



WEBXR API IN 20 MINUTES

(Coffee Break Series)

Kenwright



Copyright © 2021 Kenwright
All rights reserved.

No part of this book may be used or reproduced in any manner whatsoever without written permission of the author except in the case of brief quotations embodied in critical articles and reviews.

BOOK TITLE:
WebXR API in 20 Minutes
ISBN-13: 979-8533764735

The author accepts no responsibility for the accuracy, completeness or quality of the information provided, nor for ensuring that it is up to date. Liability claims against the author relating to material or non-material damages arising from the information provided being used or not being used or from the use of inaccurate and incomplete information are excluded if there was no intentional or gross negligence on the part of the author. The author expressly retains the right to change, add to or delete parts of the book or the whole book without prior notice or to withdraw the information temporarily or permanently.

Revision: 01072021
Author: Kenwright

1	Introduction	3
2	Getting Started	17
3	Rendering with WebXR	45
4	Augmented Reality (AR)	71
5	Virtual Reality (VR)	81
6	Immersive Sound	97
7	Engines and Libraries	103

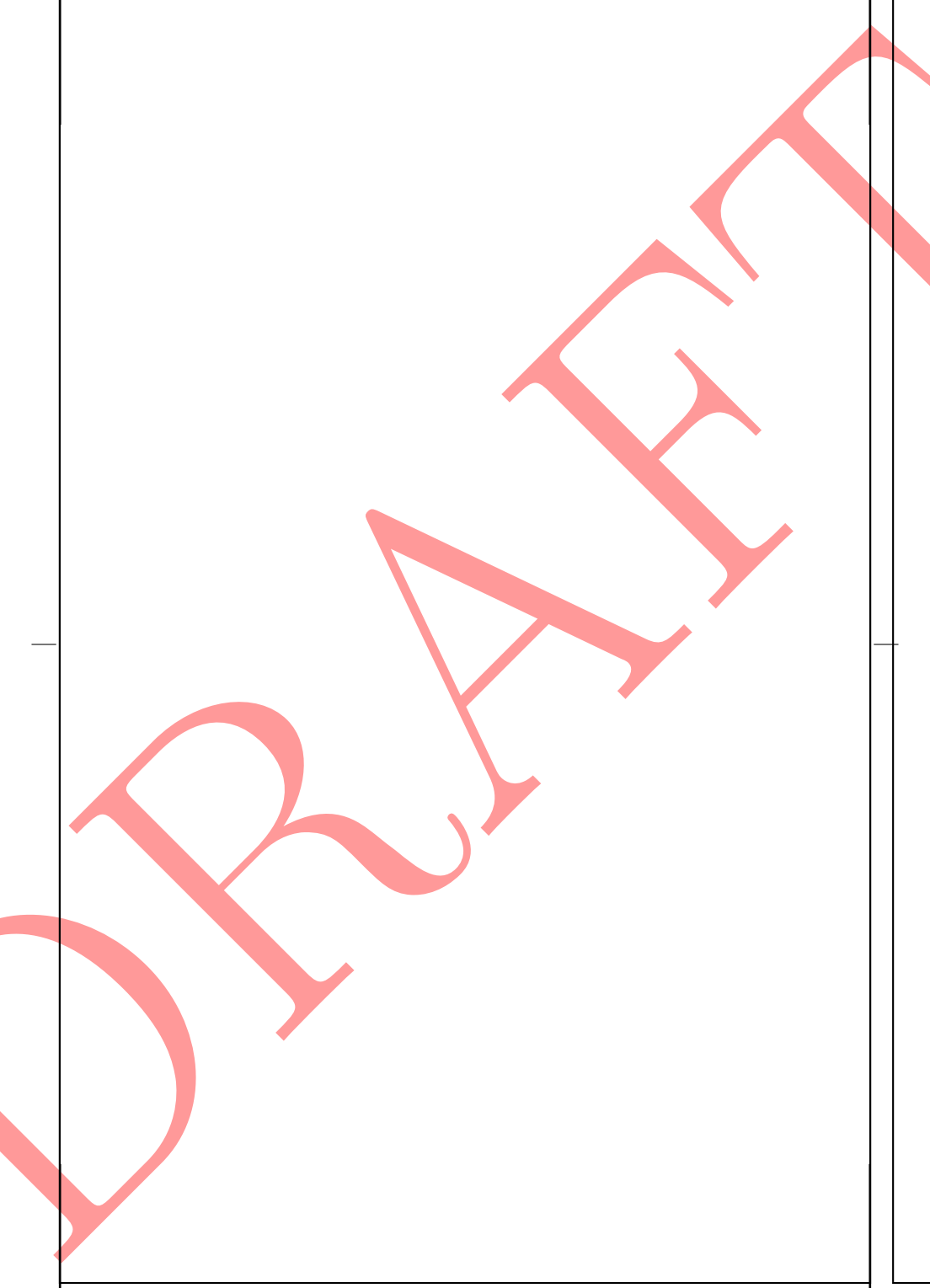


Contents

1	Introduction	3
1.1	About this Book?	3
1.1.1	What this book is Not!	3
1.2	What you'll learn	5
1.3	What is XR?	5
1.4	What is WebXR?	8
1.5	Why should you learn WebXR?	8
1.6	What can you use WebXR for?	9
1.7	Why did WebVR die/not succeed?	11
1.8	What are the Prerequisites?	11
1.9	Which devices support WebXR?	12
1.10	How does WebXR work?	12
1.11	Structure of this Book	13
1.12	Summary	14
2	Getting Started	17
2.1	Introduction	17
2.2	Setting up WebXR	18
2.3	Tools	20
2.3.1	A Code-Editor	20
2.4	Skeleton Test Web-Page	21
2.5	Running your WebXR Programs	22
2.5.1	Python HTTP server module	23
2.5.2	CodeSandBox - Online Server	23
2.5.3	Only HTTPS (Secure origin required)	23
2.6	Can you play with WebXR? (WebXR Supported?)	24

2.6.1	'isSessionSupported' is not a function	28
2.7	You don't have a VR/AR device ('inline')?	28
2.7.1	Security and Hardware enumeration	30
2.8	Graphical Output	30
2.8.1	Setting up the CANVAS renderer	32
2.9	Input Tracking	33
2.10	Update Loop	34
2.11	User Input	36
2.11.1	Input Types	37
2.11.2	Input Events	38
2.11.3	VR vs AR Input	38
2.12	End Presentation	40
2.13	Summary (Putting it all together)	40
2.14	What to do next?	43
3	Rendering with WebXR	45
3.1	Introduction	45
3.2	What is WebGL?	46
3.2.1	Vertex	48
3.2.2	Pixel (Fragment)	48
3.3	Rendering a Triangle in WebGL	48
3.3.1	Simple Vertex and Pixel Shader	49
3.4	WebGL inside WebXR	53
3.5	Spatial Transformations	56
3.5.1	No Transformation (Identity)	58
3.5.2	Translation	59
3.5.3	Scaling	59
3.5.4	Rotation	60
3.5.5	Implementation and Managing Transforms	61
3.6	Camera Transforms	63
3.6.1	Orthographic Projection	64
3.6.2	Perspective Projection	64
3.6.3	Implementing Camera	65
3.7	Shaders Cont.	66
3.8	What to do next?	70

4	Augmented Reality (AR)	71
4.1	Introduction	71
4.2	What are the benefits of AR?	72
4.3	A closer look at AR development	72
4.4	Building your WebXR augmented reality applications	73
4.5	Augmenting Your World	75
4.6	Marker aligned with the floor (detected surface)	79
4.7	What to do next?	80
5	Virtual Reality (VR)	81
5.1	Introduction	81
5.2	Viewer Perspective (e.g., Headset)	86
5.3	Poses (Reference Spaces)	87
5.4	Rendering	89
5.5	Moving/Locomotion	92
5.6	Grabbing	93
5.7	What to do next?	95
6	Immersive Sound	97
6.1	Introduction	97
6.2	How to Add Sound to the Web Browser?	98
6.3	Trigger Sounds	98
6.4	Background Music	99
6.5	3D Sounds	99
6.6	Audio and WebXR	101
6.7	What to do next?	102
7	Engines and Libraries	103
7.1	Games, Engines and Libraries	103
7.2	What to do next?	104



Preface

WebXR seamlessly combines XR technologies (VR, AR and MR) with the flexibility and accessibility of your browser to help you easily and quickly develop versatile and creative XR solutions. In this book, you'll learn definitions, terminologies and implementation details. You'll combine basic concepts with uncomplicated working examples to help you see how WebXR works. As a strong understanding of the underlying principles is important if you're to leverage the full potential of WebXR. The purpose of this text is to introduce you to WebXR from the ground-up (get you started). As you'll discover, WebXR is a powerful interface that pulls together all the elements from extensible technologies (like VR, AR and MR). WebXR's versatility and improvisation will allow you to rapidly and freely develop expressive prototypes (by seamlessly connecting hardware and software). While WebXR offers unprecedented means to immerse and interact your audiences, it also enables you to balance and manage the ever-changing and diverse XR landscape (as XR standards evolve). These benefits are partly due the fact that WebXR utilizes the power and control of your browser. As WebXR is a fusion of Javascript, WebGL and other libraries which allow you to connect movement and visuals in unique ways (interpret expressive emotions or tell stories through action and movement). WebXR will let you nurture your creativity and encourage you to explore designs that work in novel and interesting ways. Once you've mastered the basics of WebXR, you'll have opportunities to experiment with new interactive interfaces for your applications, instead of following traditional designs which may not fit the style or approach of your system. Another characteristic of WebXR is the deliberate use of Javascript (which is simple, lightweight and flexible). This lets you easily write and test prototype quickly, such as tweaking and experimenting with ideas that 'work' by embracing your user in a truly immersive/fun situations. Overall, WebXR will allow you to support specialist hardware effortlessly (let your browser manage compatibility issues), while helping you develop applications that possess coordinated, powerful visual and emotional experiences.





1. Introduction

1.1 About this Book?

This short book provides a fast-paced introduction to WebXR that will take you on a spine-tingling adventure into the amazing world of immersive virtual, augmented and mixed reality development using your web browser. While this text covers a large amount of information, the light and compelling approach combined with diagrams, simplified code listings and relaxed style will help to keep you engaged and stimulated as you learn the topic.

1.1.1 What this book is Not!

- Not about teaching you an Engine or any of the pre-developed wrapper libraries for WebXR
- Not about hiding away underlying implementation details



*Figure 1.1: **Extended Reality (XR)** - The 'X' in XR is a variable that stands for any letter. Virtual Reality (VR) encompasses **all immersive experiences**.*

- Not about writing a complete fully featured game/interactive application

Instead this book focuses on:

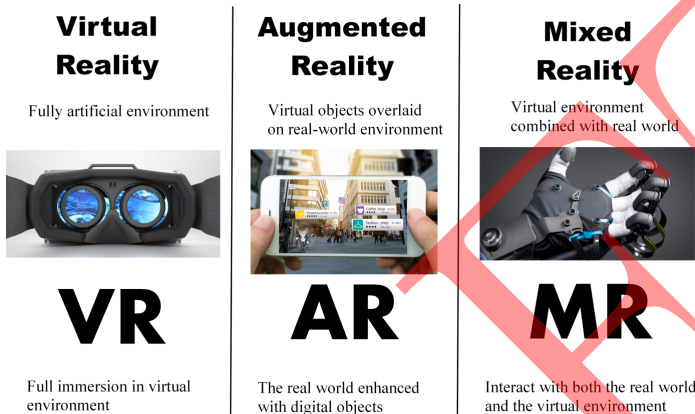
- Learning and using the WebXR API directly
- Ground-up approach using minimum working examples
- Practical book that uses a hands-on approach (writing code to test and see what’s happening and to understand why)
- Springboard for beginners to help you ‘get started’ with WebXR development (nuts & bolts)

1.2 What you’ll learn

- ✓ Discover the fundamentals of WebXR, the API, hardware and history, different applications, and the design challenges of the medium
- ✓ Learn the basics of 3D graphics, how you create objects and how to lay them out to create an environment
- ✓ Explore how you interact with a XR devices, including the concepts and technologies of XR interaction
- ✓ Utilise the skills you have learnt to create your own XR solutions

1.3 What is XR?

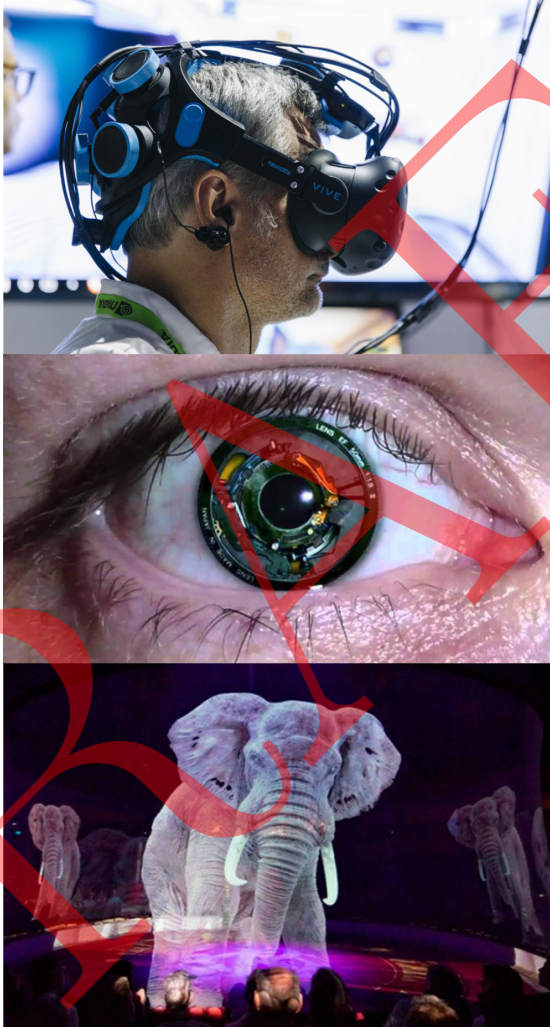
In a nutshell, **Extended Reality (XR)** refers to all real-and-virtual environments generated by computer graphics and wearables. The ‘X’ in XR is simply a variable that can stand for any letter (not limited to just VR, AR or MR - XR > VR, AR and MR). XR is the umbrella category that covers all the various forms of computer-altered reality, including (but not limited to): Augmented Reality (AR), Mixed Reality (MR), and Virtual Reality (VR). Since AR, VR and MR are heavily dependant on spatial computing, the concepts you learning in



*Figure 1.2: **WebXR** - Everything and everyone is moving online! Through WebXR, extended reality is more accessible than ever, welcoming in a new immersive frontier of storytelling, data visualization, social sharing, and so much more.*

one subset undoubtedly transfers to the others.

XR is more than Headsets and Mobile Phones As you might be wondering, is there really a need to unify VR, AR and MR under a single domain? However, you have to remember, XR is designed to go beyond current technologies - to embrace future ‘extended reality’ system, such as, brain-computer interfaces, (BCI), digital contact lenses and holograms. As these technologies evolve and become accessible, they will work together synergistically to create incredible experiences. The WebXR library is a forward thinking API, which is designed support and allow these new technologies to be integrated in over time through future updates.



*Figure 1.3: **Future XR Interfaces** - XR is not limited to headsets and mobile phones, as time goes by you will be sure to see research prototypes, such as, holograms, brain-interfaces and contact-lens systems make their way into the mainstream.*

Give It Some Thought

XR brings together VR, AR, and MR under one API (overlapping). You should consider rational behind this, for example, the graphical interface, since it's not feasible to consider displays separately in the same way it's not possible to separate optics. (Not to mention the topic of persistence which is also common and very important).

1.4 What is WebXR?

WebXR encapsulates and brings together all of the 'extended' realities (i.e., virtual, augmented and mixed reality) under one API. Previously, there was separate APIs for the different technologies (WebAR and WebVR). While WebXR provides a single set of APIs, that is integrated and built into the latest web-based technologies. Allowing you to develop and publish immersive experience through your web-browser.

While there may be many options for graphical display technologies – including libraries like WebGL, CSS, Canvas, SVG and plug-in based choices (Silverlight) - you may be asking yourself how WebXR fits in and whether or not you should learn it. So let's take a quick look at why you should.

1.5 Why should you learn WebXR?

Easy to get started with WebXR, and it lets you create immersive interactive solutions without using plug-ins. It works with most web-based platform that support the latest W3C standards, and it's so popular right now that you don't have to look far at all to find some awesome examples. Range from realistic 3D virtual worlds to augmented maps. WebXR is popping up all over the place. Currently, you'll be glad to hear, WebXR is supported by most major browsers including Inter-

net Explorer, and it even works on the vast majority of mobile platforms including iOS.



*Figure 1.4: **Browsers** - Nearly all major web-browsers are already on-board and currently support WebXR. W3C providing an open source set of standards and regulations to ensure consistency across systems/software.*

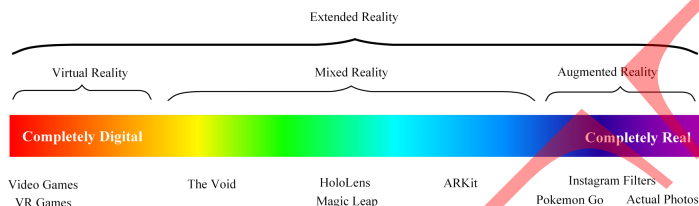
- Frictionless access
- W3C standards
- Widely supported
- Works everywhere
- Easy to use/learn

1.6 What can you use WebXR for?

Here are some instances when you should use WebXR:

- **Data visualization:** Some types of data are more useful when viewed in an immersive space, this includes things like medical MRI scans or engineering survey data
- **Games:** This is probably the most obvious. It should also be noted that games development framework (e.g., Three.js) offers libraries that support WebXR
- **Interactive page components:** This allows users to explore your product from every angle

WebXR can also offer an interesting challenge to any developer, and it gives you that extra satisfaction of having something immersive and visually attractive to demonstrate (employers or



*Figure 1.5: **Extended Reality (XR) Spectrum** - Diverse range of applications all under XR. As XR will offer new realities and play a greater role in our lives. XR industry will become increasingly important. Through technologies and libraries, such as WebXR, immersive applications will become more accessible and be integrated more into our world.*

your friends will probably be more impressed by an immersive demo than some standard visual output).

Other areas and applications include:

- 360 photo/video tours
- Web shops
- Art/music
- IoT (Internet of Things)
- Games
- Education



*Figure 1.6: **RIP WebVR** - WebVR is dead, long live WebXR.*

1.7 Why did WebVR die/not succeed?

WebVR is now discontinued. The reason the WebVR API was abandoned, was that it was **superseded by a more powerful API** that is capable of representing virtual reality and augmented reality devices simultaneously. This more powerful API (as you might have guessed) is WebXR. Also WebVR was designed to support ‘only’ VR headsets, while neglecting another important immersive technologies behind. Instead of having a WebVR, WebAR, WebMR, Web(?)R, it was decided to create a unified extensible reality API for current and future technologies. As there is nothing stopping the various approaches mixing realities to create novel hybrids/solutions. (not to mention, the graphical concepts and algorithms have similarities for the different realities - makes sense to keep them together).

1.8 What are the Prerequisites?

While the WebXR API is straightforward in itself, to understand and implement XR solutions, you need some basic knowledge of Javascript and Computer Graphics (e.g., WebGL). For this short text, you’ll benefit more if you have some previous knowledge of the following:

- Programming concepts (e.g., conditional logic, function, algorithms)
- Javascript
- HTML/CSS (basics to setup the browser window)
- Graphical principles (e.g., lighting, transforms and vectors)
- WebGL (web-based API used to generate/output graphical content to screen canvas)



Figure 1.7: **Prerequisites** - To take advantage of WebXR you should be familiar with a number of concepts, such as programming and graphics.

1.9 Which devices support WebXR?

The research/public and industry sectors were quick to adopt and take on board the WebXR API - and it's widely supported. For examples, some of the supported devices include (but not limited to):

- ARCore-compatible devices
- Google Daydream
- HTC Vive
- Magic Leap One
- Microsoft Hololens
- Oculus Rift
- Samsung Gear VR
- Windows Mixed Reality headsets
- ...

1.10 How does WebXR work?

In the fewest possible words, WebXR performs the following tasks: (1) Detect if XR capabilities are available. (2) Query the XR device capabilities. (3) Poll the XR device and associated input device state. (4) Displays imagery on the XR device at the appropriate frame rate. Simple?

Since, WebXR is an **open standard**, all industries can (and



*Figure 1.8: **WebXR, WebGL and the Browser** - WebXR uses the browser languages and the WebGL standard to manage the XR input/output.*

hopefully will) support WebXR. You have to remember, WebXR is more than just another library! It's the unification of standards that complements existing web-based tools (WebGL, WebAudio, ...). You can expect WebXR to go from strength to strength as the XR sector continues to grow and develop.

1.11 Structure of this Book

The book follows the given structure:

- The first chapter starts by explaining the fundamentals around WebXR (the whats and whys behind the WebXR API).
- This is followed by an introduction to the API using a bottom up minimum working example, which essentially gets you up and running with WebXR (i.e., initializing a running WebXR demo that doesn't do much but helps you see how all the pieces work).
- You're then introduced to WebGL (since WebXR uses WebGL for rendering).
- Then you're given two examples that focus on AR and VR specifically.
- Sound is discussed and explained (i.e., how to integrate basic sounds into your WebXR solution), since this is often an overlooked component but an important one for immersive

experiences.

- Finally, a summary of future work and projects to take your mastery of the subject further, such as, working with external libraries and engines.

1.12 Summary

XR technologies are not going to go away. They're not a passing phase! You should embracing XR technologies, which can make your life easier and offer solutions to possibilities that were previously impossible or infeasible.

To summarize:

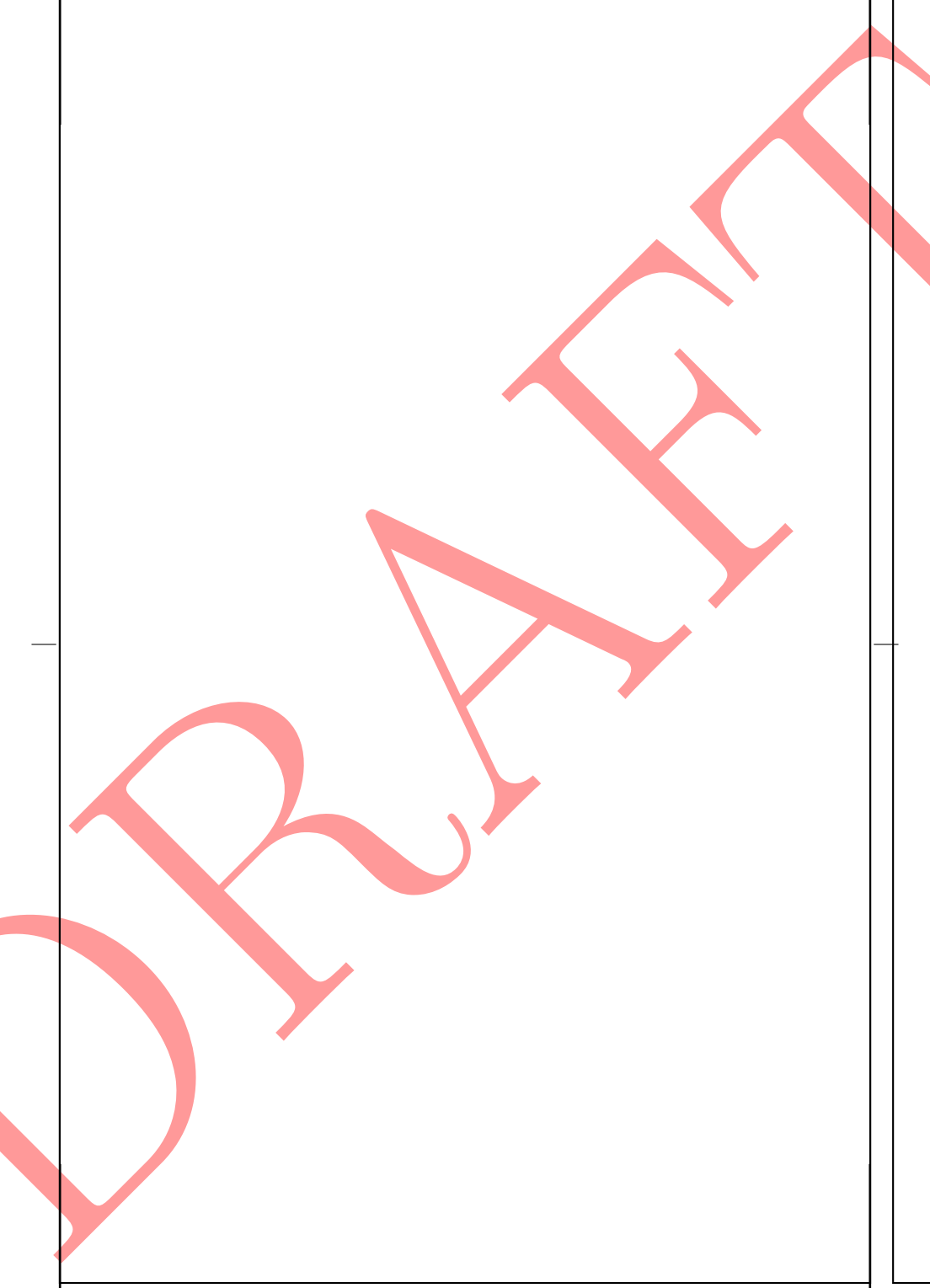
- **WebXR** is a JavaScript API that enables applications to interact with extended reality devices, such as the HTC Vive, Oculus Rift, Google Cardboard or Open Source Virtual Reality in a **web browser**
- **Extended Reality (XR)** refers to all real-and-virtual environments generated by computer technologies and wearables. The 'X' in XR is a variable that can stand for any letter
 - ✓ **Virtual Reality (VR)** encompasses all immersive experiences. These could be created using purely real-world content (360 Video), purely synthetic content (Computer Generated) or a hybrid of both
 - ✓ **Augmented Reality (AR)** is an overlay of computer generated content on the real world that can superficially interact with the environment in real-time. With AR, there is no occlusion between CG content and the real-world
 - ✓ **Mixed Reality (MR)** is an overlay of synthetic content that is anchored to and interacts with objects in the real world—in real time. Mixed Reality experiences exhibit occlusion, in that the computer-generated objects are visibly obscured by objects in the physical environment

Note: WebXR is 'new', and based on the latest exploration of virtual and augmented realities; that taps the power of the web along with the

unification of these realities, under one philosophical umbrella, making it easier to create immersive 3D art, interactive environments, VR tools and more. Once you get started with WebXR, you'll see it's an invaluable resource.

Creating mouth watering immersive experience is now more possible than ever before. The WebXR API enables you to run your XR applications in your browser, while supporting a broad set of devices and experiences. WebXR opens the door to combining different web-based technologies, like speech recognition or geolocation, not to mention, a plethora of devices that can be connected and accessed simultaneously. This can range from user/room tracking devices like the Microsoft Kinect to Cardboard-based solutions - or even control other devices around the house/office from within your virtual world. The strength of the web and these online technologies lies in its ability to connect. These connections will allow you to enrich your experiences to be more social, shareable, frictionless, open and interactive.

Just remember, the success and reach of your application is dependent on how it adapts to different ways it's experienced. If you limit your application to a specific device, you'll restrict your reach. You may well be tailoring your application to a specific piece of hardware, but you must be aware of the consequences. The trick to a successful solution with a large reach is down to the design. To be flexible in your options, to avoid leaving users behind (for example, if a user doesn't have a headset, they can still see/view the experience through their monitor). As you'll see, WebXR lets you poll for available functionality and adapt your experience to the users facilities (e.g., headset, desktop or mobile device). For a mobile devices the orientation sensors may help to visualize information accordingly, while a desktop solution would allow the mouse and keyboard to navigate and visualize the same information.





2. Getting Started

2.1 Introduction

As a web-interface, the WebXR API makes full use of the features built into many, if not all, contemporary web-browsers. You'll access these features through the dominant scripting language of the web: 'JavaScript'. The WebXR API extends the existing WebGL graphics library to allow developers to merge the writing of applications in both JavaScript and GLSL. The WebGL graphics library is a powerful API that is supported by OpenGL ES. Remember, WebXR API is more than just an interface between you and your XR applications, it provides a bridge between WebGL and the drivers for GPUs built into computers and mobile devices alike. The WebXR API is also a conduit for client-server communication which can be processed by hundreds or even thousands of processes.

Despite the WebXR API's breadth, reaching from software to hardware, from peripheral controllers to GPUs, the tools you

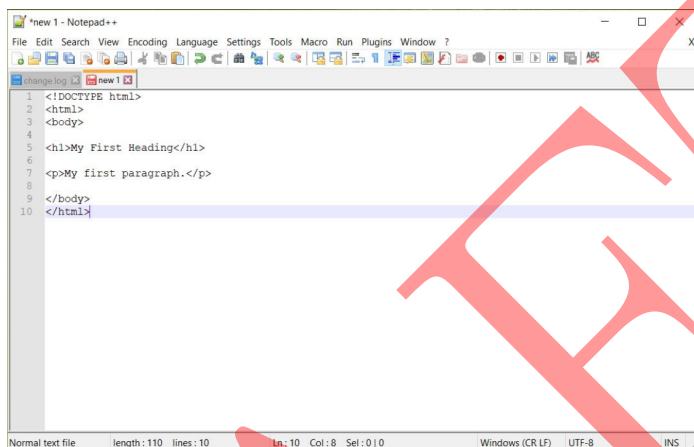


*Figure 2.1: **WebXR** - Taking a look inside (how does WebXR work).*

need as developers of WebXR content are modest. A simple code editor, like Microsoft's Visual Studio Code (or any text editor), will provide you with all the functionality you need to write HTML, CSS, JavaScript and GLSL in a document. You then host these documents locally or online to run your applications through a web-browser. You can also take advantage of the built-in debugging tools and resources of the browser itself (e.g., most browsers provide debug information, performance details and log outputs for any issues/errors while your program runs).

2.2 Setting up WebXR

Writing a minimum working WebXR program - is nothing special, but you must remember to initialize and call the WebXR API functions in the correct order. Also since most of the functions are non-blocking (asynchronous), you have to be sure each stage has finished before progressing.



*Figure 2.2: **Text Editor** - Writing WebXR programs can't be any easier - all you need is a basic text editor and a web-browser! (one example is Notepad++ (<https://notepad-plus-plus.org/downloads/>), it's free, fast, open source and has tons of features - including syntax highlighting).*

In the following sections, you'll perform the following checks:

1. Check that WebXR API is available on your browser
2. Check what options are supported by the API
3. Check for any devices
4. Enter immersive experience

While the source files are text based, which means you can write them using any .txt editor (e.g., notepad++), to run the WebXR samples you need to have a web-server running (either a local test server or an online solution). You can't just run the examples as local file in your web-browser (i.e., `file:///C:/test.html`).

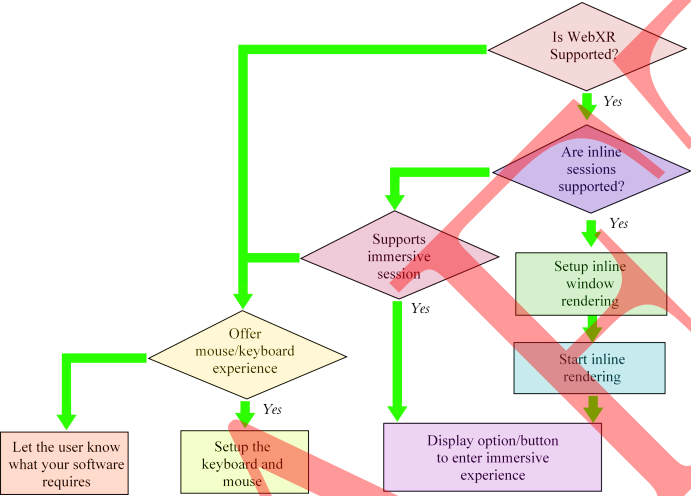


Figure 2.3: **WebXR Support** - Steps to follow when checking if WebXR is supported and what to do.

2.3 Tools

The tools described in the following sections should be easy to access and provide an uncomplicated set of tools to help you get started developing WebXR content. However, feel free to use any tools/applications that you're more comfortable with. Most of the suggested tools are free/open-source, and have been vetted by reputable parties. But remember, with any bleeding edge technologies, like WebXR, always refer to the most recent, published documentation for up-to date compatibility and requirements (just in case anything changes or certain tools/packages are required).

2.3.1 A Code-Editor

Akin to a text-editor, a code-editor allows you to type the syntax of a program into a document. Features built into a code-

editor create an environment convenient for writing, deploying, testing, and correcting code (e.g., automatic indentation, word highlighting or predictive suggestions). One such example, the Notepad++ editor, is free, cross-platform, popular and powerful. You can use it to write the HTML, JavaScript and CSS - which you need to create your XR applications for the web.

2.4 Skeleton Test Web-Page

To begin creating a web-page featuring WebXR, you must first be able to run HTML web-pages through a web-server. First, create a new text file with the '.html' extension and save it as 'index.html'. In the body of the document, type the simple code listing below. Nothing complex, it's just your minimal working HTML template for a web-page.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width,
    ↪ initial-scale=1.0">
  <title>WebXR</title>
</head>
<body>
  <script type="text/javascript">
    alert('hello');
  </script>
</body>
</html>
```

The script tags in an HTML document inform the browser's layout engine of the structure of your page. The visual content rendered to the screen occurs between the `< body >` element tags of the HTML document. You want to launch your new web-page in your web browser through your web-server. For example, if you're running a 'localhost' on port 8000, you'd navigate to the address in your browser, that is, 'localhost:8000' - which will open the web-page on your screen.

The `< script >` tag with a type attribute set to "text/-javascript" notifies the browser that what lies between these tags is distinct from HTML. In this case, you've told the

WebXR Emulator (Access to XR Devices)

If you don't have access to an XR device, then not to worry! There is the WebXR emulator. So you can still test your code and applications without a physical XR device. The WebXR emulator is a **web-browser** extension that enables you to run and test XR content using a desktop browser (without a physical XR device).

The WebXR emulator runs on the following browsers:

- Chrome
Google Chrome Web Store
- Firefox
<https://addons.mozilla.org/en-US/firefox/addon/webxr-api-emulator>

However, while you can test out XR applications using the emulator, it's worth going the extra mile to try out your applications on an physical XR device.

browser the text that will appear between the `< script >` tags will be of the type "JavaScript". For the simple example, it triggers the "alert()" popup box to be shown and displays the message 'hello'. This lets you check that your web-server is running and you can run a simple web-page/script.

2.5 Running your WebXR Programs

To test and debug Web applications written in a code-editor, developers **require the creation of a local web-server (or the use of an online server/editor)**. Mimicking the behavior of a remote server that stores and delivers web-pages and their resources to client browsers, a local web-server allows developers to launch and view web applications from their local machines (so you don't have to be connected to the internet to

test and experiment). For example, to run and test the examples in this book, you'll need to create a local server or upload the samples to a server that support http.

A popular option to create a local web-server is with Python. You require a Python installation on your machine before you can run a local server to test your programs (very easy to setup and get going).

2.5.1 Python HTTP server module

See <https://docs.python.org/3/library/http.server.html>.

2.5.2 CodeSandBox - Online Server

Another common resource for the creation, testing and development of your project is an online tool called 'CodeSandBox'. It's a popular resource and has many positive testimonials from other developers who speak favourably of its flexibility and easy of use.

See <https://codesandbox.io/>

These are just a couple of examples, and there are lots of ways to create and manage online resources. Please use whatever solution you prefer.

2.5.3 Only HTTPS (Secure origin required)

The WebXR API is considered a "powerful feature" and thus only available on secure origins (i.e., URLs using HTTPS). However, I'm sure you'll be happy to learn, for development purposes localhost counts as a secure origin, and other domains can be temporarily treated as secure via browser-specific mechanisms (e.g., using Chrome/Chromium settings).

2.6 Can you play with WebXR? (WebXR Supported?)

A few lines of code will let you check if your browser/device supports WebXR!

```
if ("xr" in window.navigator) {  
  alert( "WebXR can be used!" );  
} else {  
  alert( "Darn! WebXR isn't available");  
}
```

If you're curious what's inside the 'window.navigator' object, you can use these few lines:

```
var aa = window.navigator;  
  
var prop = "";  
  
for(var key in aa){  
  prop = prop + '<br>';  
  prop = prop + key + ' -> [' + aa[key] + '];  
}  
  
console.log( prop );
```

This will write out a lot of information, not all of it is important for WebXR, however, it shows you all of the things that your browser support and if they're active (e.g., such as the geolocation, userAgent and so on). See an example output below:

```
vendorSub -> []  
productSub -> [20030107]  
vendor -> [Google Inc.]  
maxTouchPoints -> [0]  
userActivation -> [[object UserActivation]]  
doNotTrack -> [null]  
geolocation -> [[object Geolocation]]  
connection -> [[object NetworkInformation]]  
plugins -> [[object PluginArray]]  
mimeType -> [[object MimeTypeArray]]  
webkitTemporaryStorage -> [[object DeprecatedStorageQuota]]  
webkitPersistentStorage -> [[object DeprecatedStorageQuota]]  
hardwareConcurrency -> [4]  
cookieEnabled -> [true]  
appName -> [Mozilla]  
appName -> [Netscape]
```

2.6 Can you play with WebXR? (WebXR Supported?)

25

```
appVersion -> [5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537
    ↳ .36 (KHTML, like Gecko) Chrome/86.0.4240.183 Safari/537
    ↳ .36]
platform -> [Win32]
product -> [Gecko]
userAgent -> [Mozilla/5.0 (Windows NT 10.0; Win64; x64)
    ↳ AppleWebKit/537.36 (KHTML, like Gecko) Chrome/86
    ↳ .0.4240.183 Safari/537.36]
language -> [en-US]
languages -> [en-US,en]
onLine -> [true]
getBattery -> [function getBattery() { [native code] }]
getGamepads -> [function getGamepads() { [native code] }]
javaEnabled -> [function javaEnabled() { [native code] }]
sendBeacon -> [function sendBeacon() { [native code] }]
vibrate -> [function vibrate() { [native code] }]
xr -> [[object XRSys⚠tem]]
mediaCapabilities -> [[object MediaCapabilities]]
permissions -> [[object Permissions]]
locks -> [[object LockManager]]
wakeLock -> [[object WakeLock]]
usb -> [[object USB]]
mediaSession -> [[object MediaSession]]
clipboard -> [[object Clipboard]]
credentials -> [[object CredentialsContainer]]
keyboard -> [[object Keyboard]]
mediaDevices -> [[object MediaDevices]]
storage -> [[object StorageManager]]
serviceWorker -> [[object ServiceWorkerContainer]]
deviceMemory -> [8]
presentation -> [[object Presentation]]
userAgentData -> [[object NavigatorUAData]]
bluetooth -> [[object Bluetooth]]
...
```

The important line you want to keep an eye out for is [⚠]:

```
xr -> [[object XRSys⚠tem]]
```

You can do the same as you did above, but this time, only print out the properties for the `xr` object. Here is what the code would look like:

```
var prop = "";
for(var key in window.navigator.xr){
    prop = prop + '<br>';
    prop = prop + key + ' -> [' + window.navigator.xr[key] + '>';
}
console.log( prop );
```

You should get an output in the browser console window that looks something like the following:

```
ondevicechange-> [null]
isSessionSupported-> [function isSessionSupported() { [native
  ↪ code] }]
requestSession-> [function requestSession() { [native code] }]
supportsSession-> [function supportsSession() { [native code] }]
addEventListener-> [function addEventListener() { [native code]
  ↪ }]
dispatchEvent-> [function dispatchEvent() { [native code] }]
removeEventListener-> [function removeEventListener() { [native
  ↪ code] }]
```

This is a good way of exploring your system and learning what's available. If you notice the output listed for your `xr` object, there are only seven key function properties. You'll go through what each of these lines means next. However, you should not forget, for additional details/information you can also refer to the online WebXR documentation (which is important as new updates/versions are developed).

Looking at the output for the `xr` object properties. The first line, you'll notice is 'null', which is an event callback to let you know if the device has changed. Obviously, you've not setup anything yet, so it's null.

The next important line you'll notice is 'isSessionSupported'. This is where it gets interesting.

If you call the 'isSessionSupported' method, you'll get a promise object returned:

```
let bb = navigator.xr.isSessionSupported();
console.log( bb );

>>[object Promise]
```

You access the result of a promise by using the '.then' method (or using 'await' in an async function). Your '.then' callback is then called when/if the result is made available, which will happen after you call has been resolved, or if the promise was already resolved prior to when it was called (quick/instant reply).

However, by default it will return an empty object. This is because the method takes arguments. The arguments specify which device you want to check are supported.

await vs .then() **(Performance)**

`await` is just an internal version of `.then()` (doing basically the same thing). The reason to choose one over the other doesn't really have to do with performance, but has to do with desired coding style or coding convenience. Certainly, the interpreter has a few more opportunities to optimize things internally with `await`, but its unlikely that should be how you decide which to use. If all else was equal, You would choose `await` (if you don't mind waiting). 'Await' was used for some of the examples, to make the code simpler to read/write and understand (step-by-step sequential).

Used properly, `await` can often save you a bunch of lines of code making your code simpler to read, test and maintain. That's why it was invented.

There's no meaningful difference between the two versions of your code. Both achieve the same result when the axios call is successful or has an error.

Where `await` could make more of a convenience difference is if you had multiple successive asynchronous calls that needed to be serialized. Then, rather than bracketing them each inside a `.then()` handler to chain them properly, you could just use `await` and have simpler looking code.

A common mistake with both `await` and `.then()` is to forget proper error handling. If your error handling desire in this function is to just return the rejected promise, then both of your versions do that identically. Then again, if you have multiple 'async' calls in a row and you want to do anything more complex than just returning the first rejection, then the error handling techniques for `await` and `.then().catch()` → are quite different and which seems simpler will depend upon the situation.

The different argument values are:

```
enum XRSessionMode {  
  "inline",  
  "immersive-vr",  
  "immersive-ar"  
};
```

2.6.1 ‘isSessionSupported’ is not a function

```
navigator.xr.isSessionSupported( 'immersive-vr' ).then( function  
  ↪ ( supported ) {  
  
  console.log( supported );  
  
} );  
  
/*  
or with 'await'  
let supported = await navigator.xr.isSessionSupported('  
  ↪ immersive-vr');  
console.log( supported );  
*/
```

2.7 You don’t have a VR/AR device (‘inline’)?

For testing on a desktop, you can use the “**inline**” webxr session. So even if you haven’t actually got a physical device at least you can test out the WebXR API.

```
navigator.xr.isSessionSupported( 'inline' ).then( function (  
  ↪ supported ) {  
  
  console.log( 'inline supported:', supported );  
  
} );
```

WebXR applications begin presenting XR content by calling `navigator.xr.requestSession()` with the `XRSessionMode` (e.g., “immersive-vr”). This returns a promise that resolves to an `XRSession`, which the developer keeps a reference to. **This object is what almost all further interaction is done with**

Notes (Immersive vs Inline)

- ✓ **immersive-ar** - The session's output will be given exclusive access to the immersive device, but the rendered content will be blended with the real-world environment. The session's 'environmentBlendMode' indicates the method to be used to blend the content together.
- ✓ **immersive-vr** Indicates that the rendered session will be displayed using an immersive XR device in VR mode; it is not intended to be overlaid or integrated into the surrounding environment. The 'environmentBlendMode' is expected to be opaque if possible, but might be additive if the hardware requires it.
- ✓ **inline** The output is presented 'inline' within the context of an element in a standard HTML document, rather than occupying the full visual space. Inline sessions can be presented in either mono or stereo mode, and may or may not have viewer tracking available. **Inline sessions don't require special hardware** and should be available on any user agent offering WebXR API support.

for the remainder of the VR content presentation. So if you don't have a valid 'XRSession' you can't go any further!

You should have something like this:

```
sessionType = "immersive-vr";
if (!navigator.xr) {
  alert("no WebXR supported!");
}

navigator.xr.isSessionSupported("immersive-vr")
.then((supported) => {
  // if you don't have a vr device connected fallback to '
  ↪ inline'
  // think of those poor people who want to try your demo but
  // can't because they don't have a device yet!

  if(!supported)
  {
    alert("falling back to 'inline' mode");
    sessionType = "inline";
  }
});
```

2.7.1 Security and Hardware enumeration

In WebXR, a list of connected hardware cannot be retrieved to avoid fingerprinting. Instead a single active “XR device” is implicitly picked by the user agent and all operations are performed against it. (System support for multiple XR devices at once is almost unheard of at the moment, so this isn't problematic for most any real world scenario). Changes to the available XR hardware are indicated by the 'devicechange' event of the `navigator.xr` object.

2.8 Graphical Output

This is where you bring together the WebXR device with the visual technology (i.e., the XR screen output). You configure and do this through 'Canvas'.

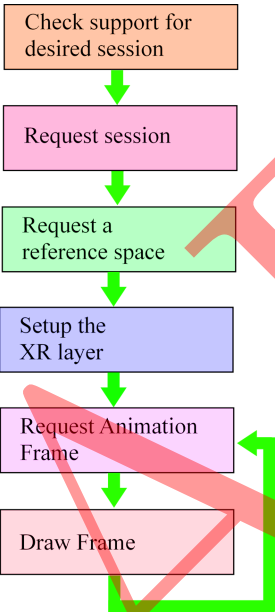


Figure 2.4: **Initialization** - Steps from checking for a supported session through to the render/update loop.

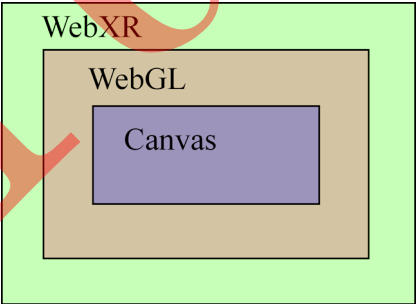


Figure 2.5: **Graphics** - WebXR presents graphics through WebGL and Canvas.

Just to note, the HTML Canvas API is used to draw graphics, on-the-fly (real-time). The Canvas API provides a means for drawing graphics via JavaScript and the HTML elements. Among other things, it can be used for animation, game graphics, data visualization, photo manipulation and real-time video processing.

2.8.1 Setting up the CANVAS renderer

WebXR applications must first make the WebGL context compatible with the active XR device. This ensures that the WebGL resources reside on the GPU that is optimal for VR rendering (for example, the one that the headset is physically connected to on a multi-GPU desktop PC). This is done by either setting the `'xrCompatible'` key to true in the `WebGLContextCreationAttributes` when creating the context or calling `gl.makeXRCompatible()` on the context after it's been created (which may trigger a context loss).

Then you construct a new `XRWebGLLayer`, passing in both the `XRSession` and an XR compatible `WebGLRenderingContext`. This layer is then set as the source of the content the XR hardware will display by passing it to `xrSession.updateRenderState()` as the `'baseLayer'` of the `XRRenderStateInit` dictionary. The canvas should not need to be resized for best results. As you can see in the following listing example:

```
let glCanvas = document.createElement('canvas');
let gl = glCanvas.getContext('webgl', { xrCompatible: true });

// *** Important***
let xrSession = await navigator.xr.requestSession('inline');
let xrLayer = new XRWebGLLayer(xrSession, gl);
xrSession.updateRenderState({ baseLayer: xrLayer });

console.log('xrSession:', xrSession );
// Now presenting to the device.
```

You can look at the `xrSession` object information:

```
xrSession:
  XRSession
  domOverlayState: null
  environmentBlendMode: "opaque"
  inputSources: XRInputSourceArray {length: 0}
```

```

interactionMode: "screen-space"
onend: null
oninputsourceschange: null
onselect: null
onselectend: null
onselectstart: null
onsqueeze: null
onsqueezeend: null
onsqueezestart: null
onvisibilitychange: null
renderState: XRRenderState {depthNear: 0.1, depthFar: 1000,
  ↪ inlineVerticalFieldOfView: 1.5707963267948966,
  ↪ baseLayer: null}
visibilityState: "hidden"
__proto__: XRSession

```

2.9 Input Tracking

WebXR also requires you to define the tracking environment you want your input/movement information communicated in. This is both to enable a wider range of hardware (like AR devices) and to simplify the creation of floor-aligned content by removing much of the matrix math.

The tracking space is specified by calling `xrSession.requestReferenceSpace()` with the desired `XRReferenceSpaceType`, which returns a **promise** that resolves to an `XRReferenceSpace`. You then supply this object any time poses requested. A ‘local’ reference space closely aligns with WebVR’s implicit tracking environment, while a ‘local-floor’ reference space aligns the virtual environment with the floor of the user’s physical environment. A ‘bounded-floor’ reference space also aligns with the user’s physical floor, with the addition of reporting `boundsGeometry`, which gives a flexible and accurate full polygonal boundary.

```

xrReferenceSpace = await xrSession.requestReferenceSpace("local"
  ↪ );

```

If you want to use reference spaces other than ‘local’ during an “immersive-vr” session you must also request consent to use it at session creation time by passing the desired type to either the `requiredFeatures` or `optionalFeatures` members of the

`XRSessionInit` dictionary passed to `navigator.xr.requestSession()`. This will cause the user-agent to prompt you for your consent to use the more detailed levels of tracking if necessary.

```
let xrSession = await navigator.xr.requestSession('immersive-vr',
  ↪ {
    requiredFeatures: ["local-floor"]
  });

let xrReferenceSpace = await xrSession.requestReferenceSpace("
  ↪ local-floor");
```

2.10 Update Loop

In WebXR, an `XRFrame` is passed into the callback provided to `xrSession.requestAnimationFrame()`. The user's pose is queried from the `XRFrame` by calling `xrFrame.getViewerPose()` with the `XRReferenceSpace` the developer wants the pose reported in. The `XRViewerPose` that's returned contains an array of `XRView`, each of which reports a `projectionMatrix` and a transform that indicates the required position of the “camera” for that view. The `projectionMatrix` is expected to be used as-is, but the transform (which is an `XRRigidTransform`) providing a position vector and orientation quaternion, as well as a matrix representation of the same transform). The `XRViewerPose` also has a top-level transform that gives the position and orientation for the VR hardware. No velocity or acceleration is exposed by WebXR at this time.

The WebXR application renders the scene **N-times**, once for each `XRView` that's reported by the `XRViewerPose`. The number of views reported may change from frame to frame.

Content is rendered into the `framebuffer` of the `XRWebGLLayer`, which is allocated by the user-agent to match the VR hardware's needs. (The default WebGL framebuffer is not used by WebXR for “immersive-vr” sessions, and can be rendered into for display on the page as usual during a VR presentation). The viewport for each view is determined by passing the `XRView` ↪ into `xrWebGLLayer.getViewport()`. The size of the `XRWebGLLayer`

→ framebuffer is determined by the VR hardware (and reported on the layer as `framebufferWidth` and `framebufferHeight` →) but can be scaled at layer creation time by setting the `framebufferScaleFactor` in the `XRWebGLLayerInit` dictionary.

WebXR automatically presents the content of the `XRWebGLLayer`'s framebuffer to the VR hardware when the `xrSession.requestAnimationFrame()` callback returns.

See the following example listing:

```
function onXRFrame(t, frame) {
  let session = frame.session;
  // Queue a request for the next frame to keep the
  // update loop going.
  session.requestAnimationFrame(onXRFrame);

  // Get the XRDevice pose relative to the Reference
  // Space created earlier. The pose may not be
  // available for a variety of reasons, so
  // you'll exit the callback early if it comes back as null.
  let pose = frame.getViewerPose(xrReferenceSpace);
  if (!pose)
  {
    return;
  }

  // Ensure you're rendering to the layer's backbuffer.
  let layer = session.renderState.baseLayer;
  gl.bindFramebuffer(gl.FRAMEBUFFER, layer.framebuffer);

  // Loop through each of the views reported by the viewer pose.
  for (let view of pose.views)
  {
    // Set the viewport required by this view.
    let viewport = layer.getViewport(view);
    gl.viewport(viewport.x, viewport.y,
               viewport.width, viewport.height);

    // Render your scene using the view's
    // matrices and the Canvas API

    // ....
  }
}
```

2.11 User Input

WebXR applications take input from multiple sources through `xrSession.inputSources`, which is an array of `XRInputSource` objects.

The `XRInputSource` contains a `targetRaySpace` (for point and click tracking) and optional `'gripSpace'` (for hand-held objects) which can be passed to the `xrFrame.getPose()` method along with the tracking space they should be reported relative to in order to get the `XRPose` of the input source.

The `XRInputSource` also has a `handedness` attribute to indicate which hand the input source is associated with, if known. For button and axis state the `XRInputSource` has an optional `gamepad` attribute, which is a `Gamepad` object (that notably lacks the non-standard extensions that was used by the old WebVR API and does not appear in the array returned by `navigator.getGamepads()`).

The `profiles` attribute of the `XRInputSource` contains an array of strings that indicate, with decreasing specificity, the type of input device and can be used to load an appropriate mesh to represent the device in the virtual scene.

The `selectstart`, `select`, and `selectend` events fired on an `XRSession` indicate when the primary trigger, button, or gesture of an `XRInputSource` is being interacted with, and can be used to facilitate basic interaction without the need to observe the `gamepad` state. The `select` event is a user activation event and can be used to begin media playback, among other things.

There is also a corresponding set of `squeezestart`, `squeeze`, and `squeezeend` events that are fired when either a grip button or squeeze gesture is being interacted with. The `squeeze` event also is a user activation event.

See the below listing example:

```
function onXRFrame(t, frame) {  
  // Queue a request for the next frame to keep
```

```
// the update loop going.
xrSession.requestAnimationFrame(onXRFrame);

// Loop through all input sources.
for (let inputSource of xrSession.inputSources) {
  // Show the input source if it has a grip space
  if (inputSource.gripSpace) {
    let inputPose = frame.getPose(inputSource.gripSpace,
                                  xrReferenceSpace);

    scene.showControllerAtTransform(inputPose.position,
                                     inputPose.orientation,
                                     inputSource.handedness);
  }
}

// Handle your rendering .....
);
```

2.11.1 Input Types

WebXR supports lots of different types of devices to handle targeting and action inputs. These devices include but aren't limited to:

- Screen taps (particularly but not necessarily only on phones or tablets) can be used to simultaneously perform both targeting and selection.
- Motion-sensing controllers, which use accelerometers, magnetometers, and other sensors for motion tracking and targeting and may additionally include any number of buttons, joysticks, thumbpads, touchpads, force sensors, and so on to provide additional input sources for both targeting and selection.
- Squeezable triggers or glove grip pads to provide squeeze actions.
- Voice commands using speech recognition.
- Spatially-tracked articulated hands, such as haptic gloves can provide both targeting and squeeze actions, as well as selection if outfitted with buttons or other sources of selection actions.
- Single-button click devices.
- Gaze tracking (following the movements of the eye to choose targets).

2.11.2 Input Events

You manage inputs through event callbacks, such as when the user clicks the screen or presses a button on a controller.

```
xrSession.addEventListener('select', onSelect);
xrSession.addEventListener('selectstart', onSelectStart);
xrSession.addEventListener('selectend', onSelectEnd);

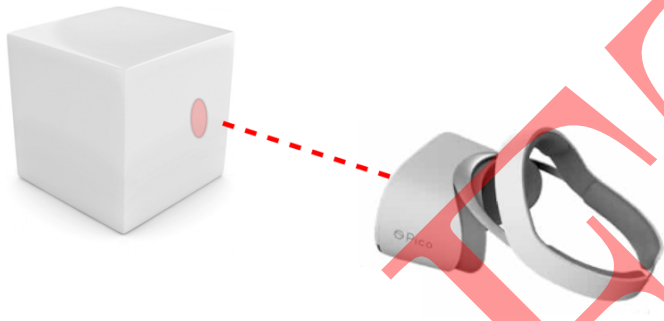
xrSession.addEventListener('squeeze', onSqueeze);
xrSession.addEventListener('squeezestart', onSqueezeStart);
xrSession.addEventListener('squeezeend', onSqueezeEnd);

function onSelect(event)
{
  // manage input event for the select
}
...
```

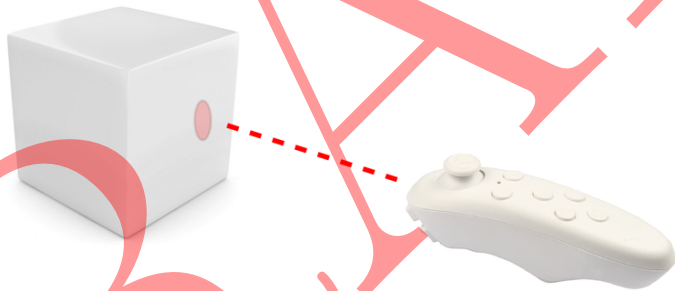
2.11.3 VR vs AR Input

Touch-based input (AR) You've got your demo running but it doesn't do much and isn't really immersive, but you'll add these parts in later. For example, touch-based interactions are used typically for inline or **immersive-ar** with tablet/phone experiences. These are a bit simpler to use since you don't need any representation, the user just touches the screen and you need to determine which object was selected in your 3D scene.

Gaze-based input (VR) Typically this approach is used when your implementation has a dedicated headset. The user will look at something and you can extract this information, based on the head pose. If you use the headset for picking options, you'll want to draw a cursor to help the user (show what they're aiming for and selecting). You might want to draw a line or lazer from the headset to the target, but be aware, it would come right out of the user's head, and can cause discomfort. Usually easier and more effective to draw a target of some kind in from of the viewers gaze to let them know what they're look at or selecting.



*Figure 2.6: **Gaze** - Viewers head position and orientation to identify the direction and target (e.g., ray-line intersection test).*



*Figure 2.7: **Pointer** - The controller provides a pointer-based means for interacting and selecting items in the virtual world.*

Pointer-based input Users will use this type of input when they have a hand-held controller, no matter what type of degrees-of-freedom (DOF) is supported. In VR, you'll typically render a representation of it in your scene, so the user can see where the controller is and where it is pointing. To represent the pointing aspect, you typically fire a ray (or line) straight from the virtual controller representation and potentially some kind of cursor on an object that the user can interact with, helping them to understand that something will happen if you select that object.

2.12 End Presentation

WebXR may have the presentation of XR content ended by the user-agent at any time.

WebXR applications may explicitly end the presentation of XR content by calling `xrSession.end()`, at which point the `XRSession` object becomes unusable. An end event is fired on the `XRSession` when it is ended by either the application or user-agent.

For example, you can catch the event using the following listing example:

```
xrSession.addEventListener("end", () => {  
  // XR presentation has ended. Do any necessary cleanup.  
});
```

2.13 Summary (Putting it all together)

A complete minimum working example that runs (doesn't do much, just initializing the basic WebXR API and sets your update loop going). The default is for the 'inline', so you can test out your program first before plugging in extra XR hardware.

To help manage concepts, let's define some typical functions (common names and conventions). You're not limited to these functions, but these provide some standardization as you go through the book - and identify key points for you to refer to in subsequent programs. You'll also add extra functions, for example, to manage events for user input and graphics initialization.

```
async function xrInit() { /* 1 */  
  ...  
}
```

```
function onNoXRDevice() { /* 2 */  
  ...  
}
```