Peter Gustafson
writing sample
Epic Games Sep 15, 2021

Handheld AR Template Technical Reference | Unreal Engine 4.27 Documentation
Choose your operating system:

The Handheld AR Template is a starting point for creating augmented reality projects for iOS and Android devices. While the [Handheld AR Template Quickstart](https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARQuickStart) provides the steps for setting up the template and deploying it on a mobile device, this guide will provide more technical guidance on how the template works, where its key features are implemented, and how to modify its functionality.

Basics

The Handheld AR Template is displayed with two parts:

* The feed from the user's camera.

* The _virtual scene_, which exists in Unreal Engine's 3D world.

The user's camera feed is captured automatically when they start an **AR Session**, while the virtual scene is rendered on top of that. As a representation of the user, the [Pawn](https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Framework/Pawn) contains logic for interacting with objects within the virtual scene, including planes defined by the initial scan and placeable objects. A HUD is overlaid on this composited display, providing configuration options and other tools.

For more information about the user journey for the Handheld AR Template, refer to the [Handheld AR Template Quickstart](https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARQuickStart).

Application Flow

The Handheld AR Template opens directly to **HandheldARBlankMap** both in the Editor and on application startup. This map does not feature any Actors apart from the helper Actors used to point to documentation on mobile device setup. Its main function is to override the Game Mode to **BP_ARGameMode** so that the application will use the [Game Framework](https://docs.unrealengine.com/4.27/en-US/

InteractiveExperiences/Framework) classes that are responsible for running the application and setting up the virtual scene.

 ![The template project displaying HandheldARBlankMap](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InitialMap.jpg)

BP_ARGameMode starts the player with **BP_ARPawn**, which is responsible for initializing the UI by adding **BP_MainMenu** to the user's viewport. Once **BP_MainMenu** is set up, it provides a prompt to the user to start scanning the environment. This begins an **AR session** and displays the user's camera feed, then tells BP_ARPawn to define a series of planes based on the available flat surfaces in the environment.

BP_ARPawn uses a simple state machine to track what step of the user journey the application is currently on. This state machine tracks these factors:

*   Has the user scanned their environment and set up planes in the virtual scene?

*   Has the user selected a plane to interact with?

*   Has the user placed an object on the selected plane?

Based on the user's progress through the user journey, the Pawn changes the way the application responds to their input and triggers prompts in the HUD to walk them through setup steps.

Actors

AR Pawn

The Handheld AR Template uses a custom Pawn called **BP_ARPawn**. This Pawn is responsible for initialization, building and updating the virtual scene, and handling the user's input.

Initialization

The AR Pawn adds **BP_MainMenu** to the user's viewport on BeginPlay. **BP_MainMenu** is then responsible for setting up the AR scene and handling prompts based on the current state of the application.

State Control

The AR Pawn implements a simple state machine to support the different steps in the user journey outlined in the above sections. The state is

controlled by a series of booleans under the **State** category.

 ![The booleans that control the state machine for this template](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/StateMachineBooleans.jpg)

These include:

*   **ScanningIsDone**: The initial scanning process is complete and planes are placed in the virtual scene.

*   **PlaneIsSelected**: The user has selected a plane.

*   **ObjectIsPlaced**: The user has placed the virtual object on a plane and can now interact with it.

As the user follows the prompts from the HUD, these change from false to true, triggering the application to move to the next step in the user journey. The Pawn refers to these when processing touch input and when handling the Tick event to update the scene.

Camera Depth Scene Texture

The AR Pawn is also responsible for fetching the camera depth scene texture, which is used for placing objects in the virtual scene. This information is fetched as soon as the user gives the application permission to use the camera, but before the application completes its AR scan. The depth texture is stored in **CameraDepthTexture**, which is then used in the **Create Plane Candidate** function when the Pawn updates the visuals for planes. If **CameraDepthTexture** is not available, it uses **CameraDepthFallback** instead.

Light Intensity Estimation

During the Tick event, the AR Pawn calls the **Feed Light Estimate** function. This obtains current light estimate data from the AR session. If the data is valid, it updates the directional light and skylight in the map with color and intensity settings based on the camera feed. This creates lighting in the virtual scene that roughly matches lighting in the user's real-world environment.

AR Plane Creation

Once the AR session has finished scanning the environment, the AR Pawn begins updating the **PlaneCandidate** Actor in the virtual scene. This process checks to see if tracked objects in the environment fit valid **AR Plane Geometry**. Typically, a device's sensors can detect anywhere between five and twenty objects.

After that, the Pawn calls **Create Plane Candidate** to spawn a **BP_Plane** Actor and place it in the appropriate location in the virtual scene. Plane candidates are updated once per second. To reduce visual complexity on screen, this process is set up so that the AR Pawn only tracks one plane candidate at any given time, favoring the closest object tracked by the AR session. As the user moves around the environment, they will see different plane candidates appear and disappear.

When the user selects a plane candidate, the BP_Plane is locked in place as the **SelectedPlane** instead.

Gesture Processing and Manipulating the Virtual Scene

The AR Pawn is the only blueprint that consumes input, so it is responsible for all gesture recognition.

The **InputTouch** event handles single-touch inputs for either selecting a plane or placing a virtual object, depending on whether or not the user has a plane selected. Because two-finger gestures and swipes are not available until the user selects a virtual object, this touch input is very simple and does not require any special input actions.

 ![The InputTouch Event](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InputTouchEvent.jpg)

After the user has both placed a plane and selected a virtual object, input is handled by the **OneFingerAction** and **TwoFingerAction** input actions, as well as an input axis called **TwoFingerMapping**. These are defined in the input bindings under **Project Settings > Input**.

 ![Input Actions displayed in the Project Settings menu](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InputActions.jpg)

The **OneFingerAction** and **TwoFingerAction** input actions are represented with events in the AR Pawn's event graph. These events do not trigger any functions when interacting with planes and objects. Instead, they record data that the AR Pawn can use to pick what type of gesture is appropriate during general input handling.

 ![Input Actions displayed in the AR Pawn's blueprint graph](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InputActionsARPawn.jpg)

The Tick event calls **Reset Recognized Gesture** to clean up

information about previous gestures, then calls the **One Finger Gesture Recognition** and **Two Finger Gesture Recognition** functions to process the data gathered from the input action events. This includes the deltas for position, direction, and travelled distance of touch inputs.

These functions set an enum for the appropriate gesture, called **Current Transformation**. A switch on **Current Transformation** then chooses whether to use the Translate, Rotate, or Scale functions on the selected placeable object. The object itself handles these transformation functions.

 ![Gesture controls displayed in the AR Pawn's blueprint graph](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InputGestureControls.jpg)

Placeable Objects

 ![An example of a placeable object](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/PlaceableObjectExample.jpg)

The **BP_Placeable** blueprint is the base class for all placeable objects in the Handheld AR Template, including planes. It encapsulates everything related to a placeable virtual object: visuals, transformation logic, and HUD interaction logic with a widget component. On BeginPlay, the placeable object will choose an asset based on what type of product you are creating with the **Assign Product Asset** function. The model selected depends on the category you choose when setting up the template.

Controlling Visuals

Placeable objects use the **Set Visuals** function to control the display of both the model's materials and the UI widgets displayed alongside them. This function is called from **Set Placeable Position**, **Set Placeable Rotation**, and **Set Placeable Scale**.

If an interaction is active, the Tick event will call **Update Interaction**. This function will update the placeable object's UI widget with new information based on how the user moves, rotates, or scales it. There are several utility functions for getting text appropriate to each transformation.

Placing Objects on AR Planes

The AR Pawn spawns a **BP_Placeable** actor from the Input Touch event and assigns it to the **Placeable Object** variable.

AR Planes

The Handheld AR Template represents virtual planes with BP_Plane actors. These are created by the AR Pawn using the **Update Plane Candidate** and **Create Plane Candidate** functions. Create Plane Candidate in turn calls **Initialize Plane**, which handles setup of **BP_Plane** materials and color.

The static mesh attached to **BP_Plane** is a simple plane consisting of two triangles, with complex collision turned on and the **Collision Presets** set to **BlockAllDynamic**. This makes it possible to detect the surface of the plane with any type of trace.

The default state of a BP_Plane actor is its unselected state. In this state, it uses **DM_Scan** as its material instance, displaying ripples to indicate that the device is currently scanning the environment.

 ![BP_Plane using DM_Scan as its material](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/ScanningMaterialExample.jpg)

The **Get Platform Scan Material** function sets **DM_Scan** to either **ScanMaterial01** or **ScanMaterial02**, depending on whether or not the user is on an Android or iOS device.

The Initialize Plane function will attempt to use the camera scene depth taken by the AR Pawn to set **DM_Scan** to visually cut it off so that it fits better into the environment. It will also set the color using **Get Plane Color** in the AR Pawn. This function fetches the AR geometry index and uses that to select a color. If the AR toolkit finds new geometry in the environment, the colors may change.

When the user selects a plane to place an object on, the AR Pawn will call the **BPPlane Is Selected** event dispatcher. The main menu uses this event to switch to the place object state. When the user places an object on a plane, the plane is hidden, creating a more immersive display. Ideally, the plane should match the scanned object in the user's environment closely enough that interactions feel intuitive even if the plane is not being displayed.

When the user uses the translate function on a placeable object, the AR Pawn will call the **Switch to Material Translate** function to change the plane's material instance to **DM_Plain**, which shows a less intrusive outline compared with **DM_Scan**. This makes it easier for the user to see the bounds where they can move the object.

This section provides a reference for the elements that make up the

HUD in the Handheld AR Template. The information below explains how the UI classes are organized and how they provide key features, such as styling and mode changes.

UI Styles and BP_StylizedUI

All menus in the Handheld AR Template use **BP_StylisedUI** as their parent class. This class encapsulates all functions and relevant data needed to switch the UI style between the Light, Dark, and Game themes. These themes are contained in the **DT_Styles** data table, which contains information for common UI colors, fonts, and icons.

 ![The DT_Styles data table](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/StyleData.jpg)

The **Get Style Data** function fetches information about the currently selected style and outputs this information for use in other functions.

 ![An example of the Get Style Data function](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/GetStyleData.jpg)

BP_MainMenu uses the **Call Switch Style** function to trigger style switching in each of its sub-menus.

 ![The Call Switch Style function called from the main menu](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/CallSwitchStyle.jpg)

These bind events for changing their UI to the **Switch Style** event dispatcher, which is defined in **BP_StylizedUI**.

 ![Switch Style event dispatcher](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/SwitchStyleEventDispatcher.jpg)

Other widgets, such as **UI_CapsuleButton**, define their own **Change Style** function, which is called manually by their parent menus.

 ![Example of a manual change style function](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/ChangeStyleExample.jpg)

This design both provides a data-driven method for defining UI styles, and also ensures that there is minimal duplication necessary to support large-scale changes in UI style and layout.

Main Menu

BP_MainMenu acts as a container and manager for all the other menus. After the AR Pawn adds it to the viewport, it initializes all the other menus, binds their event dispatchers, and binds the event dispatchers that correspond to specific steps in the user journey as well. It is also responsible for starting the AR session. Because it does not have any visual elements of its own to style, it extends the base User Widget class instead of **BP_StylizedUI**.

Initialization and Event Binding

During the **On Initialized** event, the main menu initializes the UI, sets the default UI style to **Light**, then binds the event dispatchers for all of the UI's main functions. These include steps through the template's user journey, displaying the initial tutorial pop-up, as well as the flow for taking screenshots or transitioning between other sub-menus and confirming settings changes.

 ![The On Initialized event in the Main Menu](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/OnInitializedMainMenu.jpg)

This is a highly extensive event graph, but think of it as acting like the glue between the visual elements in the sub-menus and the application's main functionality.

Starting the AR Session

The AR session starts when the user presses the button in **BP_StartMenu** to begin scanning.

 ![Starting the AR session](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/StartARSessionExample.jpg)

Handling State Changes

The Main Menu's tick function then listens for the AR session to reach the **Running** state. When this happens, it tells **BP_StartMenu** to enter the scanning state, which triggers several cosmetic changes to the UI to help highlight the state change.

 ![State Changes tracked with Tick](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/StateChangeOnTick.jpg)

Similarly, the Tick function also listens for when the user starts to place an object. These changes correspond to the state machine found in the AR Pawn.

Several sub-menus are embedded in the Main Menu. These menus primarily handle their cosmetic elements and layout, deferring to Event Dispatchers to handle their functionality except for internal styling purposes. All sub-menus are based on **BP_StylizedUI** to provide common functions for supporting style changes.

 ![The Start Menu as it appears in UMG](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/StartMenuUMG.jpg)

BP_StartMenu is the menu that initially displays when the user starts the application, and handles overlays and prompts walking the user through the user journey, providing feedback as they move from one step to the next. The Start Menu is overlaid over the entire screen, and contains a separate overlay for each discrete user journey state. For example, the Start Menu shows **UI_Scanning** when the user is in the middle of scanning their environment for planes, while it shows **UI_PlaceObject** once they have selected a plane and are ready to place an object on it.

 ![The Bottom Menu as it appears in UMG](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/BottomMenuUMG.jpg)

When the user has gone through the user journey and placed an object, **BP_BottomMenu** displays at the bottom of the screen, providing a way to access the Options Menu, the Info Menu, and the Snapshot and Reset functions.

The Bottom Menu has a separate overlay for the horizontal layout used by the Light and Dark themes, as well as the circular layout used in the Game theme. Both of these layouts' versions of the buttons bind the same custom events to avoid duplication.

 ![Binding an event for two different versions of the same button](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/DoubleBinding.jpg)

 ![The Options Menu as it appears in UMG](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/OptionsMenuUMG.jpg)

BP_OptionMenu is the menu that displays when the user taps the Options button in the Bottom Menu. This provides a handful of configuration options, including the ability to switch UI styles. The UI buttons in this menu are bound to functions in the Main Menu to provide better visibility to the AR Pawn as well as other functions that may need to trigger in other parts of the UI.

 ![An example of calling a function from the Player Pawn with an event dispatcher](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/ToggleSnapExample.jpg)

 ![The Info Menu as it appears in UMG](https://docs.unrealengine.com/4.27/Images/SharingAndReleasing/XRDevelopment/AR/ARHowTos/HandheldAR/ARTemplateReference/InfoMenuExample.jpg)

BP_InfoMenu appears when the user taps the Info button in the Bottom Menu. This displays the gestures for working with placeable objects.

Buttons

UI_CapsuleButton and **UI_ToggleButton** are used by all menus. They are custom made to support a wide range of common functions, including style changes and specific button behaviors such as toggling.