Infrastructure graphique WinUI pour les applications Windows

Application de démonstration

Veikko Krypczyk (Allemagne)

Les logiciels conçus pour le contrôle des applications d'électronique fonctionnent souvent sous un environnement Windows. L'utilisation d'une application de bureau installée localement donne un accès direct à toutes les interfaces système du PC. Ainsi, Microsoft rationalise actuellement son assistance technique aux développeurs utilisateurs d'infrastructures (ou frameworks) d'interface Windows. La nouvelle interface graphique WinUI 3 est une approche d'avenir. Dans cet article, nous allons aborder le contexte technique et montrer comment l'utiliser en créant une simple application de démonstration pour les électroniciens.

Dans de nombreux projets d'électronique, un PC Windows est indispensable pour assurer les fonctions de contrôle ainsi que l'enregistrement des données et d'autres tâches ; en général, il est nécessaire de développer une interface utilisateur graphique exécutable sous Windows pour remplir cette fonction. Les applications de bureau exécutées localement sont souvent la méthode à privilégier dans ce cas ; elles donnent un accès complet à l'environnement du système et procurent également des liens vers les périphériques connectés via les pilotes appropriés.

Développeur du système d'exploitation Windows, Microsoft connaît actuellement un bouleversement technologique majeur, qui concerne essentiellement la connectivité et la conception d'interfaces utilisateur. Les développements sont coordonnés sous les projets *WinUI 3* et *Windows App SDK*. Cet article décrit en quoi consiste cette nouvelle interface graphique et comment l'utiliser. Vous y trouverez d'abord un aperçu des possibilités de programmation d'applications Windows avec une interface utilisateur graphique. La finalité de *WinUI 3* sera précisée grâce à l'utilisation de différents types d'applications et technologies. Ne souhaitant pas nous contenter d'une analyse purement théorique de ce type d'application relativement nouveau, nous créerons également notre première application pratique.

Technologie

Pour l'essentiel, les applications actuelles dotées d'une interface utilisateur graphique pour le système d'exploitation Windows se répartissent en deux catégories. D'une part, nous avons les applications de bureau (*Desktop Applications*). Elles sont essentiellement basées sur l'utilisation de l'*API Win32* et leur développement s'appuie sur différentes approches, infrastructures et langages de programmation. Microsoft a proposé les technologies Windows Forms (WinForms) et Windows Presentation Foundation (WPF). WinForms est basé sur l'interface Windows GDI. WPF est basé sur DirectX et était à l'origine destiné à remplacer WinForms. Les deux infrastructures graphiques étaient basées sur l'infrastructure .NET, destinée aux programmes Windows. Son développement s'est arrêté à la version 4.8.

La version .NET 5, quant à elle, est le successeur technologique de .NET Core. Cette infrastructure ne se limite pas à Windows, et peut également être utilisée avec d'autres systèmes d'exploitation. De manière inattendue, Microsoft a transformé WinForms et WPF en .NET Core. Si vous optez aujourd'hui pour WinForms ou WPF en développant une application pour Windows, vous avez le choix entre l'ancien environnement .NET et .NET Core. Il est également possible d'effectuer une migration des applications existantes, mais, comme c'est souvent le cas pour ce genre de projets, cette opération s'accompagne d'un certain nombre de problèmes. Les autres outils de développement de différents fabricants sont le plus souvent basés sur l'interface graphique du système d'exploitation (GDI) et encapsulés dans leur propre infrastructure.

La deuxième catégorie d'applications Windows concerne la plateforme d'application universelle Windows *(UWP)*. Ces applications s'exécutent au sein d'une couche isolée du système d'exploitation et n'ont qu'un accès limité au système. Les utilisateurs les installent via le magasin d'applications. Cependant, ce type d'application n'a pas eu le succès escompté et leur adoption a été assez réduite. L'une des causes est l'accès très limité au système. La plateforme *UWP* a toutefois pour avantage que l'infrastructure graphique WinUI 2 utilisée ici est considérablement plus moderne que les technologies WinForms et WPF. Ses points forts résident dans un design attrayant, de nouveaux composants visuels, l'utilisation de matériaux et l'orientation vers le langage de conception *Fluent Design System*. En d'autres termes, les applications *UWP* ont un aspect moderne, contemporain et inédit, mais

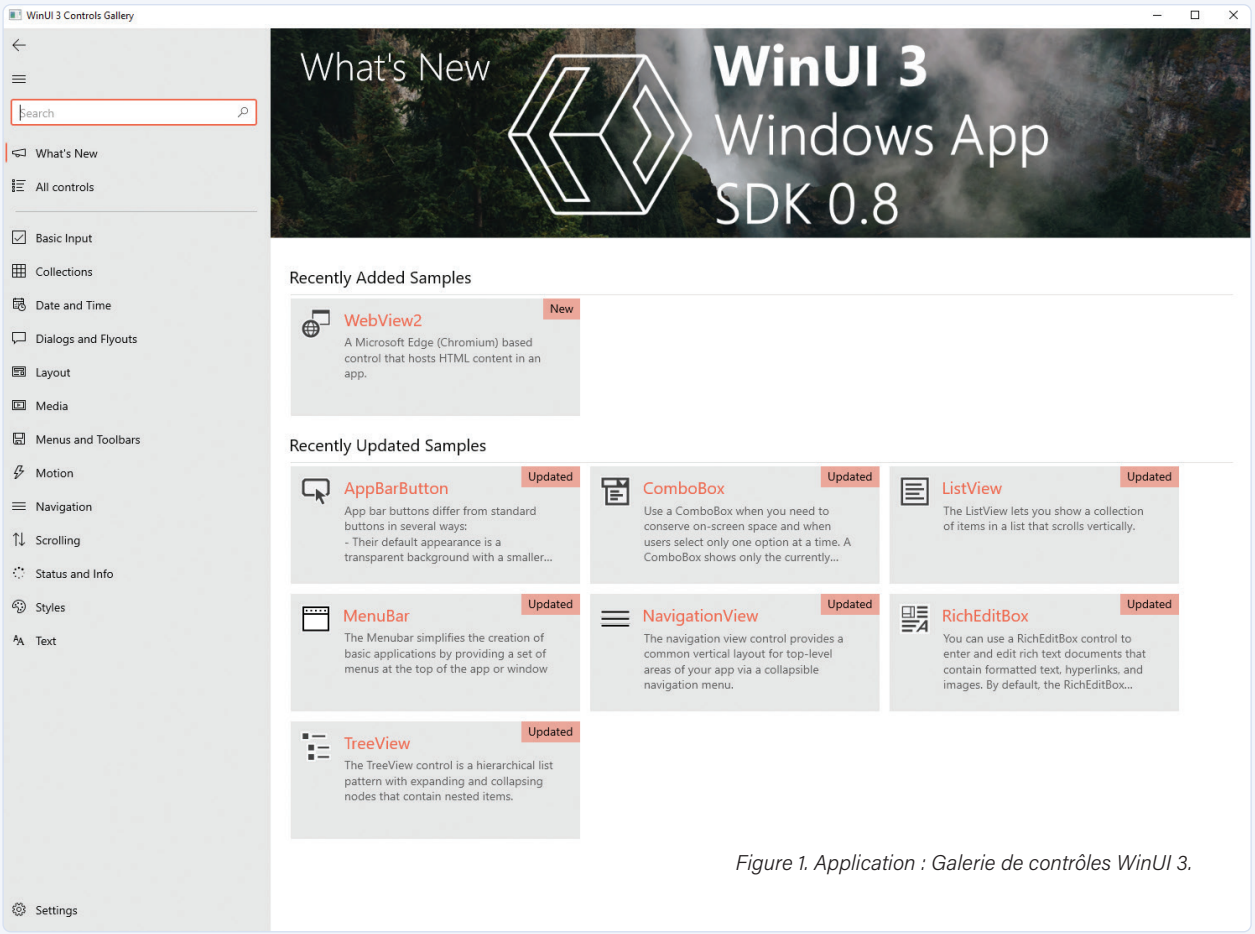

Figure 1. Application : Galerie de contrôles WinUI 3.

leur convivialité est assez limitée. Pour obtenir des effets similaires avec les technologies WinForms ou WPF, nous devons pousser les capacités dans leurs retranchements, utiliser de nombreux composants tiers ou « combiner » les technologies WinForms/WPF avec la plateforme *UWP*. Cette approche conduit toutefois rapidement à une structure d'application plus complexe et présente les inconvénients habituels, notamment une plus grande vulnérabilité aux erreurs et une plus faible maintenabilité.

Les logiciels conçus pour la commande de projet électronique, le développement, etc., sont presque tous des applications classiques de bureau. Vous pouvez également les construire en utilisant d'autres outils et cadres. Pour le langage de programmation Java, il existe, par exemple, le cadre graphique Swing qui est basé en interne sur l'interface du système d'exploitation GDI sous Windows.

Les logiciels conçus pour la commande de projets électroniques, le développement, etc., sont presque tous des applications classiques de bureau. Vous pouvez également les construire en utilisant d'autres outils et infrastructures. Pour le langage de programmation Java, il existe, par exemple, l'infrastructure graphique Swing, basée en interne sur l'interface de système d'exploitation GDI de Windows.

Infrastructure graphique WinUI 3

Avec l'introduction de WinUI 3, Microsoft souhaite permettre à toutes les applications fonctionnant sous Windows d'utiliser une interface graphique moderne. WinUI 3 est le successeur de WinUI 2 [1]. Cependant, elle est disponible pour tous les types d'applications Windows et n'est donc pas uniquement utilisée dans celles fonctionnant avec la plateforme UWP. WinUI 3 fait partie du nouveau SDK (kit de développement) Windows APP, proposé parallèlement à l'introduction de Windows 11. Ce SDK offre de nouvelles fonctionnalités pour le développement d'applications Windows. Il n'est pas seulement destiné à Windows 11, mais peut être utilisé aussi sous les versions actuelles de Windows 10. Le développement du SDK Windows App est toujours en cours, mais une première version est disponible et peut déjà être utilisée dans les nouvelles applications.

L'infrastructure WinUI 3 est techniquement et conceptuellement basée sur WinUI 2. Si vous avez déjà développé une application pour la plateforme UWP, vous vous familiariserez rapidement avec son utilisation. Elle est fondée sur les principes suivants :

- *Séparation entre le code et la conception* : l'interface utilisateur est créée de manière déclarative dans des fichiers séparés à l'aide du langage XAML, basé sur XML.
- *Éléments de contrôle de l'interface utilisateur* : il existe un ensemble de contrôles pour concevoir l'interface utilisateur. Il s'agit d'éléments de base, comme des boutons et des champs de saisie de texte, mais aussi des éléments plus complexes et avancés comme un élément de contrôle de calendrier, un composant *WebView* ou un élément d'affichage de données personnelles, que nous pouvons utiliser, par exemple, pour l'administration des utilisateurs. Si vous souhaitez explorer davantage toute la gamme des éléments de contrôle, accédez au Microsoft Store et téléchargez l'application *WinUI 3 Controls Gallery*, qui donne un aperçu des contrôles disponibles pour WinUI 3, avec une démonstration de leur utilisation et de leur intégration dans le code source (XAML). Des liens correspondants vers la documentation sont également disponibles (**figure 1**).
- *Couplage souple par liaison de données :* les propriétés et les événements des éléments de contrôle sont liés au code source par liaison de données. Les données sont ainsi échangées dans

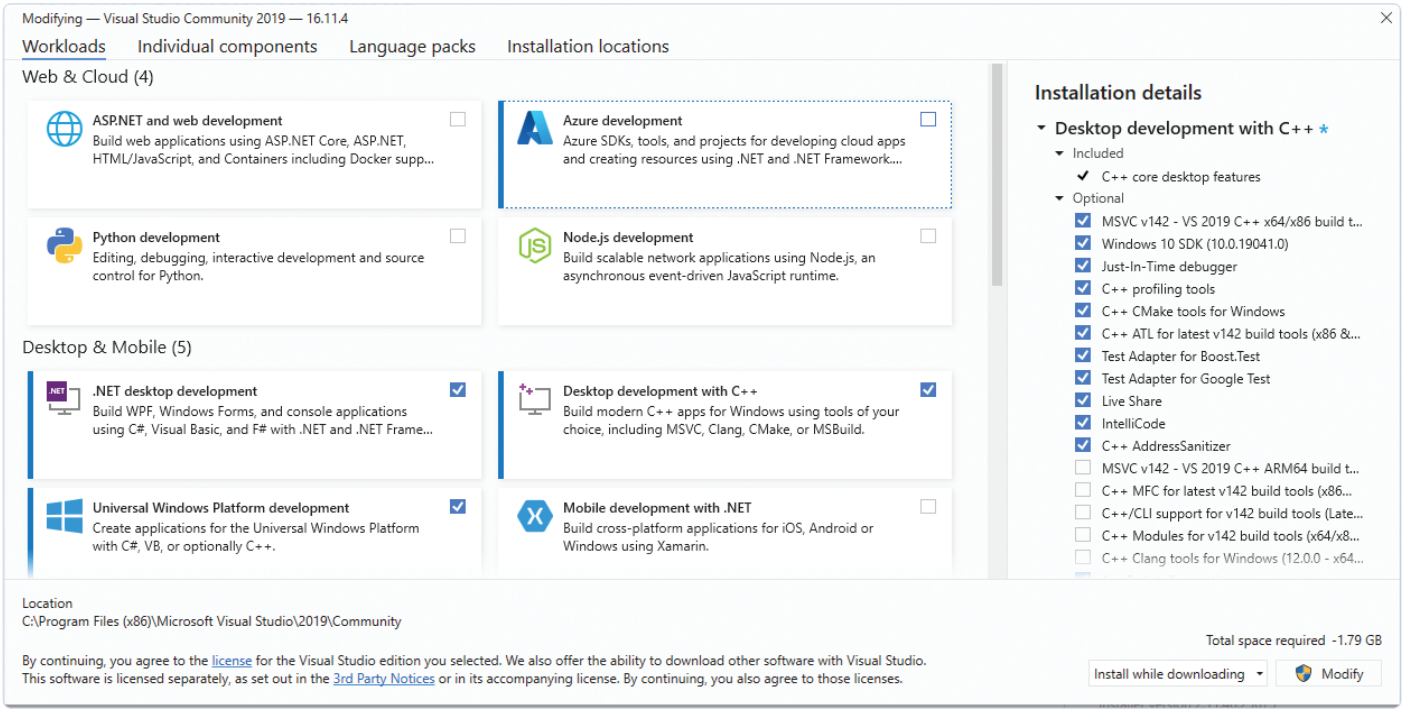

Figure 2. Sélectionnez les Workloads nécessaires pour Visual Studio.

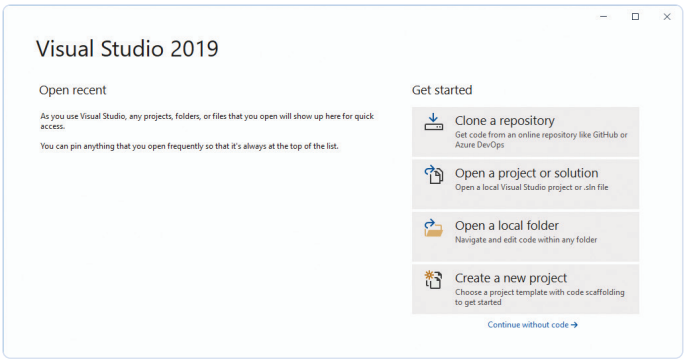

Figure 3. Premier démarrage de Visual Studio.

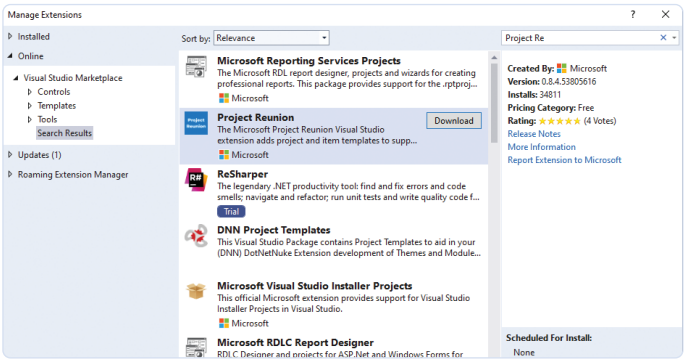

Figure 4. Installation de Windows SDK App (Projet « Reunion »).

les deux directions entre le code du programme et le contrôle de l'interface utilisateur. Les événements des éléments de contrôle, par exemple le clic sur un bouton, sont également transmis à l'algorithme correspondant, de la même manière.

- *Design moderne* : l'infrastructure WinUI 3 offre un aspect contemporain. Elle inclut le langage de conception *Fluent Design System* de Microsoft, ainsi que le matériau *Mica,* intégré dans Windows 11. Le langage *Fluent Design System* fournit les éléments d'interface utilisateur suivants : utilisation raisonnée de la géométrie et de la couleur, superposition des surfaces, utilisation de matériaux sélectionnés et application d'une iconographie et d'une typographie spécifiques pour la conception visuelle à l'aide d'images, de symboles et de polices. Les mouvements entre les éléments de l'interface utilisateur sont aussi pris en charge.

Examinons la procédure de développement d'applications avec l'infrastructure *WinUI 3*. Nous devons tout d'abord configurer l'environnement de développement.

Environnement de développement et configuration

La version actuelle de Visual Studio 2019 [2] est l'environnement de développement que nous utiliserons. L'édition Community sera suffisante pour nos besoins. Au moment où vous lirez cet article, une version stable de Visual Studio 2022 sera peut-être déjà disponible, auquel cas vous devrez l'utiliser. Il est conseillé d'installer au préalable les dernières mises à jour du système d'exploitation. Lors de l'installation de Visual Studio, il vous sera demandé de sélectionner les packages d'installation. Vous pouvez également, par la suite, appeler à tout moment le programme d'installation de Visual Studio via le

menu de démarrage. Sélectionnez maintenant les packages d'installation suivants (Workloads) : *.NET desktop development*, *Desktop development with C ++* et *Universal Windows Platform development* (**figure 2**). Après l'installation, démarrez Visual Studio et dans l'écran de démarrage, sélectionnez l'option *Continue without code* -> (continuer sans code) (**figure 3**).
Nous devons maintenant installer le modèle pour le développement avec *WinUI 3*. Pour ce faire, dans Visual Studio, choisissez l'option de menu *Extensions* | *Manage Extensions* (Extensions | Gérer les extensions). Recherchez *Project Reunion* (le nom de développement du nouveau *SDK Windows App*) et installez la version actuelle (**figure 4**). De même, installez l'extension *Windows Template Studio*. Celle-ci offre des modèles avancés pour la création d'une nouvelle application. *Visual Studio* doit être redémarré après le téléchargement des extensions ; l'installation se fera alors automatiquement.

Application pour WinUI 3

Commençons par créer un nouveau projet. Ici, comme l'indique la **figure 5**, nous sélectionnons le modèle App (*WinUI 3 in Desktop*) (*Windows Template Studio*). Dans Windows Template Studio (**figure 6**), nous pouvons configurer le projet :

- *Project type* : on spécifie ici le type de navigation, par exemple avec une barre de menu ou une barre de navigation latérale (menu *Hamburger*).
- *Design pattern* : installation et configuration directe de la boîte à outils *MVVM*. Celle-ci associe les éléments de l'interface utilisateur (définis dans *XAML*) à la logique du programme (langage de programmation C#).
- *Pages* : nous pouvons ajouter un certain nombre de pages au projet. Nous pouvons choisir parmi différents modèles, par exemple une page pour entrer les paramètres du programme.
- *Features* : ici, nous pouvons sélectionner des exemples de thèmes ou enregistrer les paramètres du programme.

En cliquant sur le bouton *Create*, nous générons l'application de bureau que *WinUI 3* utilise. *Windows Template Studio* génère un dossier avec trois projets :

- *App* : il contient le code source de l'application de bureau. Dans le sous-dossier View, par exemple, vous trouverez les fichiers XAML pour les pages qui ont été créées par Windows Template Studio. La logique du programme est stockée dans les fichiers du dossier de programme ViewModel.
- *Package* : le projet est chargé de fournir l'application de bureau. Actuellement, les applications *WinUI 3* sont installées sur l'ordinateur cible sous la forme d'un paquet d'applications. Ce format a jusqu'à présent été utilisé pour les applications *UWP*. Les paquets générés peuvent également être distribués en utilisant Store. Les futures versions du *SDK Windows App* devraient également permettre l'installation sans package d'application.
- *Core* : ce projet contient l'ensemble des services et des classes qui fournissent des services à l'application. Il n'est pas obligatoire et peut être exclu.

Démarrez l'application directement à partir de Visual Studio en utilisant la flèche verte de la barre d'outils. Félicitations - vous avez créé

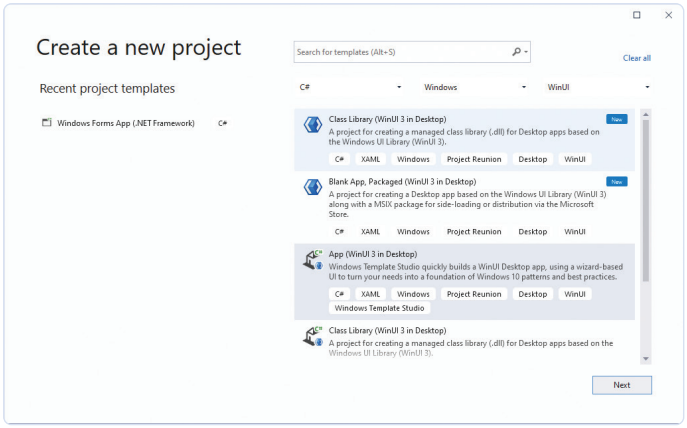

Figure 5. Modèle de projet WinUI 3 Desktop.

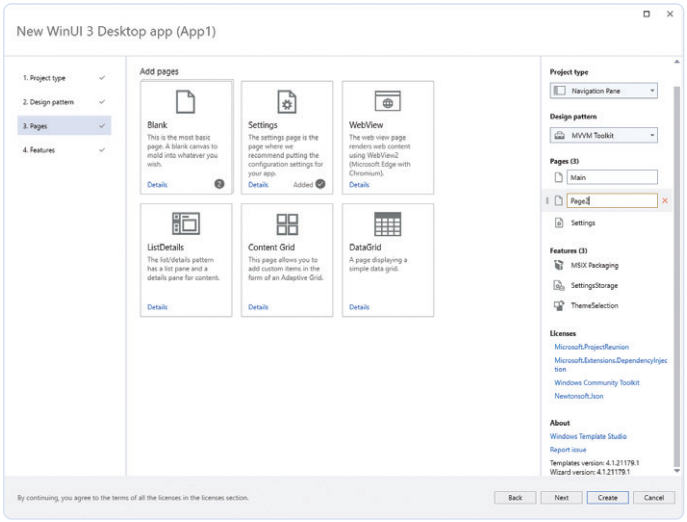

Figure 6. Windows Template Studio.

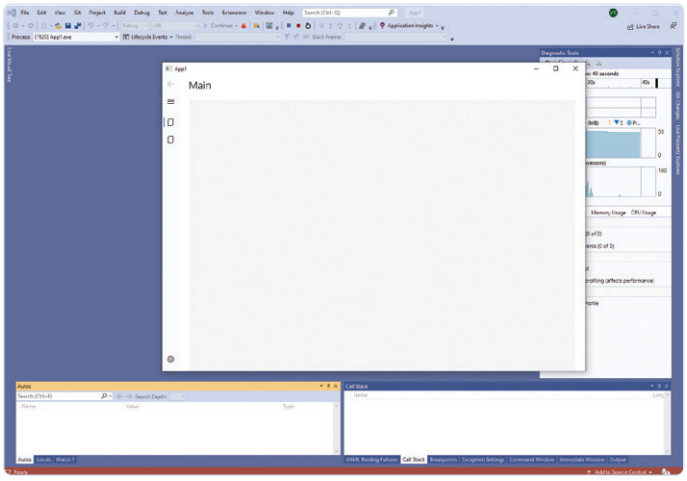

Figure 7. Une première application de bureau avec WinUI 3.

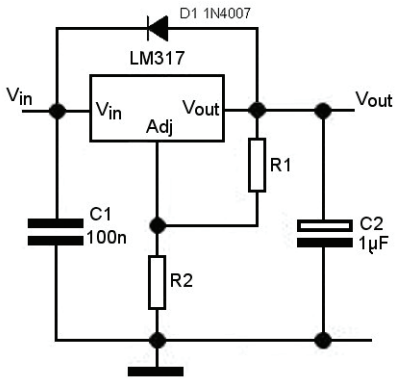

Figure 8. Schéma du circuit du régulateur de tension LM317.

votre première application avec *WinUI 3* (**figure 7**). Je dois insister à nouveau sur le fait qu'il s'agit d'une application de bureau avec un accès complet au système. Comme nous l'avons déjà mentionné, c'est un aspect important pour les logiciels destinés à contrôler des composants électroniques externes, par exemple. Il est possible de créer une barre de navigation latérale, des pages initiales et d'adapter la conception de l'application. Vous disposez de toutes les options d'accès au système, y compris la communication avec les bibliothèques et les pilotes du système. Nous pouvons maintenant commencer à expérimenter la conception de l'interface utilisateur.

Application de démonstration

La meilleure façon de se familiariser avec un nouveau système est de le tester. Nous allons concevoir une interface utilisateur simple pour notre première application (le code source de cet exemple se trouve sur la page Web de l'article [3]). Le point de départ est le fichier *XAML* de la page concernée. À titre d'expérience, nous pouvons créer un outil de calcul pratique utilisable avec le régulateur de tension linéaire ajustable

Tableau 1 : Élements de contrôle

Éléments de contrôle	Propriété	Valeur	Description
TextBoxR1	Width	200	Textbox width
	Margin	20, 20, 0, 0	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	R1:	Caption
	Text	x:Bind ViewModel.R1	Binding the property to the variable R1 in C#.
TextBoxU (out)	Width	200	Textbox width
	Margin	20, 10	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	U (out):	Caption
	Text	x:Bind ViewModel. UOut	Binding the property to the variable UOut in C#.
TextBoxR2	Width	200	Textbox width
	Background	LightGray	Background colour
	Margin	20, 10	Margin size: left, top, right, bottom
	HorizontalAlignment	Left	Left justified
	Header	R2:	Caption
	IsReadOnly	True	Read only protection
	Text	x:Bind ViewModel.R2	Binding the property to the variable R2 in C#.
Button	Width	200	Button area width
	Margin	20, 10	Margin size: left, top, right, bottom
	Background	LightGreen	Background colour
	Command	x:Bind ViewModel. CalcCommand	Binding to the CalcCommand Method in C#.
	Content	Calc	Caption
	FontWeight	Bold	Font properties

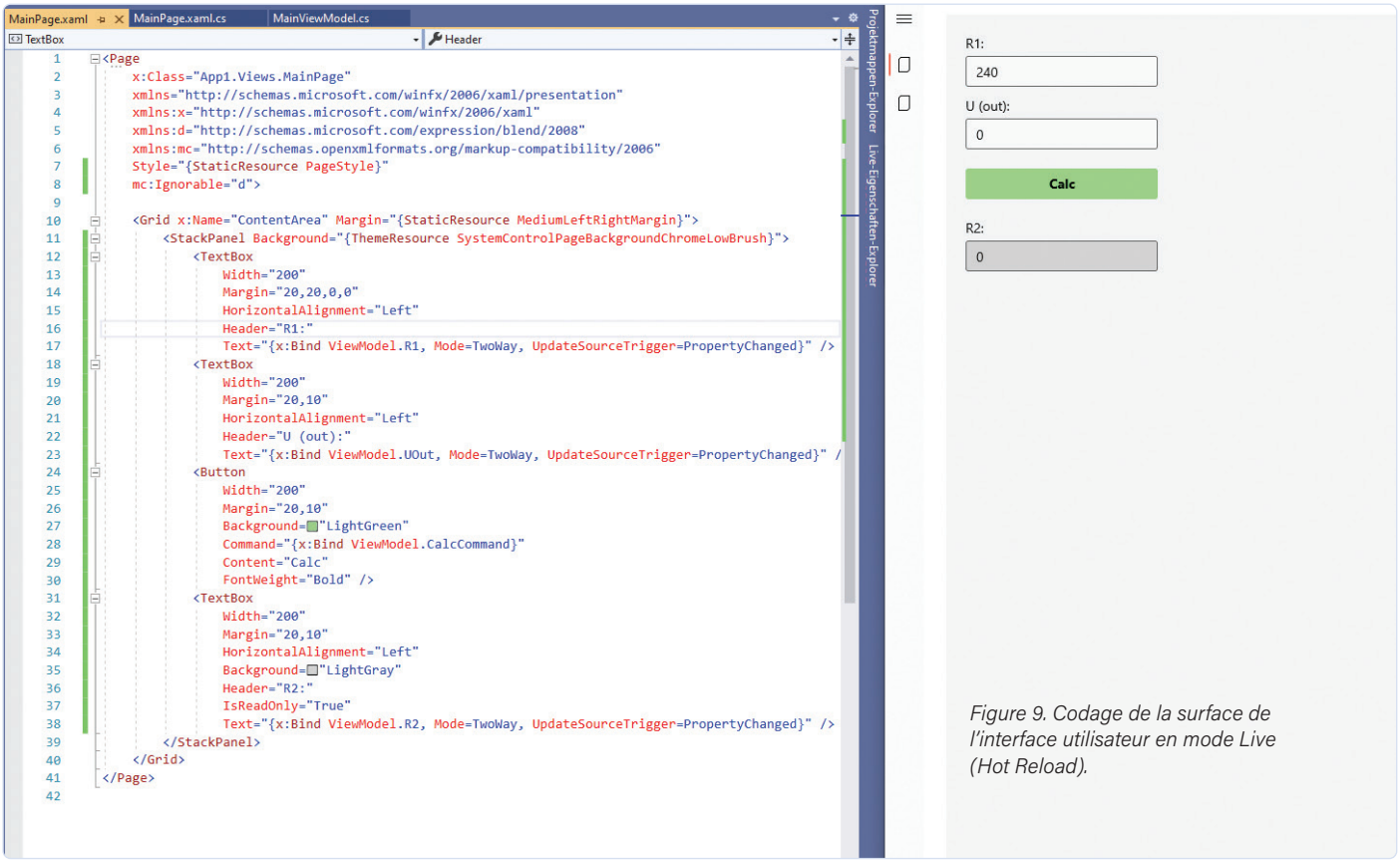

Figure 9. Codage de la surface de l'interface utilisateur en mode Live (Hot Reload).

LM317 (**figure 8**). La tension de sortie de ce dispositif est donnée par la formule *Vout = 1,25 (1 + R2/R1)*. Nous pouvons résoudre cette équation pour *R2* et ainsi calculer sa valeur pour donner la tension de sortie souhaitée. À l'aide de cet exemple, nous pouvons démontrer la marche à suivre pour programmer des applications avec *WinUI 3*. Cette procédure comprendra les étapes suivantes :

- Définir l'interface utilisateur en XAML.
- Coder la logique du programme en C#.
- Lier l'interface utilisateur à la logique du programme.
- Transférer l'interaction de l'utilisateur entre l'interface utilisateur et la logique du programme.
- Générer les données vers le formulaire.

Commençons par définir la surface. Nous avons besoin de deux champs de texte pour enregistrer les valeurs de *R1* et V_{Out}, ainsi que d'un champ de texte pour la valeur de *R2*. Il faudra un bouton (*Button*) pour lancer le calcul. Nous utilisons des contrôles *TextBox* pour l'entrée et la sortie. Tous les éléments doivent être disposés sous la forme d'une pile verticale les uns au-dessus des autres, ils sont donc insérés dans un

Listage 1. Définition des interfaces utilisateur

```
<Page
    x:Class="App1.Views.MainPage"
    …>

    <Grid x:Name="ContentArea" Margin="">
        <StackPanel Background="">
            <TextBox
                Width="200"
                Margin="20,20,0,0"
                HorizontalAlignment="Left"
                Header="R1:"
                Text="" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Header="U (out):"
                Text="" />
            <Button
                Width="200"
                Margin="20,10"
                Background="LightGreen"
                Command=""
                Content="Calc"
                FontWeight="Bold" />
            <TextBox
                Width="200"
                Margin="20,10"
                HorizontalAlignment="Left"
                Background="LightGray"
                Header="R2:"
                IsReadOnly="True"
                Text="" />
        </StackPanel>
    </Grid>
</Page>
```

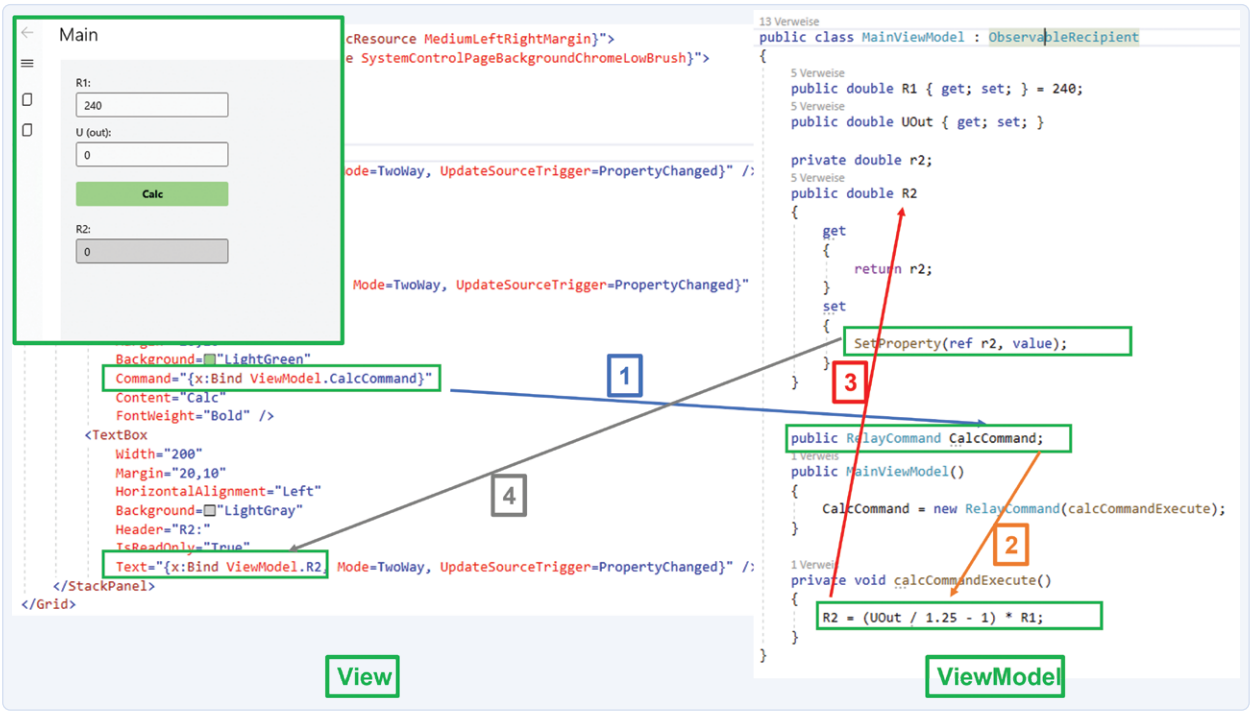

Figure 10. Exemple de relation entre View et ViewModel.

conteneur de mise en page de type <*StackPanel* />. Tous les éléments seront disposés les uns au-dessus des autres avec *StackPanel* sans nécessiter aucune autre configuration. Les contrôles sont configurés via le code *XAML*, avec les propriétés selon le **tableau 1**.
Le code source associé est présenté dans le **listage 1**. Vous pouvez coder la surface de manière interactive. Démarrez l'application et placez le fichier XAML correspondant dans Visual Studio et l'application en parallèle sur l'écran (**figure 9**). Les modifications apportées au code XAML sont immédiatement reprises au démarrage de l'application, sans enregistrement, et produisent un affichage actualisé. Cette fonctionnalité, appelée *Hot Reload*, est standard dans la création d'interfaces utilisateur graphiques.

Ce qui est intéressant ici est la liaison des éléments de contrôle de type *TextBox* avec les propriétés de *Text*. Vous trouverez ici une expression conforme au modèle dans le code XAML :

```
Text=»{x:Bind ViewModel.R1, Mode=TwoWay,
   UpdateSourceTrigger=PropertyChanged}»
```

Cela signifie que la propriété *Text* est liée à la variable *R1*. Cette liaison, basée sur le concept MVVM, est définie dans la page *ViewModel*. Les événements de l'interface utilisateur sont traités dans la couche *View* et les données sont gérées dans la couche *Model*. *ViewModel* représente la connexion entre les deux couches. Grâce au concept MVVM,

Listage 2. Logique de programme pour le calcul en C#

```
using CommunityToolkit.Mvvm.ComponentModel;
using CommunityToolkit.Mvvm.Input;

namespace App1.ViewModels
{
    public class MainViewModel : ObservableRecipient
    {
        public double R1 { get; set; } = 240;
        public double UOut { get; set; }

        private double r2;
        public double R2
        {
            get
            {
                return r2;
            }
            set
            {
                SetProperty(ref r2, value);
            }
        }

        public RelayCommand CalcCommand;
        public MainViewModel()
        {
            CalcCommand = new RelayCommand(calcComm
                andExecute);
        }

        private void calcCommandExecute()
        {
            R2 = (UOut / 1.25 - 1) * R1;
        }
    }
}
```

toutes les couches sont découplées et peuvent être développées et maintenues indépendamment les unes des autres. Vous trouverez des informations sur le modèle MVVM dans le document [4].

Logique du programme

La logique du programme est mise en œuvre en langage C# (**listage 2**). À cette fin, un fichier de programme (*ViewModel*) est attribué à chaque fenêtre de l'interface utilisateur (View). Dans notre exemple, il s'agit du fichier *MainViewModel* attribué à la vue *MainPage*. Quelques remarques concernant le code :

- **Importation de bibliothèques** : elle se fait par l'instruction *uses*. Dans notre cas, nous avons besoin de deux bibliothèques pour le modèle MVVM.
- **Définition des propriétés :** celles-ci doivent être publiques, car nous accédons aux propriétés depuis l'extérieur, dans ce cas depuis *View*.
- **Mise à jour automatique de l'interface utilisateur :** la classe `MainViewModel` est dérivée de la classe de base `Observable-Recipient`, générée par l'assistant lorsque le projet a été créé. Cette classe implémente à son tour l'événement `OnProper-tyChanged`. Cela garantit que lorsque la valeur d'une propriété change, tous les éléments liés sont avertis de cette modification. Dans notre cas, c'est la propriété `R2` qui nous intéresse. La valeur de `R2` est calculée dans le code du programme. Le code *Setter* de la propriété est appelé et l'événement `OnPropertyChanged` décrit ci-dessus est déclenché par la méthode `SetProperty (...)`. La propriété *Text* de *TextBox* R2 est liée à `R2` (dans *ViewModel*). Si `R2` est modifié, la valeur affichée est automatiquement mise à jour dans la *TextBox* associée. Cela fonctionne grâce à la liaison de données.
- **Transmettre les actions de l'utilisateur au moyen de commandes :** si l'utilisateur appuie sur le bouton, il déclenche une commande. La méthode de calcul est liée à celle-ci. Ici aussi, l'interface utilisateur et le code de programme ne sont reliés entre eux que par la liaison de données.
- **Attribution de *View* et *ViewModel*** : le code du programme (fichier : MainViewModel.cs) est attribué à l'interface utilisateur (fichier : *MainPage.xaml*). Cela se fait dans le fichier *code-behind* de la page (fichier : *MainPage.xaml.cs*). Vous pouvez le constater en examinant le code source.
- **Calcul :** le calcul de la valeur de `R2` se fait par la méthode `calcCommandExecute (...)` selon la formule ci-dessus, qui est ensuite affectée à `R2`.

L'interface utilisateur est donc liée de manière « souple » au code du programme au moyen de la liaison de données. Les connexions à la liaison de données qui viennent d'être décrites sont visualisées à l'aide de l'exemple de la **figure 10**. Lancez l'application et testez-la. La valeur de la deuxième résistance *R2* est calculée après avoir saisi les valeurs de *R1* et de V_{Out} (**figure 11**).

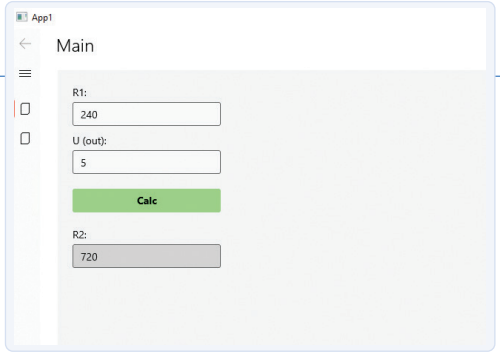

Figure 11. L'exemple complet de l'application.

Nous avons ainsi décrit le modèle de développement de base des applications de bureau avec l'infrastructure graphique WinUI 3. Dans cette perspective, elle deviendra un nouveau standard sous Windows et pourra également être utilisée par d'autres environnements et langages de développement. La variété des options graphiques permettant de créer des applications modernes est impressionnante.

Conclusions et perspectives

Vous avez le choix entre plusieurs technologies pour créer une application Windows. La tendance – avec Windows 11 en perspective – est à l'utilisation de *WinUI 3*, qui vous permettra de créer une interface utilisateur attrayante et moderne. De ce point de vue, il est intéressant de considérer *WinUI 3* pour développer une nouvelle application Windows et d'envisager les options de migration pour une application existante. Vous obtiendrez ainsi une interface contemporaine avec une bonne expérience utilisateur. Il n'y a pas non plus de limitations en termes d'accès au système comme c'est le cas avec le modèle d'application *UWP*.

210407-04

Contributeurs

Texte et images : **Veikko Krypczyk**
Rédaction : **Jens Nickel**
Traduction : **Asma Adhimi**
Mise en page : **Giel Dols**

Des questions, des commentaires ?

Contactez Elektor (redaction@elektor.fr).

LIENS

[1] Informations sur WinUI 3: https://docs.microsoft.com/en-us/windows/apps/winui/
[2] Visual Studio 2019: https://visualstudio.microsoft.com/
[3] Page du projet pour cet article: www.elektormagazine.fr/210407-04
[4] Informations sur le modèle MVVM: https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-vue_mod%C3%A8le