Welcome to the documentation of the Kinzart Automatic Updating Controller, or the KZK AUC for short. This modern marvel will be used for updating all future Kinzart avatars without the requirement of replacing parts or scripts to users who have already installed modifications to their avatar. The AUC is not a complete substitute, as older users can still transfer the parts or scripts themselves should they choose. The AUC is simply an easier and faster means of doing this task.

Performing updates is not the AUC's only function. The AUC can also install scripts to the avatar as well, meaning mod makers can make use of the AUC to make self-installing modifications for all future Kinzart avatars.

It should be noted the part must have an AUC compliant script in it first in order to make use of this feature. Possible cases in why an AUC compliant script may not be in a particular part of an avatar may be due to no major functions for the part (e.g., fore legs), the simplicity of a part (e.g., ears and tails), or due to self-removable, i.e. from the AIO Strip Down feature or manual removal of the script. This should be kept in mind when making use of the AUC as a self-installer, as it isn't always perfect. Some example parts that almost always will have an AUC compliant script in it would include the eyes, head, and paws.

If anything, the AUC can also be used to also identify what avatar the user is wearing. In this case, even if the AIO Strip Down is used, some AUC compliant scripts will still remain and will respond to AUC commands. This can be helpful for projects that require identification of not only the user but what the user may be. It should be noted that some modifications can change what the user is trying to be, so this is not the absolute best way to determine what the user is. In addition, the user may be wearing multiple AUC compliant parts from various other avatars, meaning you will get many possibilities.

How you, the user, make use of the AUC is the ultimate deciding factor in how you want to go forward with your own future projects. With the Kinzart Automatic Updating Controller, we are helping unlock a path to better allowing ourselves to help our customers while also allowing modification makers have more tools at their disposal.

We hope this document will provide you with answers to your questions regarding the AUC and how to operate it. We also would like to once again thank you for your purchase with Kinzart Productions and we look forward to doing business with you in the future.

-Sincerely,

The Kinzart Staff

Command/Reply Reference

Commands should be given on the AUC channel, which defaults to 23. The avatar parts will only listen to objects owned by the user. This means trying to send commands to an avatar not owned by the owner of the object with the script will not execute or respond.

Command	Used for…	Response
AUC:|:ProductReq	Identifying what avatar the user is wearing	AUC:|:ProductIDRes:|:[ProductName] Ex: AUC:|:ProductIDRes:|:HYENA
AUC:|:VersionReq:|:[ProductName] Ex: AUC:|:VersionReq:|:HYENA	Identifying what version the avatar is that the user is wearing.	AUC:|:VersionRes:|:<VersionInfo> Ex: AUC:|:VersionRes:|:<1,1,7,2>
AUC:|:GetKeyReq:|:[ProductName:|:[PartName] Ex: AUC:|:GetKeyReq:|:HYENA:|:HEAD	Obtaining the UUID of the object of the requested part of the avatar the user is wearing.	AUC:|:GetKeyRes:|:[ProductName]:|: [PartName]:|:[UUID]:|:[ScriptName] Ex: AUC:|:GetKeyRes:|:HYENA:|:HEAD: [UUID-here]:|:Expressions
AUC:|:IdentityReq:|:[ProductName] :|: [PartName] :|:[ScriptName] Ex: AUC:|:IdentityReq:|:HYENA:|: HEAD:|:Expressions	Receiving the actual physical name of the script with the same 'script name' as the request from the avatar the user is wearing	AUC:|:IdentityRes:|:[ProductName] :|: [PartName] :|:[ScriptName] :|: [PhysicalScriptName] Ex: AUC: :|:IdentityRes :|:HYENA :|:HEAD :|: Expressions :|: KZK AIO Avatar Script v1.7b[Expressions]
AUC :|:SetRPinReq :|:[ProductName]:|:PIN]	Setting the remote script load pin on all parts relating to a specific avatar the user is wearing.	AUC:|:SetRPinRes:|:[ProductName] Ex: AUC:|:SetRPinRes:|:HYENA
AUC :|:TermReq :|:[ProductName] :|: [PartName] :|:[ScriptName] :|: [PhysicalScriptName] Ex: AUC: :|:TermReq :|:HYENA :|:HEAD :|: Expressions :|: KZK AIO Avatar Script v1.7b[Expressions]	TERMINATING a script from an avatar the user is wearing. **[WARNING] THIS CANNOT BE UNDONE!!! BE CAREFUL WHEN USING THIS!!! MALICIOUS USE WILL RESULT IN BLACKLISTING FROM ALL KINZART SERVICES!!!**	(none)

Additional Information
Version Information = <Major, Minor, Revision, Build>
Script Name refers to the AUC ID of the script itself. This typically relates to the purpose of the script. This does NOT refer to its physical name! (See 'IdentityReq' for how to get the physical script name)
If using lists, the common separator is ":|:"
At this time, all AUC compliant scripts will respond if and only if they match the conditions to send a response. This means you will receive many response to "ProductReq," and fewer for "VersionReq."
The Pin for SetRPinReq refers to the pin to be used for llRemoteLoadScriptPin. The response means it was set.
Once finished with the AUC for installing scripts, be sure to issue SetRPinReq again to set the pin to 0!!!

Example Scripts

Get a product name

```
default {
    state_entry() {
        //Set up a listening channel for incoming AUC commands
        llListen(23, "", "", "");
        //Set our 'status' to waiting
        llSetText("...Waiting...",<1,1,1>,1);
        //Dramatic Pause
        llSleep(2);
        //Now send a request on the product name
        llSay(23,"AUC:|:ProductReq");
    }
    listen(integer c, string w, key k, string msg) {
        //Check to see if the speaker is an object we own
        if(llGetOwner()!=llGetOwnerKey(k)) return;
        //Break things up
        list lTemp = llParseString2List(msg,[":|:"],[]);
        //See if it's our response!
        if(llList2String(lTemp,0)=="AUC" &&
          llList2String(lTemp,1)=="ProductIDRes") {
            //Show our results!
            llSetText("You are wearing the "+
              llList2String(lTemp,2)+"!",<0,1,0>,1);
        }
    }
    touch_start(integer duh) {
        //If touched, reset the fun!
        llResetScript();
    }
}
```

Get a list of what products the user is wearing

```
//We will use this to keep track of what avatars
//we already know about
list lAvatars;

default {
    state_entry() {
        //Set up a listening channel for incoming AUC commands
        llListen(23, "", "", "");
        //Set our 'status' to waiting
        llSetText("...Waiting...",<1,1,1>,1);
        //Dramatic Pause
        llSleep(2);
        //Now send a request on the product name
```

```
        llSay(23,"AUC:|:ProductReq");
    }
    listen(integer c, string w, key k, string msg) {
        //Check to see if the speaker is an object we own
        if(llGetOwner()!=llGetOwnerKey(k)) return;
        //Break things up
        list lTemp = llParseString2List(msg,[":|:"],[]);
        //See if it's our response!
        if(llList2String(lTemp,0)=="AUC" &&
         llList2String(lTemp,1)=="ProductIDRes") {
            //First see if we already know about this avatar
            if(llListFindList(lAvatars,[llList2String(lTemp,2)])==-1) {
                //If we don't know about this one, add it to the list
                lAvatars+=[llList2String(lTemp,2)];
            }
            //Show our results!
            llSetText("You are wearing the following:\n"+
            "---------------------------------\n"+
             llDumpList2String(lAvatars,"\n"),<0,1,0>,1);
        }
    }
    touch_start(integer duh) {
        //If touched, reset the fun!
        llResetScript();
    }
}
```

Documentation History

Date Released	Current AUC\AIO Versions		Notes
April 22, 2012	1.0.1	1.7c	➢ Initial Release of documentation