"Introduction à WComp"

1 Découverte de WComp

Vous pouvez vous référer à la documentation technique de référence disponible en ligne ainsi qu'aux vidéos de démonstration disponibles à l'adresse suivante pour l'installation et la prise en main de l'environnement WComp :

`Installation WComp 3.2.1`
`http://www.wcomp.fr/videos`

Commencez par démarrer SharpWComp et créez un fichier WComp :

- Fichier / Nouveau / Fichier …
- WComp.Net / C# Container -> crée un nouveau fichier Container.cs (onglet en haut de la zone de travail)
- Pour pouvoir manipuler les composants il faut activer la représentation graphique du Container (onglet WComp.NET en bas de la zone de travail, passage sur cet onglet automatique en SharpWComp 3.0).

N'oubliez pas que vous pouvez sauvegarder vos assemblages de composants avec l'option Export du menu WComp.NET.

2 Composition Locale : Modèle LCA

Nous allons voir comment utiliser et créer des composants : composants fournis avec SharpWComp, ou encore créer vos propres composants pour répondre à vos besoins. Nous verrons aussi dans cette section comment relier un événement émis par un composant à un appel de méthode d'un autre composant ce qui permet d'établir des communications entre les composants.

2.1 Mise en place d'une application par assemblage de composants

2.2 Création d'un Composant Bean

Il peut s'avérer nécessaire de créer vos propres composants si aucun composant existant ne correspond à vos besoins. L'environnement offre la possibilité de créer un composant à partir d'un squelette de code. Nous allons donc créer un composant permettant d'envoyer un événement lorsqu'un seuil numérique est dépassé.

2.2.1 Création et compilation d'un composant

Après avoir quitté et redémarré SharpWComp, vous pouvez créer une nouvelle solution pour la création du composant souhaité :

- Fichier / Nouveau / Solution …
- WComp.NET / WComp Bean Combine (nommer cette solution Blink)

Ajoutez un fichier à votre nouvelle solution :

- Clic droit sur votre solution / Ajouter / Nouveau Fichier …
- WComp.Net / C# Bean with Thread -> crée un nouveau fichier BeanThread1.cs (que l'on peut renommer en Blink.cs)

Il ne vous reste plus qu'à modifier le squelette de code pour donner à ce nouveau composant le comportement souhaité.

Nous allons faire un composant qui nous permettra de faire clignoter une des lumières du feu tricolore. Ce composant disposera tout d'abord d'une **propriété** `sleepProperty` permettant d'indiquer la fréquence d'émission de ces événements. Pour coder cette propriété, inspirez-vous du code contenu dans le squelette de code d'un composant Bean sans Thread.

Ce composant aura une **méthode** `MethodBlink` qui, en fonction de la valeur d'un paramètre booléen, démarrera ou arrêtera le Thread à l'aide des fonctions `Start` et `Stop`. Vous veillerez à terminer par l'émission d'un événement

faux quand vous arrêtez le clignotement. La méthode `Stop` est utilisée par SharpWComp pour terminer proprement le thread créé par le Bean.

Ecrire le code correspondant, le compiler et ajouter la librairie créée dans le dossier contenant les composants (dossier `C:\Program Files (x86)\SharpDevelop\3.0\Beans\`). Un nouveau composant `Blink` est maintenant disponible pour vos assemblages. Si vous n'avez rien précisé dans la directive [Bean] vous trouverez votre composant dans la catégorie Beans : Basic. Si vous souhaiter créer une catégorie particulière pour y ranger ce nouveau composant, vous devrez spécifier comme directive en tête de votre classe :

```
[Bean(Category="Demo")]
```

2.2.2 Créer un assemblage pour la mise en œuvre de l'application

Créer un assemblage à l'aide du composant de `Blink` que vous avez développé. Vous utiliserez des Beans Winforms comme des radioboutons pour visualiser le comportement de votre application.

3 Introduction de Matériel type Phidget

Nous allons reprendre ici les notions vues ci-dessus en utilisant un composant Bean d'accès à du matériel type capteur/actionneur Phidget.

3.1.1 Evènements simples

Nous souhaitons pouvoir allumer et éteindre une LED de la plate-forme Phidget à l'aide de 2 buttons. Créer les 2 boutons : un bouton LedOn et un bouton LedOff, et les relier au composant. Créer les 2 boutons : un bouton LedOn et un bouton LedOff, et les relier au composant `GenericOutput` de la catégorie Phidget afin d'appeler les méthodes `On()` et `Off()` du dispositif. Vous modifierez les propriétés du composant `GenericOutput` pour que le Port corresponde au port de la LED que vous souhaitez commander.

3.1.2 Evènements complexes

Avoir deux boutons pour contrôler une des LEDs peut ne pas s'avérer intéressant surtout si on doit le faire pour toutes les LEDs connectées à la plate-forme. Nous souhaitons maintenant pouvoir contrôler la LED à l'aide d'une `CheckBox`.

Créer `1 checkBox` et la relier au composant `GenericOutput` afin de piloter le dispositif à l'aide de la méthode `SetOutputValue(boolean)`. Nous pouvons constater que l'événement émis par la checkBox (`CheckStateChanged`) n'envoie pas de données (la valeur de la case cochée). La fonction `SetOutputValue` nécessite par contre un paramètre booléen. Il faut donc sélectionner le mode « *signature incompatible* » pour voir cette fonction. Le panneau suivant vous permettra de faire appel à une fonction de « get » sur le composant émettant l'événement pour compléter la signature de la méthode appelée.

Vous devriez maintenant pouvoir contrôler la LED. Mais comment faire pour contrôler l'allumage de cette LED en fonction de capteurs ou actionneurs connectés à la plate-forme Phidgets ?

3.1.3 Créer un assemblage pour la mise en œuvre de l'application

En récupérant le composant Blink développé précédemment, faites clignote la LED sur le phidget.

1 Génération d'un composant proxy d'un service pour dispositif UPnP

Pour les plus avancés, nous étudierons comment générer un composant Bean Proxy pour accéder à un service pour dispositif de type UPnP.