Guide contents

Choose the layer you are changing

Make a first Global script

Save state and assets safely

Sources / References

Documentation: In progress **Quick start: Complete**

Offline copy from ModDocs. Source links remain clickable.

Tabletop Simulator modding

Tabletop Simulator creations combine a saved table, objects and optional Lua scripts. A game's **Global Script** can manage the table; an **Object Script** belongs to one physical object. Scripts are stored in the save's JSON, so a Workshop upload carries the scripts with the save. The official scripting introduction and editor overview explain that model.

> **Quick start:** save a simple table, add a Global Lua script that prints a message when the save loads, use **Save and Play**, then save and reopen the table. Confirm the message appears after each load before adding objects or custom assets.

Choose the layer you are changing

Goal	Main place to work	First check
Set up cards, dice, board or pieces	Table objects and saved game	Objects persist after reopening the save
Run a table-wide rule	Global Lua script	The documented event fires after load or interaction
Give one object behavior	Object Lua script	Only that object responds

Goal	Main place to work	First check
Add custom buttons or panels	UI XML plus Lua callbacks	UI appears and invokes the right handler
Use custom art or models	Custom assets and URLs	Another player can load the asset

Begin with a local saved table. Avoid publishing a Workshop item when you have only tested the scripting editor's current session. The official overview notes that **Save and Play** commits script edits and reloads, but discards non-script object changes made since the last ordinary save. Save the physical table first, then edit scripts.

Make a first Global script

Open the in-game **Scripting** menu and select the Global tab. Enter:

```lua
function onLoad()
    printToAll("This table loaded.", {r=0.2, g=0.8, b=1.0})
end
```

The event reference documents `onLoad` and the base-function reference documents `printToAll`. Use **Save and Play**, then look for the message in the chat. Save the table normally, exit to the menu, reopen the saved game and verify the message repeats. This confirms the script is in the save, not merely the editor buffer. If nothing happens, inspect Lua errors and ensure the code is in **Global**, not an object that has not spawned.

Next move a feature into an Object Script only if the behavior belongs to a specific piece. Global and Object scripts can call many of the same APIs, but an object script has `self` for its owner. Use a unique object GUID when referring to a piece from Global code; names may not be unique. The script introduction explains both contexts.

Save state and assets safely

If the script tracks a score or turn number that must survive save/reload, use `onSave` and `onLoad` with the JSON API. Store serializable values or object GUIDs, not live object references. Test a reload and an undo: the event reference notes load callbacks may also run after rewinding. A script that only works in one uninterrupted session is not ready to publish.

Custom images and models need valid locations available to other players. Check the asset-management and Custom Game Objects documentation for the specific asset type. Test with a second player or clean account before claiming a table is portable. An asset cached on your PC may hide a broken URL.

The saved table with its Global script is a complete small scripted creation: it loads, displays a player-visible message and survives save/reload. The walkthrough is source-reviewed against the linked API; it has not been run in a local Tabletop Simulator installation here. A multiplayer turn system, object UI and custom-asset publication each need additional feature-specific testing.

Sources / References

External project and documentation links cited in this guide. Check the upstream release and version before following version-sensitive steps.

- official scripting introduction
- editor overview
- event reference
- base-function reference
- JSON API
- Custom Game Objects