**Caliburn.Micro의 주요 기능 정리**

- Caliburn.Micro는 Windows Form, WPF, UWP, Silverlight, Windows Phone, 모바일(iOS, Andriod)에서 적용 가능한 MVVM 프레임워크로 다음과 같은 기능을 갖고 있습니다. 위의 모든 플랫폼에서 동일하게 개발하기 위한 프레임워크로 나름의 규칙을 갖고 있기 때문에 다음의 주요 기능들을 꼭 알아 둘 필요가 있습니다.

Caliburn.Micro를 이하 부분에서 CM으로 표시합니다.

**\* Action Messages & Action Conventions 그리고 Binding Conventions**

CM에서는 UI(View)에서 발생하는 이벤트와 이에 대한 트리거와 트리거가 발생할 때 하는 작업에 대한 정의를 쉽게 하기 위한 Action Convetions을 가지고 있습니다. View (.xaml 파일) 단에서 Action 이벤트를 발생할 컨트롤에 x:Name에 특정 명칭인 XXX를 지정하면 자동으로 View Model (.cs 파일) 단에 해당 메소드를 XXX()를 찾아서 실행을 해 줍니다.

**Caliburn.Hello 샘플**의 ShellView.xaml에서는 Button 컨트롤의 x:Name으로 “SayHello”으로 지정하고, ShellViewModel.cs에 SayHello() 메소드를 정의하고 있습니다. 따라서 x:Name이 “SayHello”인 버튼이 클릭되면 CM에 의해서 자동으로 SayHello()가 호출되어 실행됩니다.

CM에서는 View 단의 컨트롤의 데이터 바인딩을 View Model의 정의된 형식으로 자동으로 바인딩하는 기능을 갖는데 이것이 Binding Conventions입니다.

**Caliburn.Hello 샘플**의 ShellView.xaml에서는 TextBox 컨트롤의 x:Name으로 “Name”으로 지정하고, ShellViewModel.cs에 Name에 대한 요소를 정의하고 있습니다. 여기서는 Name을 Property로 정의하고 있습니다. 따라서 x:Name이 “Name”인 텍스트 박스는 CM에 의해서 자동으로 Name 프로퍼티의 값과 자동 동기화 됩니다.

참고로, CM은 Action Messages와 Action Binding, Binding Conventions을 System.Windows.Interactivity를 기반으로 구현하여 지원합니다.

Action에 다양한 Messages, Targets, Parameter 부분을 변경할 수 있습니다. 이 부분은 다음 사이트에서 상세 내용을 확인해 주십시오.

* <https://caliburnmicro.com/documentation/actions>

CM의 Conventions은 다음 내용을 참고해 주십시오.

* <https://caliburnmicro.com/documentation/naming-conventions>

꼭 필요 없지만 위의 Conventions을 .NET에서 어떻게 지원해주고 CM에서 어떻게 적용했는지 이해하기 위해서는 다음 내용을 참고해 주십시오.

* <https://caliburnmicro.com/documentation/conventions>

**\* Screens, Screen Collection, Conductors, Container**

좀 풀어서 설명하면 CM에서는 Conductor는 Main View 개념입니다. Conductor는 여러 Screen을 갖습니다. 이들은 Sub View입니다. 그리고 이러한 Screen들의 집합이 Screen Collection입니다. 이것은 개념적인 의미로 프로그램 개발 시에 프로젝트에 있는 View들로 생각하시면 됩니다. 그리고, 자세한 설명은 없지만, 코드 상으로는 Screen Collection은 Container에 자동 등록되고 Container에 등록된 Screen Collection 들 중에서 불러 오기 위해서 IoC 클래스를 통해서 IoC.Get<View모델명>() 형식으로 Screen과 Screen의 View Model을 한꺼번에 불러올 수 있습니다.

참고로 Container는 Bootstrapper 클래스에 선언되고, Condutor는 ShellViewModel, 각 View 및View Model이 Screen입니다. 이에 대한 구현은 샘플 코드 Caliburn.Tutorial에 잘 정리되어 있습니다. 먼저 Screen은 AboutViewModel.cs, CategoryViewModel.cs에서 클래스 선언을 할 때 Screen 클래스를 상속받도록 정의하고 있습니다. 참고로 앞에서 언급했던 Convetions에 의해서 View는 AboutView.xaml, CategoryView.xaml이 됩니다. Conductor는 ShellViewModel.cs에서 Conductor<object>를 상속받도록 정의되어 있습니다. Container는 Bootstrapper.cs 파일에 SimpleContainer 인스턴스 멤버를 갖도록 정의되어 있습니다. 그리고, Bootstrapper.cs 파일에는 Configure() 메소드를 통해서 DI(Dependency Injection) 작업으로 ViewModel 폴더안의 클래스들을 SimpleContainer 인스턴스에 등록합니다. 여기서 등록된 Screen 들이 Screen Collection입니다. 반대로 등록된 Screen 인스턴스를 불러오기 IoC 클래스를 사용합니다. (개념상으로 DI와 반대 개념으로 IoC (Inversion of Control)는 SimpleContainer에 등록된 Control을 찾거나 꺼내 오는 역할 입니다.) Caliburn.Tutorial 샘플에서 ShellViewModel.cs의 About 버튼 클릭시에 IoC.Get<AboutViewModel>()로 AboutViewModel과 이에 종속된 AboutView를 얻을 수 있습니다.

위의 Screens, Conductors, Screen Collection에 대한 자세한 개념을 보시기 위해서는 다음 내용을 참고하십시오.

* <https://caliburnmicro.com/documentation/composition>

\* **View Locator, View Model Locator**

CM에서 View Model로 View 위치를 찾거나 View로 View Model을 찾을 수 있습니다. 이것은 앞에서 언급한 Conventions에 의해서 가능합니다.

**\* Event Aggregator**

CM에서는 Event Aggregator (이벤트 수집기) 기능을 제공합니다. Event Aggregator (이벤트 수집기)는 서로 연관 관계가 없는 View Model 사이에 Event를 전달하기 위한 개발 패턴입니다. 이벤트 수집기를 사용할 View Model의 생성자에 IEventAggregator 인터페이스 변수를 선언하고, View Model 내부에 바인딩하여 사용하면 됩니다.

IEventAggregator 인터페이스 변수로 할 수 있는 작업은 크게 3가지입니다.

- **이벤트 발생** : PublishXXX() 메소드로 이벤트를 발생시킵니다. 그러면 이벤트 수집을 하고 있는 다른 View Model에서 발생된 이벤트를 받아서 처리합니다.

**Caliburn.Features 샘플**을 보시면 EventSourceViewModel.cs에서 EventSourceViewModel 생성자에 IEventAggregator 변수 \_eventAggregator에 생성자의 파라미터로 받을 수 있도록 선언하고 바인딩하는 코드가 있습니다. 생성자를 호출하는 것은 CM이라 CM이 자동으로 IEventAggregator 인터페이스 객체를 Runtime에 자동으로 바인딩 해 줍니다. 그리고 Publish() 메소드 (Publish 버튼의 이벤트 처리 메소드)에서는 이벤트 생성하기 위해서 PublishOnUIThreadAsync()를 호출하여 이벤트를 생성합니다.

- **이벤트 수신 / 비수신 설정** : IEventAggregator 인터페이스 변수의 SubscribeXXX() 메소드를 호출하면, 호출한 View Model에서 Event를 수신할 수 있습니다. 반대로 UnsubscribeXXX() 메소드를 호출하면 이벤트를 수신하지 않습니다.

**Caliburn.Features 샘플**을 보시면 EventDestinationViewModel.cs에서 EventDestinationViewModel 생성자에서 IEventAggregator 변수 \_eventAggregator에 생성자의 파라미터로 받을 수 있도록 선언하고 바인딩하는 코드가 있습니다. 생성자를 호출하는 것은 CM이라 CM이 자동으로 IEventAggregator 인터페이스 객체를 Runtime에 자동으로 바인딩 해 줍니다.

Subscribe() 메소드(Subscribe 버튼의 이벤트 처리 메소드)에서는 이벤트 수신 설정하기 위해서 SubscribeOnPublishedThread() 메소드를 호출합니다.

Unsubscribe() 메소드(Unsubscribe 버튼의 이벤트 처리 메소드)에서는 이벤트 미수신 설정하기 위해서 Unsubscribe() 메소드를 호출합니다.

- **이벤트 처리** : IHande 인터페이스를 상속받고 HandleAsync()와 같은 이벤트 처리 메소드를 구현 하여 이벤트 발생시에 처리 동작을 정의할 수 있습니다.

**Caliburn.Features 샘플**을 보시면 EventDestinationViewModel.cs에서는 이벤트 처리를 위해서 HandleAsync() 메소드를 구현해 놓았습니다. 만약 이벤트 수신으로 설정된 경우 HandleAsync() 메소드 부분의 코드가 실행됩니다.

위의 Event Aggregator에 대한 자세한 개념을 보시기 위해서는 다음 내용을 참고하십시오.

* <https://caliburnmicro.com/documentation/event-aggregator>

\* **Coroutines**

C#에서는 기본적으로 coroutine(코루틴)을 제공하지 않습니다. corutine(코루틴)은 개념은 상당히 복잡하지만, 여기서는 CM에서 제공하는 coroutine(코루틴) 개념에 대해서 얘기하면, 서로 다른 비동기 작업들에 대해서 순차적으로 실행하기 위한 방법을 제공합니다. 그래서 A, B, C 비동기 작업이 있으면 이것을 특정 메소드 안에서 A, C, B 순서로 정의해 놓으면 A -> C -> B 순서로 실행이 됩니다. 개별적인 비동기 작업에 대한 순차화 기능을 CM에서 제공하고 있습니다.

CM에서는 IResult 인터페이스를 구현하는 클래스들에 Execute() 메소드를 재정의하여 작업을 정의해서 개발자가 원하는 작업을 구현할 수 있습니다. 그리고 완료 여부를 표시하기 위한 이벤트로 Completed 이벤트를 제공하여 IResult 인터페이스를 구현한 클래스의 작업이 완료됨을 알릴 수 있습니다.

**Caliburn.Features 샘플**을 보시면 EventDestinationViewModel.cs에서는 VisualStateResult.cs와 MessageDialogResult.cs에서 IResult 인터페이스를 구현했습니다. 그리고 코루틴으로 처리할 작업을 Execute() 메소드를 재정의하여 구현해 놓았습니다.

CM에서는 코루틴으로 묶어서 처리할 메소드를 정의할 때 IEnumerable<IResult> 리턴값을 갖는 메소드를 정의합니다. 이 메소드에서 실제로 코루틴으로 작업에 대한 순서를 지정합니다. 그리고 메소드 안에서 각 작업에 yield 키워드를 사용하여 각 작업들을 정의할 수 있다. 이 메소드에 정의한 순서대로 비동기 메소드들이 실행됩니다.

**Caliburn.Features 샘플**을 보시면 CoroutineViewModel.cs에서는 Execute() 메소드에서 코루틴을 정의한 메소드를 표시했습니다. 각 작업은 비동기 작업들로 정의되어 있지만, Execute()메소드 순차적으로 실행되는 것을 확인할 수 있습니다.

**\* Window Manager**

WPF에서는 큰 의미 없지만, CM에서 Window의 창을 관리할 수 있는 관리자를 제공합니다. WindowManager 클래스와 IWindowManager 인터페이스로 제공합니다. Event Aggregator에 설명했듯이 IEventAggregator 인터페이스 변수를 생성자로 받는 것처럼 IWindowManager 인터페이스 변수 선언하고 View Model 생성자의 파라미터로 바인딩하여 사용할 수 있습니다. IWindowManager나 WindowManager Windows 창(Modal 혹은 Non Modal 창)을 관리하기 위한 메소드를 제공합니다.

**Caliburn.Tutorial 샘플**을 보시면 ShellViewModel.cs의 About() 메소드 (About 버튼 이벤트 처리 메소드)에서는 WindowManager 클래스 객체를 직접 생성해서 사용했습니다. 이 코드를 IWindowManager 인터페이스 변수를 사용하는 코드로 변경하여 사용 가능합니다.

**\* PropertyChangedBase and BindableCollection**

C# 혹은 .NET에서 제공하는 PropertyChangedBase 추상 클래스 혹은 INotifyPropertyChanged 인터페이스, BindableCollection 클래스처럼 데이터(Model 혹은 Model의 Collection)이 변경되었을 때 UI에 이벤트를 알리기 위한 기능을 CM도 동일하게 제공합니다.

**\* Bootstrapper**

Bootstrapper는 CM이 실행하기 위한 기본 클래스 정의입니다. 모든 샘플 예제에서 보듯이 CM을 사용하기 위해서는 Bootstrapper.cs 혹은 BootstraperBase를 상속받은 클래스의 정의를 하고 App.xaml의 ResourceDictionary에 등록해야 사용 가능합니다.

Bootstrapper 클래스에는 보통 다음의 변수 혹은 메소드를 재정의해서 사용합니다.

* Container 변수 : 모든 View Model, View, Service, … 이 추가될 컨테이너입니다.
* Configure () 메소드 재정의 : Container 정의, Container에 DI(Dependency Injection)로 View Model, View, Service, … 를 넣어줍니다.
* OnStartup() 메소드 재정의 : 처음 시작할 ViewModel를 정의합니다. Main 화면을 정의합니다.
* GetInstance() 메소드 재정의 : Container에 포함된 특정 하나의 객체(View Model, View, Service, ...)를 얻기 위한 메소드입니다.
* GetAllInstances() 메소드 재정의 : Container에 포함된 모든객체(View Model, View, Service, ...)를 얻기 위한 메소드입니다.
* BuildUp() 메소드 재정의 : 동적(프로그램이 실행 중 : Runtime이라고 합니다.)으로 Container에 DI(Dependency Injection)로 View Model, View, Service, … 를 넣어줄 때 호출하는 메소드입니다. 프로그램 외부 객체 동적 참조가 가능하게 하기 위한 메소드입니다.

더 자세한 내용은 아래 내용을 참고해 주십시오.

* <https://caliburnmicro.com/documentation/configuration>
* <https://caliburnmicro.com/documentation/bootstrapper>

**\* Logging**

CM에서는 Logging 기능을 제공하지만 우리는 NLog를 사용할 예정이라서 여기서 자세한 설명은 생략합니다.

**\* 그 밖에 내용들 : 아래 내용 꼭 보십시오.**

1. **람다 (Lambda) 식 :** <https://learn.microsoft.com/ko-kr/dotnet/csharp/language-reference/operators/lambda-expressions>
2. **C# 비동기 :** <https://learn.microsoft.com/ko-kr/dotnet/csharp/programming-guide/concepts/async/>