VRML Standard
Version 2.0

March 2025

What is VRML?

The Virtual Reality Markup Language (VRML) is designed to describe how a location in the VRWeb needs to be displayed. A software that supports VRML and its associated protocols (HOPPER), can take the information that is stored in this data file to download the assets and create the experience that was defined in such a file.

What's new?

Version 1.0 of the VRML Standard failed at the attempted to be universal and platform independent. It relied to much on specific, existing software solutions and was neither flexible nor future prove. The new Standard v2.0 on the other hand is designed to solve these identified key weaknesses of the first version and extend the standard with the accumulated knowledge of the last couple of years.

VRML v2.0 is designed as an Open Standard to be extended for future needs.

Unfortunately, because of the substantial changes, the new Version 2.0 is not backwards compatible at all. For existing projects, the VRML-Files need an update to work with an VRML v2.0 compatible Hopper if the Hopper doesn't support both versions simultaneously. But you'll find it straight forward to update your VRML-Files from v1.0 to v2.0 because the information is now part of individual designed protocols or is not needed any more.

How is a VRML-File structured?

VRML follows the guidelines of an XML data file. So, the data is enclosed with a VRML-Tag and specification of the used version.

Example:

```
<vrml version="2.0">

</vrml>
```

The first part is a general section with a domain (rootDomain) for connecting the location to a world. If this is not specified a VRML compatible hopper should use the base of the link as the domain. With %ROOT% you can patch this root domain in the entire VRML-File.

The next part is a mandatory section that contains the creator/publisher information of the location. By ensuring that this section is mandatory all locations and worlds will have a legal notice that defines who is owner and responsible for the content.

Example:

```
<vrml version="2.0">
      <rootDomain>https://example.com/MyWorld</rootDomain>
      <creatorInfo>
            <name>Creator's Name</name>
            <legalNotice type="Text">
                  Legal notice as text or link
            </legalNotice>
            <copyright>© Creator 2025</copyright>
      </creatorInfo>
</vrml>
```

The last part is a collection of sections that can be defined individually. They require a defined protocol and a version. Thes protocols are defining how and what should be displayed. A Hopper can read each of these sections and can figure out if or if not, it supports those display protocols and then follow the portal or inform the user that this portal requires additional software that supports this special protocol.

There are some predefined protocols available.

Complete Example:

```
<vrml version="2.0">
      <rootDomain>https://example.com/MyWorld</rootDomain>
      <creatorInfo>
            <name>Creator's Name</name>
            <legalNotice type="Text">
                  Legal notice as text or link
            </legalNotice>
            <copyright>© Creator 2025</copyright>
      </creatorInfo>


      // Protocols
      <protocol name="WORLD_BUILDER_PROTOCOL" version="1">
            <locationPath>%ROOT%/MyLocation.wbz</locationPath>
      </protocol>

      <protocol name="META_INFO" version="1" optional="true">
            <name>Name of the location</name>
            <description>Description of the location</description>
      </protocol>
</vrml>
```

Protocols

Age Restriction Protocol

Protocol: AGE_RESTRICTION
Protocol Version: 1

Use Case:

Use this protocol if you want to deny access for under or overage users.

Values:

<u>min</u> *integer - optional*	
Minimum age	
<u>max</u> *integer - optional*	
Maximum age	

Example:

```
<protocol name="AGE_RESTRICTION" version="1">
      <min>16</min >
      <max>99</max >
</protocol>
```

Download And Run Protocol

Protocol: DOWNLOAD_AND_RUN

Protocol Version: 1

Use Case:

Use this protocol to download and run an app. Please be aware that this is designed to be untrusty. All Hopper implementations should ask the user if they want to download and run the app.

Values:

<u>appName</u> *string/text*

Name of the Hopper

<u>waitOnReturn</u> *bool*

This is for a return to this Hopper if the other Hopper/App is closed

<u>appPath</u> *string/text*

Specify the download link for the application here

Example:

```
<protocol name="DOWNLOAD_AND_RUN" version="1">
      <appName waitOnReturn="true">MyApp</appName>
      <appPath>https://example.com/MyApp_Installer.exe</appPath>
</protocol>
```

Error Protocol

Protocol: ERROR

Protocol Version: 1

Use Case:

Use this protocol if you want to inform the user about a non valid portal or other types of errors.

Values:

id *integer*
Error message Id e.g. 404, 500 etc.

message *string/text*
Additional error message text

Example:

```
<protocol name="ERROR" version="1">
      <id>500</id>
      <message>Internal server error!</message >
</protocol>
```

Hopper Multi User Space Protocol

Protocol: HMUS

Protocol Version: 1

Use Case:

This protocol provides the required server address and port to connect to if the location is a multi user space and uses a Hopper Multi User Space server.

Values:

host *string/text*
Address of the hosting server

port *string/text*
Port to connect to

Example:

```
<protocol name="HMUS" version="1">
      <host>127.0.0.1</host>
      <port>4711</port>
</protocol>
```

Meta Info Protocol

Protocol: META_INFO

Protocol Version: 1

Use Case:

This protocol provides necessary information for search engines and for user before they use the associated portal.

Values:

name	*string/text*
Insert the title of the location	
description	*string/text*
Insert the description for the location	

Example:

```
<protocol name="META_INFO" version="1" optional="true">
      <name>Name of the location</name>
      <description>Description of the location</description>
</protocol>
```

Package Manager Download And Run Protocol

Protocol: PACKAGE_MANGER_DOWNLOAD_AND_RUN
Protocol Version: 1

Use Case:

This protocol is designed to give information for a package manger to download a Hopper or app that should be use with this vrml. The package manager addresses are stored in the hopper to query them when needed for this application. They are trusted sources for apps.

Values:

appName	*string/text*
Name of the Hopper	
waitOnReturn	*bool*
This is for a return to this Hopper if the other Hopper/App is closed	
appId	*string/text*
Use this for defining an app id for the package manager, this might be a guid	

Example:

```
<protocol name="PACKAGE_MANGER_DOWNLOAD_AND_RUN" version="1">
      <appName waitOnReturn="true">AppName</appName>
      <appId>c55ff15d-826f-4534-bb03-7d82a9bebbaa</appId>
</protocol>
```

Portal Hopper Protocol

Protocol: PORTAL_HOPPER_PROTOCOL

Protocol Version: 1

Use Case:

Use this for locations that were build as Unity's Asset Bundles in combination with the SDK for Portal Hopper. Here all the information like Asset Bundle download address and scene path are specified. Also some other configuration settings are included.

Values:

assetBundlePath *string/text*
This is the download address of the asset bundle

scenePath *string/text*
This is the path of the scene inside the asset bundle

loadMode *enum as string/text*
Use one of this 3 modes to define how the scene should be loaded:

Add
Adds the scene to the already loaded scenes

AddAsMain
Adds the scene to the already loaded scenes and set it as main scene

Replace
Unload the loaded scenes and load the new scene as main scene

Example:

```
<protocol name="PORTAL_HOPPER_PROTOCOL" version="1">
      <assetBundlePath>https://example.com/MyLocation</assetBundlePath>
      <scenePath>Assets/Scenes/MyLocationScene.unity</scenePath>
      <loadMode>Replace</loadMode>
</protocol>
```

Recommended Hopper Protocol

Protocol: RECOMMENDED_HOPPER
Protocol Version: 1

Use Case:

This protocol should be use to define a recommended Hopper. If the currently in use Hopper does support all the necessary protocols this is ignored. If not this information can be use to decide witch Hopper should be used to display this VRML. Use this to inform which hopper was used for testing.

Values:

hopper *string/text*
Name of the recommended Hopper

waitOnReturn *bool*

This is for a return to this Hopper if the other Hopper/App is closed

Example:

```
<protocol name="RECOMMENDED_HOPPER" version="1">
      <hopper waitOnReturn="true">OtherHopperName</hopper>
</protocol>
```

Steam Protocol

Protocol: STEAM
Protocol Version: 1

Use Case:

Use this protocol to start a steam app.

Values:

steamId *integer*
Specify the steam id for the app you want to run. The app needs to be installed ahead.

waitOnReturn *bool*
This is for a return to this Hopper if the other Hopper/App is closed

Example:

```
<protocol name="STEAM" version="1">
      <steamId waitOnReturn="true">45000</steamId>
</protocol>
```

Streaming Protocol

Protocol: STREAMING
Protocol Version: 1

Use Case:

This is a protocol that defines entire locations with every asset in it. It's a very large protocol. For further information please have a look into its dedicated definition file.

Unknown Protocol

Protocol: UNKNOWN
Protocol Version: 1

Use Case:

This is a very special protocol and can be used only internally inside a hopper for not supported protocols. It is listed here to reserve this protocol name.

Values:

This protocol has no values!

World Builder Protocol

Protocol: WORLD_BUILDER_PROTOCOL
Protocol Version: 1

Use Case:

This protocol is for World Builder export loading. It provides a Hopper with the necessary download address of the World Builder export file, so that a Hopper can download, load, and run the experiences generated with the World Builder.

Values:

locationPath *string/text*
Download address for a World Builder export file

Example:

```
<protocol name="WORLD_BUILDER_PROTOCOL" version="1">
      <locationPath>https://example.com/MyLocation.wbz</locationPath >
</protocol>
```