chaîne youtube)

Nom param OUT (json format)	Format param Out (json format)
SubscriberCount	integer (nombre d'abonnés de la chaîne)

Le principe reste le même à chaque fois : un exemple de code ici avec la fonction Search (liste des vidéos) que l'on va trouver ici : https://developers.google.com/youtube/v3/docs/search/list

```
//fonction de recherche de toutes les vidéos d'une chaîne entre deux dates
public async Task<Dictionary<string, ModelYoutubeItems>> rechercheItems(DateTime dateDebut,
        DateTime dateFin)
    {

       Dictionary<string, ModelYoutubeItems> mapItems = new Dictionary<string, ModelYoutubeItems>();

        var searchListRequest = youtubeService.Search.List("snippet");
        searchListRequest.ChannelId = channelId;
        searchListRequest.PublishedAfter = dateDebut;
        searchListRequest.PublishedBefore = dateFin;
        //impossible de récupérer plus de 50 résultats de toute façon
        searchListRequest.MaxResults = 50;

//Appel de l'api rest
      var searchListResponse = await searchListRequest.ExecuteAsync().ConfigureAwait(false);



        // boucle sur tous les résultats
        foreach (var searchResult in searchListResponse.Items)
        {
          ModelYoutubeItems model = new ModelYoutubeItems();
//Récupération de tous les champs qui nous intéressent
          model.ChannelId = searchResult.Snippet.ChannelId;
          model.ChannelTitle = searchResult.Snippet.ChannelTitle;
          model.Description = searchResult.Snippet.Description;
          model.PublishedAt = searchResult.Snippet.PublishedAt;
          model.Title = Utiles.gestionCaracteresSpeciaux(
            searchResult.Snippet.Title);
          model.VideoId = searchResult.Id.VideoId;
          model.Thumbnails = searchResult.Snippet.Thumbnails.Default__.Url;
 //Ajout à la liste de référence
          mapItems.Add(searchResult.Id.VideoId, model);
    }
return mapItems ;
  }}
```

Le programme obtiendra une erreur si le nombre de vidéos est supérieur à 50. Mettre des plages de dates plus petites si vous publiez beaucoup de vidéos. Attention aux caractères spéciaux aussi dans les noms des vidéos et aux champs null.
Il faudra ensuite appeler d'autres fonctions pour avoir le nombre de likes, le nombre de commentaires, le nombre d'abonnés.
Pour chaque contrôleur, on retourne le code http (200 si ok), l'objet de sortie et le format (ici Json).

```
//Retour controleur
return Request.CreateResponse(HttpStatusCode.OK,
          v.MapItemsGlobale.Values, Configuration.Formatters.JsonFormatter);
```

Pour l'écriture en base de données, entityframework est l'outil idéal, voici un exemple de code que vous pouvez utiliser :

```
public static int insertionYoutubeBDD
        (List<ModelYoutubeItems> modelListe, int currentThreadId)
    {
//Contexte BDD : vous avez accès aux tables, aux procédures stockées, …
        using (var context = new BDD_JFO_AZUREEntities())
        {
          try
          {
            List<youtube_video> listInsertion = new List<youtube_video>();
            foreach (ModelYoutubeItems model in modelListe)
            {
//mapping champs métiers avec les champs de la table, petites transformations/conversions si besoin
              youtube_video i = new youtube_video();
              i.channelTitle = model.ChannelTitle;
              i.commentCount = (model.CommentCount == null) ? 0 : (int)model.CommentCount;
              i.description = model.Description;
              i.dislikeCount = (model.DislikeCount == null) ? 0 : (int)model.DislikeCount;
              i.likeCount = (model.LikeCount == null) ? 0 : (int)model.LikeCount;
              i.thumbnails = model.Thumbnails;
              i.Categorie = model.Category;
              i.title = model.Title;
              i.viewCount = (model.ViewCount == null) ? 0 : (int)model.ViewCount;
              listInsertion.Add(i);


            }
//ajout d'une liste complète (ne pas mettre trop d'éléments à chaque appel, trouver le juste
 milieu pour le temps optimal d'insertion de l'ensemble)
            context.youtube_video.AddRange(listInsertion);
//commit en base, on retourne le nombre de lignes insérées
            return context.SaveChanges();
          }
```

Ne pas oublier de créer une table de logs en base de données et d'y mettre les différentes étapes de traitements (appel de la fonction xxxx, temps de traitements, exception si problème, …) que l'on pourra superviser.

Publication du web service sur Azure

Une fois le webservice développé en local, il faut le mettre sur Azure. Au préalable, il faut créer un service « App service » sur le portail. Ce service hébergera le site web du type API REST. On peut choisir la pile d'exécution : Node js, Asp net, Python, Ruby, Net core, Java sous Linux ou Windows. On peut également pousser une image Docker. Une supervision est proposée pour voir les différents appels http. On peut également choisir la typologie de machine : dev/test, production, … On peut ajuster la mémoire et le stockage : bref ce que l'on appelle le scale up/scale down. Cela permet d'ajuster (en fonction de la charge du site) la taille de la machine pour conserver le meilleur service au client. Ce service app peut être arrêté et démarré à souhait. Il ne faut pas oublier que si on appelle le code une fois par jour, autant ne pas payer le reste du temps

avec un service non utilisé. Une fois le service créé, on peut publier (« publish ») le projet avec Visual Studio. Vous pourrez, avec le wizard, vous connecter avec votre compte Azure. On choisit le service app service web créé précédemment. On peut publier avec via un zip, une connexion ftp, répertoire partagé ou webdeploy.

Appel des fonctions du web service avec Azure fonction

Le code est maintenant publié. Il faut dès lors appeler le traitement en automatique. Pour cela, il existe ce que l'on appelle « Azure fonction ». C'est une application dite serverless. En réalité, c'est un morceau de code qu'on lance sur un serveur que l'on ne maintient pas. Pour cela, il faut créer dans un premier temps une application de fonction.

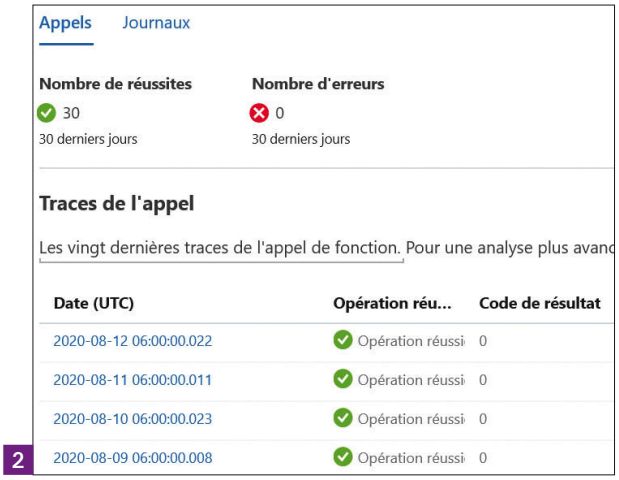
On choisit la stack technique d'exécution (powershell, .net core, …), comme pour l'app service du paragraphe précédent. Ensuite, on crée la fonction. Elle peut être déclenchée sur plusieurs évènements : http triggers, timer trigger, blob storage trigger, cosmos trigger,… J'ai utilisé le timer trigger : à heure fixe donc. C'est une expression Crontab qui permet le lancement de la fonction.

On peut créer sa fonction Azure avec un projet dans Visual Studio. On peut la créer aussi en ligne.

Voici un exemple du code Powershell :

```
# Input bindings are passed in via param block.
param($Timer)

# Get the current universal time in the default string format.
$currentUTCtime = (Get-Date).ToUniversalTime()

# The 'IsPastDue' property is 'true' when the current function invocation is later than scheduled.
if ($Timer.IsPastDue) {
    Write-Host "PowerShell timer is running late!"
}

Write-Host "AZURE FONCTION lancement début: $currentUTCtime"
Write-Host "Appel de l'api rest"
# avec cette fonction, on appelle une fonction du Web service
Invoke-WebRequest -Uri https://pocyoutubesapplicationwebservice.azurewebsites.net/api/xxxxx/NomFonction?channelId=UCp9mU94-_J-32aClu -Method GET  -ContentType "application/json" -TimeoutSec 0
Write-Host "Appel de l'api rest AZURE effectué "

Write-Host "AZURE FONCTION lancement effectué: $currentUTCtime"
```

Il faut bien penser à loguer dans le code des messages dans la sortie standard. On peut superviser l'ensemble des traitements et voir les logs successifs (info, erreur, code retour, json output, …) 2



2

Hébergement des données et modèle BI

La fonction déclenche l'appel. Le webservice écrit dans une base de données sur Azure. Il faut pour cela créer un serveur de base de données avec le service « SQL DATABASE ». On choisit un nom de base de données, un nom de serveur, un type de machine (nombre de cpu, stockage, …), une zone d'hébergement. On peut aussi changer ces caractéristiques après la création. SQL database donne la possibilité de filtrer l'accès à ses bases de données à une ip ou plage d'ip. On peut répliquer les bases de données sur un autre datacenter chaque jour.)

Il existe un outil de consultation des données/création des tables/procédures stockées et vues SQL en ligne Personnellement, je préfère utiliser sql server management studio.

Pour préparer la donnée de manière à pouvoir en faire l'analyse via un rapport power bi, je conseille de faire un modèle en étoile avec une table de fait et des dimensions. Il faut bien prendre le temps de le concevoir en amont. Bien se poser les bonnes questions : quels sont les indicateurs dont j'ai besoin ? Quels sont les axes d'analyses de mes mesures ? Il faut construire ce qu'on appelle une matrice de bus, c'est-à-dire le lien entre les indicateurs et les axes d'analyses ; ni plus ni moins.

Ici, une vidéo (dimension dim_video) est associée à une date photo et une date de publication (dimension dim_date), à une chaîne youtube (dimension dim_channel). Côté SQL pur, la table de fait contient des clés étrangères vers les dimensions et une valeur (champ fact_value). 3

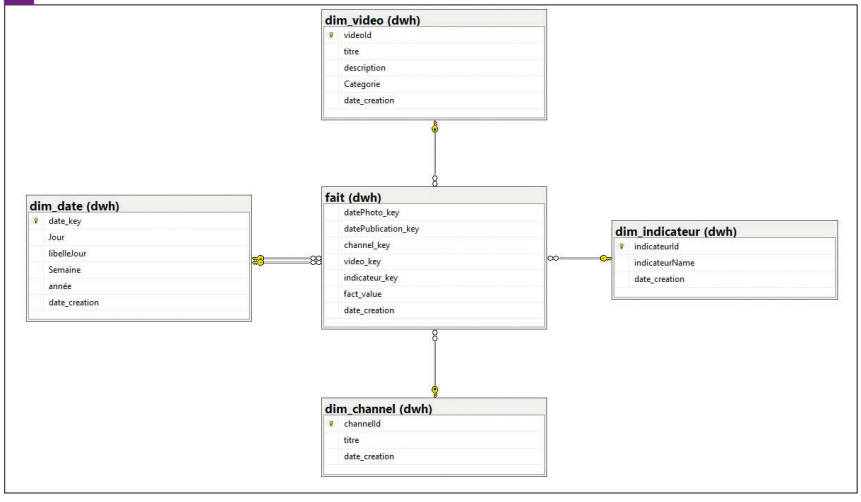
Pour une vidéo donnée, on aura plusieurs lignes dans la table de fait car plusieurs indicateurs sont utilisés.

Dans la table dim_indicateur, on aura le Nb vues vidéo, Nb likes vidéo, Nb dislikes vidéo, …

On stocke à chaque appel la quantité totale des données par date photo de toutes les vidéos.

Ce modèle sera alimenté tous les jours via la fonction Azure.

3



La partie Dataviz avec Power BI

Le modèle BI dans la base de données Azure est en place et alimenté tous les jours. Il faut maintenant passer de données stockées à un affichage simple qui permettra de piloter l'ensemble au quotidien. C'est le rôle de Power BI dans le dispositif. L'outil est gratuit en téléchargement, avec des nouvelles versions tous les mois. Les supports et tutoriels sont très nombreux. Il est simple à utiliser dans la plupart des cas d'usages. De nombreux connecteurs (> 80) sont proposés pour récupérer vos datas : csv, Excel, Sharepoint, SQL Server, Azure, boite mail, Google Analytics, … (graphique en dessous à gauche). Il existe également de nombreux visuels par défaut (graphique en dessous à droite). Il existe d'autres visuels en ligne. On peut même en développer et les pousser en ligne pour les partager 4

La première étape est d'importer les données depuis la base de données sous Azure (en mode direct query). Chaque table, lors de l'import, est dans un flux séparé. Les clés étrangères en base permettent d'avoir les relations entre les flux. C'est ce que l'on appelle les relationships. Par exemple, on va lier une vidéo, une chaîne, une date de publication et une mesure. Une fois importé, on renomme les tables dans Power BI pour pouvoir avoir les axes d'analyse dans l'ordre que l'on veut (c'est l'ordre alphabétique). Il faut masquer les champs que l'on ne va pas utiliser : date de création ou de modification de la ligne SQL en base par exemple.

On peut créer ses propres indicateurs avec un langage qui s'appelle le DAX. Il est également possible de créer des indicateurs « rapides » sans connaissance DAX (chiffres d'affaires cumulés depuis le début de l'année, …) avec des indicateurs templates proposés. Il existe également beaucoup de tutoriels sur le net, on trouve facilement ce que l'on veut. Voici un exemple de quelques formules que j'ai mises en place :

```
Nb vues vidéo = CALCULATE(SUM(Mesures[fact_value]), dim_05_INDICATEUR[indicateurId]
=1,dim_03_VIDEO[Categorie] <> "N\A")

Nb de vues de la veille = CALCULATE([Nb vues vidéo], PREVIOUSDAY( dim_02_DATE_
PHOTO[Jour date photo]))

Nb de vue en plus sur la journée = if( ISBLANK( [Nb de vues de la veille]), BLANK(), CALCULATE
([Nb vues vidéo]-[Nb de vues de la veille]))
```

Avec le DAX et Power BI, l'autocomplétion est en place et est très utile. La liste des fonctions est très fournie : temporelle, texte, filtrage, mathématique, statistiques, … Tout est ici : https://docs.microsoft.com/fr-fr/dax/dax-function-reference#. Les mots clés calculate, isblank, totalytd, all, previousday, sum, max reviennent très souvent.

Une fois les formules créées, on peut construire les graphes dans les différents onglets. Les plus fréquents sont les segments (filtre), histogramme empilé, table, matrice, graphique en secteurs, … Le choix est vaste.

Les onglets et les graphes créés en local, il faut publier le rapport pour pouvoir le partager à d'autres. Il faut un compte office 365 pour le faire. Un bouton « publier » est présent en haut à droite, on se connecte avec son compte email et la publication est effective. C'est dans le rapport en ligne que l'on pourra partager à quelqu'un de votre entreprise.

Voilà quelques exemples de graphiques que j'ai faits : 5

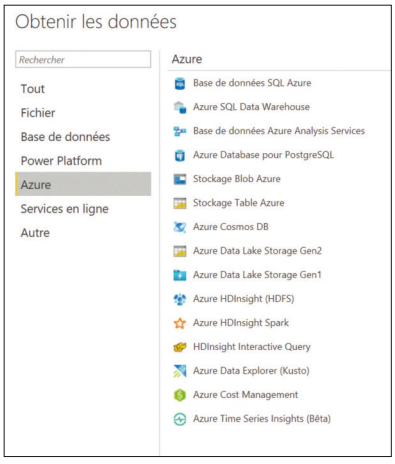
Une première page de synthèse globale ci-dessus. 6

Le service marketing pourra utiliser l'onglet ci-dessus pour voir le détail des vidéos par année de publication, par ordre de vues. On peut imaginer avoir l'évolution des vues jour par jour de la chaîne, en intégrant une sous-catégorie pour chaque vidéo (qui n'existe pas dans YouTube) pour voir les vidéos qui marchent le mieux : vidéo sur teams, vidéo programmation, vidéo retour d'expérience client, … La liste des possibilités est celle que l'on se fixe !

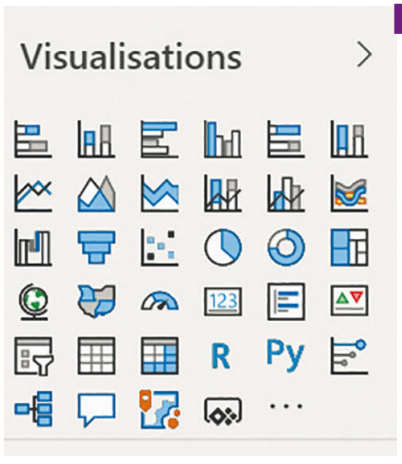
Pour aller plus loin

On peut imaginer placer les sources de l'api REST, SQL pour la BDD et pour la fonction Azure sous un gestionnaire de configuration comme Git ou Azure Devops. Mettre en place, ensuite, un CI/CD, c'est-à-dire une intégration continue et déploiement du code à chaque commit (modification du code) sur Azure. L'objectif est de se concentrer sur le code pour les développeurs et de diminuer les actions manuelles.

On peut également piloter la chaîne dans son ensemble, c'est-à-dire avoir dans un autre Power BI, orienté technique, une supervision de l'ensemble de la chaîne : le lancement de la fonction Azure et de ses logs, le nombre de lignes insérées en base de données, les performances du serveur de la base de données, les statistiques d'utilisation du rapport, … La liste est non exhaustive :-)



4



6

Jour date photo: jeudi 13 août 2020 — année de publication: 2020

année de publication	Categorie	titre	Nb vues vidéo
2020	Science & Technology	Infeeny Back from IGNITE 2019	23
2020	Science & Technology	Back To Ignite 2019	27
2020	Science & Technology	[Learn and Lunch] Kubernetes and AKS par Infeeny	50
2020	Science & Technology	Architectures Serverless : Nouveau buzzword !	51
2020	Science & Technology	[Learn and Lunch] Infrastructure as Code	82
2020	Science & Technology	Tutoriel Microsoft Teams 'How to Live'	100
2020	Science & Technology	VS2019 Windows Containers	130
2020	Science & Technology	Tutoriel DATA : votre premier cube SSAS depuis ze'ro	179
2020	Science & Technology	AccorInvest : Un nouveau SI avec le Cloud Microsoft	292
2020	Science & Technology	Guide du de'butant : 10 conseils pour Microsoft Teams	1574
2020	Science & Technology	Les 5 erreurs à éviter sur Microsoft TEAMS	30577
Total			33085

5

Guillaume Verret
Titulaire d'un diplôme d'expert en technologie de l'information à l'EPITECH, Guillaume intègre SoftFluent à l'issue de son stage de fin d'études en 2018. Aujourd'hui consultant .NET, il intervient sur un projet lui permettant d'exploiter des technologies telles que C#, .NET Framework, .NET Core et Angular.

Edge sous Chromium : y a-t-il du changement pour les développeurs ?

Microsoft a sorti le 15 janvier 2020 une nouvelle version de son navigateur internet : **Microsoft Edge Chromium**. D'abord construit sur son moteur de rendu EdgeHTML, Microsoft l'a abandonné au profit du projet open source Chromium. Ce navigateur, successeur d'**Internet Explorer**, étant né après quelques controverses, une mise en perspective est nécessaire.

Petit historique

Chromium est un projet open source initié par Google en 2008. Ce projet est composé en réalité de 3 gros projets : il y a **Chromium**, le navigateur internet proprement dit ; **Blink (**un fork de WebKit, NDLR), le moteur de rendu et d'exécution Javascript (« V8 ») dans ce navigateur ; et enfin Chrome OS, le système d'exploitation.

Etant donné que Chromium est un projet Open source, quiconque peut partir de ce projet pour créer son propre navigateur. Aujourd'hui on ne compte pas moins de 20 navigateurs qui fonctionnent grâce à Chromium, dont maintenant Microsoft Edge.

1 *Logo Microsoft Edge Chromium*

Une des questions que l'on peut se poser aujourd'hui est : pourquoi Microsoft a abandonné son moteur maison EdgeHTML, pour passer sous Chromium ? 1

En 2015 Microsoft annonce la fin d'**Internet Explorer** et dans le même temps la création d'un tout nouveau navigateur Internet : **Microsoft Edge**. Celui-ci sera basé sous un nouveau moteur de rendu, EdgeHTML, plus respectueux des standards du web contrairement à son grand frère. Malheureusement en 2019, la firme de Redmond prend tout le monde de court en annonçant le portage de ce navigateur **Microsoft Edge** vers Chromium. La vision de l'éditeur est claire : « *créer une meilleure compatibilité Web pour les clients et une fragmentation moindre du Web pour tous les développeurs* ». Ce que l'éditeur de Windows omet de dire, c'est que son navigateur n'avait que 3,77% de part de marché contrairement à **Google Chrome**, son concurrent direct, qui en avait 68%. (1)

L'échec de **Edge**, est dû à plusieurs facteurs. Le plus important étant l'arrivée très tardive de **Edge** dans ce marché. Son concurrent principale, Google Chrome, est arrivé en 2008 il a donc pu construire son catalogue d'extensions au fil des années contrairement à **Edge**. En 2019 **Edge** comptait seulement 119 extensions ce qui est infime comparé à Chrome qui en compte des milliers. Autre point important : la réputation d'**Internet Explorer** est venue entacher celle de **Edge**. En effet des failles de sécurité d'**Internet Explorer** sont aussi apparues dans ce nouveau navigateur(2). Il n'en fallait pas plus pour entacher la réputation de **Edge** et se demander combien de problèmes perdureraient dans celui-ci.

Nous voilà donc avec un tout nouveau navigateur, et qu'est-ce que cela implique pour les développeurs ?

Pour le développeur

Pour un utilisateur lambda, ce navigateur a le même rendu que son concurrent **Google Chrome**, moyennant la mise en avant des services tiers de Microsoft 365 à la place des services tiers de Google.

Pour les développeurs, le premier bénéfice de sites Web Internet/Extranet est une moindre charge de travail et de tests pour le support des navigateurs principaux du marché, si et seulement si, à terme, le pari de Microsoft d'un bon déploiement effectif de son navigateur réussit. Mais intéressons-nous au plus court terme.

Dans les outils de développement, on retrouve les classiques présents dans de nombreux navigateurs : panneau des Eléments (HTML), la Console, un panneau Sources, un panneau Réseau, un panneau Performances, un panneau Mémoire, un panneau d'Application et un volet de Sécurité. Mais un petit nouveau est apparu, l'outil de Google : LightHouse. 2

LightHouse : LightHouse est un outil développé par Google. Déjà présent dans Google Chrome, il permet de faire un audit simple et rapide de votre page web. Il passe en revue 5 caractéristiques de celle-ci. L'analyse de type ***Performances*** va mesurer la « vitesse » du site internet en se basant sur 6 critères :

First Contentful Paint : mesure le temps nécessaire à l'affichage du premier contenu de la page (texte ou image).

First Meaningful Paint : mesure le laps de temps entre le chargement de la page et l'affichage des principaux contenus du site Web.

Speed Index : indique la rapidité avec laquelle le contenu de la page web est affiché.

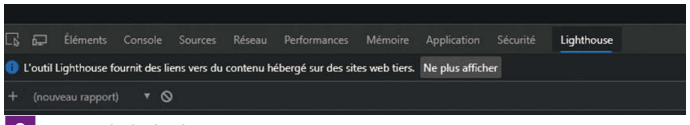
Time To Interactive : mesure le laps de temps entre le chargement de la page et l'activation de ses fonctions d'interaction.

First CPU Idle : indique l'heure à laquelle, pour la première fois, l'activité du thread principal du site Internet est suffisamment faible pour permettre le traitement des saisies de l'utilisateur.

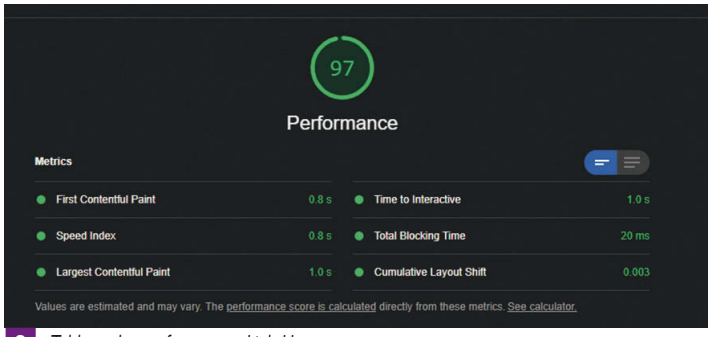
Estimated Input Latency : donne une estimation du nombre de millisecondes qu'il faudra à la page pour réagir aux saisies de l'utilisateur. Lorsque la latence

(1) https://netmarketshare.com

(2) https://www.lemonde.fr/pixels/article/2017/02/27/une-faille-informatique-serieuse-decouverte-dans-internet-explorer-et-microsoft-edge_5086422_4408996.html



2 *Les outils de développement.*



3 *Tableau des performances LightHouse.*

est supérieure à 50 millisecondes, les utilisateurs trouvent souvent la page ou l'application lente. 3

En plus d'analyser ces aspects, LightHouse fournit des suggestions pour l'amélioration du site. Ces suggestions sont, pour la plupart, des indications d'optimisation ayant pour objectif la réduction du temps de chargement. Par exemple : compression et taille limite des images, fourniture des textes et Javascript dans des formats compressés, etc.

Dans le domaine des bonnes pratiques, l'outil analyse principalement les aspects liés à la sécurité des sites web. Il contrôle ici l'utilisation de technologies de chiffrement telles que TLS, la sécurité des sources des ressources intégrées au site web, ou si les bibliothèques JavaScript peuvent être classées comme sécurisées. 4

L'analyse de type ***Application Web Progressive*** (PWA) est une fonctionnalité intéressante de Google LightHouse : elle permet d'analyser le fonctionnement de votre site attendu s'il cible une architecture PWA(3). Il va vérifier l'affichage de tous les éléments et des contenus dynamiques ainsi que si tous les prérequis de PWA sont respectés. L'utilisation de l'onglet **Application** permet alors de debugger les aspects locaux importants des applications PWA : manifeste, workers, stockage et cache local (voir plus loin).

Dans la catégorie de l'***Accessibilité***, Google LightHouse examine dans quelle mesure le site Web peut être utilisé par des personnes ayant un handicap. Il vérifie de manière approfondie que les boutons et les liens sont correctement décrits, ou encore que les images et les graphiques disposent d'une fonction de « voix off » pour permettre aux personnes malvoyantes de comprendre les contenus via des messages vocaux. 5

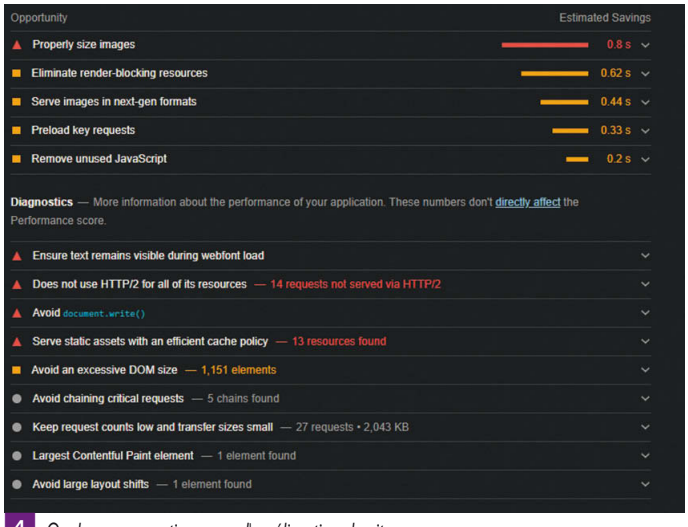
Enfin la dernière catégorie de LightHouse est le ***SEO***. Ces tests analysent dans quelle mesure le site Web sera pris en compte par les différents moteurs de recherche. Ces tests sont assez limités contrairement à la page de performance qui est ultra détaillée. Ici on ne retrouve qu'un score. 6

Microsoft Edge Driver

Un second outil est apparu à la suite du portage de Edge sous Chromium. C'est le WebDriver Microsoft Edge Driver. WebDriver permet aux développeurs de créer des tests automatisés qui simulent l'interaction de l'utilisateur. Si vous utilisez Sélénium 3 ou Sélénium 4 pour vos tests sur Google Chrome, il faut télécharger et configurer le Web Driver destiné à Microsoft Edge. Je vous invite à regarder la documentation de Microsoft(4) pour mettre en œuvre de tels tests. Une particularité supplémentaire de ce WebDriver est de permettre d'utiliser les propriétés et les méthodes propres à Chromium. 7

(3) https://docs.microsoft.com/fr-fr/microsoft-edge/progressive-web-apps-chromium/

(4) https://docs.microsoft.com/fr-fr/microsoft-edge/webdriver-chromium?tabs=c-sharp



4 *Quelques suggestions pour l'amélioration du site.*

WebViewer

Pour les développeurs d'applications **Windows Presentation Foundation (WPF)**, la nouvelle version du contrôle WebViewer (package Microsoft.Web.WebView2), s'appuie sur le moteur de rendu de Microsoft Edge Chromium pour afficher des pages Web de façon embarquée dans vos applications. Ces évolutions du navigateur sont une conséquence du portage de Edge vers Chromium mais **Microsoft** n'en n'oublie pas les nouvelles technologies Web. Notamment les WebAssembly et les Progressive Web App (PWA).

Débogage pour les applications Progressive Web App

Edge Chromium, permet de déboguer des applications PWA, un écran dans les outils de développements leur étant entièrement dédié. Il se trouve dans l'onglet « Application ».

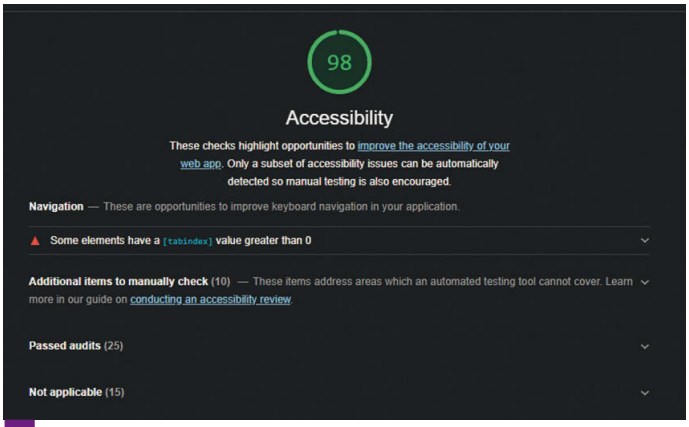
Mais avant de se demander comment déboguer une Progressive Web App, il faut remettre un peu de contexte.

Qu'est-ce qu'une Progressive Web App ? C'est une version optimisée d'un site web mobile intégrant des fonctionnalités d'applications natives (normalement indisponibles sur un navigateur). Les PWA combinent ainsi les fonctionnalités des applications mobiles et le meilleur des technologies Web.

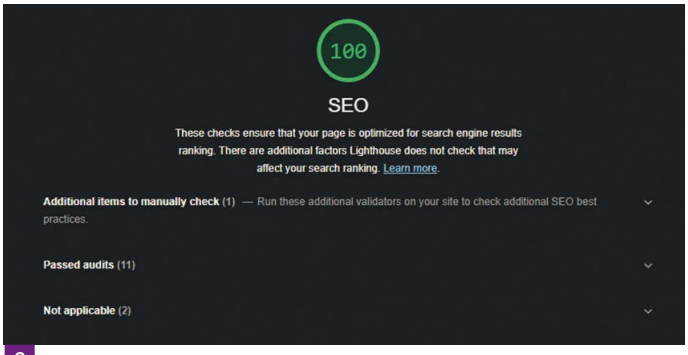
Elles permettent par exemple de créer un raccourci sur la page d'accueil d'un smartphone, de recevoir des notifications push, d'accéder aux fonctionnalités du téléphone comme l'appareil photo, la géolocalisation ou le micro et d'atteindre du contenu hors-connexion. 8

```
var options = new EdgeOptions();
options.UseChromium = true;
options.AddArgument("headless");
options.AddArgument("disable-gpu");
```

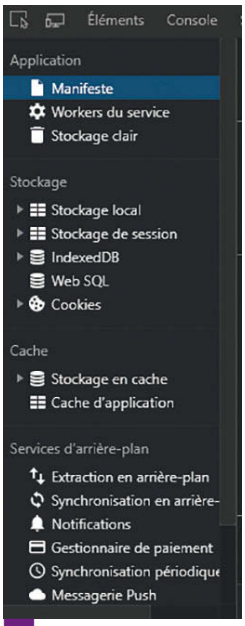
7 *Option spécifique à Chromium.*



5 *Analyse de l'Accessibilité.*



6 *Analyse du SEO.*



8

L'onglet «Application» se décompose comme ceci : un menu avec plusieurs catégories comme Application, Stockage, Cache et les services d'arrière-plan.

Manifeste

Le manifeste est obligatoire si vous souhaitez que vos utilisateurs puissent ajouter votre application à leur mobile. Il va définir les icônes utilisables selon les mobiles pour le raccourci de l'application, l'apparence de l'application au moment du lancement et enfin la page d'accueil au lancement de celle-ci. C'est un fichier JSON référencé dans les pages du site en question.

Le Worker de service

Les **Workers**, sont des scripts JS que le navigateur exécute en arrière-plan. Ces scripts permettent d'exploiter des fonctionnalités qui n'ont pas besoin d'une interaction utilisateur, comme les notifications ou la synchronisation en arrière-plan des données nécessaires pour permettre un mode déconnecté. Si un **Worker** est installé sur la page actuellement ouverte, il va apparaître dans ce volet. Dans l'illustration qui suit, le worker se trouve dans le fichier **service-worker.js** et il est limité au site https://webpushdemo.azurewebsites.net/.

Pour forcer le **Worker** à procéder à une mise à jour à chaque rechargement, il suffit de cocher la case correspondante. De même cocher « Hors ligne » permet de le déboguer sans réseau. Ultra pratique pour debugger en local une application PWA. Enfin pour émuler une notification, vous pouvez cliquer sur « Emission ». Attention, la notification ne contient aucune donnée. Pour recevoir des notifications avec des données, je vous invite à lire cette documentation(5). **Workers de service** est l'endroit le plus important pour inspecter et déboguer vos **Workers**.

Enfin la section **Caches** fournit une liste en lecture seule des ressources qui ont été mises en cache par les workers de service.

Cet onglet est un outil très complet pour le débogage de ce type d'application. Il permet notamment le débogage, en cas de notifications, des évènements de synchronisation, et la mise au point du fichier manifeste.

Je vous invite pour approfondir cet article à lire la documentation de Microsoft à ce sujet(6). Le navigateur supporte aussi les WebAssembly de façon similaire à Chrome. Si vous décidez de développer votre site web en utilisant **Blazor Client Side**, c'est donc maintenant une cible de choix : je vous invite à lire l'article à ce sujet présent dans ce hors-série. Malgré ces bons outils pour le développement, une question reste toujours en suspens.

Quel est l'impact de Edge Chromium dans l'avenir du Web ?

Imaginons quelques instants que les navigateurs basés sur Chromium ont une part de marché importante, il devient alors plus facile pour les développeurs Web et les entreprises de décider de cibler ces navigateurs pour leurs services et leurs sites pour diminuer leur charge de tests de compatibilité. C'est ce qui s'est passé lorsque Microsoft avait le monopole des navigateurs au début des années 2000. Et cela pourrait se reproduire. Les navigateurs internet, et par extension leur moteur, déterminent la façon dont nous consommons le Web. La décision de Microsoft, a un sens d'un point de vue commercial, mais elle donne surtout à Google plus de capacité à décider des possibilités d'évolution du Web par sa contribution principale au projet open source Chromium…si Microsoft relâche sa participation à ce projet au fil du temps(7), cela renforcera Google sur de nombreux fronts : la recherche, la publicité, les smartphones et la capture de données. Une prépondérance importante qui peut inquiéter à l'avenir. Cette dernière partie peut être à la première lecture dramatique voire anti-Google, mais ce que j'ai voulu exprimer dans cet article, c'est que l'échec de Edge Html est dû plutôt à une mauvaise réputation d'**Internet Explorer** et à un retard trop important dans la guerre des navigateurs. Néanmoins malgré cette décision d'abandonner son moteur maison, la firme de Redmond veut se différencier de Google, notamment par l'intégration de Edge à son écosystème d'outils, pour développer des applications « classiques » ou plus modernes. C'est un navigateur tourné vers l'avenir avec un énorme potentiel. Cependant c'est à nous, développeurs, de ne pas oublier qu'il y a d'autres navigateurs internet, et qu'il est de notre responsabilité de laisser l'utilisateur choisir son outil pour naviguer sur le Web.

(5) https://docs.microsoft.com/fr-fr/microsoft-edge/progressive-web-apps-chromium/serviceworker

(6) https://docs.microsoft.com/fr-fr/microsoft-edge/devtools-guide-chromium/progressive-web-apps

(7) https://chromium-review.googlesource.com/q/author:*.microsoft.com+AND+status:merged

Nefer-IT

Une publication Nefer-IT, 57 rue de Gisors, 95300 Pontoise - redaction@programmez.com
Tél. : 09 86 73 61 08 - Directeur de la publication : François Tonic
Rédacteur en chef : François Tonic
Secrétaire de rédaction : Olivier Pavie

Nos experts techniques : A. Giretti, M. Bernard, C. Pichaud, J.Jeanson, A. Zanchetta, J. Thiriet, P. Laroche, J. Chomarat, M. Jandar, L. Charaner, V. Thavonekham, M. Billermaz, J. Larrieu, G. Verret, J. Fourre
Couverture : © Microsoft, logo VS - **Maquette** : Pierre Sandré.
Publicité : François Tonic / Nefer-IT - Tél. : 09 86 73 61 08 - ftonic@programmez.com.
Imprimeur : SIB Imprimerie
Marketing et promotion des ventes : Agence BOCONSEIL - Analyse Media Etude - Directeur : Otto BORSCHA oborscha@boconseilame.fr
Responsable titre : Terry MATTARD Téléphone : 09 67 32 09 34

Contacts : Rédacteur en chef : ftonic@programmez.com - Rédaction : redaction@programmez.com - Webmaster : webmaster@programmez.com
Evenements / agenda : redaction@programmez.com

Dépôt légal : à parution - Commission paritaire : 1220K78366 - ISSN 2729-5001 - © NEFER-IT / Programmez, octobre 2020

Abonnement :
Programmez - service abonnement
57 rue de Gisors, 95300 Pontoise
abonnements@programmez.com

Tarifs
Abonnement (magazine seul) : 1 an - 11 numéros France métropolitaine : 49 € - Etudiant : 39 € CEE et Suisse : 55,82 € - Algérie, Maroc, Tunisie : 59,89 € - Canada : 68,36 € - Tom : 83,65 € - Dom : 66,82 € - Autres pays : nous consulter.

PDF
39 € (monde entier) souscription sur www.programmez.com

Cybersécurité :
RESTEZ INFORMÉ

- **L'actualité quotidienne**
  News, avis d'experts, témoignages, livres blancs, etc.
  https://www.solutions-numeriques.com/securite/

- **La newsletter**
  Chaque lundi, comme 40 000 professionnels et décideurs, recevez la synthèse des informations.
  C'est gratuit, inscrivez vous :
  https://www.solutions-numeriques.com/securite/inscription/

- **L'annuaire : Guide papier et en ligne**
  Trouvez l'éditeur de solution, le prestataire de services qu'il vous faut.
  https://www.solutions-numeriques.com/securite/annuaire-cybersecurite/

SoftFluent est fière d'être,
cette année encore, au palmarès

Best Workplaces™
De 50 à 250 salariés
Great Place To Work.
FRANCE 2020

Merci aux SoftFluentees

engagés pour la qualité des
applications logicielles !

SoftFluent, c'est 15 ans d'aventure humaine !

Rejoignez une équipe dynamique et participez aux projets ambitieux de nos clients et aux produits innovants du groupe sur jobs@softfluent.com

softfluent | CONSEIL & EXPERTISE | DÉVELOPPEMENT D'APPLICATIONS .NET | INNOVATION DIGITALE