게임 저장 샘플

*이 샘플은 Microsoft 게임 개발 키트 미리 보기(2019년 11월)와 호환됩니다.*

# 설명

이 샘플에서는 게임 코어 **XGameSave** API를 사용하여 게임 저장 데이터를 저장하고 로드하는 방법을 보여 줍니다. 또한 일반적인 “전체 동기화” 방법을 사용하여 클라우드 동기화를 처리하는 방법(사용자가 다른 콘솔에서 타이틀을 재생할 때 사용자의 게임 저장 컨테이너가 모두 동기화됨)과 “요청 시 동기화” 모드를 사용하는 방법(컨테이너를 동기화하는 시간 및 방법을 세부적으로 제어할 수 있는 권한 제공)을 보여 줍니다. 일반적으로 이 기능은 동기화하는 데 시간이 걸릴 수 있는 대규모 컨테이너에 사용하기 위한 것입니다.

또한 이 샘플에서는 새 게임 코어 API 노출 영역과 관련된 다양한 기술을 보여 줍니다. 이와 관련된 대부분의 파일은 \Helpers\ 폴더에 있으며 C++ 클래스에서 비동기 작업 배치, RAII 래퍼, 단일 플레이어 사용자 관리, 컨트롤러 바인딩 등과 같은 기술을 보여 줍니다.

이 샘플은 간단한 단일 플레이어 낱말 맞추기 게임 형식입니다. 게임 패드를 사용하여 보드의 지점을 선택하고 해당 위치에 넣을 문자를 선택합니다. 인접한 문자는 함께 연결되며, 유효한 영어 단어가 만들어지면 문자 빈도에 따라 단어 점수가 계산됩니다.

이 샘플에서 다루는 게임 저장 시나리오는 다음과 같습니다.

**전체 동기화 또는 요청 시 동기화 모드 사용**

샘플을 시작할 때 전체 동기화 API(콘솔과 타이틀 저장소 서비스 간에 모든 게임 저장 동기화)를 사용할지

요청 시 동기화 API(필요한 경우에만 게임 저장 데이터 동기화)를 사용할지 여부를 선택할 수 있습니다. 일반적인 실제 타이틀에서는 게임의 요구 사항에 따라 이 결정이 하드 코드되어 있습니다.

오늘 다룰 타이틀 중 일부에서만 요청 시 동기화 모드를 사용하므로, 타이틀에서 매우 **크고** 많은 컨테이너를 사용할 경우 모든 게임 데이터가 동기화될 때까지 기다리는 것이 불편할 수 있습니다.

참고: 전체 동기화를 이미 수행하고 요청 시 동기화를 시도해 보려는 경우 다른 사용자로 로그인하거나 게임 저장 데이터의 로컬 캐시를 지워야 합니다.

Xbox One용 로컬 캐시를 지우려면 XDK 명령 프롬프트에서 “xbstorage.exe reset /force” 명령을 실행합니다.

Windows 10 PC의 로컬 캐시를 지우려면 관리자 명령 프롬프트에서 "gamesaveutil.exe reset" 명령을 실행합니다.

참고: Windows 2019년 5월 SDK를 설치한 경우 **gamesaveutil.exe**는 다음 디렉터리에 있습니다.  
%ProgramFiles(x86)%\Windows Kits\10\Extension SDKs\XboxLive\1.0\Bin\x64

**게임 저장 데이터 로드, 저장, 삭제**

메뉴 옵션을 사용하여 게임 보드를 로드, 저장 및 삭제합니다. 최대 9개의 보드를 저장할 수 있습니다.

**컨테이너 및 BLOB 목록**

메뉴 옵션을 사용하여 컨테이너와 BOLB을 열거합니다. 출력이 게임 화면의 스크롤 가능한 디버그 출력 영역에 표시됩니다.

**마지막으로 수정한 날짜 및 남은 할당량 보기**

이 정보는 게임 화면에서 제목 바로 아래에 표시됩니다.

**사용자 로그아웃 시 자동 저장**

현재 게임 보드를 아직 저장하지 않은 경우 사용자가 로그아웃할 때 자동으로 저장됩니다.

**일시 중단 시 자동 저장**

일시 중단 이벤트에 반응하여 현재 게임 보드가 아직 저장되지 않은 경우 자동으로 저장합니다.

# 샘플 빌드

Xbox One 개발 키트를 사용하는 경우 활성 솔루션 플랫폼을 Gaming.Xbox.XboxOne.x64로 설정하세요.

Project Scarlett을 사용하는 경우 활성 솔루션 플랫폼을 Gaming.Xbox.Scarlett.x64로 설정하세요.

*자세한 내용은 GDK 문서에서* 샘플 실행하기*를 참조하세요.*

# 샘플 사용

**Xbox Live 샌드박스 요구 사항**

* Xbox One devkit: 콘솔의 샌드박스를 XDKS.1로 설정

**시작 메뉴**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | 작업 | 게임패드 | 키보드 | | "전체 동기화" 모드와 "요청 시 동기화" 모드 중 선택 | 왼쪽 스틱 또는 D-패드 | 화살표 키 | | 메뉴 항목 선택 | A 단추 | Enter | |

**게임 보드**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  |  | | --- | --- | --- | | 작업 | 게임패드 | 키보드 | | 커서 이동 | LS 또는 D-패드 | 화살표 키 | | 메뉴 항목 선택 | A 단추 | Enter | | 게임 저장 슬롯 선택 | LB/RB 단추 | 1~9 키 | | 커서 아래 문자 타일 변경 | RS 왼쪽/오른쪽 | A-Z 키 | | 커서 아래 문자 지우기 | X 단추 | Delete 키 또는 스페이스바 | | 디버그 출력 스크롤 | RS 위쪽/아래쪽 | Page Up/Dn  및  Home/End | |

# 게임 각주/미주

* **보드 읽기**여기서 대부분의 작업은 **GameSaveManager::ReadBlocking**에 의해 수행됩니다. 여기서는 **XGameSaveCreateContainer** 및 **XGameSaveReadBlobData** API를 사용하여 현재 게임 저장 슬롯에 대한 게임 보드를 로드합니다.
* **보드 저장**여기서 대부분의 작업은 **GameSaveManager::SaveBlocking**에서 시작됩니다. **XGameSaveCreateContainer, XGameSaveCreateUpdate, XGameSaveSubmitBlobWrite, XGameSaveSubmitBlobDelete** 및 **XGameSaveSubmitUpdate** API를 사용하여 현재 게임 저장 슬롯에 대한 게임 보드를 저장합니다.
* **보드 초기화**  
  모든 문자의 보드를 지웁니다. 보드를 이전에 저장한 경우 보드가 "더티"로 표시됩니다. 보드를 아직 저장하지 않은 경우 보드가 "더티"로 표시되지 않습니다.
* **보드 삭제**이 작업은 **GameSaveManager::DeleteBlocking** 내에서 수행됩니다. **XGameSaveDeleteContainer** API를 사용하여 현재 게임 저장 슬롯에 대한 게임 보드를 삭제합니다.
* **보드 BLOB 삭제**작업은 주로 **GameSaveManager::DeleteBlobsBlocking**에 의해 처리됩니다. **XGameSaveCreateContainer, XGameSaveCreateUpdate, XGameSaveSubmitBlobDelete** 및 **XGameSaveSubmitUpdate** API를 사용하여 빈 컨테이너를 남겨 두고 현재 게임 보드에 대한 BLOB만 삭제합니다.
* **컨테이너 목록**여기서 작업은 **GameSaveManager::EnumerateContainersBlocking**에서 수행됩니다. **XGameSaveEnumerateContainerInfo** API를 사용하여 게임 화면의 디버그 출력 영역에 모든 컨테이너를 열거합니다.
* **컨테이너 및 BLOB 목록**여기서 작업은 **GameSaveManager::EnumerateContainersBlocking**에서 수행됩니다. **XGameSaveEnumerateContainerInfo, XGameSaveCreateContainer** 및 **XGameSaveEnumerateBlobInfo** API를 사용하여 게임 화면의 디버그 출력 영역에 모든 컨테이너와 BLOB을 열거합니다.

게임 플레이 메모

**게임 플레이**

게임은 5 x 5 눈금에서 재생됩니다. 눈금의 아무 곳에나 문자를 입력할 수 있습니다. 보드에서 십자 또는 아래로 연속하는 문자가 인식 가능한 영어 단어를 형성할 경우 단어 내 각 문자에 대한 점수 값을 합산한 점수를 획득합니다. 목표는 점수를 최대화하는 것입니다. 각 보드에 입력할 수 있는 각 문자 수는 제한됩니다. 게임 보드 바로 위에 남은 문자 수가 추적됩니다.

**게임 보드 로드**

데모용으로 제작된 게임 보드는 게임 보드 화면이 나타나거나 새 게임 저장 슬롯으로 전환할 때 **자동으로 로드되지 않습니다**. 따라서 특정 게임 저장 슬롯에서 로드 작업과 저장 작업을 완벽히 제어할 수 있습니다.

**변경된 게임 보드 및 자동 저장**

게임 보드에서 문자를 변경하거나 초기화 메뉴 명령을 사용할 경우 게임 보드는 "더티"로 표시되고 화면 상단의 보드 이름 뒤에 별표가 표시됩니다. 더티 게임 보드는 다음과 같은 경우 자동으로 저장됩니다.

* 다른 게임 보드로 전환(게임 패드 LB/RB)
* 사용자 로그아웃
* 게임 일시 중단

# 구현 참고 사항

**GameSaveManager** 클래스는 게임 저장 작업을 관리합니다. **InitializeForUser()** 메서드는 플레이어의 연결된 저장소 저장 컨텍스트를 설정합니다. 저장 데이터를 로드, 저장, 열거 및 삭제하는 메서드도 있습니다. 클래스의 각 메서드에 대한 사용 참고 사항은 헤더 파일의 메모를 참조하세요.

이 샘플은 일부 사용자 관리 및 게임 패드 관리 기능을 포함하지만 *이러한 기능을 설명하기 위한 것이 아니며*, 다양한 환경에서 게임 저장을 사용하는 방법을 이해하기 위한 것입니다.

게임에서는 두 가지 유형의 게임 데이터 구조(색인 및 게임 보드)가 사용됩니다. 서식 파일 형식의 **GameSave** 클래스를 사용하면 일반적으로 **GameSaveManager**가 모든 유형의 게임 데이터를 로드 및 저장할 수 있습니다. GameSaveManager.h에서 **GameBoardIndex** 구조체로 정의되는 색인은 주로 플레이어에서 사용되는 마지막 저장 슬롯을 추적하는 데 사용됩니다("활성 보드"). 게임 보드 데이터는 GameBoard.h에 **GameBoard** 구조체로 표시됩니다.

# 추가 샘플 코드

이 샘플에는 사용자 코드에서 게임 코어 API 작업을 도와주는 일련의 추가 도우미 클래스가 함께 제공됩니다. 대부분의 파일은 \Helpers\ 폴더에 있습니다.

|  |  |  |
| --- | --- | --- |
| 폴더 | 파일 이름 | 설명 |
| \Helpers\ | HandleWrapperBase.h  XGameSaveHandleWrappers.h XTaskQueueHandleWrapper.h XTaskQueueHandleWrapper.cpp XUserHandleWrapper.h | 샘플에 사용되는 주요 API에 정의된 게임 코어 핸들 형식에 대한 RAII 클래스 래퍼입니다. |
| \Helpers | User.h  User.cpp | 타이틀에 보유하고 쿼리할 수 있는 사용자 개체입니다. 중심에는 XUserHandle이 있습니다. 단일 로그인 사용자 관리자가 반환하는 개체 유형입니다. |
| \Helpers | UserManager.h  UserManager.cpp | 게임 중인 한 명의 사용자를 추적하고 다른 사용자를 선택할 수 있는 단일 로그인 사용자 관리자입니다. |
| \Helpers | UTF8Helper.h  UTF8Helper.cpp | Utf-8 텍스트 변환 함수 |
| \Helpers | Buffer.h  Buffer.cpp | BLOB 처리 클래스로서, 개체가 제거되면 메모리를 해제하여 메모리를 할당하고 할당된 버퍼의 시작 위치와 크기를 추적합니다. 또한 이동 의미 체계를 지원합니다. |
| \Helpers | StateMachine.h | 상태 시스템의 잠금 없는 구현입니다. 또한 잠금이 없는 버전이 제공되며 상태 시스템이 특정 상태로 변경될 때까지 소비자가 기다릴 수 있습니다. |
| \Helpers | AsyncOp.h  AsyncOp.cpp | API에서 비동기 호출의 진행 상황을 추적하기 위해 확장할 수 있는 기본 클래스입니다. 또한 비동기 작업 큐에서 작업을 수행할 수 있는 AsyncTask 구현을 제공합니다. |
| \Helpers | AsyncAction.h | 비동기 작업 큐에서 std::function 또는 람다를 수행하고 생성된 결과(있는 경우)를 가져올 수 있는 서식 파일 기반 클래스입니다. 이 클래스는 ppltasks와 유사하게 작동합니다. |
| \Helpers | ScopedLockWrappers.h | Slim Reader/Writer 잠금 주위의 범위가 지정된 래퍼이며, 입력 시도를 지원하고 잠금 범위 내 페이로드 데이터에 대해서만 액세스를 제공합니다. |
| \Helpers | TaskQueue.h TaskQueue.cpp | 초기화할 때 코어당 작업자 작업 발송 큐 하나와 작업 완료 큐 하나를 만들고, 해당 코어의 작업 큐를 쿼리할 수 있도록 해줍니다.  **참고:** 이 코드에서는 시작할 때 *일시 중단*되었다가 다시 시작하기 전에 특정 코어에 연결되는 스레드를 올바르게 만드는 방법을 보여 줍니다. 따라서 기존 코어의 작동 또는 코어 홉핑을 방해하지 않고 스레드를 회전할 수 있습니다. |
| \Common\ | InputDeviceManager.h  InputDeviceManager.cpp  ScopedGameInputDeviceInfo.h GamePad.h GamePad.cpp | 게임 패드 연결/연결 끊기 이벤트, 사용자 장치 연결, 기존 DirectXTK 게임 패드 구현에 GameInput 판독값 연결 등을 추적하는 게임 패드 관리자입니다. |
| \ | Assets.h Assets.cpp SampleSpecificAssets.inl | 간단한 스트리밍 자산 관리 시스템의 토대가 되며, 이전 버전에 비해 샘플에 대한 자산 관리를 중앙 집중화하여 PC에서 분실된 디바이스 처리를 간소화하는 데에도 사용할 수 있습니다. |

## 비동기 큐 사용

샘플에서는 기본 시스템 스레드 풀에서 작업 및 알림을 제출하는 작업 큐를 만듭니다. 샘플 코드로 만든 모든 비동기 작업은 이 큐에서 대기합니다. 이러한 작업은 스레드 풀에서 사용 가능한 다음 스레드로 실행됩니다.

모든 사용자 디바이스 연결, 사용자 이벤트 및 GameInput 관련 콜백 작업은 Common\InputDeviceManager.h에 정의된 **DEFAULT\_INPUT\_WORK\_AND\_CALLBACK\_CORE**의 큐에 대기합니다. 이는 이러한 이벤트를 단일 스레드로 직렬화하여 모든 이벤트를 간소화하기 위한 것입니다.

## 스레드 안전

사용자 관리자 클래스 및 게임 패드 관련 클래스에서는 스레드 간 동기화에 기존 중량 잠금 메커니즘을 사용하며 스레드로부터 안전해야 합니다. 하지만 재입력으로부터 안전하지 않을 수 있습니다.

# 알려진 문제

* **GameInput** 컨트롤러가 전원 차단 유휴 상태가 되면 GameInput 라이브러리가 중단됩니다.

# 업데이트 기록

**초기 릴리스: 2019년 8월**

새 버전은 주로 새 작업 시스템인 게임 코어 API를 활용하고, 동기식(차단) GameSave 호출, 새 자산 로더 코드, 새 사용자 관리 작업을 사용하고, 모든 WinRT 관련 코드를 제거하도록 다시 작성되었습니다. 이 버전은 향후 릴리스에 다시 추가될 PC 지원을 제거합니다.

# 개인정보처리방침

샘플을 컴파일하고 실행할 때 샘플의 사용을 추적하는 데 도움이 되도록 샘플 실행 파일의 파일 이름이 Microsoft에 전송됩니다. 이 데이터 수집을 옵트아웃하려면 프로젝트 설정의 C/C++/전처리기/전처리기 정의 목록에서 ATG\_ENABLE\_TELEMETRY를 제거합니다.

Microsoft의 일반 개인정보취급방침에 대한 자세한 내용은 [Microsoft 개인정보처리방침](https://privacy.microsoft.com/en-us/privacystatement/)을 참조하세요.