Chunity Reference

For the most up-to-date information, please visit
http://chuck.stanford.edu/chunity/.

For the most up-to-date documentation, including code examples, please visit
http://chuck.stanford.edu/chunity/documentation/.

Chunity : Documentation

version: 1.4.x.x (numchucks)

(up): chunity

Welcome to the documentation for Chunity! Below are lists of all the available functions you might want to use in Chunity, as well as a few functions. You can also check out the tutorials.

ChuckMainInstance / ChuckSubInstance

These components have identical APIs. In most cases, you will be sending commands to a ChuckSubInstance, which will send on those commands to whichever ChuckMainInstance it refers to, but you can also send commands directly to a ChuckMainInstance.

Running Code

Here's an example of how to use a ChuckMainInstance / ChuckSubInstance, taken from the tutorials.

```
void OnTriggerEnter( Collider other )
{
    if( other.gameObject.CompareTag( "Pick Up" ) )
    {
        other.gameObject.SetActive( false );
        count = count + 1;
        SetCountText();

        GetComponent<ChuckSubInstance>().RunCode(@"
            SinOsc foo => dac;
            repeat( 5 )
            {
                Math.random2f( 300, 1000 ) => foo.freq;
                100::ms => now;
        ");
    }
}
```

[function]: bool **RunCode** (string **code**);

- *add a new ChucK program to this VM*
- ***code**: the literal text of the program to run*

[function]: bool **RunFile** (string **filename**, bool **fromStreamingAssets** = true);

- *add a new ChucK program to this VM, from filename*
- ***filename**: the location of the file to run*
- ***fromStreamingAssets**: if true, **filename** is relative to the path of the StreamingAssets folder*

[function]: bool **RunFile** (string **filename**, string **colonSeparatedArgs**, bool **fromStreamingAssets** = true);

- *add a new ChucK program to this VM, from filename*
- ***filename**: the location of the file to run*
- ***colonSeparatedArgs**: the arguments to pass to the ChucK file, joined together by colons*
- ***fromStreamingAssets**: if true, **filename** is relative to the path of the StreamingAssets folder*

Global Ints

Here's how it would look to set and get a global int. See also ChuckIntSyncer below for a simpler pattern that doesn't involve writing your own callback.

```
ChuckSubInstance myChuck;
Chuck.IntCallback myIntCallback;
long latestGottenInt;

void Start()
{
    myChuck = GetComponent<ChuckSubInstance>();
    myChuck.RunCode(@"
        global int myGlobalInt;
        // ...
    ");
    // create a callback to use with GetInt in Update()
    myIntCallback = myChuck.CreateGetIntCallback( MyGetIntCallbackFunction );

    // call SetInt
    myChuck.SetInt( "myGlobalInt", 5 );
}

void Update()
{
    // call the callback
    myChuck.GetInt( "myGlobalInt", myIntCallback );

    // do something with the gotten value
    Debug.Log( latestGottenInt );
}

void MyGetIntCallbackFunction( long newValue )
{
    latestGottenInt = newValue;
}
```

[function]: bool **SetInt** (string **variableName**, long **value**);

- *set the value of global int variableName*
- ***variableName**: the name of the global int to set*
- ***value**: the value to set it to. this is a long because ChucK ints are 64 bit integers*

[function]: bool **GetInt** (string **variableName**, Chuck.IntCallback **callback**);

- *eventually call the callback with the value of global int variableName*
- ***variableName**: the name of the global int to get*
- ***callback**: the function to call when we know the value*

[function]: Chuck.IntCallback **CreateGetIntCallback** (Action< long > **callbackFunction**);

- *constructs the callback necessary for GetInt. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when ChucK has the value of your GetInt operation. this is a function that has one argument, a long*

Global Floats

See Global Ints above for an example of how to get and set global variables. See also ChuckFloatSyncer below for a simpler pattern that doesn't involve writing your own callback.

[function]: bool **SetFloat** (string **variableName**, double **value**);

- *set the value of global float variableName*
- ***variableName**: the name of the global float to set*
- ***value**: the value to set it to. this is a double because ChucK floats are 64-bit (i.e. a "double" in C++ and C#)*

[function]: bool **GetFloat** (string **variableName**, Chuck.FloatCallback **callback**);

- *eventually call the callback with the value of global float variableName*
- ***variableName**: the name of the global float to get*
- ***callback**: the function to call when we know the value*

[function]: Chuck.FloatCallback **CreateGetFloatCallback** (Action< double > **callbackFunction**);

- *constructs the callback necessary for GetFloat. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when ChucK has the value of your GetFloat operation. this is a function that has one argument, a double*

Global Strings

See Global Ints above for an example of how to get and set global variables. See also ChuckStringSyncer below for a simpler pattern that doesn't involve writing your own callback.

[function]: bool **SetString** (string **variableName**, string **value**);

- *set the value of global string variableName*
- ***variableName**: the name of the global string to set*
- ***value**: the value to set it to*

[function]: bool **GetString** (string **variableName**, Chuck.StringCallback **callback**);

- *eventually call the callback with the value of global string variableName*
- ***variableName**: the name of the global string to get*
- ***callback**: the function to call when we know the value*

[function]: Chuck.StringCallback **CreateGetStringCallback** (Action< string > **callbackFunction**);

- *constructs the callback necessary for GetString. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when ChucK has the value of your GetString operation. this is a function that has one argument, a string*

Global Events

Here's an example of how to trigger and listen to global Events. See also ChuckEventListener below for a simpler pattern that doesn't involve communicating back from the audio thread to the Update() thread.

```
ChuckSubInstance myChuck;
Chuck.VoidCallback myEventCallback;

void Start()
{
    myChuck = GetComponent<ChuckSubInstance>();
    myChuck.RunCode( @"
        global Event startNotifying;
        global Event notifier;

        startNotifying => now;
        while( true )
        {
            notifier.broadcast();
            250::ms => now;
        }
    " );


    // call BroadcastEvent to trigger startNotifying
    myChuck.BroadcastEvent( "startNotifying" );

    // create a callback to use with StartListeningForChuckEvent
    myEventCallback = myChuck.CreateVoidCallback( MyEventReaction );

    // call my callback every time notifier broadcasts
    myChuck.StartListeningForChuckEvent( "notifier", myEventCallback );
}

// keep track of how many times the Event has
// broadcast on the audio thread
int numTimesEventHappenedOnAudioThread = 0;
void MyEventReaction()
{
    numTimesEventHappenedOnAudioThread++;
}

// use that information on the Update() thread
void Update()
{
    while( numTimesEventHappenedOnAudioThread > 0 )
    {
        numTimesEventHappenedOnAudioThread--;
        // Do something in Unity, e.g....
        transform.Rotate( ... );
    }
}
```

[function]: bool **SignalEvent** (string **variableName**);

- *calls .signal() on global Event variableName (i.e. awake the next listener)*
- ***variableName**: the name of the global Event to signal*

[function]: bool **BroadcastEvent** (string **variableName**);

- *calls .broadcast() on global Event variableName (i.e. awake all listeners)*
- ***variableName**: the name of the global Event to broadcast*

[function]: bool **ListenForChuckEventOnce** (string **variableName**, Chuck.VoidCallback **callback**);

- *calls the callback the next time that global Event variableName signals it*
- ***variableName**: the name of the global Event to listen to*
- ***callback**: the function to call when the global Event signals*

[function]: bool **StartListeningForChuckEvent** (string **variableName**, Chuck.VoidCallback **callback**);

- *calls the callback every time that global Event variableName signals it (until cancelled)*
- ***variableName**: the name of the global Event to listen to*
- ***callback**: the function to call when the global Event signals*

[function]: bool **StopListeningForChuckEvent** (string **variableName**, Chuck.VoidCallback **callback**);

- *stop calling the callback registered to global Event variableName*
- ***variableName**: the name of the global Event to listen to*
- ***callback**: the function to stop calling when the global Event signals*

[function]: Chuck.VoidCallback **CreateVoidCallback** (Action **callbackFunction**);

- *constructs the callback necessary for global Events. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when the ChucK Event signals. this is a function that has no arguments*

Global Int Arrays

See Global Ints above for an example of how to get and set global variables. Note that "associative array" is ChucK's version of a dictionary with string keys mapping to values (see ChucK documentation).

[function]: bool **SetIntArray** (string **variableName**, long[] **values**);

- *set all values at once of global int variableName[]*
- ***variableName**: the name of the global int array to set*
- ***value**: the values to assign to this array. these are longs because ChucK ints are 64 bit integers*

[function]: bool **GetIntArray** (string **variableName**, Chuck.IntArrayCallback **callback**);

- *eventually call the callback with the value of global int variableName[]*
- ***variableName**: the name of the global int array to get*
- ***callback**: the function to call when we know the value*

[function]: Chuck.IntArrayCallback **CreateGetIntArrayCallback** (Action< long[], ulong > **callbackFunction**);

- *constructs the callback necessary for GetIntArray. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when ChucK has the value of your GetIntArray operation. this is a function that has two arguments, a long array (the values) and an unsigned long (the number of values)*

[function]: bool **SetIntArrayValue** (string **variableName**, uint **index**, long **value**);

- *set the value of global int variableName[index]*
- ***variableName**: the name of the global int array to set a value in*
- ***index**: the index of the array at which to set the value*
- ***value**: the value to assign to this index on this array. this is a long because ChucK ints are 64 bit integers*

[function]: bool **GetIntArrayValue** (string **variableName**, uint **index**, Chuck.IntCallback **callback**);

- *eventually call the callback with the value of global int variableName[index]*
- ***variableName**: the name of the global int array to get a value from*
- ***index**: the index of the array at which to get the value*
- ***callback**: the function to call when we know the value*

[function]: bool **SetAssociativeIntArrayValue** (string **variableName**, string **key**, long **value**);

- *set the value of global int variableName[key]*
- ***variableName**: the name of the global int array to set a value in*
- ***key**: the key of the associative array for which to set the value*
- ***value**: the value to assign to this key on this array. this is a long because ChucK ints are 64 bit integers*

[function]: bool **GetAssociativeIntArrayValue** (string **variableName**, string **key**, Chuck.IntCallback **callback**);

- *eventually call the callback with the value of global int variableName[key]*
- ***variableName**: the name of the global int array to get a value from*
- ***key**: the key of the associative array for which to get the value*
- ***callback**: the function to call when we know the value*

Global Float Arrays

See Global Ints above for an example of how to get and set global variables. Note that "associative array" is ChucK's version of a dictionary with string keys mapping to values (see ChucK documentation).

[function]: bool **SetFloatArray** (string **variableName**, double[] **values**);

- *set all values at once of global float variableName[]*
- ***variableName**: the name of the global float array to set*
- ***value**: the values to assign to this array. these are doubles because ChucK floats are 64-bit*

[function]: bool **GetFloatArray** (string **variableName**, Chuck.FloatArrayCallback **callback**);

- *eventually call the callback with the value of global float variableName[]*
- ***variableName**: the name of the global float array to get*
- ***callback**: the function to call when we know the value*

[function]: Chuck.FloatArrayCallback **CreateGetFloatArrayCallback** (Action< double[], ulong > **callbackFunction**);

- *constructs the callback necessary for GetFloatArray. you should store this callback yourself to avoid it being garbage collected*
- ***callbackFunction**: the function to call when ChucK has the value of your GetFloatArray operation. this is a function that has two arguments, a double array (the values) and an unsigned long (the number of values)*

[function]: bool **SetFloatArrayValue** (string **variableName**, uint **index**, double **value**);

- *set the value of global float variableName[index]*
- ***variableName**: the name of the global float array to set a value in*

- ***index**: the index of the array at which to set the value*
- ***value**: the value to assign to this index on this array. this is a double because ChucK floats are 64-bit*

[function]: bool **GetFloatArrayValue** (string **variableName**, uint **index**, Chuck.FloatCallback **callback**);

- *eventually call the callback with the value of global float variableName[index]*
- ***variableName**: the name of the global float array to get a value from*
- ***index**: the index of the array at which to get the value*
- ***callback**: the function to call when we know the value*

[function]: bool **SetAssociativeFloatArrayValue** (string **variableName**, string **key**, double **value**);

- *set the value of global float variableName[key]*
- ***variableName**: the name of the global float array to set a value in*
- ***key**: the key of the associative array for which to set the value*
- ***value**: the value to assign to this key on this array. this is a double because ChucK floats are 64-bit*

[function]: bool **GetAssociativeFloatArrayValue** (string **variableName**, string **key**, Chuck.FloatCallback **callback**);

- *eventually call the callback with the value of global float variableName[key]*
- ***variableName**: the name of the global float array to get a value from*
- ***key**: the key of the associative array for which to get the value*
- ***callback**: the function to call when we know the value*

Unique Variables

You might want many copies running of a script that needs a global variable for communication between ChucK and Unity, but they all should not be referring to the same global variable. In this case, you should determine the name of each script's global variable at runtime.

```
ChuckSubInstance myChuck;
string myGlobalVariableName;

void Start()
{
    myChuck = GetComponent<ChuckSubInstance>();
    myGlobalVariableName = myChuck.GetUniqueVariableName();

    myChuck.RunCode( string.Format( @"
        global int {0};
        // ...
    ", myGlobalVariableName ) );
}
```

[function]: bool **GetUniqueVariableName** (string **prefix** = "v");

- *returns a string which is guaranteed not to be the name of a variable already returned by this* function. use this function to generate names for global variables for scripts that are run many times, since global variables are shared by all scripts running on a VM

- ***prefix**: the variable name will start with this string, and end in a number. if you don't specify a prefix, then "v" is used. choosing a prefix is mostly useful for debugging purposes*

Specific to ChuckMainInstance

A ChuckMainInstance has one setting: the microphone used as input to its ChucK VM.

[value]: string **microphoneIdentifier**

- *this ChuckMainInstance will use the first microphone whose name contains this string. if this is not specified, your system's default microphone is used*

Specific to ChuckSubInstance

A ChuckSubInstance has two parameters for setting itself up. See the initial setup tutorial and the spatialization tutorial.

[value]: ChuckMainInstance **chuckMainInstance**

- *all ChuckSubInstances must have a reference to a ChuckMainInstance, with which they share a VM and a set of global variables. if this is not specified, then Chunity will look for a ChuckMainInstance with a TheChuck component attached. if that also fails, then your ChuckSubInstance will not work*

[value]: bool **spatialize**

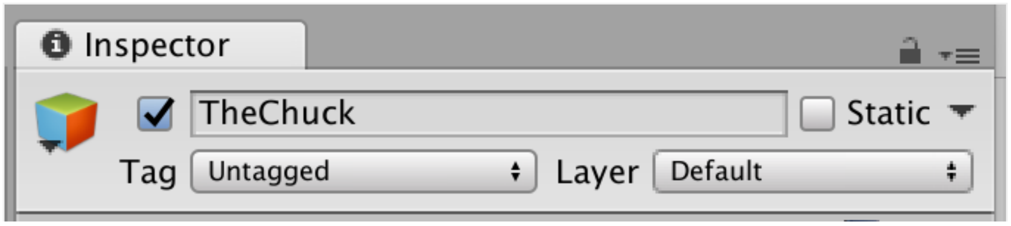
- *if this bool is true, then the ChuckSubInstance will be spatialized to the location of its GameObject. if this bool is false, then the ChuckSubInstance will not be spatialized and will be delivered directly to your audio output*

[function]: void **SetRunning** (bool **r**);

- *set whether the ChuckSubInstance is currently outputting sound*

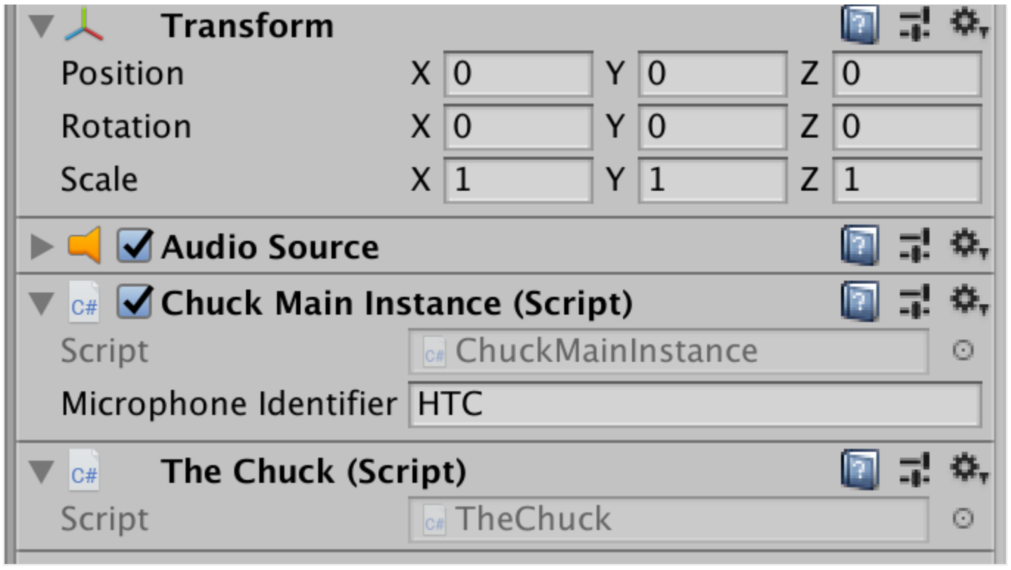
TheChuck

If you add a **TheChuck** component to a game object with a **ChuckMainInstance** component, then you will not need to set the **chuckMainInstance** field of any new **ChuckSubInstances** you spawn; they will instead refer to the game object that has a **TheChuck** attached.


Inspector

✓ TheChuck ☐ Static

Tag Untagged Layer Default



Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

Audio Source

Chuck Main Instance (Script)
Script ChuckMainInstance
Microphone Identifier HTC

The Chuck (Script)
Script TheChuck

Helper Components

These components can be used by adding them to a GameObject at runtime. They are useful for dealing with global ChucK variables without having to write callbacks that are called from the audio thread.

ChuckEventListener

ChuckEventListener is a helper component that makes it easier to write programs like the example in Global Events.

```
ChuckSubInstance myChuck;
ChuckEventListener myEventListener;

void Start()
{
    myChuck = GetComponent<ChuckSubInstance>();
    myChuck.RunCode( @"
        global Event startNotifying;
        global Event notifier;

        startNotifying => now;
        while( true )
        {
            notifier.broadcast();
            250::ms => now;
        }
    " );

    // call BroadcastEvent to trigger startNotifying
    myChuck.BroadcastEvent( "startNotifying" );

    // create an EventListener to call my callback
    myEventListener = gameObject.AddComponent<ChuckEventListener>();

    // call my callback every time notifier broadcasts
    myEventListener.ListenForEvent( myChuck, "notifier", MyEventReaction );
}

// this function will be called from the Update() thread
void MyEventReaction()
{
    // Do something in Unity
    transform.Rotate( ... );
}
```

[function]: void **ListenForEvent** (ChuckSubInstance **chuck**, string **eventToListenFor**, Action **callback**);

- *start checking **chuck** for global Event **eventToListenFor** so that **callback** will be called during Update() whenever **eventToListenFor** broadcasts*

[function]: void **StopListening** ();

- *stops calling the callback that was previously set up with **ListenForEvent** on this component*

ChuckIntSyncer

ChuckIntSyncer is a helper component that makes it easier to write programs like the example in Global Ints.

```
ChuckSubInstance myChuck;
ChuckIntSyncer myIntSyncer;

void Start()
{
    myChuck = GetComponent<ChuckSubInstance>();
    myChuck.RunCode( @"
        global int myGlobalInt;
        // ...
    " );
    // create a callback to use with GetInt in Update()
    myIntSyncer = gameObject.AddComponent<ChuckIntSyncer>();

    // start syncing
    myIntSyncer.SyncInt( myChuck, "myGlobalInt" );
}

void Update()
{
    // get the most recently synced value
    Debug.Log( myIntSyncer.GetCurrentValue() );
}
```

[function]: void **SyncInt** (ChuckSubInstance **chuck**, string **intToSync**);

- *start checking **chuck** for global int **intToSync** so that later, you can get its value whenever you want* [function]: void **StopSyncing** ();

- *stops syncing the int that was previously set up with **SyncInt** on this component*

[function]: int **GetCurrentValue** ();

- *returns the most recently fetched value for your synced int*

[function]: void **SetNewValue** (int **newValue**);

- *sets the value of the currently syncing int to **newValue***

ChuckFloatSyncer

See ChuckIntSyncer above for an example of how to use this component.

[function]: void **SyncFloat** (ChuckSubInstance **chuck**, string **floatToSync**);

- *start checking **chuck** for global float **floatToSync** so that later, you can get its value whenever you want*

[function]: void **StopSyncing** ();

- *stops syncing the float that was previously set up with **SyncFloat** on this component*

[function]: float **GetCurrentValue** ();

- *returns the most recently fetched value for your synced float*

[function]: void **SetNewValue** (float **newValue**);

- *sets the value of the currently syncing float to **newValue***

ChuckStringSyncer

See ChuckIntSyncer above for an example of how to use this component.

[function]: void **SyncString** (ChuckSubInstance **chuck**, string **stringToSync**);

- *start checking **chuck** for global string **stringToSync** so that later, you can get its value whenever you want*

[function]: void **StopSyncing** ();

- *stops syncing the string that was previously set up with **SyncString** on this component* [function]: string **GetCurrentValue** ();

- *returns the most recently fetched value for your synced string*

[function]: void **SetNewValue** (string **newValue**);

- *sets the value of the currently syncing string to **newValue***

(up): chunity

chuck | soundlab | ccrma