KHRONOS GROUP

Interaction in OpenXR

Ryan A. Pavlik, Ph.D.
Principal Software Engineer, Collabora, Ltd.
OpenXR Working Group Spec. Editor
XR Kaigi, November 2020

Hello everyone, and welcome to our class on interaction in OpenXR.

Agenda

Slides, with speaker notes and links, will be available at khronos.org

- **About Me**
- **Handle and atom types**
- **Modeling interaction:**
  - Actions, Action Sets, and Interaction Profiles
  - Sample Walkthrough

KHRONOS GROUP

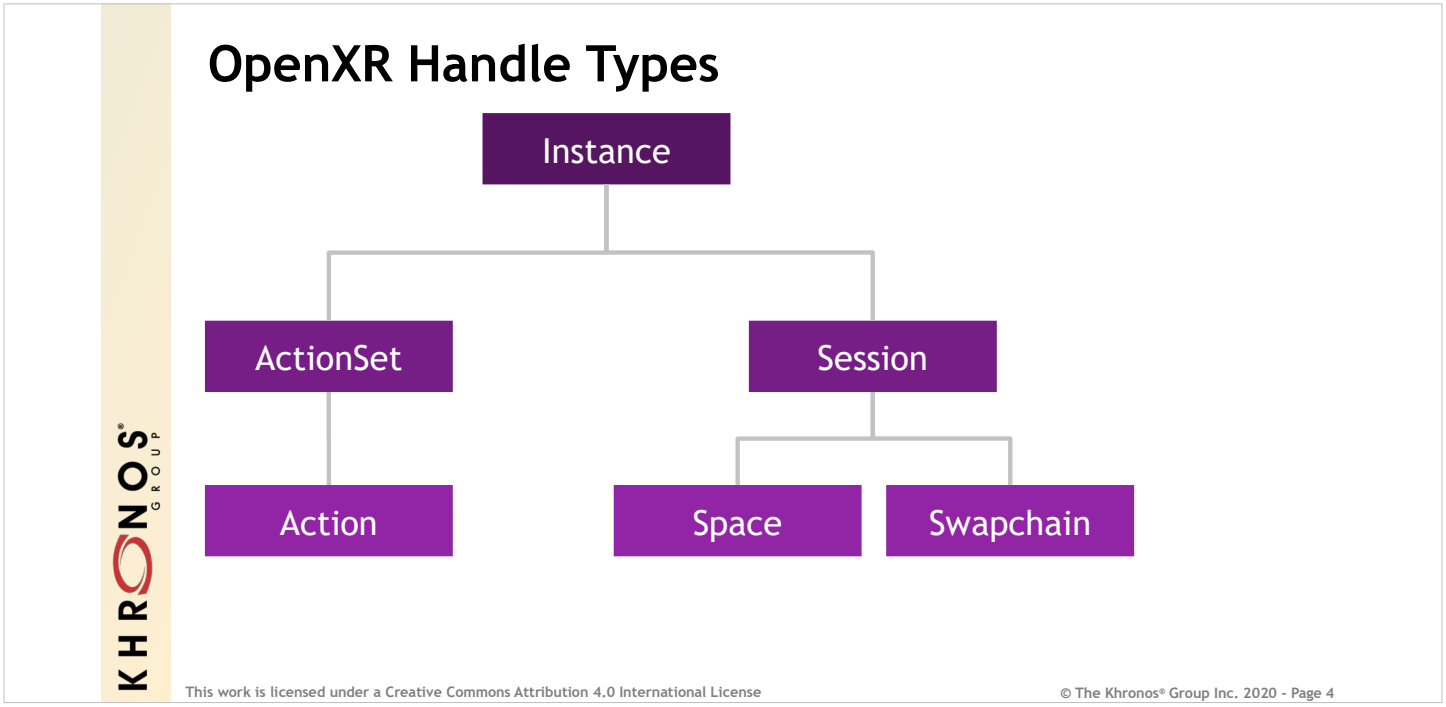
The plan for the talk today is to first spend a few moments introducing myself. We will briefly discuss the basic object handle types in OpenXR. At that point, we'll have the background to explore how OpenXR models interaction. Then, we'll take a deep dive into OpenXR application structure and API usage.

About Me: Ryan Pavlik

- **Open-source VR software developer since 2009**
- **OpenXR working group**
  - participant since the first official meeting in January 2017
  - elected specification editor in April 2019
- **Principal Software Engineer at Collabora**
  - Focusing on XR client projects
  - Leading our OpenXR contributions
  - Developer on Monado

COLLABORA

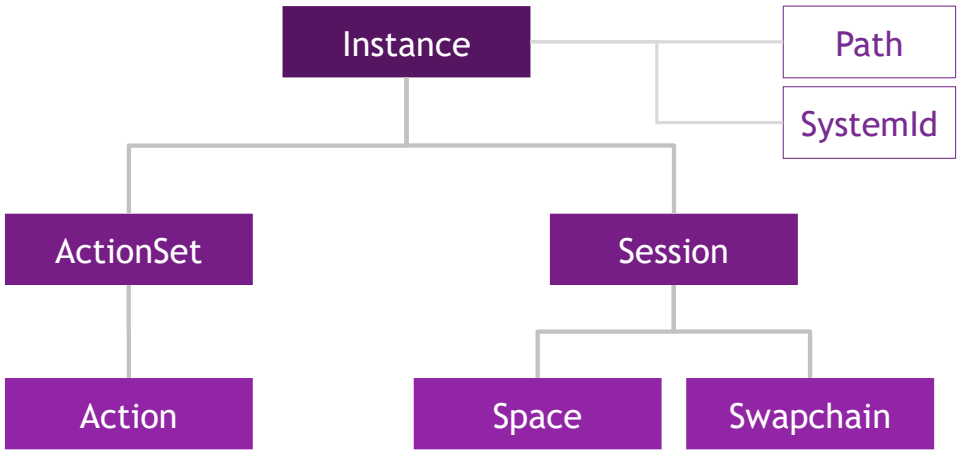
My name is Ryan Pavlik. I've been working in the VR realm since around 2009. I've been involved with the OpenXR working group since the first official meeting in 2017. I was elected specification editor for the OpenXR working group in 2019. I am a principal software engineer at Collabora, where I work on customer projects in the XR team, and contribute to our OpenXR runtime Monado.



There are a number of important objects in OpenXR, which are represented by handle types.

These are the main handle types. All handle types except XrInstance have a parent handle type. In OpenXR, destroying a parent handle also destroys all handle that come from it. For example, if you destroy a Session, that also destroys the associated spaces and swapchains. If you destroy the Instance, that destroys all of the handles shown here, since they all come from the Instance.

Atoms



In addition to handles, there are two additional types known as atoms. They're not objects, and they don't have a explicit lifetime. They're just coded numbers that represent some fixed thing in the runtime. The one you'll work with most often is a path. An XrPath is a number that corresponds to a string representing a semantic path.

Although these have no explicit lifetime, they only have meaning in the Instance they're retrieved from. Do not save or reuse these between runs.

Modeling Interaction with Actions

- **Focus first on *what* users do, not *the hardware* they do it with**
- **Important for hardware-independence.**
- **Action: A semantic (meaningful) bit of interaction**
  - Types: Boolean (button), Float (analog), Vec2, Pose (tracked object), Haptic
  - e.g. “grab_object”, “teleport”, “hand_pose”
- **ActionSet: a group of related actions for a context, environment, etc.**
  - e.g. “menu”, “gameplay”, “driving”
  - One or more active at any time

An important part of immersive technology is interaction. Before we get into the details of how an OpenXR application is structured, let’s take a few minutes to look at how OpenXR models interaction, since it might be a new approach for you. OpenXR focuses on the actions that a user takes in your application instead of on the buttons and controllers that are used to perform those actions. This is an important part of the hardware independence provided by OpenXR.

In OpenXR, an Action is a semantic or meaningful bit of interaction: it's something that you do. "Grab object" and "Teleport" are examples of names for actions.

Actions that should be available in a given context or environment are grouped into Action Sets. For example, you might have an Action Set called "menu", one called "gameplay", and one called "driving". You can have one or more Actions sets active at a single time, so you can have "gameplay" and "driving" both active. You do not need to duplicate Actions between sets or create additional Action Sets combining others.

There are several different types of Actions available.

- Boolean is essentially a button action: it has either on or off.
- Float actions are things like an analog trigger.
- Vector2 are two-dimensional floats: things like a thumb stick or trackpad.
- Pose is a special kind of action that's a tracked object. Frequently, these represent hands.
- Finally, haptic actions are an output action allowing you to provide rumble or tactile feedback to the user.

Suggested Bindings and Interaction Profiles

- **How you customize for hardware you've tested, without excluding the rest**
- **For each controller type you've tested ("interaction profile"), suggest bindings for actions**
  - With as many or few action-binding pairs as you like - OK if not all actions have a suggested binding
  - Can suggest multiple bindings per action in a call: e.g. both left and right hands can "grab_object"
  - Binding is an XrPath atom representing a path string like /user/hand/right/input/select/click
- **If your application is used on different hardware, the runtime may re-map your actions to the available hardware**
- **Set up actions, action sets, and suggested bindings once, at startup**

interaction profiles added by vendor extensions XR_MSFT_hand_interaction, XR_HUAWEI_controller_interaction, and multi-vendor extensions XR_EXT_eye_gaze_interaction, XR_EXT_hp_mixed_reality_controller, XR_EXT_samsung_odyssey_controller

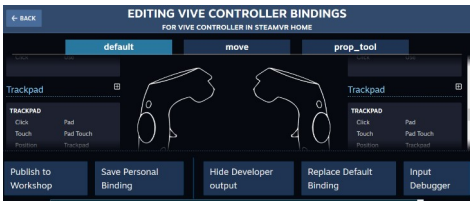
Once you've established which logical Actions a user of your application will make, you can then customize how they actually perform those Actions for the hardware that you're testing. This is done using suggested bindings for interaction profiles. For each controller type that you've tested, look up its interaction profile. These are listed in the specification, and cover a number of well-known devices. Additional interaction profiles will be added through extensions. For each device you test with, you submit Actions and their suggested bindings for the corresponding interaction profile. The suggested bindings are the logical part of the controller that you'd like to use to drive that Action: the specific button, for example.

You can suggest multiple bindings per Action in a call. For instance both your left and right hands, and thus your left and right controller, could both cause Action "grab_object". The binding path that you'd like to suggest is a hierarchical string like /user/hand/right/input/select/click. Here you can see that we are referring to the select button on the right hand's controller, and specifically the "click" operation of that select button. For the last two path levels there are naming conventions and standardized names that are detailed in the specification. All these paths are listed in the interaction profile definitions in the specification.

It's important to only suggest bindings for generic profiles, as well as interaction profiles matching devices that you actually tested. Your application will still be able to run on other runtimes: those runtimes may automatically map your Actions to the controller available. As runtimes continue to advance, you can expect that these rebindings will increase in quality, and can be updated independently from your application. They may also provide an interface for the user to customize how the Actions are mapped to their controller, and perhaps even share these bindings with other users. Remapping or rebinding can also be done to improve comfort or accessibility, to support a wider array of users. This part of the specification design is strongly influenced by the success Valve has found with their similar SteamInput and SteamVR Input Action binding and remapping systems.

One last Action setup step

- **Set up Actions, Action Sets, and provide suggested bindings at application start.**
- **Before you can use them, one more call is required later:**
  - xrAttachSessionActionSets
  - Associates them with the session
  - Makes them immutable
  - Editor authors: tear down session, actions, action sets and re-create to modify them
- **Why is action setup done all up front and immutable?**
  - Good rebinding experience needs maximum information on interaction early in execution



You will typically set up Actions, Action Sets, and suggest bindings for interaction profiles in a single block of code during the startup process of your application. This model is actually enforced by OpenXR: before you can use actions in your main loop, you need to make a call to "Attach" your Action Sets to a Session. This tells the runtime that you're all done setting up my Actions, Action Sets, and suggested bindings; that you're going to use them with this Session; and that you're not going to change them anymore. This does make your Actions and Action Sets immutable: you can't modify them after this point and there's a special error code that you'd get if you tried to do that. If you happen to be writing an editor for a game engine, the solution for editing Actions is that each time, you need to tear down the Session, the Actions, and the Action Sets, and then recreate them in order to modify them.

This seems like a pain but there's an important reason that the Action setup is done all up front. The main reason is rebinding. We want the runtime to be able to provide the user with the maximum ability to configure how they're interacting with their application right away. To do this, your application must provide the runtime with the maximum information about its interactions as early as it can. That way, when a user launches your application, if you do not have suggested bindings for the hardware that they have, the runtime can remap the actions automatically, or pop up a UI that lets them map your specified Actions to the hardware that they have available. This would happen behind the scenes and goes unnoticed by your application: you just get compatibility.

If you were able to add Actions and Action Sets later on during execution, not up front, it would then interrupt the flow of your application if rebinding needed to be done a second time. Additionally, if a runtime supports sharing bindings between users, you'd be able to compose a binding that supports all Actions only if Action setup is all done at once. Otherwise, for example, if there's an Action or Action Set that's only used in the last scene or two of your game, then a community created rebinding might be incomplete if that scene was not yet reached or was on a path that wasn't reached, making it less useful in general.

Sample of Actions

- These are the actions from “hello_xr” - see **OpenXrProgram::InitializeActions**
- All in one action set, “**gameplay**”, due to simplicity of the app
- All except “Quit” are specified for both left and right hand as “subaction paths” because we might want to know which hand did an action
  - which hand grabbed object, etc.

actionName	localizedActionName	actionType	subaction path
grab_object	Grab Object	Float Input	/user/hand/left /user/hand/right
hand_pose	Hand Pose	Pose Input	/user/hand/left /user/hand/right
quit_session	Quit Session	Boolean Input	
vibrate_hand	Vibrate Hand	Vibration Output	/user/hand/left /user/hand/right

To make this a bit more concrete, I've gone through the sample hello_xr application that's in the OpenXR-SDK-Source and summarized the Actions and bindings that are used there. The InitializeActions member function is where these get set up, if you want to look at the source code on your own later. All these Actions are in a single Action Set because the application is very simple. Additionally, all of these Actions except “Quit” are specified with what we call sub-action paths: in this case, /user/hand/left and /user/hand/right. The concept of sub-action paths lets the user perform an Action with either of two hands, and also let you know which one hand actually did it. It's similar to the SteamVR Input concept of restrictToDevice, in case you're familiar with that. The Quit Session action is different because we are not interested in which hand chose to quit session, just that some hand did. We will still suggest bindings for it on the left and right hands, but we didn’t specify sub-action paths because we don’t need to know the hand that performed the action.

The four Actions represent a variety of Action types. We have

- grab_object
- hand_pose
- quit_session
- vibrate_hand

xrSuggestInteractionProfileBindings 1

- **Standard defines “khr/simple_controller” as a minimal subset profile**
- **Note here that grab_object is float, but suggested to bind to “select/click” (boolean)**
  - Runtime will automatically convert boolean to a 1 or 0.

actionName	actionType	subaction path	/interaction_profiles/khr/simple_controller
grab_object	Float Input	/user/hand/left	/user/hand/left/input/select/click
		/user/hand/right	/user/hand/right/input/select/click
hand_pose	Pose Input	/user/hand/left	/user/hand/left/input/grip/pose
		/user/hand/right	/user/hand/right/input/grip/pose
quit_session	Boolean Input		/user/hand/left/input/menu/click
			/user/hand/right/input/menu/click
vibrate_hand	Vibration Output	/user/hand/left	/user/hand/left/output/haptic
		/user/hand/right	/user/hand/right/output/haptic

After creating that action set and those four actions, it's time to suggest bindings. There are a number of calls in hello_xr to suggest bindings. I've picked a few of them for three different interaction profiles to illustrate some points.

This first one uses the interaction profile khr/simple_controller. Unlike most other interaction profiles, this does not correspond to any particular specific piece of hardware. Instead, it's a generic lowest-common-denominator sort of device that can be mapped to a wide variety of hardware. In the case of Simple Controller, essentially all runtimes will be able to map these bindings on to their available controllers.

There are a few things to notice here. Overall, as you'll see is a pattern, the suggested binding path for the interaction profile starting with /user/hand/left will be marked as for the subaction path of /user/hand/left, and similarly similarly for /user/hand/right. In this application all Actions can be done with either hand, but you would not see this same pattern if you had Actions that could only be performed by one hand. Additionally, “Quit Session” is suggested for a button on the left and right hand, but since it was not defined earlier with sub-action paths, we do not specify sub-action paths for the suggested bindings.

One point about the simple controller is that its select input is boolean: it only has **on** or **off**. We're binding grab_object, which is a float input, to /user/hand/left/input/select/click, which is boolean. This is fine: the runtime will automatically convert that boolean value to a float one or zero. There are conversion rules that are described in the specification for these common, simple cases.

xrSuggestInteractionProfileBindings 2

- HTC Vive controller
- The **grab_object** action is here suggested for the “trigger/value” input
  - trigger/value instead of select/click
  - float instead of boolean: no conversion required

actionName	actionType	subaction path	/interaction_profiles/htc/vive_controller
grab_object	Float Input	/user/hand/left	/user/hand/left/input/trigger/value
		/user/hand/right	/user/hand/right/input/trigger/value
hand_pose	Pose Input	/user/hand/left	/user/hand/left/input/grip/pose
		/user/hand/right	/user/hand/right/input/grip/pose
quit_session	Boolean Input		/user/hand/left/input/menu/click
			/user/hand/right/input/menu/click
vibrate_hand	Vibration Output	/user/hand/left	/user/hand/left/output/haptic
		/user/hand/right	/user/hand/right/output/haptic

The second example is the HTC Vive controller. Here you can see that we've bound the grab_object action to a different path. That's because the Vive controller has an analog trigger rather than a button simply labeled "Select". This analog trigger, which is a float, is now being suggested as our binding for grab object. The runtime will not need to convert a boolean to a float. The grab_object action will get something that's not just limited to 0 or 1 but might be anywhere in that entire range.

xrSuggestInteractionProfileBindings 3

- Oculus Touch controller
- Has a float input suitable for **grab_object** action - called "squeeze/value"
- Only left controller has a menu button, so not suggesting a binding for **quit_session** on the right hand.

actionName	actionType	subaction path	/interaction_profiles/oculus/touch_controller
grab_object	Float Input	/user/hand/left	/user/hand/left/input/squeeze/value
		/user/hand/right	/user/hand/right/input/squeeze/value
hand_pose	Pose Input	/user/hand/left	/user/hand/left/input/grip/pose
		/user/hand/right	/user/hand/right/input/grip/pose
quit_session	Boolean Input		/user/hand/left/input/menu/click
vibrate_hand	Vibration Output	/user/hand/left	/user/hand/left/output/haptic
		/user/hand/right	/user/hand/right/output/haptic

The last example of suggested bindings is the Oculus Touch controller. The Oculus Touch controller has a float squeeze input: it can report how hard you are squeezing using a floating point value between 0 & 1. The grab_object action will get a value anywhere in that range, just like the analog trigger on the Vive controller. Additionally, only the left controller has a menu button, so in this case we're only suggesting a binding for the left hand for quit_session which uses the menu button. We're not suggesting a binding for /user/hand/right and that's okay.

Wrap-up

- **Outline**
  - About Me
  - Interaction in OpenXR
  - Sample Actions Walkthrough
- **Resources**
  - Landing page with news: khronos.org/openxr
  - API registry (links to the spec, ref pages, all the repos, etc) khronos.org/registry/openxr

Thank you!

- **Community**
  - Source, issue trackers, etc github.com/KhronosGroup?q=openxr
  - Chat khr.io/slack
  - Forum community.khronos.org/c/openxr
- **Open-Source Runtime for Linux: Monado**
  - Community project founded by Collabora, not a Khronos/OpenXR WG project
  - Repos, including additional (cross-platform) OpenXR-related projects gitlab.freedesktop.org/monado

Thanks for your time! Hopefully you found this introduction to the OpenXR interaction model to be helpful. I've put a number of resources on this slide that you can follow for additional information. If you have questions that I didn't answer during this session, please feel free to drop by one of the community locations for the OpenXR group and leave a note there with your question. I or someone else in the community will be happy to respond to you.

Interaction in OpenXR

Ryan A. Pavlik, Ph.D.
Principal Software Engineer, Collabora, Ltd.
OpenXR Working Group Spec. Editor