Creating Custom Content and Mods for The Sims™ 4

Part 1 – Intro & Create A Sim

Version 1.0 (8/31/2014)

Introduction

From the launch of the first installment of The Sims™, Maxis has strived to support the creativity of our community. We know that for many of you, the creation of Custom Content or Mods is an important part of your game experience. While it's important to remember that Maxis cannot officially support any Custom Content or Mods, we have taken some steps to make it easier for you to continue to create amazing stuff and enjoy The Sims that way you want.

Specifically, we've incorporated some features that will make it possible for you to add, manage or remove Custom Content and Mods without modifying your core game data. We've also taken steps to lessen the risk that your Custom Content will be affected when we enhance or update the game.

This document provides technical information around those features and provides best practices to create Custom Content that is more compatible with The Sims 4 game.

Reminders

- The terms and conditions set out in The Sims 4 EULA and EA's Terms of Service are applicable to the use of The Sims 4 product and services and all materials distributed by Maxis that relate to creating Custom Content and Mods for The Sims 4.
- Maxis does not pre-screen, endorse or specifically support any particular Custom Content or Mod. Players should use Custom Content or Mods with caution and understand that there may be risk.
- Maxis expects the community to respect the rights of others when creating Custom Content or Mods and ensure they do not infringe the intellectual property of Maxis or others or contain inappropriate or harmful material.
- This document does not fully specify all data formats or contain complete information required to make Custom Content or Mods but is intended to provide a foundation on which to build.
- The information in this document may be updated from time-to-time to reflect updates and new releases in The Sims 4 line of products.

Installing Custom Content

Custom Content should be installed in the user documents directory, as follows:

The Sims 4 Create A Sim Demo

Windows XP:

```
C:\Documents and Settings\<user>\My Documents\Electronic Arts\The Sims 4 Create A Sim Demo\Mods
```

Windows Vista, Windows 7 or Windows 8:

```
C:\Users\<user>\Documents\Electronic Arts\The Sims 4 Create A Sim Demo\Mods
```

The Sims 4

Windows XP:

```
C:\Documents and Settings\<user>\My Documents\Electronic Arts\The Sims 4\Mods
```

Windows Vista, Windows 7 or Windows 8:

```
C:\Users\<user>\Documents\Electronic Arts\The Sims 4\Mods
```

Note: This directory location will vary depending on your installed language so non-English installation locations will differ.

When the application starts it will create this directory if it does not already exits.

You can disable all Mods and Custom Content by running the game with the command line "-ignoremods". Command line options can be set in Origin by right clicking on The Sims 4 tile and selecting "Game Properties".

The Resource.cfg File

Inside of the "Mods" directory, the game will create a file called "Resource.cfg". This file describes the locations and file types that will be loaded by the application out of this directory. The Sims 4 uses .package files (described in more detail below) for its content. The default contents of the Resource.cfg file are:

```
Priority 500
PackedFile *.package
PackedFile *\*.package
```

This tells the application to load .package files out of the "Mods" directory, or any subdirectory. The "priority 500" line tells the application to treat the content as high priority - which effectively means that it overrides any Maxis authored content in cases where the "Mods" directory contains records with the same resource key as Maxis authored content. More information on resource keys below.

This Resource.cfg file can be modified by the user and it will not be replaced by the application. However it is recommended that Custom Content or Mods do not require users to change this file in order to simplify the setup process for users.

Important Note about the Install Directory

It is strongly recommended that Custom Content and Mods do not require the user to modify any files under the Windows "Program Files" directory, which is where the application is installed. This is because modifying these files may interfere with game updates, and make it difficult for users to diagnose problems introduced by such changes.

Note that if the game files under the Program Files directory are believed to be corrupted or modified, you can run "Repair Game" in Origin which will re-download the files. (Warning: this process can be time consuming.)

Data Formats

A number of .bt files have been provided along with the document. These files describe some of our key data formats and can be loaded by "010 Editor" by SweetScape software (http://www.sweetscape.com/),

which is a binary data viewer that is well suited to reverse engineering data formats. We encourage you to support this software. However the supplied .bt files are easily readable in a text file editor as well.

Package Files

The application uses .package files to bundle its data. Package files are conceptually similar to a zip file, it is a file that contain a collection of other files. Please see the supplied file **PackageTemplate.bt** which describes the format of package files. Custom Content should be distributed as .package files, which users can then drop into their "Mods" directory to install.

Unlike a zip file, which uses a human readable text name for the records it contains, the records within a .package file are identified by "resource key".

Resource Keys

Resource keys are a fundamental concept in how the application loads data, they are what is used to identify data within package files.

Resource keys are 64 bits and are constructed as follows:

32 Bits: Type
32 Bits: Group
64 Bits: Instance

Where:

Type: Is the file type. You can think of this as mapping to a file extension. Examples of resource types would be: textures, geometry, etc.

Group: Each resource type may choose to use the group however it likes.

Instance: The instance is generally a hash of the source resource name. We rely on the hashing algorithm to generate unique instances to avoid conflicts between resources.

All three of these fields when combined uniquely identify a resource. If a piece of Custom Content or Mod is meant to override Maxis created content, then the resource key should be the same as the content it is overriding. However a more common case will be the addition of new content, in which case it is important resource keys are unique.

There are a number of best practices we recommend for resource key generation to ensure that content does not collide with Maxis authored content, or collide with content generated by other Custom Content creators.

Resource Key Best Practices

In order to avoid conflicts with Maxis created content and other user created content, the following best practices for resource key generation are strongly recommended:

- **Group:**
  - **Set the top bit of the group to 1 for all resources.**
  - Note that it is possible that some resource may not support this (there are many different types of resources in the game, and not all have been tested). If you find a resource that

does not appear to allow setting the top bit of the group to 1 please post it in the Maxis modding forums at http://forums.thesims.com/ .

- **Instance:**
  - ALWAYS use a hash for the instance of Custom Content or Mod records. You will notice that some records authored by Maxis use an incrementing number for the instance (for example, see resource type 0x034aeecb) - **Custom Content should not do this as it will lead to collisions**. See "Catalog Instances" section below for more information.
  - When generating the hash, it is recommended that it is constructed from a string that includes as much data as possible that is unique to the user generating the content. For example, including the author's name and the name of the Mod. This reduces the probability of collisions between content generated by different users.
  - FNV-1 hash algorithm is recommended.
  - Furthermore, **the top bit of the instance should be set to 1 to flag the content as Custom Content**. Just generate the 64 bit hash and then force the top bit to 1.

Create A Sim Catalog Instances

"Catalog Instances" are records which are used to populate the game catalog with items. The application will scan .package files for these resources and add them all to the catalog presented to the user. Since Custom Content will generally want to add new items to the catalog, these are important resources to understand.

For the Create A Sim ("CAS") catalog, these are resources are of type 0x034aeecb. See the provided .bt file **PartDataTemplate.bt** which specifies the format of these records.

As mentioned above, Catalog instances authored by Maxis use an incrementing number and not a hash. Custom Content should use a hash as described above.

Note that this record contains a list of resource keys that make up the item. If the Custom Content or Mod is a variation of Maxis authored content, where appropriate this record can point to the data in Maxis authored data packages and only override the records that are necessary to achieve the customization of the content. This will reduce the size of the Custom Content or Mods package.

Part 2 – Gameplay: Objects, Interactions and Tuning

Overview

The Sims 4 is very data-driven; you can create new objects, interactions, traits, skills, careers, aspirations, and many more features entirely using data Mods. This section of the document will discuss tuning files, how they are structured, and how they should be packaged, as well as a few of the other resource types necessary to add custom game play features to The Sims 4. We also are releasing the tuning description files for The Sims 4 as a separate package.

Tuning files

The game loads tuning files in three different ways: From a "combined" tuning resource, from individual XML resources, and from binary "simdata" resources. The combined tuning resource contains Maxis tuning, and should not generally be overridden by Mods. This resource is compressed to speed up load times and cannot be partially overridden, so if two Mods update it they cannot be used simultaneously – and future Maxis updates will almost certainly break the Mod.

Instead, Mods should add or override loose XML tuning and simdata binary tuning. The XML tuning are loaded by the simulation, and the simdata resources are loaded by the UI and renderer; some features use only XML tuning, some use only simdata resources, and some features use both. The Sims 4 asset packages do not include any loose XML tuning, but this document will explain how to construct them so they can be used in Mods.

XML tuning has a variety of resource types; the following section will describe how to determine the correct type. Simdata resources all use type **0x545ac67a**; these are discussed in more detail later.

Tuning descriptions

A tuning description file (or tdesc) documents the structure of the tuning files that are loaded by the game. Tuning description files are not themselves loaded by the game, but if you want to create tuning files it is essential to understand tuning descriptions.

A (partial) example tuning description follows on the next page. The file has been broken up onto separate lines so that each line can be documented separately; the actual tdescs are more compact.

Line 1: Tdesc files are XML, so this is just a standard XML header.
Line 2: There are two types of tdesc files, "Instance" and "Module" tuning. Module tuning defines the values of global variables, while Instance tuning defines new features. Most Mods will add or update Instance tuning, but it's also possible to override Module tuning. Note: It is not possible to partially override tuning, so a Mod that overrides a Module tuning will need to set all values.
Line 3: This is the Python module associated with this tdesc.
Line 4: If use_guid_for_reference is True, tuning resources created by Maxis have sequential IDs. If False, resources use hash IDs. All Mods should use 64-bit FNV-1 hash IDs to prevent collisions.
Line 5: If instance_subclasses_only is True, this tdesc cannot be used directly to create tuned instances. Instead, instances must be created from a derived tdesc.
Line 6: This defines the resource type of instances created using this tdesc. In particular, the resource type is the 32-bit FNV-1 hash of this instance type. As an example, the resource type of "topic" is **0x738e6c56**. More on FNV-1: http://www.isthe.com/chongo/tech/comp/fnv/index.html
Line 7: A string documenting this tuning instance. Descriptions exist on most tunables; these are intended to explain how the tunable type or field should be used.
Line 8: The Python class associated with this tuning instance.
Line 9: A relative file path, you can ignore this for Mods.
Line 10: The remainder of the tdesc documents the individual tuning fields that can appear in a tuning file of this type. A "TunableVariant" is a special type of tuning field that can take on multiple different types of values. Specifically, for any instance only one of the nested tunables can be selected and assigned values; the others are ignored.
Line 11: The "name" of a tunable defines the field that will be set in Python with the value of this tunable. It also is the key that must be used to assign a value to this field in a tuning file.
Line 12: This is the Python class associated with this tuning field. This document will not cover the different classes used by tunables, but many of them should be self-evident. Knowledge of the tuning class is not necessary to create a tuning file, but a tool for creating tuning might use the class to determine which UI element to use to modify tuning values.
Line 13: The underlying type of the tuning field. This is most relevant for "Tunable" fields, for TunableVariant it is irrelevant.
Line 14: The default value of this tunable. For TunableVariant, this defines which sub-field is selected by default – in this case, it is the "disabled" variant. Tuning files generally should not include fields

with default values; omitting them will lead to tuning that is smaller, loads faster, and is more robust if default values change in future game updates. The only reason to explicitly set a default value is if it is important for the value to not change in the future, even if the default changes.

Line 15: This is a user-friendly name for this field. A tool for editing tuning values should display this instead of the "name". It is not used in tuning files.

Line 16: The "filter" attribute can be used to selectively hide or show tuning fields in a tool for editing tuning. It is not used in tuning files.

Line 17: This is an organizational category for this tuning field. It can be used in a tuning tool to categorize or group related fields. It is not used in tuning files.

Line 18: This is a descriptive string that provides additional documentation for this field. It can be displayed in a tuning tool as help text. It is not used in tuning files.

Line 19: A "Tunable" is a basic tuning field. The valid values for a tunable are defined by the "class" and "type" fields.

Line 21: The "TunableExistance" class means that this tunable doesn't have values. It either exists or does not exist. As such, it is really only useful inside TunableVariant tunables, as it is used here.

Line 23: A "TunableTuple" defines a structure containing multiple nested tunables. Each sub-tunable is assigned its own name, and can be accessed via the enclosing tunable tuple.

Line 30: This is a basic "Tunable" that can actually have values.

Line 33: This means that the values of this tunable must be integers.

Line 34: The default value for this field (if not specified in a tuning file) will be one.

Line 38: The "tuning_state" attribute is a hint to a tuning tool about how important it is to set this field. A value of "NeedsTuning" means that someone creating a new tunable should probably look at this field and decide whether to set a value.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<Instance
        module="topics.topic"
        use_guid_for_reference="True"
        instance_subclasses_only="False"
        instance_type="topic"
        description="A topic element."
        class="Topic"
        path="Topics\Descriptions">
    <TunableVariant
            name="guaranteed_content"
            class="OptionalTunable"
            type="None"
            default="disabled"
            display="Guaranteed Content"
            filter="0"
            group="General"
            description="If enabled, will force content set generation to add options for&#xA;this topic.">
        <Tunable
                name="disabled"
                class="TunableExistance"
                display="Disabled" />
        <TunableTuple
                name="enabled"
                class="TunableTuple"
                display="Enabled"
                filter="0"
                group="General"
                description="If enabled, will force content set generation to add options for&#xA;this topic.">
            <Tunable
                    name="count"
                    class="Tunable"
                    type="int"
                    default="1"
                    display="Count"
                    filter="0"
                    group="General"
                    tuning_state="NeedsTuning"
                    description="The number of options to force into the content set." />
            <Tunable
                    name="priority"
                    class="Tunable"
                    type="int"
                    default="0"
                    display="Priority"
                    filter="0"
                    group="General"
                    tuning_state="NeedsTuning"
                    description="The priority of this Topic vs. other Topics. Ties are randomized." />
        </TunableTuple>
    </TunableVariant>
    <Tunable name="is_timed_relevancy" class="Tunable" type="bool"
            default="False" display="Is Timed Relevancy" filter="0" group="General"
            description="If set, relevancy value is treated as number of minutes until topic&#xA;is removed." />
    <Tunable name="score_bonus" class="Tunable" type="int"
            default="0" display="Score Bonus" filter="0" group="General" tuning_state="NeedsTuning"
            description="Score bonus for matching topic tag." />
</Instance>
```

There are several other types of fields defined in tuning descriptions. The most important are:

"TunableList": This field type defines a repeated list of values. The type of each element in the list is defined by the nested tunable.

"TunableMapping": This is a special type of list that defines a mapping between "key" and "value" tunables. The mapping defines a "mapping_key" and "mapping_value". Keys should be unique.

XML Tuning details

This section will walk through the elements that make up an XML tuning resource based on the tdesc in the previous section. As before, the file has been broken up into multiple lines to make it easier to document, however the line breaks are not required or recommended in actual tuning files. Tuning files will load (marginally) faster if all line breaks and unnecessary white space are removed, so the file below should actually be formatted this way (without line wrapping):

<?xml version="1.0" encoding="utf-8"?><I i="topic" n="topic_Modding" c="Topic" m="topics.topic" s="17242299340165163524"><V t="enabled" n="guaranteed_content"><U n="enabled"><T n="count">2</T><T n="priority">3</T></U></V><T n="is_timed_relevancy">True</T></I>

Tuning descriptions (.tdesc files) are intended to be loaded by a tool or read by humans, so they have fairly descriptive element and attribute names, but XML tuning is loaded by the game so all elements are single characters so that it loads faster.

The XML tuning resource should have a resource key as follows:

Type: **`0x738e6c56`** (the FNV-1 hash of topic)
Group: Tuning currently must have a group of 0 (which is an exception to the rule stated above about setting the top bit).
Instance: The 64-bit FNV-1 hash of your name and the tuning instance name, with the high bit set to 1. Including your name will help avoid collisions with other Modders.
Example: "simgurumodsquad:topic_modding" -> **`0xe86b5482a0536889`**.

Example XML tuning resource contents

```
<?xml version="1.0" encoding="utf-8"?>
<I
  i="topic"
  n="topic_Modding"
  c="Topic"
  m="topics.topic"
  s="17242299340165163524">
  <V
    t="enabled"
    n="guaranteed_content">
    <U
      n="enabled">
      <T
        n="count">
        2
      </T>
      <T
        n="priority">
        3
      </T>
    </U>
  </V>
  <T
    n="is_timed_relevancy">
    True
  </T>
</I>
```

Line-by-line documentation follows:

Line 1: Standard XML header
Line 2: The “I” element indicates this is an instance tunable. “M” is used for module tuning.
Line 3: The “i" attribute is the instance type, corresponding to “instance_type” in the tdesc.
Line 4: The “n” attribute is the name of this instance. Note that this is not a localized name – it will never be displayed in the game.
Line 5: The “c” attribute is the Python class of this instance. It corresponds to “class” in the tdesc.
Line 6: The “m” attribute corresponds to the “module” in the tdesc.
Line 7: The “s” attribute is the instance ID of this tuning resource as a decimal number.
Line 8: The “V” element is a tunable value.
Line 9: The “t” attribute describes which variant type was selected. “enabled” refers to the TunableTuple starting on line 23 of the tdesc.
Line 10: The “n” attribute defines the name of the field that is being set. This corresponds to the TunableVariant starting on line 10 of the tdesc.
Line 11: The “U” element represents a Tunable Tuple.
Line 12: The name of this Tunable Tuple.
Line 13: This is a nested tunable.
Line 14: The name of the nested tunable, identifying this as the field starting on line 30 of the tdesc.
Line 15: The value of this tunable. Note that if the value were 1, this field should not exist in the tuning resource (because 1 is the default value for this field).
Line 16: The end of this tunable.
Line 17-20: Another nested tunable inside the “enabled” Tunable Tuple, this time setting priority = 3.
Line 21: The end of the Tunable Tuple.
Line 22: The end of the tunable value (the variant).
Line 23-26: Another top-level tunable, this time with a bool type. Boolean values can be either True or False. The default is False, so this tunable exists to set is_timed_relevancy = True.
Line 27: The end of this tuning instance.

Note that even though “score_bonus” exists in the tdesc, it does not have a corresponding tunable in the tuning resource, so this field will be loaded in game with the default value of zero.

Other elements and attributes that can appear in tuning files:

“L” a tunable list
“E” a tunable enum
“ev” an enum value

Simdata files

Most tuning data is used in Python, but some tuning is loaded instead or in addition by C++ code. This is determined by the “export_modes” attribute, which appears on some tuning fields in tdescs, for example in: Traits\Descriptions\Traits.tdesc

It is very important that XML tuning and binary tuning match. If a tunable feature uses both XML and simdata resources but the versions get out of synch, the UI and simulation may mismatch or the game may even crash.

There are three export modes:

- server_xml: The default; this field is exported to XML tuning resources and loaded by the simulation.
- server_binary and client_binary: This field is exported to a binary tuning "simdata" resource, and loaded by C++ code. The two variants refer to whether the tuning is loaded by the simulation thread or the rendering thread.

Binary tuning is a self-contained format; each simdata file contains a schema describing the format of data in the resource, as well as the data for each field. Unlike XML (which is nested hierarchically), simdata resources are table-oriented. This format is too complex to fully document here, but we have provided a Binary Template file that can be used to help understand how binary tuning files are constructed. See "simdata Template.bt" for more details.

Localized strings

All text that appears in the UI for things like interaction, trait or buff names must be localized. It is not possible to define interaction names directly in tuning; instead the tuning file contains a localization key which is looked up by the UI from a string table. The string table binary format is documented in "StringTable Template.bt". To decrease the chance of string key collisions, you should prefix your string keys with your name, and generate the string keys using the 32 bit FNV-1 hash function.

In the resource key for a string table, the top byte of the instance corresponds to its language (for example, EN-US is 0x00). So for a custom string table resource, you should not set the high bit of the instance in the resource key.

The string table system will not be discussed more here, as it a fairly straightforward system and works similarly to previous Sims games.

Part 3 – Detailed Mod Example

Overview

The goal of this Mod is to show a few simple additions to the game. We're going to create a new trait (Novelist) that is selectable in CAS and makes a Sim better at writing in a few ways. We'll then add an object with a unique interaction for Novelist Sims that can give them a moodlet in-game.

Note that the sample Mod created in this example is supplied with this document, the file is named **simsmodsquad-novelist.package**.

Part I: Setting Up the Trait

Trait

Let's make a Novelist trait. The novelist will be naturally better at writing, become skillful in writing faster, and maybe even get a new computer mimic.

First, we'll create an XML file called *trait_Novelist.trait*.

Next, we'll want to edit the main fields that we care about on the trait:

1. Display Name (display_name): this is a reference to a string entry that will show up as the name of the trait. We should call this trait "Novelist" and add that string to our local string database.

2. Icon (icon): this is a reference to the icon value that should show up for this trait. For this trait, let's point it to the existing writing icon for now, which has a key of 2f7d0004:00000000:a6c70d2325617d80.
3. Trait Description (trait_description): this is a reference to a string entry that will show up as the description of the trait, similar to the Display Name.
4. Ages (ages): this is a list of valid ages for this trait. Most traits are valid for all age groups, which are BABY, CHILD, TEEN, YOUNGADULT, ADULT, and ELDER.
5. Buffs > Buff Type (buffs / buff_type): this is a reference to a buff file ID, an invisible moodlet that we'll give to any Sim who has this trait. This is enclosed in a buffs list. We'll make this file in the next step and add a reference to it here.
6. Trait Type (trait_type): we'll set this to PERSONALITY to make this a personality trait (as opposed to HIDDEN, for example, which would make it invisible to the player).
7. Tags (tags): We'll want 2 tags for this trait to get it to show up appropriately in CAS – TraitPersonality and TraitGroup_Hobbies. The latter tag makes it show up in the right group; the other 3 groups for personality traits are TraitGroup_Emotional, TraitGroup_Lifestyle, and TraitGroup_Social.
8. Bonus – Trait Asm Param (trait_asm_param): if a new animation clip is created in the CAS Animation State Machine (ASM) and attached to the right parameter (CAS_Traits) in the right state (Traits), this animation will play when the trait is chosen in CAS. Alternatively, this can be set to a value that already exists (usually just the name of the trait). For the novelist, let's use the "Creative" parameter. Note that the CAS ASM data has the resource key 02d5df13:00000000:d8d5c1186ba97fdc.

Traits are objects that require an accompanying binary simdata file as well.

Buff (Invisible)

We need to make a buff to go along with our trait. This buff will be invisible to the player but will change a few stat-based things to make the Sim a better writer.

Create a file called *Buff_Trait_Novelist.buff*.

There's a lot of stuff that buffs can do to affect Sims. We're going to only edit a few fields that make sense for a writer.

1. Interactions (interactions): if we set this to enabled, we can add an Interaction Item to the list. Let's do that and reference an interaction that already exists – Idle_Trait_Creative (ID 13500). If we had wanted to create our own interaction, we could have. This reference will mean that any Sim with this trait will play this interaction from time to time; for novelists, it will be a creative idle that they share with Sims who have the Creative trait.
2. Game Effect Modifiers (game_effect_modifiers): this is the tuning field that will do most of the heavy lifting in terms of actually getting our trait to feel like it is affecting our Sim.
    a. Let's add an item to the Game Effect Modifiers list and set it to be an Autonomy Modifier. This should allow us to change the way that many behind-the-scenes stats work when this trait is active.
    b. Within the Autonomy Modifier tuning, we'll add a Stat Use Multiplier. In that, we'll set the key to statistic_Skill_AdultMajor_Writing (ID 16714), and under value, set

apply_direction to INCREASE (so it only applies when the skill is going up), and the multiplier to 1.5. What this will do is make it so any time our Sim's writing skill goes up, it goes up 50% faster.

c. Let's add another item to the Game Effect Modifiers list, and this time, let's make it an Effective Skill Modifier.
d. Under the Effective Skill Modifier we just created, we'll set the modifier_key to skill_type (we can also use skill_tag, if we wanted this to affect a number of skills that all have the same tag), and set the skill_type to reference the writing skill (same ID 16714). Lastly, we'll add a modifier_value of 2. What this will do is make it so that the game treats our Sim as if he is 2 skill levels above his actual current writing skill for most behind-the-scenes skill checks. While he won't see that increase in his skill panel or unlock interactions earlier, he should be noticeably faster at some interactions and generally play more success animations and less failures than normal.

3. Visible (visible): one last setting that we want to ensure we have for this buff is that it is invisible, so it doesn't show up in-game for the player. Set the visible value to False.

Be aware that buffs need companion binary simdata files as well, so create one for our XML file.

That's it for part one! We only had to create four files (two XML, two simdata), but with those two files, we've now created a trait that we should be able to select in CAS, that has an icon, and that will significantly influence our Sim's writing ability and skill gain when we play the game.

Part II: New Object, New Interaction, New Moodlets!

For part two, we'll create a new object for our Sim, make a new interaction on it that gives a visible moodlet, and tie to only be available for Sims who are novelists.

Object Catalog Instance

This section discusses Object Catalog Instances. Object Catalog Instances are a similar concept as CAS Catalog Instances described in the "Create A Sim Catalog Instances" section above but for build/buy objects. However we have not supplied a .bt file for Object Catalog Instances, so at this time it is an exercise for the reader to decode these binary files. A quick primer:

A catalog "buy" object such a door, chair, table, or in this case typewriter - is actually defined by two resources:

- An Object Definition (type 0xc0db5ae7). Which contains rendering information for the object. Such as references to the model, texture and rig the object uses. And important for this example, it also contains a reference to the object's tuning.
- A Catalog Object Instance (type 0x 319e4f1d). Which contains information about how to display the object in the catalog. Such as its name and description, what type of object it is and which section of the catalog it should appear.

To create a new buy object we need to make both resources. For the purposes of this example, we will just talk about the changes that are required for us to make our new typewriter for this Mod.

You can compare the standard typewriter out of the Maxis data the one in the sample Mod package to better understand the binary changes that were made to these resources for this example.

Maxis resources:
c0db5ae7:00000000:0000000000006e4e and 319e4f1d:00000000:0000000000006e4e

The equivalent example resources:
c0db5ae7:80000000:889a604f0a722f69 and 319e4f1d:80000000:889a604f0a722f69

Items that were tweaked for this new typewriter:

1. **Catalog Name and Catalog Description:** creating two strings and tying them to the catalog name and description will make it much easier to find the object in the Buy Catalog (and a chance to show off those witty writing skills). These string keys are specified in the Catalog Object Instance.
2. **Simoleon Price:** this is simply the cost of the object in the catalog. This specified in the Catalog Object Instance.
3. **Object Tuning ID:** every object catalog file will need to reference an object tuning file. This is specified in the Object Definition resource. The next section will describe how to create tuning for our new typewriter.

Object Tuning File

Every object that has any interactions or behaviors requires an object tuning file to go along with its object catalog file.

Let's create one for the sculpture we just created. Let's create *object_TypewriterNovelist.object* file.

Now let's add an interaction to the object.

1. Super Affordances (_super_affordances): The only thing we actually want to add to our special sculpture is a new interaction, which is under the super affordances list. We don't have a reference to what interaction we're adding yet, since we haven't created it! Let's go make it now and then add it in here.

Interaction

An interaction is something a Sim can do, whether it is with an object, another Sim, or all by themselves. Here, we'll add a new interaction on our decorative typewriter that is only available for novelists and gives them a buff (or two!) after they finish the interaction.

Let's start by creating a file called *typewriterNovelist_Reminisce.interaction*.

We'll make it a super interaction (most object interactions are super interactions, while most socials are mixer interactions), and give it the following tuning:

1. Basic Content (basic_content): set the basic content to one_shot and then, under the animation_ref, reference the factory *12069*, which is an existing animation that has the Sim look thoughtful.
2. Contraints (_constraints): because the animation we're using is a single-Sim animation, it can happen anywhere. Within the constraints, we'll set up a circle and facing constraint so that the Sim will route over near the sculpture before doing the animation.
3. Icon (_icon): instead of creating a custom icon for everything, we can always use the icon to refer to the object we're targeting for an interaction. Let's set the icon to participant, and under

participant_type, use the enum Object. This means that when this interaction is in the queue, it will have the icon of the typewriter sculpture.

4. Display Name (display_name): we'll add a string here – "Reminisce" – that will show up in the pie menu when we click the object
5. Test Globals (test_globals): to tie this to the Novelist trait, we'll make this interaction only show up for Sims who have that trait. To do so, we'll add a new test to the global tests. We will be able to just use the trait test and add a reference to the trait we created in part 1 in the whitelist_traits list.
6. Outcome (outcome): outcome is what dictates what happens after the basic content in an interaction and the primary place where we can apply gameplay changes to the Sim or object involved. For this interaction, let's create a dual outcome for now and tune the base_chance under success_chance to 75 - this is a % chance that we'll choose the success outcome instead of the failure outcome.

Let's go create two moodlets that we can apply to the Sim depending on if the interaction succeeds or fails. After doing that, we'll come back and hook them back into the outcome tuning for this interaction.

Buff (visible) x 2

Let's make two files, named *Buff_TypewriterNovelist_Success.buff* and b*uff_TypewriterNovelist_Fail.buff*.

While there are a variety of things visible buffs (aka moodlets) can do, we'll focus on the major aspects that are required for a moodlet to function properly:

1. Emotion (mood_type): this details what emotion the moodlet will try to put the Sim into. For our two buffs, let's use *14641* (Inspired) for the success and *14635* (Embarrassed) for the failure.
2. Weight (mood_weight): this is a number that determines how strong the moodlet is. When a Sim has multiple moodlets, the weights of all moodlets of the same emotion get added together, and the highest weighted emotion wins. For now, let's make the weights just 1 so these moodlets don't swing your Sim's emotion too wildly.
3. Temporary Commodity (_temporary_commodity_info): most simple moodlets are controlled by a temporary commodity, which is a fancy way of saying the amount of time that a moodlet is given before it goes away. Let's enable the temporary commodity here and give it two pieces of information – an *Inspired_Buffs* or *Embarrassed_Buffs* enum as its category and a max_duration of 120. The max_duration is in Sim minutes, so this moodlet should last 2 hours.
4. Name & Description (buff_name / buff_description): each buff will have two strings – one will be the name and the other the description. These should both point to unique string keys.
5. Icon & Icon Highlight (icon / icon_highlight): these fields point to an image reference that is the icon that will be shown to indicate the actual buff when the Sim has it. These fields can be identical unless the icon should be different when the Sim has the emotion of the moodlet vs. when they don't.
6. Optional: Audio Stings (audio_sting_on_add / audio_sting_on_remove): these are tuned primarily because these audio references are the standard stings that play when any moodlet is added or removed.

Each buff will also require a binary simdata file for associated UI information.

Loot x 2

We're almost ready to hook these moodlets up to our interaction outcome, but we need two more files in order to do it properly. Because so much of the tuning for The Sims 4 is data driven, and to give ourselves more flexibility, interaction outcomes don't directly give buffs. Instead, they have a tuning field called Loot Lists, which define lists of "loot" to give. Loot comes in many different flavors, with buffs being one of them, but the same idea could be extended to give things such as stat changes, inventory items, career promotions, and the like.

To hook our interaction to our buffs then, let's create two final files: *Loot_Buff_TypewriterNovelist_Fail.action* and *Loot_Buff_TypewriterNovelist_Success.action*.

There isn't too much tuning we'll need to do in these files:

1. Buff Type (buff_type): in the loot_actions list, we'll add a buff variant and attach the appropriate buff_type (which will be a reference to the buff we created in the last step). Make sure to attach the fail buff to the fail loot and the success buff to the success loot!
2. Buff Reason (buff_reason): enable buff_reason and point it to a string. Both loots can probably share the same string, which should roughly be of the format *(i.e. "From reminiscing")*. This string is what appears below the moodlet to help the player understand what their Sim may have done to have gotten that moodlet.

Back to our Interaction

Now that we have IDs for our two loots, we can go back to our interaction and hook them up to the appropriate loot lists (look for loot_list under the failure_actions and success_actions).

Come to think of it, let's add another global test as well, under the test_globals list. It seems silly to allow a Sim to reminisce over and over again, probably getting both the positive and negative moodlets at the same time. That makes no sense! We'll make it so that a Sim can't do the reminisce interaction if they've done it recently by only allowing Sims who don't currently have one of the two moodlets that result from the interaction to actually reminisce.

To do that, we'll add a *buff* variant to the test_globals list, create a *blacklist* list under that variant, and then add references to the two buffs we just created as tunables in that list.

Note: you may notice that there is an additional buff that appears under the test_globals (26171). This buff is a blacklist buff on almost every interaction and is a best practice to add as it checks that a Sim is not currently dying; it turns out that having a dying Sim attempt to do other interactions causes lots of problems! You may also notice that there is a sim_info test that appears as well. While no ages are explicitly specified, this structure uses the default ages variant tuning, which makes the interaction available for teens, young adults, adults, and elders. If you wish for the interaction to be available for all ages, simply remove this test. If you wish the interaction available for a different set of ages, feel free to specify the ages in the ages specified list. Be aware that making an interaction that only has adult animations available for a child may result in horrifying animations never meant for human eyes.

Once we've done the above, we should be able to do the following things in game:

1. Give our Sim a new trait in CAS called Novelist.
2. See that Sim gain writing skill faster and appear to be a better writer at all times.

3. Be able to purchase a new typewriter sculpture from the Living Room Sculpture portion of the catalog.
4. If our Sim has the Novelist trait, see a new interaction called Reminisce on that object.
5. Upon doing that interaction, have a moodlet be applied to our Sim (positive 75% of the time, negative 25% of the time).

That's it! With a bit more exploring of the associated tdesc documentation and the many different ways that tuning files can be modified, you'll find that there are tons of things that can be done simply with a little bit of data in The Sims 4.

Packaging

The instance IDs for tuning should be generated using the following process:

Step	**Result**
Resource Name	`Buff_Trait_Novelist`
Your Name	`SimGuruModSquad`
Combine Them	`SimGuruModSquad:Buff_Trait_Novelist`
Convert to Lower Case	`simgurumodsquad:buff_trait_novelist`
64-bit FNV-1 Hash	`0x48a4d404a5a3bb0e`
Set High Bit ('OR' with 0x8000000000000000)	`0xc8a4d404a5a3bb0e`

The group IDs for tuning resources must be zero (at this time setting the group to another value is not supported).

For the resources in this example mod, this process results in the following instance IDs:

Resource Name	**Instance ID**
SimGuruModSquad:Buff_Trait_Novelist	`0xc8a4d404a5a3bb0e`
SimGuruModSquad:Buff_TypewriterNovelist_Fail	`0xef763f593d7b1d07`
SimGuruModSquad:Buff_TypewriterNovelist_Success	`0x86c68a94e577adc8`
SimGuruModSquad:Loot_Buff_TypewriterNovelist_Fail	`0xfecfa9ccb71b7a4c`
SimGuruModSquad:Loot_Buff_TypewriterNovelist_Success	`0xb6172a3815ab8ef1`
SimGuruModSquad:object_TypewriterNovelist	`0x889a604f0a722f69`
SimGuruModSquad:trait_Novelist	`0xed7000a97ea8f032`
SimGuruModSquad:typewriterNovelist_Reminisce	`0xa586620a4681b704`
SimGuruModSquad:typewriterNovelistObject	`0x889a604f0a722f69`
SimGuruModSquad:NovelistStrings_ENG_US	`0x00d4c1ffd5a99fdb`

Part 4 – Script Mods

Although many features can be modified and added using tuning resources, some features will require modifying or extending scripts. The Sims 4 uses Python 3.3.5. It is possible to add your own scripts, and to override scripts from the game.

Scripts can be added to the game via the "Mods" directory (see "Installing Custom Content" section) using two methods:

1) In a zip file containing compiled Python scripts:

a. Create an empty zip file in the “Mods” directory
b. Add optimized, compiled Python files to the zip file (compiled files end with the extension .pyo). Here is a very basic way to generate .pyo files:
    i. Run “python.exe -O”
    ii. Import your file using “import my_file
    iii. Go into the “__pycache__” folder
    iv. Rename the file “my_file.cpython-33.pyo” to “my_file.pyo”
    v. Put this into your zip file
c. If you want people to be able to see what your Mod does, you can optionally put .py (and .pyc) files into the zip file as well.

2) As loose files in a “Scripts” directory:
    a. Create a directory for your Mod in the Mods folder, with the name of your Mod.
    b. Inside of that directory, create a subdirectory called “Scripts”.
    c. Put your loose Python scripts in this folder.

Both approaches support Python packages (nested subdirectories containing Python files).

It is recommended that Mods are distributed as a zip containing .pyo files for ease of installation, and because .pyo files are optimized for performance. The loose file method listed above is primarily intended to simplify authoring of Scripts prior to publishing.

You can override Maxis Scripts, however to avoid breaking the game your Scripts should preserve the existing Maxis APIs where possible. To override a Script in a sub-package (a Script in a subdirectory containing an “__init__.pyo” file), your Mod must include all other files in the same directory. This type of Mod may need to be updated when the game is updated so it is recommended that this is avoided where possible.

An example Script Mod is included with this document: **simsmodsquad-maslow.zip**. This Mod adds a new console cheat that fills all commodities of all Sims on the current lot. Place in the “Mods” directory. In game, press Ctrl+Shift+C to bring up cheat console and run the command “|maslow true”.

Conclusion

Thank you for your interest in creating Custom Content and Mods for The Sims 4. We are always inspired by your creations and look forward to seeing what you come up with! Please visit the official forums for The Sims 4 at http://forums.thesims.com/.

Appendix A : Reference Files

The follow files are included with this document:

PackageTemplate.bt	BT File describing the binary format of .package files.
PartDataTemplate.bt	BT File describing the binary format of PartDataResource files (resource type 0x034aeecb)
simdata Template.bt	BT File describing the binary format of binary tuning data.
StringTable Template.bt	BT File describing the binary format of localized string resources (resource type 0x220557da)

TuningDescriptions.zip	Tdesc files for all tuning.
simsmodsquad-maslow.zip	Example python script mod: fills all commodities of all Sims on the current lot
simsmodsquad-novelist.package	Example tuning “Novelist” mod: The creation of this mod is described in this document.