JsonLibrary Plugin

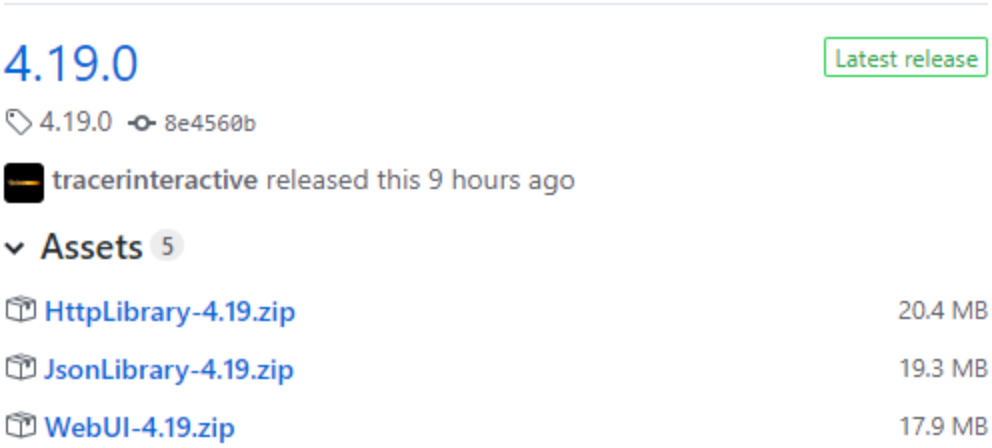
The JsonLibrary plugin supports Windows, Mac, Linux, Android, and iOS. Since it is not compatible with 4.16 or below a minimum engine version of **4.17** is required to install.

PC Android iOS

DOWNLOAD

This plugin can be downloaded from GitHub at the following address:

https://github.com/tracerinteractive/UnrealEngine/releases



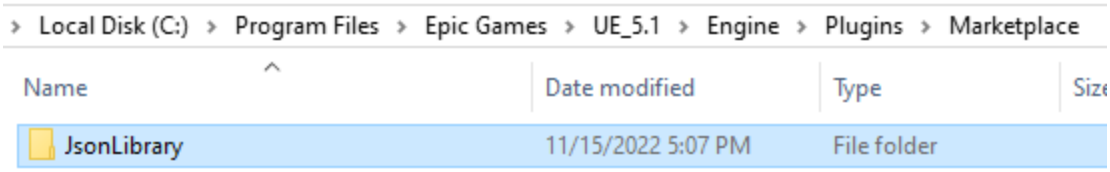
Each version is also compatible with all minor engine updates which means the 4.19.0 version of the plugin will work with any corresponding hotfix such as 4.19.1 or 4.19.2 as well.

You must have a GitHub account linked to your Epic Games account!
Setup Instructions: https://www.unrealengine.com/ue4-on-github

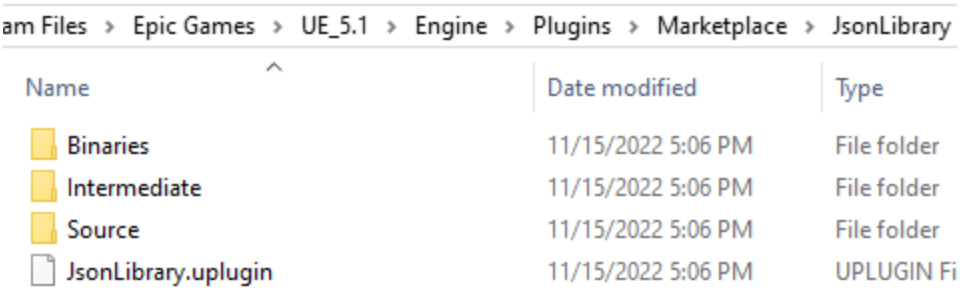
Otherwise you will receive the previous 404 error if you are not signed in with a linked account.

INSTALL

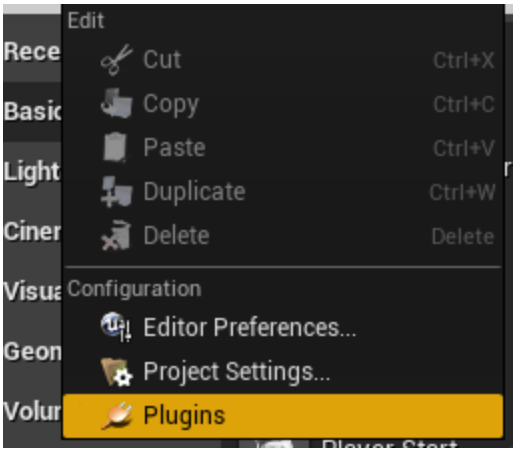
To install the JsonLibrary plugin extract the downloaded files to the following engine folder:

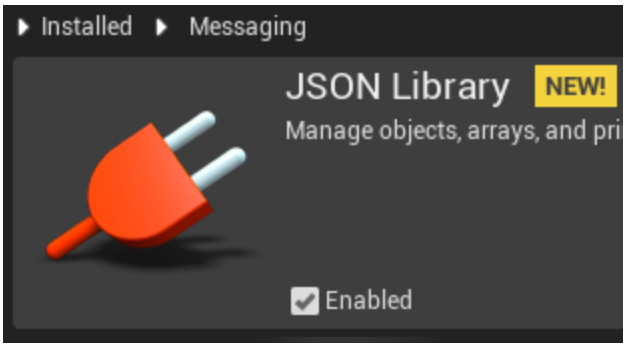


Also take note of the **UE_5.1** directory in the screenshots. You need to change this folder to the version that corresponds to the plugin version that was downloaded. *If you did not install your engine to the default directory then navigate to your custom installation folder instead.*



Then open your project and go to the “Plugins” option in the edit drop-down. Click on the “Messaging” category and enable the JsonLibrary plugin if it is not already enabled.



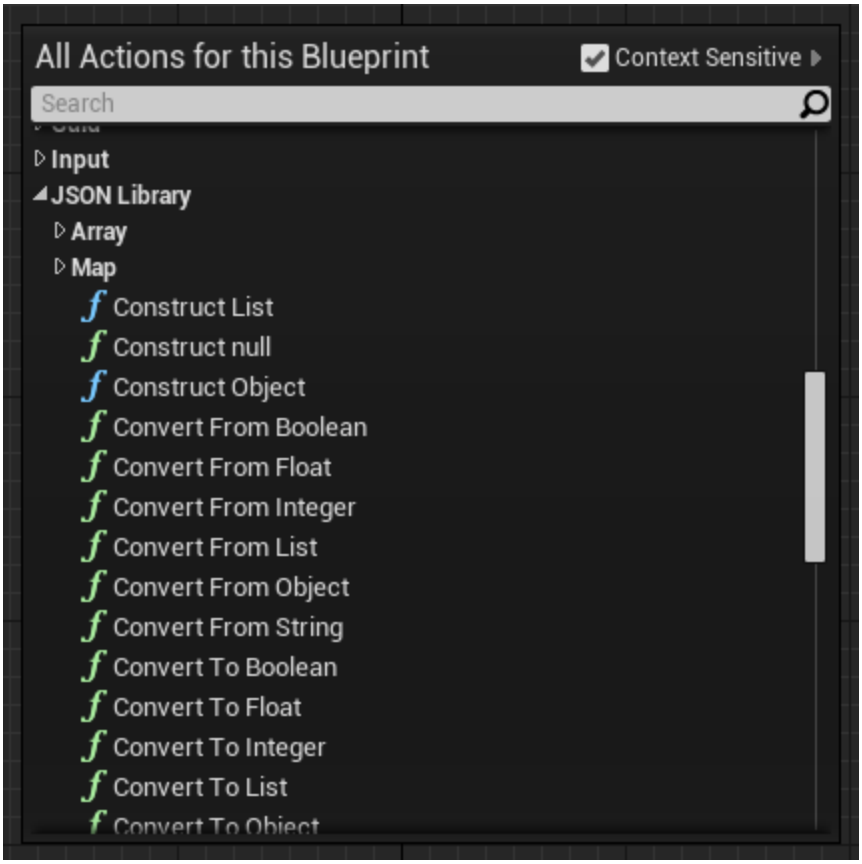


You have now successfully installed the JsonLibrary plugin. Restart the editor to continue.

TABLE OF CONTENTS

MENU

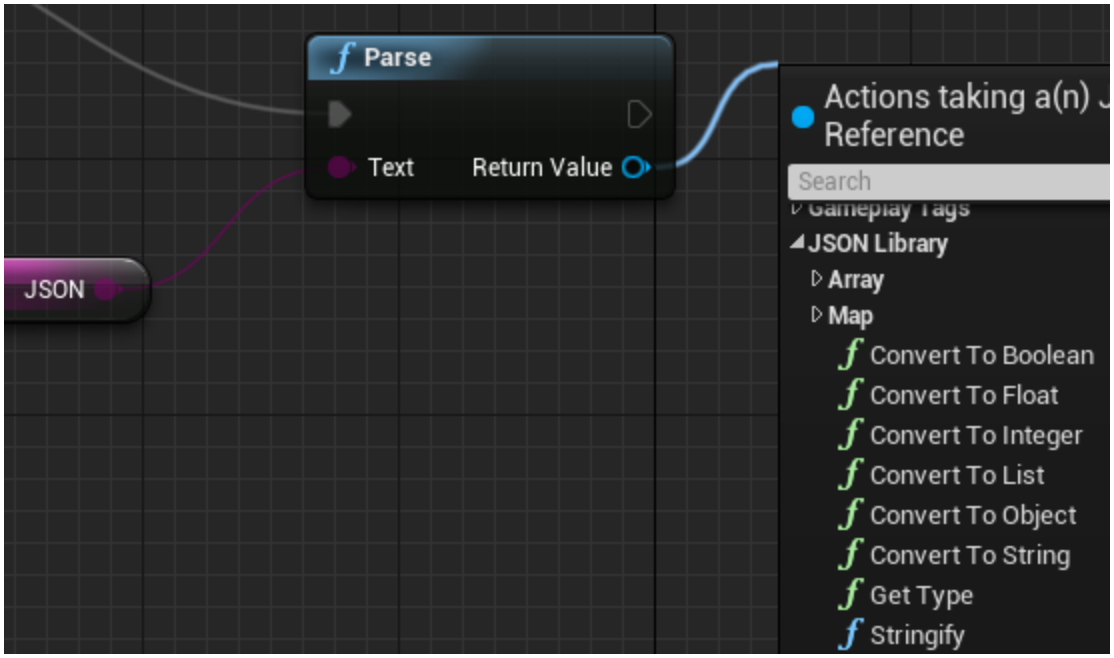
Once the JsonLibrary plugin is installed and enabled start by right clicking in any blueprint to bring up the context menu and navigate to the *JSON Library* section.



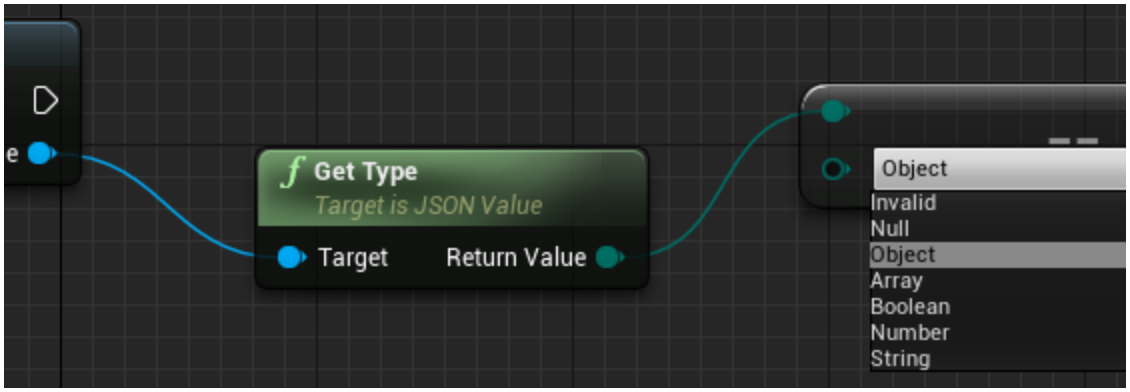
The “Construct Object” and “Construct List” nodes can be used to create JSON objects and arrays. To convert various data types into a JSON value use the “Convert From …” nodes. The “Construct null” helper is also available to create a `null` value.

PARSE

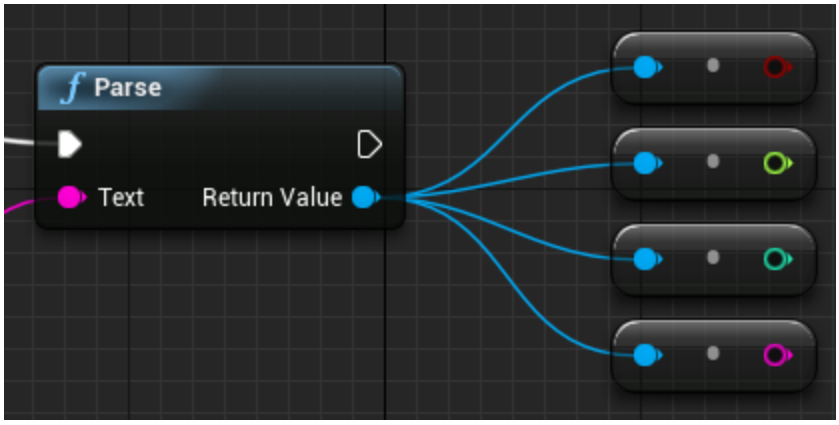
Call the “Parse” node to deserialize a valid JSON string. This will return a generic JSON value that contains any primitive or complex data type. It is the equivalent of using `JSON.parse(...)` in JavaScript. Now drag a connection from the *Return Value* pin to see the available functions.

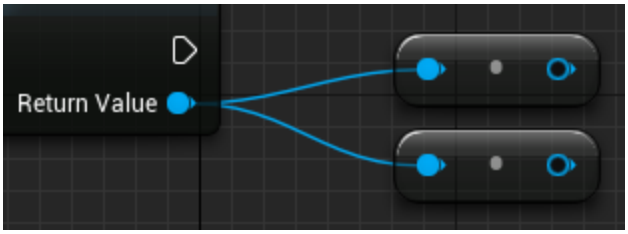


Use the “Get Type” node to check the current type of the JSON value. This will return an enum for the various primitive and complex data types. However if a JSON value is not initialized, not valid JSON, or `undefined` then “Get Type” will return *Invalid*.

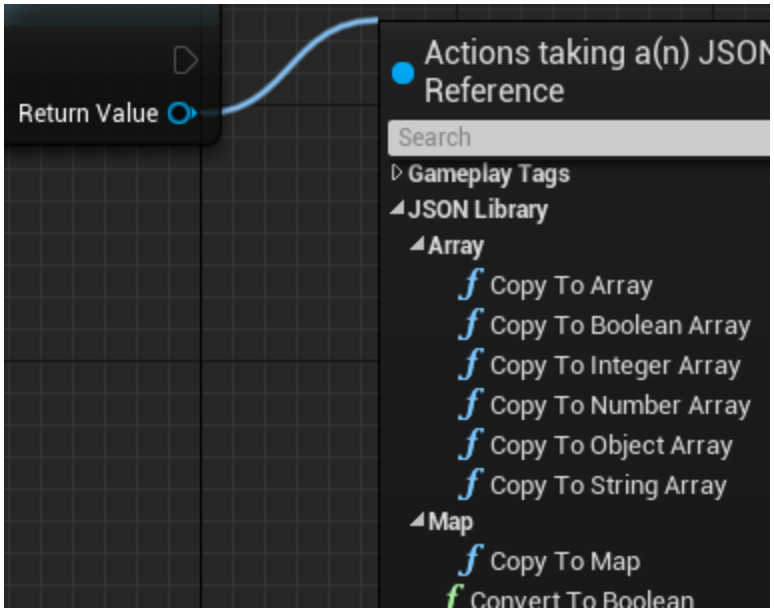


The “Convert To…” nodes will convert JSON values into primitive or complex data types. If the JSON value being converted does not match the conversion type then a default value will be returned. This is `false` for booleans, zero for integers or floats, or an empty string.



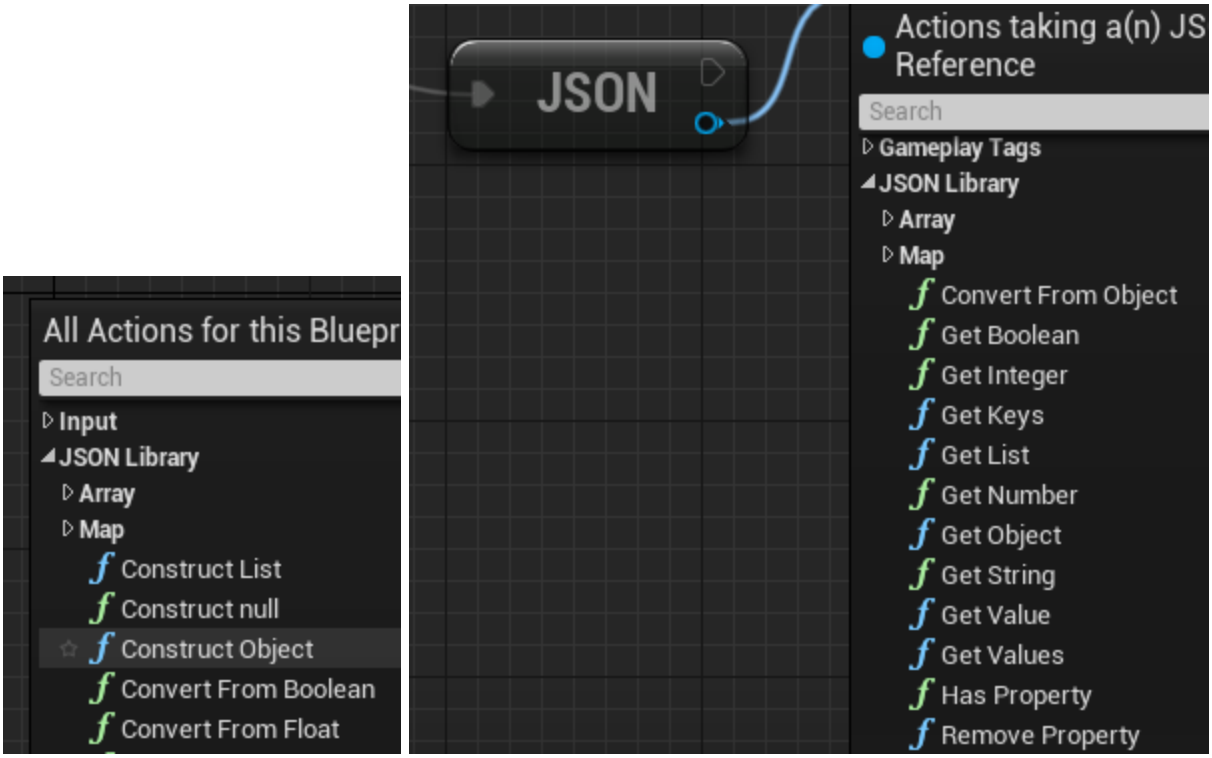


These JSON values can also be copied to native UE4 arrays or maps.

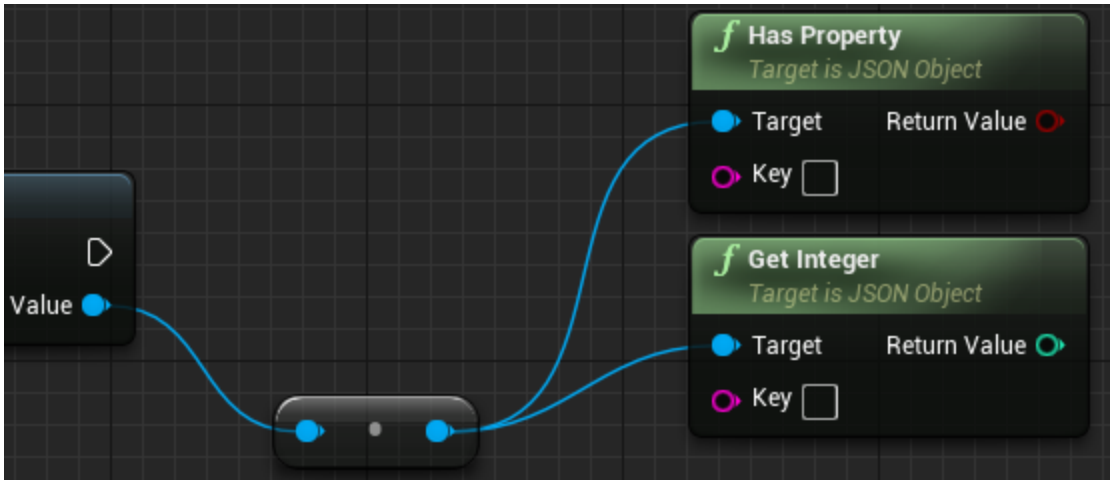


OBJECTS

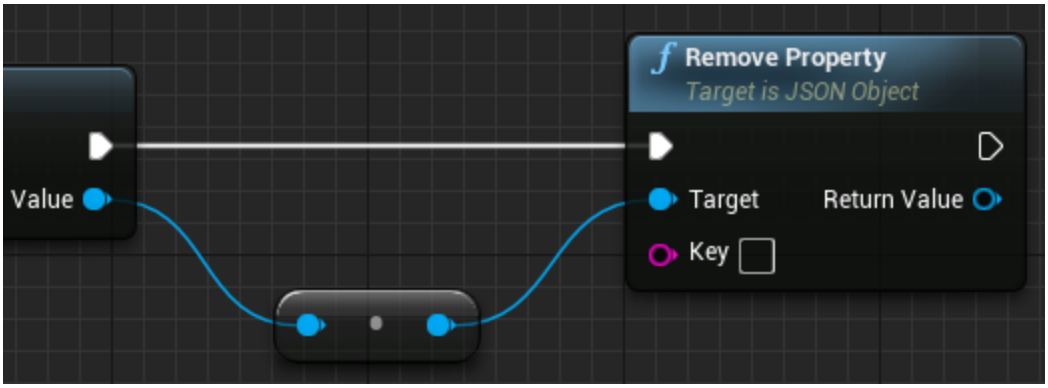
The “Construct Object” node is used to create a JSON object. Start by dragging a connection from the return value to see the available functions.



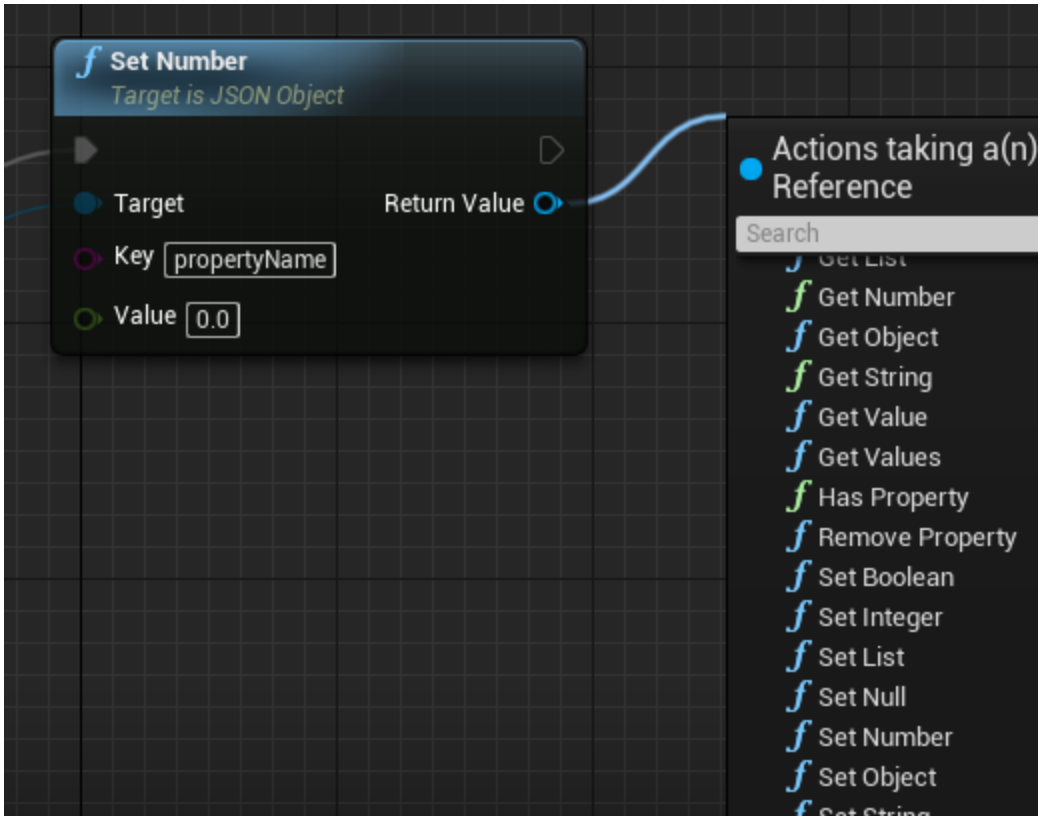
If an existing JSON value is of the type `object` then it can also be converted to a JSON object using the “Convert To Object” helper. The “Has Property” node can be used to check if the object contains an existing key while the “Get …” nodes provide access to the current values.



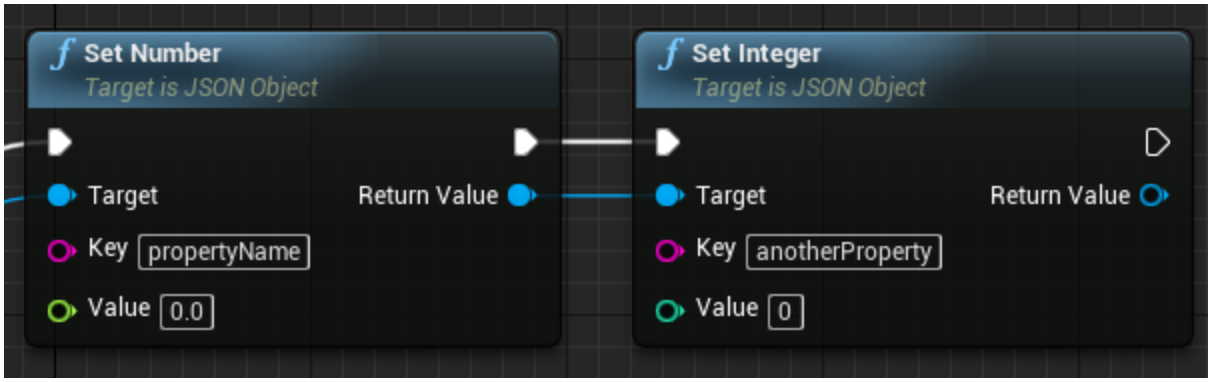
The “Remove Property” node is used to remove an existing key. There are also “Set …” functions for various data types that add a new property to the JSON object.



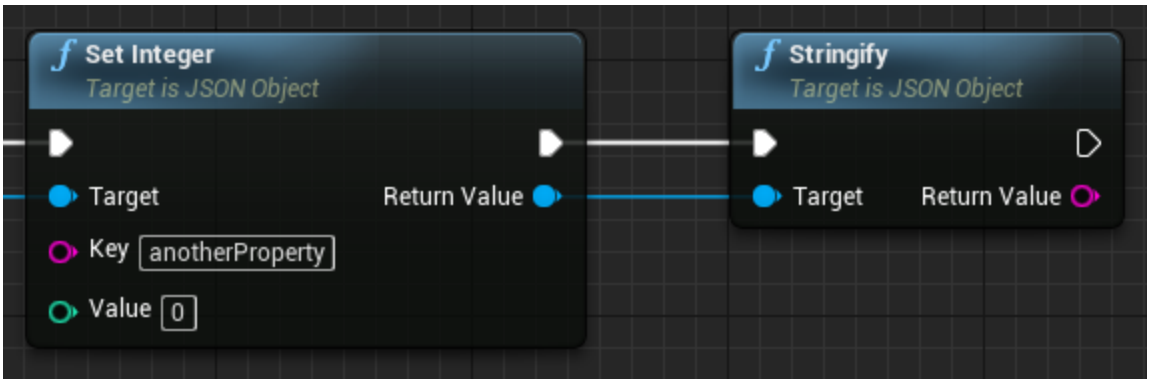
These “Set…” or “Remove Property” nodes also return a reference to the *Target* pin so that they can be daisy chained with more functions.



After dragging a connection from the *Return Value* pin select another “Set …” function to daisy chain them together. This allows for cleaner connections when manipulating JSON objects.



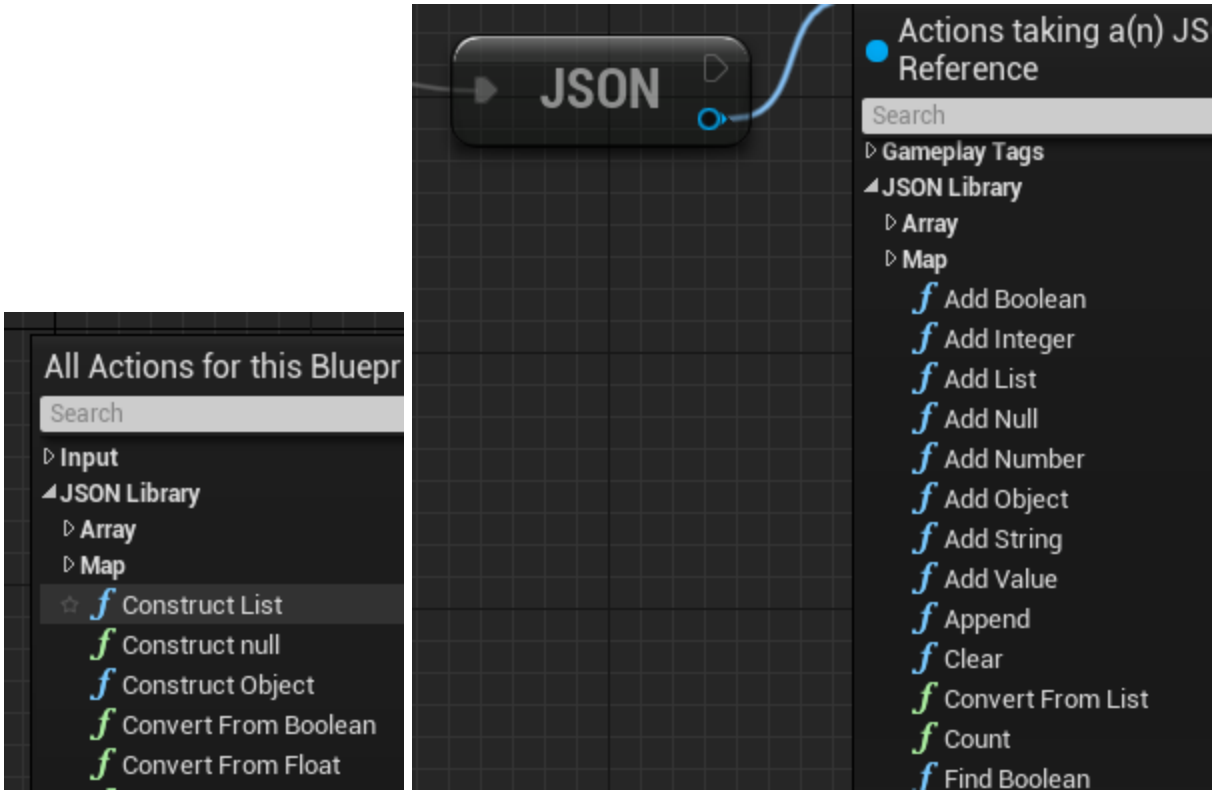
The “Stringify” node is also used to serialize a JSON object into a string. This is the equivalent of using `JSON.stringify(...)` in JavaScript.



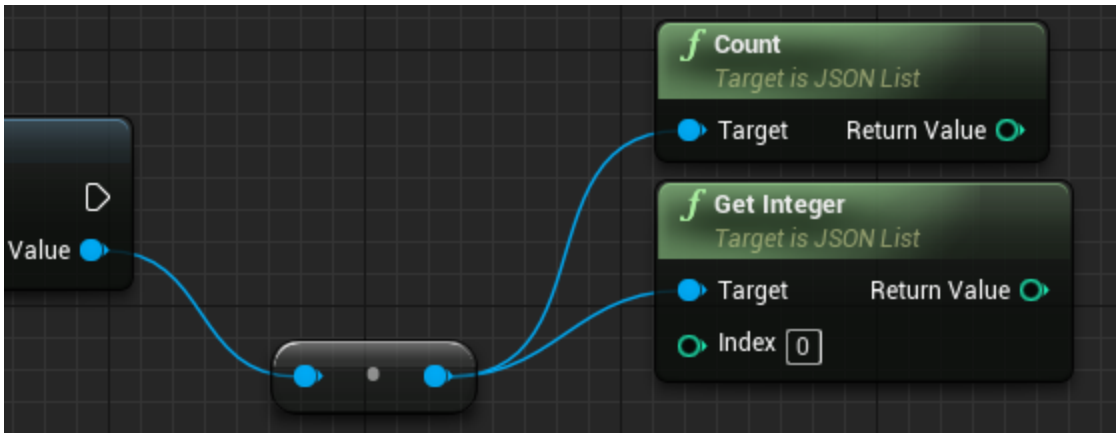
This will always return a string in the format `"{...}"` unless the JSON object is `undefined`.

ARRAYS

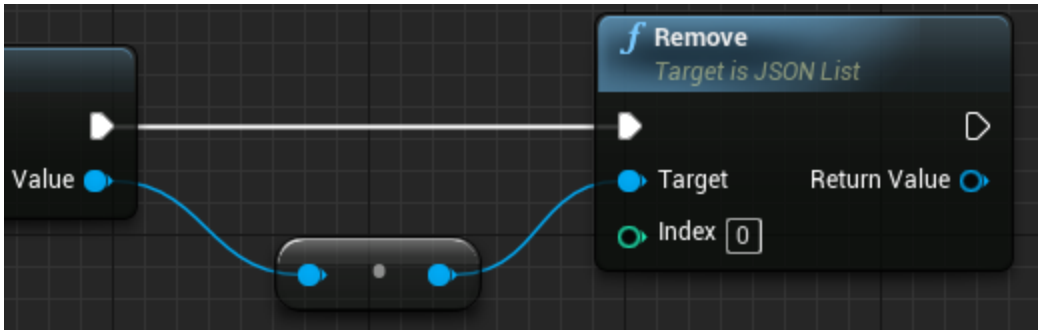
The “Construct List” node is used to create a JSON array. Start by dragging a connection from the return value to see the available functions.



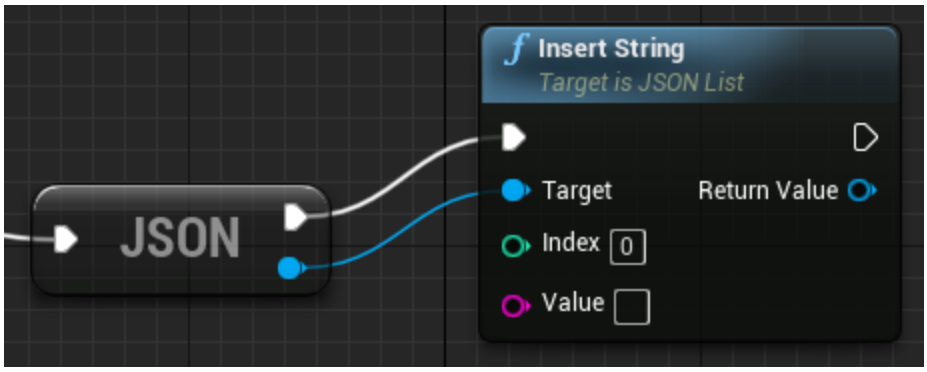
If an existing JSON value is of the type `array` then it can also be converted to a JSON array using the “Convert To List” helper. The “Count” node is used to check the current length of the array while the “Get …” nodes provide access to the values.



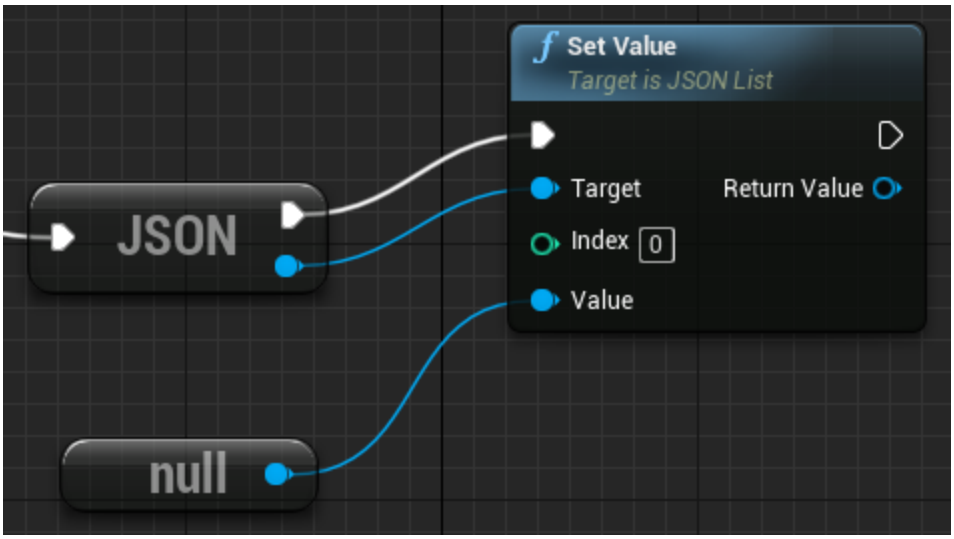
The “Remove” node is used to remove an existing index. Other “Remove …” nodes are provided as well in order to remove a specific value from the JSON array.



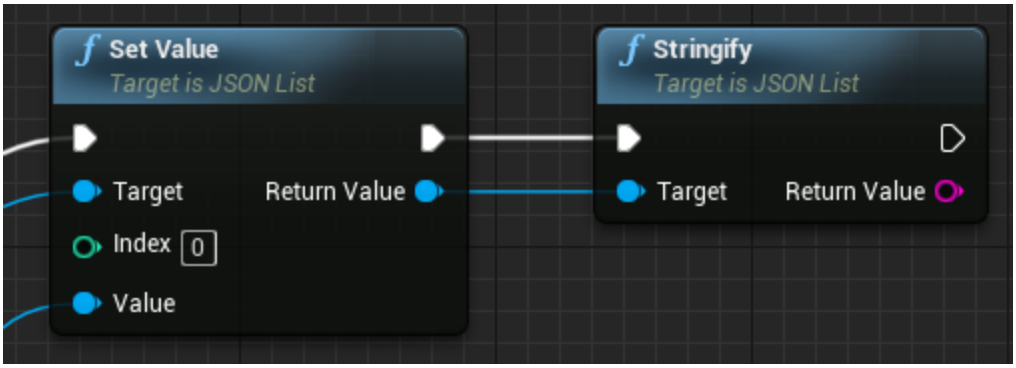
There are “Add …” and “Insert…” functions for various data types that add or insert new elements to the JSON array, respectively.



There are also “Set …” functions for various data types that set an existing element in the JSON array. These nodes return a reference to the *Target* pin as well so they can be daisy chained.



The “Stringify” node is also used to serialize a JSON array into a string. This is the equivalent of using `JSON.stringify(...)` in JavaScript.



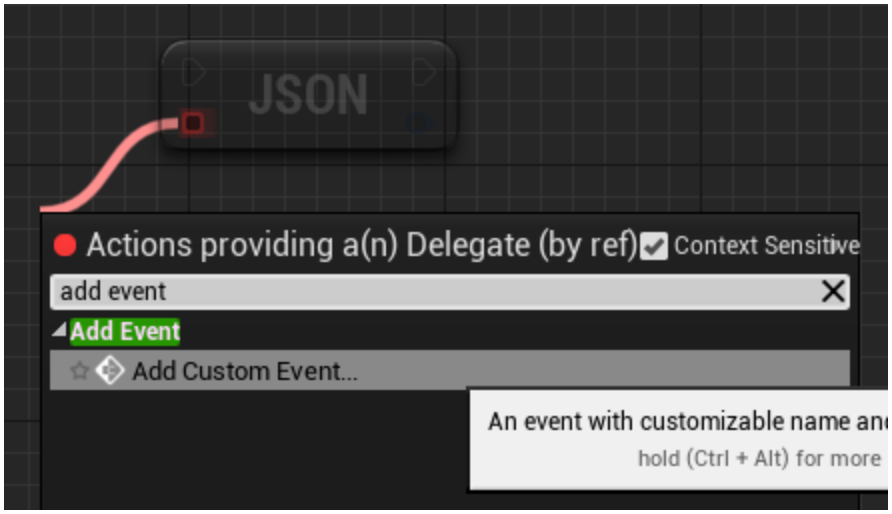
This will always return a string in the format `"[...]"` unless the JSON array is `undefined`.

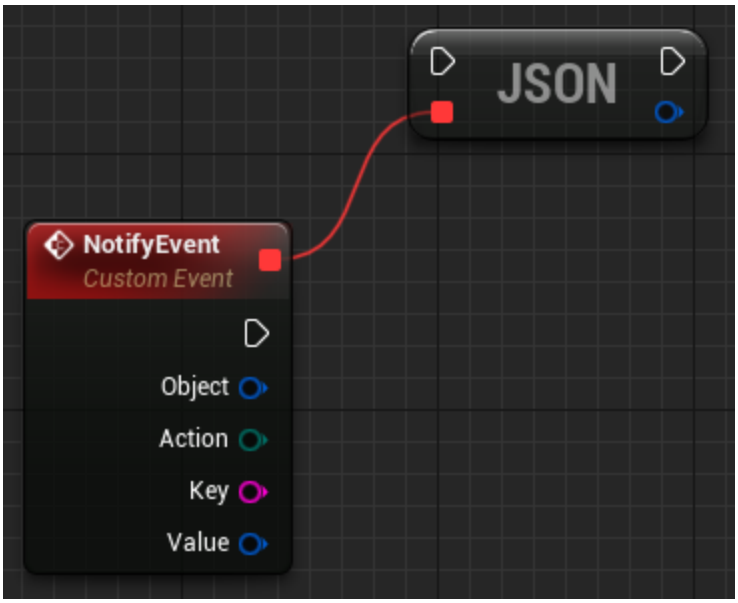
STRUCTURES

The “Construct List” node is used to create a JSON array. Start by dragging a connection from the return value to see the availa in the format `"[...]"` unless the JSON array is `undefined`.

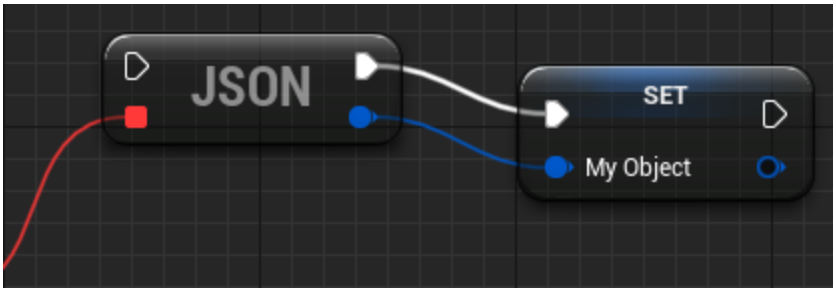
EVENTS

The “Construct Object” and “Construct List” nodes can also be connected to a delegate for change tracking. Start by dragging a connection from the delegate pin and select the following option to add a custom event:

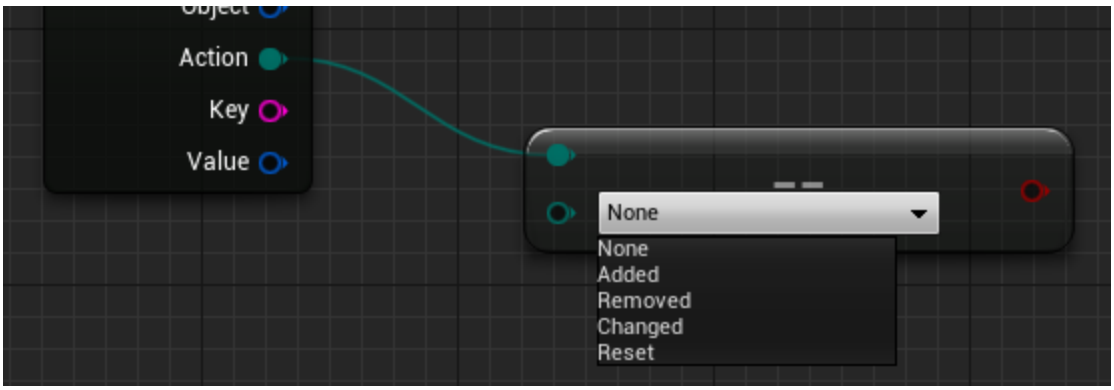




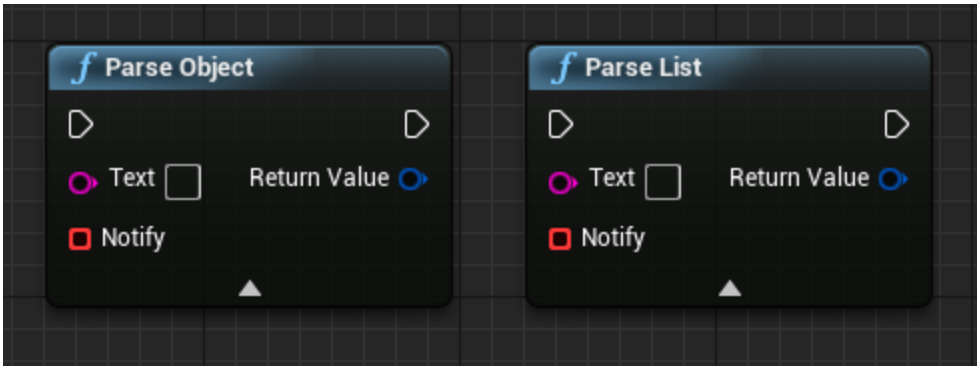
This event is only valid for the current structure and will not trigger if accessed via a parent object or converted into a generic JSON value. Therefore you should store this reference in a variable to access it in different parts of your blueprint, otherwise only the internal JSON data will be referenced when added to another object/list or converted to a value:



The “Action” pin on the event delegate logs whether the object/list had a property/element *added*, *removed*, or *changed*. It also provides the value that is being added or removed. If the object/list was cleared then the *reset* action will be triggered with no key/index or value. Lastly when a property/element is set to the same value the *none* action will be triggered:



These delegates are also available on the “Parse Object” and “Parse List” nodes:



Again note that these delegates will only have a binding to the root object/list and only apply to the current reference. Any subobjects or subarrays that were parsed will not trigger this event.

FORMAT

The “Construct Object” and “Construct List” nodes can also be connected to a delegate for change tracking. Sere parsed will not trigger this event.

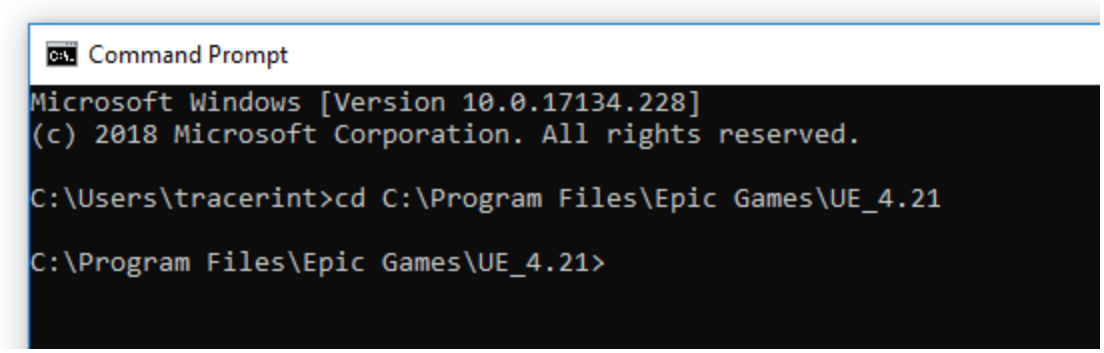
COMPILE

This plugin can be manually compiled for other platforms or engine versions. First open the command prompt (by searching for “cmd” in the start menu) and type the following command:

```
cd "C:\Program Files\Epic Games\UE_4.21"
```

You can copy this command and paste it by right-clicking on the command prompt. Also take note of the **UE_4.21** directory. You need to change this folder to the version that corresponds to the engine version you are using. *If you did not install your engine to the default directory then type the path to your custom installation folder instead.*

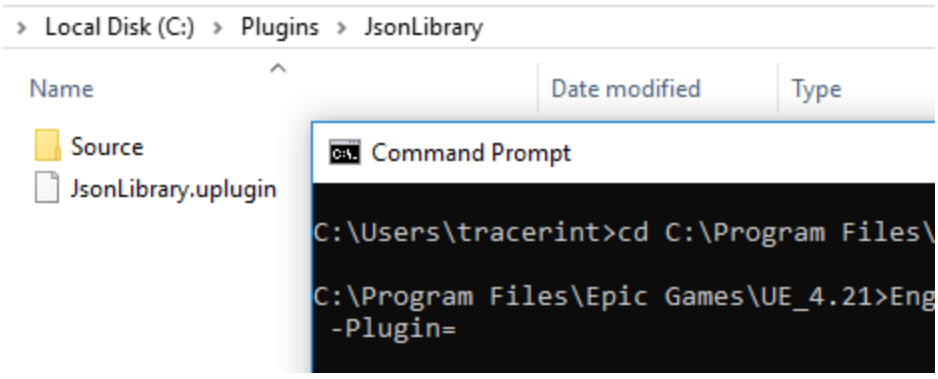
Press ENTER to run the command. You should now see output similar to the following:



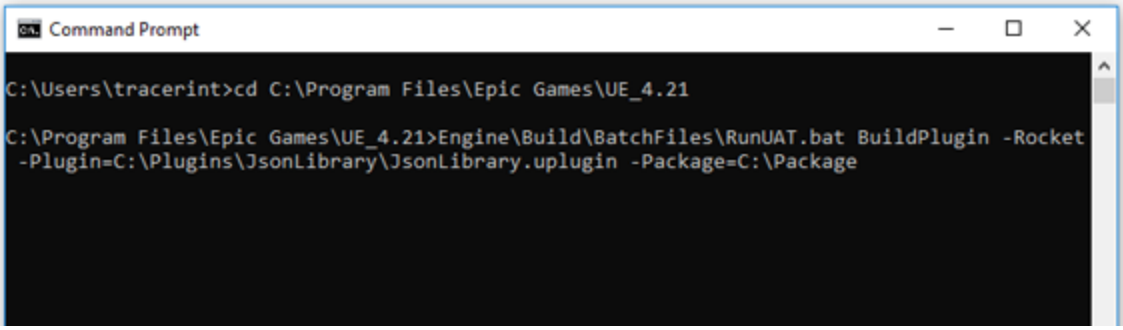
Then type the following command to build the plugin:

```
Engine\Build\BatchFiles\RunUAT.bat BuildPlugin -Rocket -Plugin="..." -Package="..."
```

Replace the first "..." with the path to `JsonLibrary.uplugin` and the last "..." with the path to a temporary “package” folder. You can also drag and drop the `.uplugin` file or your temporary folder directly onto the command prompt and it will automatically type the path:



Make sure these paths are not inside the engine directory or the build will fail. Once you have the full command typed out it should look similar to the following:

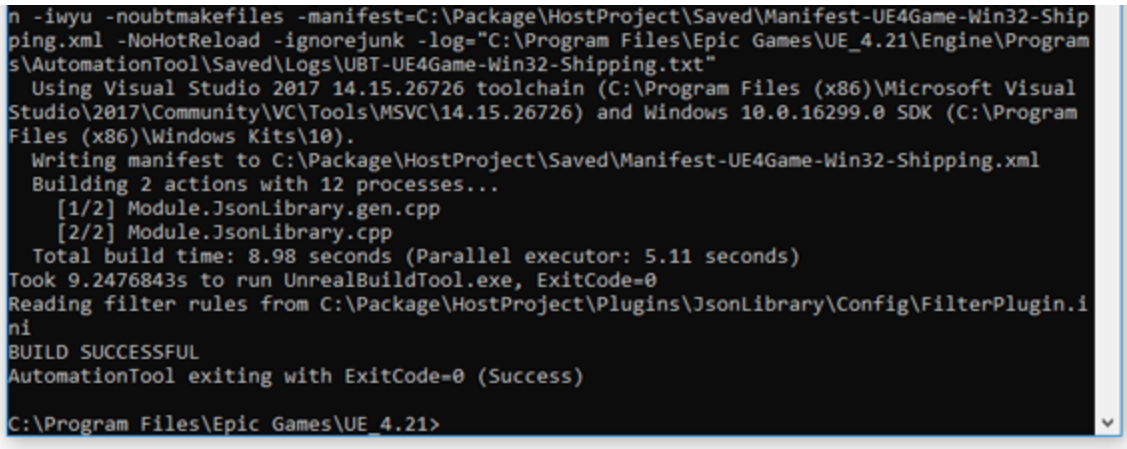


If your machine does not support Mac or Linux builds then you will most likely have to remove the "`Mac`" and "`IOS`" or "`Linux`" platforms from `JsonLibrary.uplugin` before compiling:

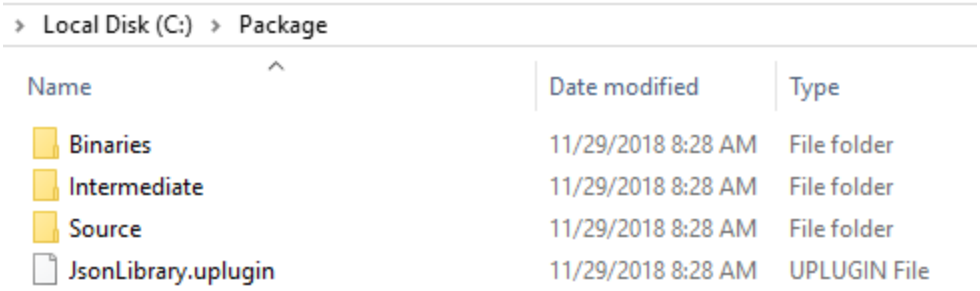
```
        "Name": "JsonLibrary",
        "Type": "Runtime",
        "LoadingPhase": "PreDefault",
        "WhitelistPlatforms": [
            "Win64",
            "Win32",
            "Linux",
            "Android"
        ]
```

Now hit the ENTER key to run the command. If everything was setup correctly you'll see many different versions of the plugin being compiled for various platforms.

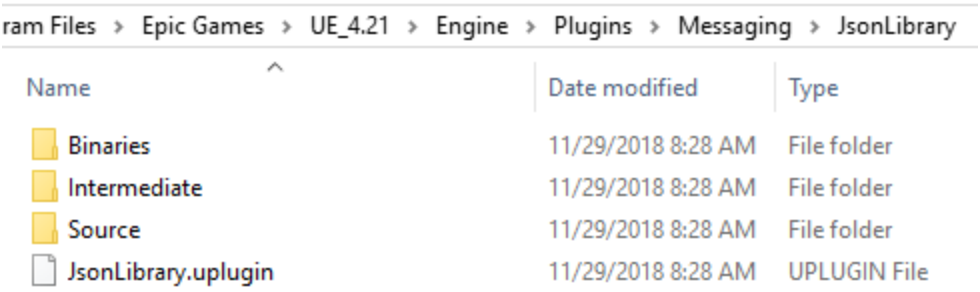
Once the build is complete you should see the "BUILD SUCCESSFUL" message:



Check your temporary “package” folder to ensure it looks similar to the following:



Then copy the files in this temporary folder into your engine installation directory. You must do this beforehand if you are compiling any other plugins that require the JsonLibrary plugin.



You have now successfully compiled the JsonLibrary plugin for your engine version.