**WordGame のゲーム セーブ サンプル (C++/WinRT)**

このサンプルには、Xbox One XDK 2017年3月以降が必要です。

# 概要

このサンプルでは、Connected Storage API を使用して完全なエンド ツー エンドのゲーム エクスペリエンスを構築する、Xbox One XDK のゲーム保存機能を紹介します。**C++/WinRT** プロジェクションヘッダーを使用した Connected Storage の使用方法を示します。このサンプルでは、C++/WinRT の予測と互換性がある **IBuffer** インターフェースの単純な実装が含まれており、並列プログラミングのための **CoRoutines** の使用方法を示しています。

このサンプルでは 以下のゲームのセーブ シナリオのオプションを提供します。

* **「完全同期」または「同期オンデマンド」モードを使用する**

デモンストレーションの目的で、サンプル起動時に、完全同期 API (コンソールとタイトル保存サービス間で、すべてのゲームのセーブ データを同期する)　または　同期オンデマンド API (ゲームのセーブ データのみを同期する)　のいずれかを使用することを、必要に応じて選択できます。  
注: 既に完全同期を実行していて、オンデマンド同期を試したい場合は、別のユーザーでサインインするか、ゲームのセーブ データのローカルキャッシュを消去する必要があります。

* + Xbox One のローカルキャッシュを消去するには、XDK コマンドプロンプトから「xbstorage.exe reset/force」コマンドを実行します。
* **ゲームのセーブ データのロード、保存、削除**

メニューオプションを使用して (Get または Read API を使用して) ゲーム ボードをロードし、保存してから削除します。最大 9 枚のボードを保存できます。

* **コンテナと Blobs をリストする**

メニューオプションを使用して、コンテナと Blobs を列挙します。出力はゲーム画面のスクロール可能なデバッグ出力領域に表示されます。

* **最終変更日と残りのクォータを表示する**

この情報はゲーム画面のタイトルのすぐ下に表示されます。

* **ユーザーサインアウト時に自動保存する**

現在のゲーム ボードがまだ保存されていない場合は、ユーザーがサインアウトしたときに自動的に保存されます。

* **中断時に自動保存する**一時停止イベントに反応して、現在のゲーム ボードがまだ保存されていない場合、自動的に保存されます。

# サンプルをビルドする

**XDK ビルド要件**

* Visual Studio 2017 (15.3アップデート) もしくはそれ以降
* Xbox One XDK

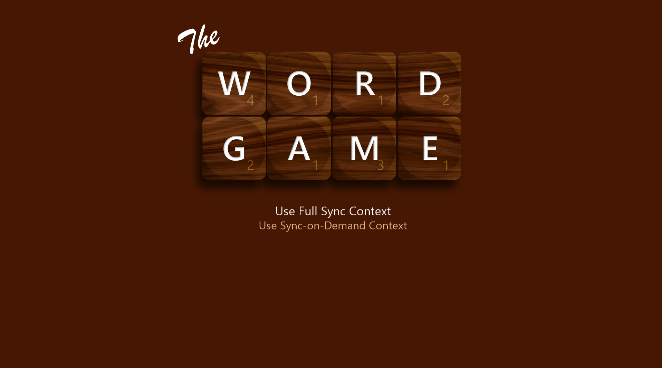
# サンプルの使用

**Xbox Live サンドボックスの要件**

* 開発キットのサンドボックスを **XDKS.1** に設定します。

**スタート メニュー**

|  |  |
| --- | --- |
| 作用 | ゲームパッド |
| 「完全同期」 モードと 「同期オンデマンド」 モードの間で選択 | LS または D-Pad |
| メニュー項目を選択 | ボタン |



**ゲーム ボード**

|  |  |
| --- | --- |
| 作用 | ゲームパッド |
| カーソルを移動 | LS または D-Pad |
| メニュー項目を選択 | ボタン |
| ゲーム セーブ スロットを選択 | LB/RB ボタン |
| カーソルの下の文字のタイルを変更 | RS 左/右 |
| カーソルの下の文字を消去する | X ボタン |
| デバッグ出力をスクロールする | RS 上/下 |



# ゲーム メニューのメモ

* **ボードを入手  
  GetAsync** API を使用して、現在のゲーム セーブ スロット用のゲーム ボードをロードします。
* **ボードを読む**  
  **ReadAsync** API (GetAsync の代替メソッド) を使用して、現在のゲーム セーブ スロット用のゲーム ボードをロードします。
* **ボードを保存**  
  **SubmitUpdatesAsync** API を使用して、現在のゲーム セーブ スロット用にゲームボードを保存します。
* **ボードをリセット**  
  ボードからすべての文字を消去します。ボードが以前に保存されている場合、ボードは「ダーティ」とマークされます。ボードがまだ保存されていない場合、ボードは「ダーティ」とマークされません。
* **ボードを削除  
  DeleteContainerAsync** API を使用して、現在のゲーム セーブ スロットのゲーム ボードを削除します。
* **ボード Blob を削除**  
  **SubmitUpdatesAsync** API を使用して、現在のゲーム ボードの Blobs だけを削除し、空のコンテナを残します。
* **コンテナの一覧表示**  
  **GetContainerInfo2Async** API を使用して、すべてのコンテナを列挙し、それらをゲーム画面のデバッグ出力領域に一覧表示します。
* **コンテナと Blobs の一覧表示  
  GetContainerInfo2Async** および **GetBlobInfoAsync** API を使用して、すべてのコンテナと Blobs を列挙し、それらをゲーム画面のデバッグ出力領域に一覧表示します。

ゲーム プレイのノート

**ゲーム プレイ**

ゲームは 5x5 のグリッドで行われます。グリッド上の任意の場所に文字を配置できます。認識された英語の単語を形成する連続した文字は、その単語の各文字のポイント値の合計に基づいて、ポイントを獲得します。目的はあなたのスコアを最大にすることです。各ボード上に配置する各文字の数は限られています。残りの数はゲーム ボードのすぐ上で追跡されます。

**ゲーム ボードのロード**

デモンストレーションの目的で、ゲーム ボード画面が表示されたとき、または新しいゲームセーブ スロットに切り替えたときに、ゲーム ボードは**自動的にはロードされません**。これにより開発者は、特定のゲーム セーブ スロットでロード方法 (Get または Read) を選択し、ロードおよびセーブ操作を完全に制御することができます。

**ゲーム ボードと自動保存の変更**

ゲーム ボードで文字が変更されたときや、[リセット] メニュー コマンドを使用した場合、ゲーム ボードは「ダーティ」とマークされます (画面上部のボード名の後にアスタリスクが表示されます)。ダーティー ゲームボードは以下の条件で自動保存されます。

* 別のゲームボードに切り替える (ゲームパッド LB/RB)
* ユーザー サインアウト
* ゲーム中断

# 実装上の注意

**GameSaveManager** のクラスはゲームのセーブ操作を管理します。**InitializeForUser（）** メソッドはプレーヤーの Connected Storage 保存コンテキストを設定します。セーブ データのロード、保存、列挙および削除のためのメソッドもあります。クラス内のメソッドの使用上の注意については、ヘッダーファイル内のコメントを参照してください。

ゲームで使用されるゲーム データ構造には、インデックスとゲームボード の 2 種類があります。テンプレート化された **GameSave** クラスは、GameSaveManager の使用する、任意の種類のゲーム データの一般的なロードおよび保存のためのメソッドを提供します。GameSaveManager.h の **GameBoardIndex**　構造体によって定義されたインデックスは、主にプレイヤーによって使用された最後のセーブ スロット (「アクティブ ボード」) を追跡するために使用されます。ゲーム　ボードのデータは、GameBoard.h の **GameBoard**構造体によって表されます。

# 既知の問題点

なし

# 更新履歴

**初回リリース：*2017 年 10 月***

2017 年 10 月の XDK でリリースされました

# プライバシーステートメント

サンプルをコンパイルして実行すると、サンプルの実行可能ファイルの名前がMicrosoftに送信され、サンプルの使用状況の追跡に役立ちます。このデータ収集を無効にするには、Main.cppの「Sample Usage Telemetry」というラベルの付いたコードブロックを削除します。

マイクロソフトのプライバシーポリシー全般に関する詳細については、 [Microsoftのプライバシーステートメント](https://privacy.microsoft.com/ja-jp/privacystatement/)をご参照ください。