游戏保存示例（电脑）

此示例与 Microsoft 游戏开发工具包（2019 年 11 月）兼容

# 说明

此示例演示如何使用 Game Core XGameSave API 保存和加载游戏保存数据。它展示了如何使用常规的“完全同步”方法处理云同步（当用户在另一个主机上播放标题时，所有用户的游戏保存容器都将同步），以及如何使用“按需同步”模式更精细地控制同步容器的时间和方式。本示例通常适用于较大的容器，这类容器用于同步的时间可能较长。

本示例还展示了与新 Game Core API 外围应用相关的各种其他技术。与此相关的大多数文件均可以在 \Helpers\ 文件夹中找到，这些文件演示了在 C++ 类中包装异步操作、RAII 包装器、单个玩家用户管理等技术。

本示例采用简单的单个玩家单词字谜游戏的形式。使用键盘选择板上的点，然后选择要放置在这些位置的字母。相邻字母串联在一起，如果你创建出一个有效的英语单词，它们将根据字母频率计算单词得分。

本示例涵盖以下游戏保存方案：

使用“完全同步”模式或“按需同步”模式

启动示例后，可以选择示例是使用完全同步 API

（在主机和标题存储服务之间同步所有游戏保存数据），还是使用按需同步 API（仅在需要时才同步游戏保存数据）。通常，实际标题会根据游戏需求对选择的结果执行硬编码。

目前发售的标题很少使用“按需同步”模式；使用此模式的标题往往倾向于使用大小和数目都很庞大的容器，在这种情况下，用户需要等待完成所有游戏数据同步，非常不方便。

注意：如果你已执行完全同步，但需要尝试使用按需同步，应使用其他用户登录，或清除游戏保存数据的本地缓存。

若要清除 Windows 10 电脑的本地缓存，请在管理员命令提示符处运行“gamesaveutil.exe reset”命令。

注意：安装好 Windows 2019 年 5 月版 SDK 后，可以在以下目录中找到 gamesaveutil.exe：  
%ProgramFiles(x86)%\Windows Kits\10\Extension SDKs\XboxLive\1.0\Bin\x64

加载、保存和删除游戏保存数据

使用菜单选项加载、保存和删除游戏板。最多可以保存 9 个不同的板。

列出容器和 blob

使用菜单选项枚举容器和 blob。输出显示在游戏屏幕的可滚动调试输出区域中。

查看上次修改日期和剩余配额

此信息显示在游戏屏幕标题的正下方。

用户注销时自动保存

如果尚未保存当前游戏板，当用户注销时，它会自动保存。

暂停时自动保存

发生暂停事件时，如果尚未保存当前游戏板，它会自动保存。

# 构建示例

构建要求

* Visual Studio 2017（15.7 更新或更高版本）
* Microsoft 游戏开发工具包（2019 年 9 月）

# 使用示例

沙盒要求

* Windows 10 电脑：[将电脑的沙盒设置为 XDKS.1](https://docs.microsoft.com/en-us/gaming/xbox-live/xbox-live-sandboxes)

启动要求

该应用程序与 Xbox Live 服务进行通信，以提供游戏保存功能，因此需要一个正确的标识才能正常运行。要应用标识，必须从“开始”菜单（或“任务栏”，若已固定）注册并启动游戏。不能通过 F5 运行，也不能直接执行 .exe。如果执行上述操作，XGameSaveInitializeProvider() 将失败，并显示 E\_GAMEUSER\_NO\_PACKAGE\_IDENTITY (0x89245110)。

若要设置本地生成的示例版本以正常运行，请执行以下操作：

1. 构建示例
2. 打开 Desktop Gaming VS 2017 游戏命令提示符
3. 运行以下命令以注册应用：  
   wdapp register [Gaming.Desktop.x64\Debug 文件夹的绝对路径]
4. 从“开始”菜单启动应用
5. 按需附加调试器

如果需要在注册后修改应用，请在“开始”菜单中右键单击该应用，将其卸载。之后，可以重新生成该应用，然后使用上述步骤进行重新注册。

启动菜单

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | 操作 | 键盘 | | 在“完全同步”模式和“按需同步”模式之间进行选择 | 箭头键 | | 选择菜单项 | Enter | |

游戏板

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | |  |  | | --- | --- | | 操作 | 键盘 | | 移动光标 | 箭头键 | | 选择菜单项 | Enter | | 选择游戏保存槽 | 1 - 9 键 | | 更改光标下的字母磁贴 | A - Z 键 | | 清除光标下的字母 | Delete 或空格键 | | 滚动调试输出 | Page Up/Dn  和  Home/End | |

# 游戏菜单备注

* 读取板其中大部分操作是由 GameSaveManager::ReadBlocking 执行的。 将使用 XGameSaveCreateContainer 和 XGameSaveReadBlobData API 加载当前游戏保存槽的游戏板。
* 保存板其中大部分操作都是从 GameSaveManager::SaveBlocking 开始的。使用 XGameSaveCreateContainer、XGameSaveCreateUpdate、XGameSaveSubmitBlobWrite、XGameSaveSubmitBlobDelete 和 XGameSaveSubmitUpdate API 保存当前游戏保存槽的游戏板。
* 重置板  
  清除板上的所有字母。如果之前保存过该板，会将它标记为“脏”。如果尚未保存过板，不会将它标记为“脏”。
* 删除板此操作在 GameSaveManager::DeleteBlocking 内执行。使用 XGameSaveDeleteContainer API 删除当前游戏保存槽的游戏板。
* 删除板 blob此操作主要由 GameSaveManager::DeleteBlobsBlocking 处理。使用 XGameSaveCreateContainer、XGameSaveCreateUpdate、XGameSaveSubmitBlobDelete 和 XGameSaveSubmitUpdate API 仅删除当前游戏板的 blob，留下一个空容器。
* 列出容器此操作在 GameSaveManager::EnumerateContainersBlocking 中执行。使用 XGameSaveEnumerateContainerInfo API 枚举所有容器，并在游戏屏幕的调试输出区域中列出容器。
* 列出容器和 blob此操作在 GameSaveManager::EnumerateContainersBlocking 中执行。使用 XGameSaveEnumerateContainerInfo、XGameSaveCreateContainer 和 XGameSaveEnumerateBlobInfo API 枚举所有容器和 blob，并在游戏屏幕的调试输出区域中列出容器。

开始游戏备注

开始游戏

将在 5 x 5 网格上开始游戏。可以在网格的任意位置放置字母。无论是横排还是竖排，组成一个可识别的英语单词的连续字母都将根据单词中每个字母的分值总和进行评分。目标是尽可能多得分。可在每个板上放置的字母数量有限。可以在游戏板上方跟踪剩余计数。

游戏板加载

为便于演示，在出现游戏板屏幕或切换到新的游戏保存槽时，不会自动加载游戏板。你可以在特定的游戏保存槽中完全控制加载和保存操作。

更改后的游戏板和自动保存

更改游戏板上的字母或使用“重置”菜单命令后，游戏板将被标记为“脏”（在屏幕顶部的板名称后标星号来表示）。脏游戏板将在以下情况下自动保存：

* 切换到其他游戏板（游戏手柄左前按钮/右前按钮）
* 用户注销
* 游戏暂停

# 实现说明

GameSaveManager类管理游戏的游戏保存操作。InitializeForUser()方法为玩家设置“连接存储”保存上下文。另外，还提供用于加载、保存、枚举和删除保存数据的方法。有关此类中每个方法的使用说明，请参阅头文件中的注释。

尽管此示例展示一些用户管理和游戏手柄管理功能，但并不是为了演示这些功能。 本示例旨在介绍不同情况下游戏保存的用法。

游戏使用两种类型的游戏数据结构：索引和游戏板。模板化的 GameSave 类提供了由 GameSaveManager 使用的方法，用于加载和保存数据，这些数据通常为任何类型的游戏数据。该索引由 GameSaveManager.h 中的 GameBoardIndex 结构定义，主要用于跟踪玩家使用的最后一个保存槽（“活动板”）。游戏板数据由 GameBoard.h 中的 GameBoard 结构表示。

# 其他示例代码

此示例随附一系列其他帮助程序类，你可以在自己的代码中用来处理 Game Core API。大部分文件可在 \Helpers\ 文件夹中找到。

|  |  |  |
| --- | --- | --- |
| 文件夹 | 文件名 | 说明 |
| \Helpers | HandleWrapperBase.h  XGameSaveHandleWrappers.h XTaskQueueHandleWrapper.h XTaskQueueHandleWrapper.cpp XUserHandleWrapper.h | 位于 Game Core 句柄类型周围的 RAII 类包装器，由示例使用的关键 API 定义。 |
| \Helpers | User.h  User.cpp | 一个用户对象，它可由标题保留且可供查询。核心是 XUserHandle。它是单一登录的用户管理器返回的对象类型。 |
| \Helpers | UserManager.h  UserManager.cpp | 一个单一登录的用户管理器，用于跟踪正在玩游戏的那个用户，同时允许选择其他用户。 |
| \Helpers | UTF8Helper.h  UTF8Helper.cpp | UTF-8 文本转换函数。 |
| \Helpers | Buffer.h  Buffer.cpp | blob 处理类；它们分配内存并跟踪已分配缓冲区的起始位置和大小，同时会在销毁对象后释放内存。它们还支持移动语义。 |
| \Helpers | StateMachine.h | 状态机的无锁实现。（还提供了一个版本，该版本无锁，并允许使用者等待状态机更改为特定状态为止）。 |
| \Helpers | AsyncOp.h  AsyncOp.cpp | 一个基类，你可以扩展它以跟踪 API 中异步调用的进度。此外，还提供一个 AsyncTask 实现，可用于对异步任务队列执行操作。 |
| \Helpers | AsyncAction.h | 一个基于模板的类，它在异步任务队列上执行 std::function 或 lambda，并允许获取生成的结果（如果有）。这与 ppltasks 的工作方式类似。 |
| \Helpers | ScopedLockWrappers.h | Slim 读写器锁周围的作用域包装器，支持 try-enter，并仅支持提供对锁作用域内的有效负载数据的访问权限。 |
| \Helpers | TaskQueue.h TaskQueue.cpp | 在初始化时，对每个核心创建一个工作人员任务分派队列和一个工作完成队列，并允许查询该核心的任务队列。  备注： 此代码展示如何正确创建在启动时挂起的线程，然后在恢复该线程之前将其关联到特定的核心。这样，它就能够在不干扰现有核心的工作或不存在核心切换的情况下启动。 |
| \ | Assets.h Assets.cpp SampleSpecificAssets.inl | 这是简单流式处理资产管理系统的基础，与之前的版本相比，它可以集中管理示例资产，简化电脑端丢失设备的处理流程。 |

## 异步队列使用情况

本示例创建一个任务队列，该任务队列在默认系统线程池上提交任务和通知。由示例代码创建的所有异步工作负载都在此队列中排队。这些任务将在线程池的下一个可用线程中运行。

所有用户设备关联、用户事件以及与 GameInput 相关的回调工作都会被列入到 GameSave\_Desktop.cpp 中定义的 DEFAULT\_INPUT\_WORK\_AND\_CALLBACK\_CORE 中。其目的是通过强制将这些事件全部序列化到一个线程上，简化事件处理过程。

## 线程安全

用户管理器类和与游戏手柄相关的类使用传统的重型锁定机制进行跨线程同步，应该是线程安全的。但是，它们并非重新进入也安全。

# 更新历史记录

初始发布：2019 年 9 月

新版本，重新编写它主要是为了使用新任务系统、Game Core API，使用同步（阻止）GameSave 调用、新的资产加载程序代码、新的用户管理工作，删除了所有与 WinRT 相关的代码。

# 隐私声明

在编译和运行示例时，示例可执行文件的文件名将发送给 Microsoft，用于帮助跟踪示例使用情况。若要选择退出此数据集合，请从项目设置的 C/C++ / 预处理器/预处理器定义列表中删除 ATG\_ENABLE\_TELEMETRY。

有关 Microsoft 的一般隐私策略的详细信息，请参阅《[Microsoft 隐私声明](https://privacy.microsoft.com/en-us/privacystatement/)》。