ChatGPT

第五章：模組設定檔與 Scribe 序列化

學習目標

本章將帶領你建立 **模組設定系統**，讓玩家可以透過遊戲內的介面調整模組參數，並學習如何在 **執行期間讀取、修改與保存** 這些設定值。你將學會： - 建立 RimWorld 模組所需的 **設定類別** 與 **模組類別**，使模組擁有可調整的參數 [1] 。 - 使用 **Scribe 序列化系統**（如 `Scribe_Values`, `Scribe_Collections` 等 API）將設定值保存到 XML 檔案，模擬 RimWorld 對模組設定的序列化機制 [2] 。 - 在 Unity 專案中模擬 RimWorld 模組的 **資料夾結構**，實作一個簡易的 **設定檔 XML** 儲存與存取流程。 - 建立一個 **使用者介面**（UI）供玩家調整設定，並整合模組初始化流程，使遊戲啟動時自動載入設定、關閉設定介面時自動保存設定。

技術重點

- **雙類別架構**：了解 RimWorld 模組設定系統由兩個類別構成：一個繼承自 `Mod`（模組主類別），一個繼承自 `ModSettings`（模組設定類別） [3] 。模組主類別負責初始化設定並提供設定介面，模組設定類別則保存實際的設定值並負責序列化。
- **Scribe 序列化**：掌握 RimWorld 提供的 Scribe 靜態類別方法將資料序列化到載入/保存流程中。例如，`Scribe_Values.Look` 用於保存/讀取簡單數值（int、float、bool、字串等）以及部分結構類型 [4] ；`Scribe_Collections.Look` 用於保存集合類型（如 List、Dictionary、HashSet），需要指定集合中元素的序列化模式（LookMode） [5] 。透過 Scribe，你可以在 `ExposeData()` 方法內描述如何保存和載入每個欄位的值 [2] 。
- **XML 文件保存**：設定值將保存到 XML 檔案中。我們會設計對應的 XML 結構，每個設定對應一個節點名稱，使模組在下次啟動時能從該檔案讀回之前的設定。理解如何使用 C# 的檔案與 XML 操作（如 `System.IO` 和 `System.Xml`）來寫入與讀取設定檔。
- **Unity 實作與模組結構**：在 Unity 中模擬 RimWorld 的模組資料夾結構（如 `About/`, `Defs/`, `Assemblies/` 等），並使用 ScriptableObject 或自訂的 C# 類別來管理設定資料。瞭解 Unity 中 ScriptableObject 可用於儲存預設設定，以及如何撰寫序列化器將設定保存成 XML 檔。
- **使用者介面整合**：學習如何以 Unity UI 元件建立一個簡易 **設定面板**（例如使用 Toggle、Slider 等），讓使用者在遊戲中變更模組設定。了解如何在 UI 和設定資料之間做資料繫結，並處理使用者輸入（例如點擊儲存按鈕或關閉面板時觸發保存）。
- **初始化與保存流程**：將模組設定的載入與保存整合進模組的生命週期。在模組初始化時載入設定檔，使預設值生效；在玩家調整參數後及時更新設定物件；在關閉設定介面或遊戲退出時保存設定檔 [6] 。確保整個流程穩定運作，包含預設值處理以及檔案不存在時的例外處理（第一次運行時建立新設定檔）。

Unity 內部實作方式（ScriptableObject、C# 設定類、序列化器）

在開始實作之前，我們先討論在 Unity 中管理設定資料的幾種方式，以及如何模擬 RimWorld 的 Scribe 序列化系統。

使用 ScriptableObject 管理預設設定

Unity 的 **ScriptableObject** 是一種方便的資料容器，常用於在編輯期間保存設定資源。我們可以建立一個 ScriptableObject 來保存模組設定的**預設值**。例如，創建 `ModConfig.asset` 來存放各項設定的初始值，然後在遊戲開始時讀取此資源，將值載入我們的設定系統。使用 ScriptableObject 有以下優點： - 可直接在 Unity 編輯器中編輯預設參數，調整方便。 - 資料以資源檔案形式存在於專案中，適合保存**初始設定**或固定不變的配置。

不過，ScriptableObject **並不會**在玩家改變設定時自動保存到外部檔案。它主要適用於編輯器內的設定調整。在運行時讓玩家修改設定並保存，仍需要透過程式碼將改動寫出。例如，我們可以在 ScriptableObject 的基礎上，另外撰寫保存功能，將變更後的值輸出到 XML。總而言之，可將 ScriptableObject 作為模組設定的預設配置容器，但實際**執行期的讀寫**需要其他機制配合。

自訂 C# 設定類別與序列化

為了完全掌控執行時的設定管理，我們可以建立一個**自訂的 C# 設定類別**來模擬 RimWorld 的 `ModSettings`。此類別（如 `MyModSettings`）將包含所有可調整的設定欄位，例如布林值開關、數字參數等。舉例來說：

```
public class MyModSettings /* 模擬 RimWorld 的 ModSettings */
{
  public bool enableFeature = false;    // 範例布林設定
  public float someThreshold = 1.5f;    // 範例浮點設定
  public List<string> nameList = new List<string>(); // 範例集合設定

  // 序列化方法，模擬 RimWorld 的 ExposeData
  public void ExposeData()
  {
    // 在此處調用序列化工具保存/讀取每個欄位
    // 例如使用 Scribe_Values 和 Scribe_Collections 模擬：
    Scribe_Values.Look(ref enableFeature, "enableFeature", defaultValue: false);
    Scribe_Values.Look(ref someThreshold, "someThreshold", defaultValue: 1.5f);
    Scribe_Collections.Look(ref nameList, "nameList", LookMode.Value);
  }
}
```

上述 `ExposeData()` 方法的概念來自 RimWorld 的 `ModSettings.ExposeData()` 虛擬方法。在 RimWorld 中，遊戲會在適當時機呼叫此方法以保存或讀取設定檔 [2]。`Scribe_Values.Look` 和 `Scribe_Collections.Look` 是 RimWorld 提供的靜態方法，用於序列化欄位到保存檔中 [7]。在我們的模擬中，可以透過自行實作這些方法或使用 .NET 提供的序列化工具達到類似效果。重點是：**我們需要在程式碼中明確列出每個需要保存的設定欄位**，確保它們會寫入 XML 並在載入時被正確讀取。

> **小提示：**RimWorld 的 Scribe 系統允許我們為 `Look` 方法提供一個預設值參數（如上例 `someThreshold` 預設為 1.5）。在載入時，如果發現舊的設定檔中沒有對應項目，就會使用此預設值 [8]。這對於日後擴充設定（新增新的設定欄位）相當有用，可以避免因缺少舊值而出錯。同時，你也可以透過一個布林參數控制是否**強制保存**某個值，即使它等於預設值 [8]。

模擬 Scribe 序列化機制

RimWorld 的 Scribe 系統本質上是在**保存階段**建立 XML 樹狀結構，並在**載入階段**讀取它。為了模擬這個行為，我們可以選擇以下兩種途徑： 1. **使用 .NET 序列化庫**：例如使用 `System.Xml.Serialization.XmlSerializer` 將 `MyModSettings` 物件直接序列化為 XML 字串並寫入檔案，或從檔案反序列化回物件。這種方式省去了手動逐欄位寫入的麻煩，但需要在類別上使用可序列化屬性，並注意某些型別可能無法直接序列化（例如 Unity 特有的型別，需要自訂轉換）。 2. **手動構建 XML（模仿 Scribe API）**：直接使用 `System.Xml` 或 `System.Linq.Xml` 去建立 XML 文件節點。這類似 RimWorld Scribe 的作法：逐個欄位寫入。例如： - 寫出：建立根節點 `<MyModSettings>`，然後為每個欄位建立子節

點 `<enableFeature>true</enableFeature>` 等，最後保存為如 `MyModSettings.xml` 的檔案。 - 讀取：載入 XML，尋找對應節點並解析其文字內容，賦值回 `MyModSettings` 實例的欄位。

如果希望更貼近 RimWorld 的風格，可以實作自己的靜態類別 `Scribe_Values` 和 `Scribe_Collections`，其內部封裝上述 XML 讀寫邏輯。例如 `Scribe_Values.Look(ref boolVal, "boolVal", defaultVal)` 可以被設計為：在**保存模式**時寫出節點，在**載入模式**時讀取節點並賦值給 `boolVal`。你可以在程式中用一個全域狀態（如 `enum ScribeMode { Writing, Reading }`）來表示目前是保存還是讀取操作，`Look` 方法根據狀態執行相應動作。由於篇幅限制，我們不會在此完整實作一個 Scribe 系統，但上述概念可以協助你了解 Scribe 的運作原理。

值得一提的是，`Scribe_Values` 能處理的型別相當廣泛，包括基本數值類型（int、float、double、bool 等）、列舉、字串、`System.Type`、Unity 向量/結構類型（如 `Vector2`, `Vector3`, `Color`）、以及範圍類型（如 `IntRange`）等 [4]。而對於集合物件，`Scribe_Collections` 則提供對 `List`, `Dictionary`, `HashSet` 等的支持，只要集合中的元素是可序列化的，並透過參數指定使用何種方式保存元素（例如純值、引用等） [5]。了解這些有助於你在擴充模組設定時選擇正確的保存方法。

XML 檔案結構與存取邏輯

在設定系統中，**XML 檔案**是實際保存設定值的地方。我們需要設計一個簡單明瞭的 XML 結構，方便未來的讀寫與維護。通常，我們會以**類別名稱**或其他標識作為 XML 的根節點，然後每個設定項目作為一個子元素。例如，對於 `MyModSettings`，我們可以設計設定檔 `MyModSettings.xml` 內容如下：

```
<MyModSettings>
  <enableFeature>True</enableFeature>
  <someThreshold>1.5</someThreshold>
  <nameList>
    <li>Alice</li>
    <li>Bob</li>
  </nameList>
</MyModSettings>
```

如上所示： - 根節點 `<MyModSettings>` 表示這是此模組的設定檔。你也可以使用模組ID或名稱作為根，以確保區分不同模組的設定。 - 每個子元素名稱對應設定類別中的欄位名稱，如 `<enableFeature>` 對應布林值 `enableFeature`。 - 對於集合類型（例如 `nameList` 是一個字串清單），可以在 XML 中巢狀多個子元素來表示清單內容。我們這裡用 `<nameList>` 包含多個 `<li>` 項作為例子。在實作時，也可採用其他約定格式，只要寫讀雙方一致即可。

存取邏輯： - 寫入（保存）：當需要保存設定時（例如玩家關閉設定面板或退出遊戲時），我們的程式應該：1. 建立一個新的 XML 文檔物件，創建根節點 `<MyModSettings>`。 2. 將當前設定物件的每個欄位寫入：對於基本值，創建對應節點並將值轉為文字；對於集合，創建父節點後迴圈建立子元素列表。 3. 將整個 XML 文檔保存到預定的路徑下的設定檔案。例如，可以保存到 Unity 的持久資料路徑（`Application.persistentDataPath`）或模組資料夾中的特定位置。 - 讀取（載入）：在模組初始化或遊戲啟動時，我們需要從 XML 檔載入先前保存的設定： 1. 檢查預定路徑是否存在設定檔案。如果不存在（表示玩家第一次安裝此模組或尚未保存過設定），則使用預設值初始化設定物件，並準備在首次保存時創建檔案。 2. 如果檔案存在，載入 XML 文檔，定位根節點，然後依次讀取子節點內容： - 可以使用節點名稱比對我們的設定欄位，將字串轉換回相應型別後賦值。例如讀到 `<enableFeature>True</enableFeature>` 就將 `enableFeature` 設為 `true`。 - 對於集合節點，如 `<nameList>`，遍歷其子元素收集值，重建清單物

件。 3. 完成讀取後，現在內存中的 `MyModSettings` 實例就承載了檔案中的所有數值，可供模組各部分使用。

路徑與檔案名：實際 RimWorld 中，模組設定檔會存放在玩家的設定資料夾下（非模組安裝路徑），以避免覆蓋模組原始檔案並允許不同玩家保存自己的配置。如 Windows 平台位於：`%USERPROFILE%/AppData/LocalLow/Ludeon Studios/RimWorld by Ludeon Studios/Config/Mods/<你的模組ID>.xml`（此路徑僅供參考）。在我們的 Unity 模擬中，可以選擇將設定檔保存在 `Application.persistentDataPath` 下的一個子目錄，例如 `ModsConfig/<模組名稱>.xml`，或直接放在專案資料夾（注意若專案發行後在 Program Files，下寫入可能需要權限）。為了便利教學，你可以暫時將其保存在 Unity 編輯器可存取的位置，如專案的 Assets/StreamingAssets 或PersistentDataPath。

總之，無論路徑如何，**保持檔名或路徑帶有模組識別**是重要的，這樣多個模組的設定才不會混淆。同時，在讀寫時要考慮檔案不存在或格式錯誤的情況，做好錯誤處理，例如初次運行時自動創建新設定檔，或在檔案損壞時重置為預設值並提示玩家。

使用者介面設計（簡易設定面板）

有了後端的設定資料結構和保存機制，接下來要提供前端介面讓玩家調整這些設定。在 RimWorld 中，模組設定介面通常從「選項 > 模組設定」中進入，會顯示各模組的設定選項列表。我們將在 Unity 中模擬一個簡易的設定面板。

介面要素：我們以先前定義的幾個設定為例，設計以下 UI 元件： - 一個標示模組名稱或設定標題的文字（UI 標頭）。 - 一個布林值選項的 **核取方塊**（Toggle）對應 `enableFeature`，附帶一段描述文字說明此選項效果。 - 一個浮點數參數的 **滑桿**（Slider）對應 `someThreshold`，旁邊有文字說明和當前值顯示。可設定滑桿範圍，例如最小值 0，最大值 3，以符合參數的合理範圍。 - 一個對字串列表進行編輯的區域（為簡化，可以只顯示當前清單內容，或者提供簡單的新增/移除項按鈕）。 - **按鈕**：包含「確定/保存」按鈕（或者在玩家修改後自動即時保存，也可以在關閉面板時保存），以及一個「取消/還原預設」按鈕（可選，用於將改動撤銷或恢復成預設值）。

你可以利用 Unity 的 **UI 元件**來構建上述介面。例如： 1. 在場景中創建一個 Canvas，並在其下建立一個 Panel 作為設定面板的背景。 2. 在 Panel 中加入 Text 元件作為標題，內容填入模組名稱或「模組設定」字樣。 3. 加入 Toggle 元件作為布林開關，附加一個 Text 子元件作為標籤（說明此選項）。可以用 `Toggle.isOn` 對應 `enableFeature` 值。 4. 加入 Slider 元件作為數值調整，設定其最小/最大值和初始值。旁邊加一個 Text 顯示目前數值（可每次滑動時更新該文字）。Slider 對應 `someThreshold` 值。 5. 若需要編輯列表，可以簡化為一個 Input Field 讓玩家輸入新值並按按鈕添加到列表，列表內容可用 Text 顯示，或使用 ScrollView 列出項目。由於此部分較繁瑣，可以作為進階挑戰項目，非本章重點。 6. 加入兩個 Button 元件作為「保存」和「關閉」按鈕。保存按鈕點擊時手動調用保存函數（如 `MyModSettings.Save()`），關閉按鈕則退出設定面板（並可順便觸發保存）。

資料繫結與事件處理：為了讓 UI 與我們的設定物件聯動，需要撰寫一個腳本（例如 `ModSettingsUIController`）來處理初始化和事件： - 在 `Start()`（或當開啟設定面板時）從當前的設定物件載入值到 UI：例如 `toggle.isOn = myModSettings.enableFeature; slider.value = myModSettings.someThreshold;` - 註冊事件監聽：如 `toggle.onValueChanged.AddListener(val => myModSettings.enableFeature = val);`，`slider.onValueChanged.AddListener(val => { myModSettings.someThreshold = val; UpdateThresholdLabel(val); });`，這樣玩家拖動滑桿或點擊開關時，設定物件會即時更新，同時更新滑桿旁的文字顯示。 - 保存按鈕的事件：`saveButton.onClick.AddListener(() => { SaveSettings(); });`，其內實作 `SaveSettings()` 函數調用我們先前實作的保存邏輯（例如調用 `MyModSettings` 實例的 `ExposeData()` 然後寫檔）。 - 關閉按鈕的事件：通常直接將面板隱藏或返回上

一介面即可。**在 RimWorld 中，關閉設定視窗會自動觸發保存** [9]；我們在模擬時可以選擇在關閉時也呼叫 `SaveSettings()` 以模擬相同行為，確保玩家改動確實被保存。

設計介面時請保持布局簡潔、元素對齊，確保說明文字清晰易懂。RimWorld 本身使用了 `Listing_Standard` 這類輔助類來簡化佈局，在 Unity 中我們可使用垂直佈局或Grid佈局來讓元件整齊排列。如果項目不多，手動擺放也無妨。重點是介面要直觀，讓玩家一眼就能理解每個設定的作用並方便地調整。

模組初始化與設定讀取整合流程

現在我們將前述各部分串連起來，說明模組從啟動到設定讀取、再到玩家調整設定並保存的完整流程。在 RimWorld 中，典型的模組設定初始化發生在模組類別的建構子中 (繼承自 `Mod` 類別)，而設定檔的寫入則發生在關閉設定視窗或遊戲關閉時 [9]。我們的模擬流程如下：

1. **模組啟動（載入）**：當遊戲或模組載入時，創建並初始化 `MyModSettings` 設定物件。這一步可以在我們的模組主類別（例如繼承 MonoBehaviour 的 `MyModController`）的 `Awake()` 或 `Start()` 方法中執行。
2. 程式嘗試從既有的設定檔讀取資料：調用 `MyModSettings.Load()`（你需要實作此靜態方法或另寫一個函數來載入），該方法會執行前面描述的 XML 讀取邏輯。成功時將讀出的值賦給新的 `MyModSettings` 實例；若沒有找到檔案，則使用預設值建立一個設定實例並準備好後續保存。
3. 將這個設定物件存放在一個全域可及的地方，例如設為 `MyModController` 的成員或乾脆做成 `MyModSettings.Instance` 單例（Singleton）。RimWorld 的實踐是透過 `GetSettings<T>()` 函數取得或初始化設定對象並保存引用 [10] [11]。無論哪種方式，我們要確保後續無論是遊戲內容還是 UI，都能拿到這個設定物件來讀寫。
4. **模組功能讀取設定**：模組的其他部分（例如遊戲邏輯）在需要時應該讀取 `MyModSettings` 中的值，以決定行為。例如，如果有一個功能會根據 `enableFeature` 開關來啟用/停用，那麼在執行該功能時檢查 `MyModSettings.Instance.enableFeature`，從而按玩家偏好調整模組行為。最好在每次使用時都讀取當前值，或者在設定變更時更新相應狀態，確保**遊戲行為與最新設定同步**。 [12]
5. **打開設定介面**：當玩家選擇調整模組設定（例如點擊某個按鈕開啟設定面板），我們的 `ModSettingsUIController` 會啟動。它讀取目前的設定物件值，更新 UI 元件的狀態（這部分我們在上一節已討論）。玩家即可查看當前設定並進行修改。
6. **玩家修改設定**：當玩家與 UI 互動（勾選/取消選項，拖動滑桿，輸入文字等），`ModSettingsUIController` 已經透過事件即時將改變反映到 `MyModSettings` 物件中。因此設定物件始終保持著玩家最新選擇的值。同時，如果某些設定的變更需要立即作用於遊戲（例如立即音效靜音與否的開關），你的程式可以在事件觸發時就立即執行對應的操作，使改動即時生效（這需要在設計時考慮哪些設定應當即時套用）。
7. **保存與退出**：當玩家關閉設定面板時（可能按下「確定」或「X」關閉按鈕），我們進行設定保存。具體來說，`ModSettingsUIController` 應呼叫 `MyModSettings.Save()` 方法將當前設定物件寫出XML檔案。保存成功後，可在UI上給予玩家反饋（例如顯示“設定已保存”訊息片刻），然後關閉面板返回遊戲。下次遊戲啟動或模組載入時，就會讀取到這次修改後的值，做到**持久化**設定。RimWorld 中則是自動在 UI 關閉時調用保存，因此玩家無需手動按保存 [13]。

8. **遊戲關閉**：在遊戲正常關閉時，也應確保模組設定已保存（假設玩家可能直接關閉遊戲而忘了進入設定面板再次確認）。為保險起見，可以在模組控制腳本的 `OnApplicationQuit()` 中再次調用一次保存函數（除非已在UI關閉時保存過且設定未再改變）。這樣可以避免設定遺漏。

完整的流程示意如下：

- 初始化階段：載入設定檔 → 初始化設定物件 → 模組功能讀取設定值應用。
- 運行階段：玩家開啟設定UI → 設定值載入UI顯示 → 玩家調整參數（即時寫入設定物件）。
- 結束階段：玩家關閉UI或遊戲 → 保存當前設定物件到XML檔。

通過上述流程，我們就將**模組設定檔的讀取/寫入**融入了模組生命週期，達到模組可以動態配置的目的。在實作和測試時，務必反覆驗證：**改變設定是否正確保存**，**重新進入遊戲是否能讀取先前的設定**，以及**模組功能是否根據設定改變**。

範例代碼與資料夾結構建議

本節我們提供一個簡化的範例代碼片段，串聯先前介紹的概念。同時討論如何在專案中組織這些檔案。

資料夾結構

在 Unity 專案中，我們建議模擬 RimWorld 模組的資料夾結構來管理檔案 [14]：

```
Assets/Mods/ExampleMod/            ← 模組根目錄
├─ About/                          ← 模組資訊，RimWorld模組必備
│   └─ About.xml                   ← 關於模組的描述、版本、作者等資訊
├─ Assemblies/                     ← 模組程式碼（編譯後的 DLL 檔）
│   └─ ExampleMod.dll              ← 編譯好的模組程式碼
├─ Defs/                           ← 模組自訂的遊戲定義（XML 格式）
│   └─ ThingDefs/ExampleItem.xml   ← 例如：物件定義檔案
└─ Textures/                       ← 模組使用的貼圖資源
    └─ exampleIcon.png             ← 例如：一些圖示
```

上述只是參考，實際內容可依你的模組需要增減。關鍵是 `About.xml` 一定要存在且正確配置，否則 RimWorld 不會識別此模組 [15]。在我們的教學模擬中，你可以仿照這結構組織資源和腳本。**模組設定檔（XML）**不一定要放在模組資料夾內，正如前述可放在玩家本地的Config資料夾中。然而，為了除錯方便，你可以在 `Assets/Mods/ExampleMod/` 下暫時建立一個 `ExampleModSettings.xml` 作為保存位置。實作完成後，視需求再調整路徑。

範例代碼

以下提供簡化的 C# 程式碼，示範模組設定系統的重點部分：

1. 模組設定類別（保存設定值並提供序列化方法）：

```
using System.Collections.Generic;
using System.Xml;
using UnityEngine;
```

```
public class ExampleModSettings  // 模擬 RimWorld 的 ModSettings
{
    public bool enableFeature = false;
    public float someThreshold = 1.5f;
    public List<string> nameList = new List<string>() { "Alice", "Bob" };

    // 將當前設定保存到XML的方法
    public void Save(string filePath)
    {
        XmlDocument doc = new XmlDocument();
        XmlElement root = doc.CreateElement("ExampleModSettings");
        doc.AppendChild(root);
        // 布林值設定
        XmlElement boolNode = doc.CreateElement("enableFeature");
        boolNode.InnerText = enableFeature.ToString();
        root.AppendChild(boolNode);
        // 浮點值設定
        XmlElement floatNode = doc.CreateElement("someThreshold");
        floatNode.InnerText = someThreshold.ToString();
        root.AppendChild(floatNode);
        // 清單設定
        XmlElement listNode = doc.CreateElement("nameList");
        foreach (string name in nameList)
        {
            XmlElement li = doc.CreateElement("li");
            li.InnerText = name;
            listNode.AppendChild(li);
        }
        root.AppendChild(listNode);
        // 保存檔案
        doc.Save(filePath);
        Debug.Log($"設定檔已保存至 {filePath}");
    }

    // 從XML檔案載入設定值的方法
    public void Load(string filePath)
    {
        if (!System.IO.File.Exists(filePath))
        {
            Debug.Log("未找到設定檔，將使用預設值。");
            return;
        }
        XmlDocument doc = new XmlDocument();
        doc.Load(filePath);
        XmlNode root = doc.SelectSingleNode("ExampleModSettings");
        // 讀取布林值
        XmlNode boolNode = root.SelectSingleNode("enableFeature");
        if (boolNode != null)
            enableFeature = boolNode.InnerText.ToLower() == "true";
        // 讀取浮點值
        XmlNode floatNode = root.SelectSingleNode("someThreshold");
```

```
        if (floatNode != null && float.TryParse(floatNode.InnerText, out float fVal))
            someThreshold = fVal;
        // 讀取清單值
        nameList.Clear();
        XmlNode listNode = root.SelectSingleNode("nameList");
        if (listNode != null)
        {
            foreach (XmlNode li in listNode.ChildNodes)
            {
                nameList.Add(li.InnerText);
            }
        }
        Debug.Log("設定檔載入完成。");
    }
}
```

上述程式碼展示了如何將三種設定（bool、float、List<string>）寫入和讀出 XML。需注意我們在寫入時使用 `ToString()` 將值轉為字串，讀取時需要轉型（例如字串 "true"/"false" 轉回 bool）。這裡簡單地透過字串比較實現布林解析、透過 `float.TryParse` 解析浮點數。實務上你可能想對錯誤情況（例如格式不對）做更多檢查。

2. 模組主控制類別（負責初始化設定和整合UI）：

```
using UnityEngine;
using UnityEngine.UI;

public class ExampleModController : MonoBehaviour  // 模擬 RimWorld 的 Mod 類別
{
    public ExampleModSettings settings;
    public string settingsFileName = "ExampleModSettings.xml";
    private string settingsFilePath;

    // 引用UI元素（在Inspector中連結）
    public Toggle featureToggle;
    public Slider thresholdSlider;
    public Text thresholdLabel;
    public InputField nameInput;
    public Text nameListText;

    void Awake()
    {
        // 決定設定檔路徑，例如放在應用程式的持久化資料夾
        settingsFilePath = System.IO.Path.Combine(Application.persistentDataPath, settingsFileName);
        settings = new ExampleModSettings();
        // 載入設定檔（如果有的話）
        settings.Load(settingsFilePath);
    }

    void Start()
    {
```

```
        // 初始化UI顯示
        featureToggle.isOn = settings.enableFeature;
        thresholdSlider.value = settings.someThreshold;
        thresholdLabel.text = settings.someThreshold.ToString("F2");
        nameListText.text = string.Join(", ", settings.nameList);
        // 綁定UI事件
        featureToggle.onValueChanged.AddListener(OnFeatureToggleChanged);
        thresholdSlider.onValueChanged.AddListener(OnThresholdSliderChanged);
    }

    // Toggle改變時
    void OnFeatureToggleChanged(bool value)
    {
        settings.enableFeature = value;
        // 可以在這裡即時改變遊戲中對應功能的狀態
    }

    // Slider改變時
    void OnThresholdSliderChanged(float value)
    {
        settings.someThreshold = value;
        thresholdLabel.text = value.ToString("F2");
        // 若此設定需要即時應用，亦可在此處理
    }

    // 點擊“新增名稱”按鈕時的處理
    public void OnAddNameButtonClicked()
    {
        if (!string.IsNullOrWhiteSpace(nameInput.text))
        {
            settings.nameList.Add(nameInput.text);
            nameInput.text = "";
            // 更新顯示清單
            nameListText.text = string.Join(", ", settings.nameList);
        }
    }

    // 點擊“保存設定”按鈕時的處理
    public void OnSaveButtonClicked()
    {
        settings.Save(settingsFilePath);
    }
}
```

在此 `ExampleModController` 中，我們透過 `Awake()` 初始化設定檔路徑並載入設定，`Start()` 中設置UI初始值並註冊事件。`OnFeatureToggleChanged`、`OnThresholdSliderChanged` 直接將 UI 改變寫回設定物件，並可選擇在當下對遊戲行為做出調整（依設定需求決定）。我們也示範了一個輸入框和按鈕，用於向列表增加新名字，這在點擊時更新設定物件和UI文字。最後，`OnSaveButtonClicked` 調用我們在 `ExampleModSettings` 定義的保存方法，將目前設定寫入檔案。

注意：在 Unity 編輯模式下測試時，`Application.persistentDataPath` 可能位於 `C:\Users\<使用者>\AppData\LocalLow\CompanyName\ProductName` 下。你可以透過除錯訊息或直接輸出 `settingsFilePath` 確認檔案位置，方便打開檢視 XML 內容是否正確[16]。另外，如果你將專案Build成獨立應用程式，記得測試設定檔的讀寫是否仍正常運作。

3. About.xml（模組資訊檔）：

```
<?xml version="1.0" encoding="UTF-8" ?>
<ModMetaData>
 <name>範例模組</name>
 <author>你的名字</author>
 <id>ExampleMod</id>
 <version>1.0</version>
 <description>這是一個示例模組，包含可調整的設定檔功能。</description>
</ModMetaData>
```

雖然 About.xml 在本章焦點之外，但為了模擬完整性，我們列出一個簡單範例。其中 `<id>` 對應模組的唯一識別碼，建議與你的設定檔名稱/根節點保持一致以方便對應（如這裡使用 "ExampleMod"）。當 RimWorld 載入模組時，它會根據此 ID 來讀寫對應的設定檔案。

完成以上步驟後，我們的模組就具備了完整的設定檔功能。請將程式碼編譯後放入 `Assemblies` 資料夾（或直接作為 Unity 腳本掛在場景物件上執行，依照你目前模擬的架構）。啟動遊戲場景，測試開啟設定面板，調整各項參數並保存，再次進入時是否保持之前的設定。

練習與挑戰

要鞏固本章所學，請嘗試完成以下練習並確保通過列出的驗收項目：

- **挑戰任務：**
- **新增更多設定**：為模組新增一項設定，例如整數型別的參數（使用者可調整一個整數值）或顏色設定（可使用三個滑桿代表RGB）。修改設定類別、新增UI元件，並確保它們能正確保存和載入。提示：整數可以使用 `Scribe_Values.Look(ref myInt, "myInt", 預設值)` 保存；顏色可以拆成三個 float 屬性或者轉為字串保存。
- **預設值與重置**：實作一個“恢復預設”按鈕，點擊後將所有設定恢復成預設值（可以在 `ExampleModSettings` 中預先保存一份預設值，或重新 New 一個設定物件覆蓋當前，再更新UI）。確認重置後再保存並重啟遊戲，設定確實回到預設狀態。
- **即時生效**：如果有些設定改變需要即時影響遊戲（例如開關立即改變某些物件的顯示），嘗試在事件監聽中加入相應的邏輯，驗證改變設定後遊戲狀態立即發生預期變化，而不需重啟遊戲。
- **錯誤處理**：模擬一個損壞的設定檔（例如手動編輯XML使內容不合法），確保你的載入程式能妥善處理，不會因 Exception 導致遊戲卡住。可以在 `Load()` 時用 `try-catch` 包裹，或先檢查節點是否存在再訪問。還可以設計當發現檔案損壞時，自動備份舊檔並生成新預設檔的策略。
- **驗收項目：**

- 啟動遊戲後，修改幾項模組設定，點擊保存並關閉遊戲，再次啟動時這些設定能正確載入並反映在 UI 上（例如先前打勾的仍為打勾，調整的數值仍保持調整後的值）[17]。
- 新增的設定項可以成功保存和載入，UI 展示與功能邏輯皆正常。

- “恢復預設”按鈕能如期運作，並且會將修改反映到檔案（重置後保存，再重新進入遊戲時確實恢復預設）。
- 在設定檔載入過程中，對於缺失的節點會使用程式中定義的預設值而不會拋錯（可透過在升級版本時新增設定項來測試，未來版本程式有新設定但玩家舊設定檔沒有相應節點時應有預設值填入 [8]）。
- 整個流程無明顯效能問題；即使設定項很多，保存/讀取也應迅速完成。此外，頻繁打開設定面板不會累積狀態錯誤（例如不會一次打開就應用以前重複的值增長等問題）。

完成以上練習後，你應該對 RimWorld 模組設定檔的運作原理以及在 Unity 中的實作方式有深刻的理解。這不僅讓你的模組更具彈性，也為玩家提供了更佳的體驗。繼續努力，在後續章節中我們將探討更進階的主題，讓你的 RimWorld 模組開發技能更上一層樓！ [3] [2]

---

[1] [2] [3] [6] [7] [9] [10] [11] [12] [13] [16] [17] Modding Tutorials/ModSettings - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/ModSettings

[4] [5] [8] RimWorld Saving For Dummies | RimWorld Modding Resources
https://spdskatr.github.io/RWModdingResources/saving-guide.html

[14] [15] Mod Folder Structure - RimWorld Wiki
https://rimworldwiki.com/wiki/Modding_Tutorials/Mod_Folder_Structure