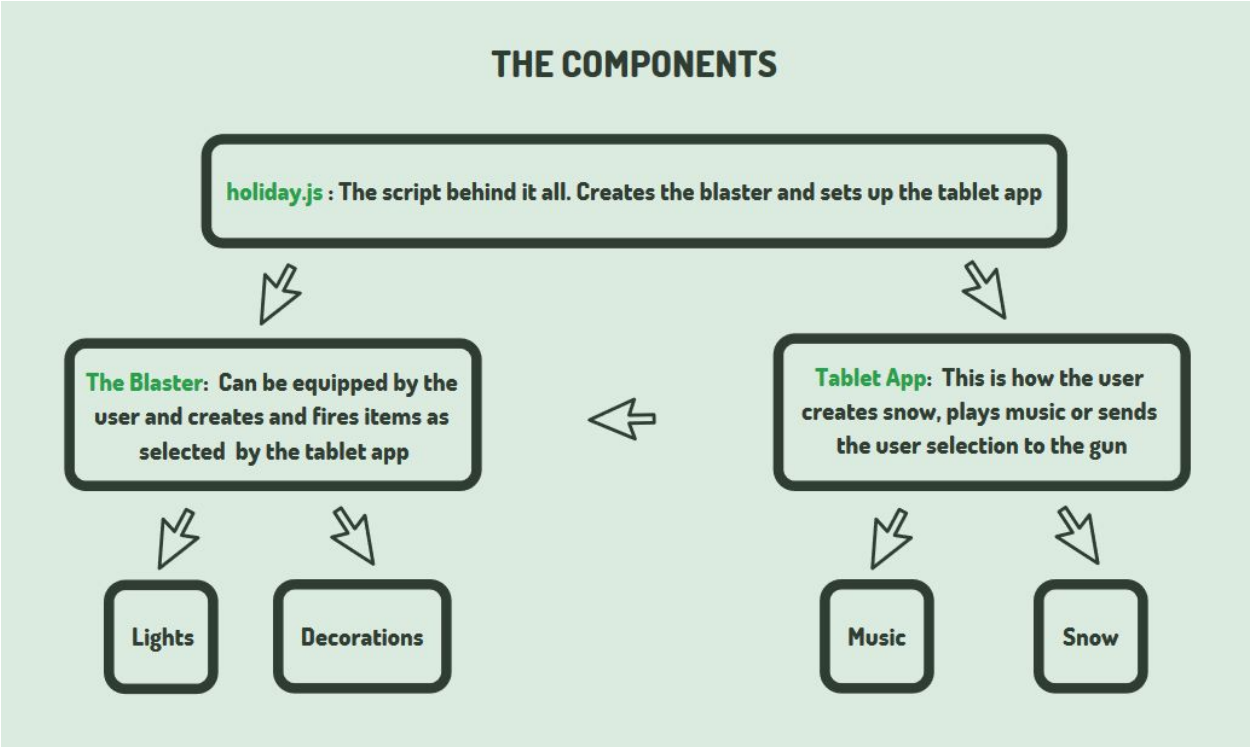
Holiday Blaster with App

This blueprint item was created by the High fidelity Experiences Team as a template to be copied and modified by our users. The code is heavily commented for clarity and by following this ReadMe, users should be able to gain some understanding of how to create using our API. Whether a user wants to completely overhaul the blaster and it's app or just swap out the items it can shoot, we encourage people to create their own variations.

The Components



There are three main parts to this item, the app, the gun, and the objects to be fired. The basis for all of this is a client script, *holidayApp.js*, that sets up the tablet app. A client script is run on a user's computer to allow direct interaction between that user and the high fidelity interface. Any changes that would affect other users, creating a cube for instance, will then pass through the High Fidelity servers. A server script, on the other hand, is carried out directly on the servers and immediately affects everyone present. When a user runs the holidayApp client script, only they will see a button for the Holiday Blaster appear on their tablet. The front end UI of the app is created in

holiday.html, styled with *holiday.css*, and allows the user to select which item to shoot next. It also plays music and sound effects and creates or deletes falling snow.

When the Holiday app is opened, the blaster is created in front of the user as an equippable gun that can shoot several different types of items. It has one client script, *gun.js*, which handles the desktop equipping and firing. On closing the app or changing tablet screens, the gun is deleted as most of its functionality requires interaction with the holiday app.

The final component of the holiday blaster are the objects which it fires. The blueprint item comes with five lights which are not grabbable and are set up with only the basic item client script, *item.js*, which makes them “stick” when they collide with another object. There is also a stocking, an icicle ornament and a ball ornament which have the same client script but are also grabbable to allow growing, shrinking, and more detailed placement. Some of the objects are created to grow on impact such as the tree, snowman, and gifts. These items have the same client script as those listed above, but in addition, they also have a server script, *itemGrow.js*, which slowly makes them larger up to a preset height. The star also has the basic item client script but adds a server script, *starSpawnLights.js*,that creates a yellow light above it when it stops moving. The last two items, the gingerbread cookie and candy cane are edible items and have a special client script for that, *edibleItem.js*. The candy cane also has the server script that makes it grow once it is in position. Both of these items are grabbable to allow for picking up to eat or stretch the items.

The Main Script: Creating the App

The tablet app is loaded when *holiday.js* runs although the user will still need to manually open it by clicking its tablet button. Two button images are linked in order to show whether the app is active. These images will be hosted .svg or .png files.

```
var TABLET_BUTTON_IMAGE = Script.resolvePath('assets/icons/christmasTree-i.png');
var TABLET_BUTTON_PRESSED = Script.resolvePath('assets/icons/christmasTree-a.png');
```

The app is then added to the tablet or toolbar by getting a reference to the tablet using the ‘Tablet’ API, pointing the page to the correct html file, and creating the button. The name that will appear under the app image can be changed here as well.

```
var tablet =
Tablet.getTablet('com.highfidelity.interface.tablet.system');
var appPage = Script.resolvePath('holiday.html');
var button = tablet.addButton({
text: 'HOLIDAY',
icon: TABLET_BUTTON_IMAGE,
activeIcon: TABLET_BUTTON_PRESSED
});
```

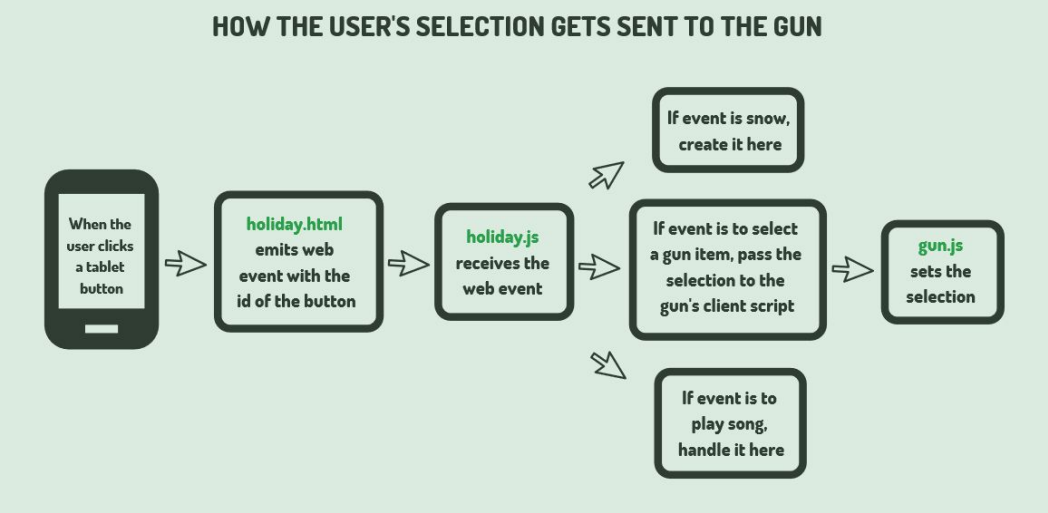
The final requirement to get the app working is to add listeners for when the user clicks the tablet button, changes tablet screens, or closes the app.

```
button.clicked.connect(onClicked);
tablet.screenChanged.connect(onScreenChanged);
Script.scriptEnding.connect(appEnding);
```

When the app's tablet button is clicked, we run the 'onClicked' function which will check to see if the app is open. If not, the tablet will be opened and sent to the app home page, a jingling bell sound will play, the gun will be created in front of the user, and the app will connect the web event listener to receive data sent from the html page. If the app was already open, it will be closed and the gun removed.

The script also checks the status of the app when the tablet screen changes. If the app is no longer open the web event listener is disconnected to avoid unnecessary actions. When the script is stopped, the 'appEnding' function will clean up by disconnecting the listeners, deleting the gun, and removing the tablet button.

Tablet App Interaction



Tablet choices are presented as buttons or images with a unique id that will be used to send the choice to the app. The snow and music are buttons created using the input tag with a button type.

```
<input type="button" class="gray" id="song3" value="Rudolph">
```

The text set by the value key is what the tablet app will show as a label on the button and the class is used to set the way the button will be formatted according to the linked style sheet, *holiday.css*.

The rest of the tablet options use the input tag with an image type.

```
<input type="image" width="48" height="48" id="snowman" alt="Snowman" src="IMAGE URL HERE">
```

The height and width of the image are set here, along with some alternative text to show in the case that the image cannot load. The src key will be a string URL that points to the hosted image.

The html file has a javascript section that includes listeners for each button, listed by id. When the button is clicked, it sets up an event and sends it over the EventBridge to be received by *holiday.js*. The event includes a unique identifier for this app to prevent other apps from trying to process the event.

```
$('#song3').click(function() {
      var event = {
            app: 'Holiday1982',
            type: "song3"
      };
      EventBridge.emitWebEvent(JSON.stringify(event));
});
```

Using the button id for the event type makes it easy to keep track of what is happening throughout the entire process, from tablet click to gun selection.

The Main Script: Sounds

When the tablet app is opened, the listener for web events is connected, enabling *holiday.js* to receive information about user selections from *holiday.html*. When an event is received, the script will read the data to make sure the info is coming from the correct app and if so, to find out which button was pressed. If the user has requested to play a song, it is handled here. At the top of the script several song URL's were prepared. Each URL points to a mono .wav or .mp3 file recorded at a rate of 48000Hz. The file is processed using the SoundCache API and is ready for use.

```
var SONG_1 = SoundCache.getSound(Script.resolvePath('assets/sounds/holly.mp3'));
```

When the script is ready to play the file, it will check that the sound was downloaded, stop any previous music, and create an audio injector using the 'Audio' API.

```
musicInjector = Audio.playSound(sound, {
      position: getPositionSnow(),
      volume: volume,
      clientOnly: false
      });
```

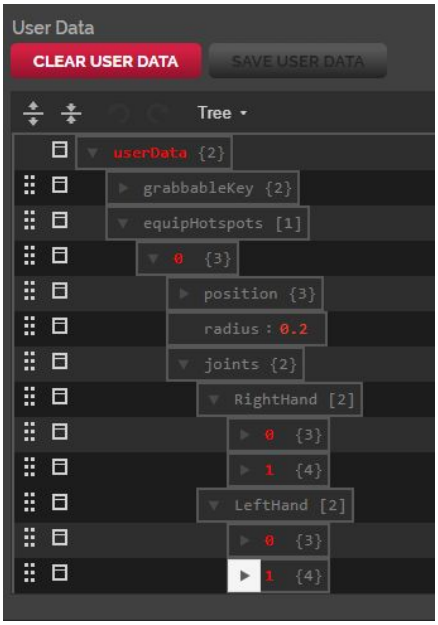
The position and volume are set upon creation of the injector and also the option to make the sound audible to all nearby users or only the user running the script. When the script is stopped, it will always check for any running music injector and stop it to prevent sounds that cannot be stopped.

The Main Script: Snow

If *holiday.js* receives a request to toggle snow on or off, it will also be handled directly by calling the 'createSnow' function. Because particles can be heavy on rendering, the app only creates one snow particle at a time. It first searches for a snow particle that already exists. If one is found, it is deleted to turn the snow off. If no snow is found, one is created via the 'Entities' API. The entity is added by passing in all key properties that should be changed from the default properties. The gun is created upon opening the app in this same manner and, later on, the gun will create the items it fires in the same way as well.

The Gun: Equippables

Objects are made equippable by setting the "equipHotspots" property in their user data which can be accessed through scripts or by selecting the item in create mode. An equippable item has a hotspot that will show as a grid sphere when a user puts their hand near the item. The radius and position of the hotspot relative to the item can be changed in the user data. When the user pulls the grip trigger on their controller while their hand is inside the hotspot, the item will attach to the nearest avatar joint of those listed in the "joints" property. In this case, the gun can attach to either the left or right hand. The position as a vector and rotation as a quaternion, of the item around the specified joint is set beneath the joint name in the properties "0" and "1". A simple way to set equip position and rotation is to copy this user data to the item, equip it, and then in HMD, grab it with the other hand to adjust to the desired placement. Then, while the item is still equipped, open a console window from the menu in Developer > Scripting. Get the item's localPosition and localRotation from the console with these commands, substituting the item's id number which can be found near the top of the create menu.




```
JSON.stringify(Entities.getEntityProperties("{*IDNUMBER*}",'localPosition').localPosition)
JSON.stringify(Entities.getEntityProperties("{*IDNUMBER*}",'localRotation').localRotation)
```

The commands will print the results which can then be pasted into the "0" and "1" properties under "joints". Follow this process for each joint listed.

Following this process will allow the gun to be equipped by clicking it in desktop, but it will simply be attached to the user's hand hanging at their side. To make holding and using the gun look realistic, the script must override the avatar's normal animation. A position is calculated for the hand holding the gun and the rest of the arm will position itself around this. At the same time, an overlay is created to show the user how to unequip or fire the blaster. An overlay is client only, meaning the only the person running the script that creates the overlay will see the overlay.

The Gun: Firing

Firing the gun takes advantage of High Fidelity's advanced physics system by simply calculating the correct direction and then adding velocity to the item in that direction. These calculations should not need to be adjusted as long as the gun's local barrel offset and direction are set correctly. The local offset is simply the position of the barrel relative to the center of the gun's collision shape. The direction shows which axis the items should fire along, also relative to the gun. On firing, the preset item is created with velocity aligned with this direction. Entities have a linear and and angular damping property with a range of 0-1 that will cause the object to stop moving over time. The blaster items have an angular damping set to 1 to prevent spinning and most have linear damping set to 0 so that they will keep moving until they collide with another object rather than stopping in mid-air. The trees also have a gravity property set in order to make them fall toward the ground.

The Items: Stopping and Growing

All items are fired from the gun at a small size to look realistic, but some will need to get bigger once they stop moving. When an object is fired from the gun, it should hit something and "stick", but due to physics, items will often bounce or ricochet. To prevent this, all items have a basic client function that listens for a collision. When a collision happens, it will check that it was not simply a collision with the gun as the object passed through the barrel and if it was not, will set the object's velocity to 0 along each axis and remove its dynamic property to prevent further motion due to physics.

For items that grow, the server script begins checking the item's velocity on creation. Every 50ms, the velocity is read and once the velocity along every axis is 0, the item has stopped moving and the script immediately begins the growing function. A preset maximum height for each object was set when the script loaded and the name of

the item was checked in order to decide which one should apply. When the item is ready to grow, its physics properties will be edited to prevent unwanted motion and a new interval will be set in which new dimensions are calculated and applied every 50ms until the maximum height is reached.

The Items: Edibles

The edible items have a different client script that adds an extra function to the basic script applied to all other objects. When a user grabs an edible item, either by triggering in HMD or clicking in desktop, a function will begin checking the distance between the item and the user's head or neck joint. Ideally, the food would be eaten when near the head, but not all avatars have a head joint, so including the neck joint as a fall back option increases the chances of the script working for more users. As long as the user is holding the food, the script continues checking that distance and if it is less than a set value, the item is considered close enough to be "eaten". A crunch sound will play and the item will disappear. If the user releases the item without eating it, the script stops checking the distance. This portion of the edible item script can be easily used on other items to make them more interactive too!