

Coroutines와 XDK 샘플

# 설명

이 샘플은 Coroutines을 XDK와 함께 사용하는 방법을 보여줍니다. 이 코드는 Coroutines 기술 사양에서 다음과 같은 몇 가지 기능을 구현합니다.

* 기본 미래형
* 기본 대기 유형
* 기존 유형에 사용할 co\_wait 연산자 오버로딩

또한 비동기 프로그래밍을 위해 C++/WinRT 기본 헤더에 정의된 대기 유형을 사용하는 방법을 보여줍니다.

# 샘플 사용하기

Coroutines은 /await 컴파일러 스위치를 사용해야 하는 컴파일러 확장입니다. Visual Studio를 사용할 때 프로젝트 속성의 "추가 옵션"섹션에 추가할 수 있습니다. Coroutines에는 적어도 Visual Studio 2015 Update 3 또는 Visual Studio 2017이 필요합니다. 이 샘플은 Visual Studio 2017 (15.5 업데이트) 이상을 위해 작성되었으며, 가장 잘 작동합니다.

# 샘플 사용하기

이 샘플에서는 다음 작업을 사용할 수 있습니다.

|  |  |
| --- | --- |
| 액션 | 게임 패드 제어 |
| 계정 선택기 실행 | A | |
| 콘솔에서 사용자 쿼리 | X | |
| 가상 키보드 실행 | Y | |
| 샘플 종료하기 | 왼쪽 트리거 + 오른쪽 트리거 + 오른쪽 범퍼 | |

각 작업은 프로그램 흐름을 처리하기 위해 기다릴 수 있는 유형과 동시 루틴을 사용하는 다른 메커니즘을 보여줍니다.

## 오버로드된 co\_await 연산자로 계정 선택 도구 처리

단항 **co\_await** 연산자는 코 루틴의 중지를 요청하는 데 사용됩니다. 실행은 coroutine을 호출한 함수로 리턴됩니다. 이 연산자는 **awaitable** 타입이라고 하는 특정 타입에서만 사용할 수 있습니다. Awaitable 타입에는 다음 공용 멤버 함수가 포함되어야 합니다.

* **await\_ready** - co\_await 연산자를 사용하여 동시 루틴을 일시 중단할 때 정지가 필요한지 여부를 결정하기 위해 호출됩니다. 작업이 예상보다 빨리 완료되면 이 함수는 **true** 값을 반환 하여 보류할 필요가 없다는 신호를 보냅니다. 그렇지 않으면 **false** 값을 반환하여 Coroutine을 중지합니다.
* **await\_suspend** -이 함수는 Coroutine이 일시 중단될 때 호출됩니다. 만약 **await\_ready** 에서 **true** 값이 반환되면 이 함수는 호출되지 않습니다. 이것은 Coroutine이 일시 중단되기 전에 기다릴 수 있는 객체에 정보를 저장하는 것과 같이 필요한 모든 작업을 처리하는데 사용할수 있습니다.
* **await\_resume** - Coroutine이 다시 시작될 때 이 함수는 co\_await 연산자가 반환하는 값을 결정합니다.

계정 선택 도구를 여는 API는 **IAsyncOperation<AccountPickerResult^>^** 객체이지만, 이것은 awaitable 타입이 아닙니다. 글로벌 **연산자 co\_await** 과부하는 co\_wait 연산자가 이 타입의 객체와 함께 사용될 수 있도록 클라이언트 코드에 의해 제공됩니다. 그것은 서스펜션 로직을 처리하기 위해 기다릴 수 있는 대체 객체 **accountpicker\_await\_adapter를 제공합니다.** **await\_ready** 함수는 **IAsyncOperation의; 상태를 검사하고**이미 완료된 경우 Coroutine을 중단하지 않고 완료까지 실행할 수 있습니다. 그만큼 **await\_suspend** 함수는 IAsyncOperation의 **Completed** 이벤트를 사용하여 계정 선택기가 닫힌 후 Coroutine을 계속 진행하는 데 필요한 새 컨텍스트를 만듭니다. 이 샘플에서 **accountpicker\_await\_adapter의 구현은** XDK와 함께 제공되는 C++/WinRT 프로젝션 헤더에 정의되어 있는 **winrt::accountpicker\_await\_adapter** 구조체에 기초합니다.

## 사용자 정의 future 타입으로 가상 키보드 처리

Coroutine의 서명은 다른 서브 루틴과 동일하게 보입니다. 반환 형식, 이름 및 인수 목록이 있습니다. 유일한 차이점은 반환 타입은 **future** 타입이라는 것입니다. **std::future** 타입을 사용할 수 있으며 대부분의 목적에 충분합니다. 이 샘플은 기본 future 타입 **awaitable\_future**을 구현하며, 다른 구현을 위한 참조로 사용할 수 있습니다. awaitable 타입과 달리 future 타입은 특정 public 인터페이스를 요구하지 않습니다. 대신, future 타입은 자체 범위 내에서 **promise\_type** 구조체를 제공해야 합니다. promise\_type은 다음 public 인터페이스를 제공해야 합니다.

* **get\_return\_object** -이 함수는 coroutine이 호출될 때 coroutine의 본체가 실행되기 전에 호출됩니다. coroutine이 일시 중단 된 경우 호출자에게 반환되는 future 개체의 생성을 처리합니다.
* **(선택 사항) get\_return\_object\_on\_allocation\_failure** -이 함수를 제공하면 호출할 때 coroutine 컨텍스트에 사용할 수 있는 메모리가 없는 경우를 정상적으로 처리할 수 있습니다.
* **initial\_suspend** - 이것은 coroutine의 본체가 실행을 시작하기 전에 호출되어 호출자에게 실행을 즉시 일시 중단하고 실행해야 하는지 여부를 결정합니다. 이 함수가 **true** 값을 반환하는 경우 coroutine이 실행되기 시작합니다. **false** 값이 반환되는 경우, coroutine은 실행을 시작하기 전에 중단되지만 future 객체가 생성된 후에 중단됩니다.
* **final\_suspend** - coroutine의 끝까지 실행되거나 **co\_return** 문을 만날때 호출됩니다. 이 함수가 **true** 값을 반환하는 경우 coroutine 컨텍스트는 promise\_type과 함께 즉시 릴리스되어 포함된 데이터를 무효화합니다. 이 함수가 **false** 값을 반환하는 경우 promise\_type이 보유한 데이터는 호출자가 쿼리할 수 있도록 유효하지만 콜러는 또한 coroutine 컨텍스트를 정리하여야 합니다.
* **return\_value** 또는 **return\_void** - promise타입은 이러한 함수 중 하나만을 구현해야 합니다. coroutine이 **co\_return** 명령문을 사용하여 호출자에게 값을 반환하면 그 값은 호출자가 질의 할 promise\_type에 저장하기 위하여 **return\_value**에 전달됩니다. co\_return 문이 없거나 값이없는 co\_return 문인 경우 **return\_void를** 구현하여야 합니다. 두 경우 모두 함수는 coroutine이 실행을 마칠 때 **final\_suspend** 직전에 호출됩니다.
* **(선택 사항) operator new** - promise\_type에 이것이 제공되면,**::operator new** 대신에 coroutine 컨텍스트의 메모리를 할당하는 데 사용됩니다. 이는 메모리 사용량을 추적하거나 최적화된 할당자를 제공하는 데 유용합니다.
* **(선택 사항) 연산자 삭제** - 연산자 new가 오버로드되면, corotuine 컨텍스트를 해제한 후에 메모리를 비우기 위해 이 연산자를 제공해야 합니다.

**accountpicker\_await\_adapter와 달리**, 비동기 TCUI 함수가 완료되면 클라이언트 코드에서 제공하는 **keyboard\_await\_adapter**는 자동으로 coroutine을 다시 시작하지 않습니다. 대신 호출자가 적절한 시간에 코루틴을 다시 시작해야 합니다.

## 동일한 함수 범위에 있는 다중 스레드로 사용자 컬렉션 쿼리

Querying User::Users는 렌더링 스레드에서 액세스하는 경우 매우 오랜 시간이 걸릴 수 있습니다. C++/WinRT 투영 헤더에는 C++/WinRT 투영을 사용하지 않는 게임에서도 단일 함수 내에서 스레딩을 원활하게 처리하는 데 사용할 수 있는 여러 가지 awaitable 타입이 있습니다. **resume\_background** 의 인스턴스를 사용하여 co\_await 연산자가 있는 객체는 즉시 coroutine을 호출 한 함수를 다시 실행하고 새로운 스레드를 생성하여 coroutine을 즉시 계속 실행합니다. 이렇게하면 coroutine 본문의 나머지 부분과 동일한 범위에 있는 백그라운드 스레드의 User::Users에 액세스할 수 있습니다. 다른 유용한 awaitable 타입의 소스 코드는 XDK와 함께 제공되는 C++/WinRT 프로젝션 헤더의 base.h에서 볼 수 있습니다.

# 업데이트 기록

2018 년 2 월 - 최초 출시

# 개인정보처리방침

샘플을 컴파일하고 실행할 때 샘플 실행 파일의 이름이 Microsoft로 보내져 샘플 사용을 추적 할 수 있습니다. 이 데이터 수집을 거부하려면 Main.cpp에서 "샘플 사용 텔레메트리"라는 코드 블록을 제거하면 됩니다.

Microsoft의 개인 정보 취급 방침에 대한 일반적인 내용은 [Microsoft 개인 정보 취급 방침](https://privacy.microsoft.com/ko-kr/privacystatement/)을 참조하십시오.