XML Injector Version 2
Modder's Documentation

TABLE OF CONTENTS

Basic Concepts and Introduction

The purpose of the XML Injector mod library is to eliminate the need for many The Sims 4 mods to include their own scripts merely to inject affordances to the various affordance lists of game objects. Performing these injections via scripting is frequently necessary to eliminate the possibility of conflicts between mods that want to modify the same game objects, particularly the object_sim tuning.

Using the XML Injector eliminates any need for the modder to understand the methods of writing scripts to modify XML tuning at game startup, compiling and maintaining these additional scripts which only serve to accomplish a simple task. Instead, the modder simply creates an easy to write XML snippet and includes this in their mod package.

This documentation is not intended to teach modders how to create interactions. You are expected to already understand the basics of The Sims 4 XML modding, what interactions are, and what the object tuning affordance lists are intended for. If you are creating mods using a mod creation tool and do not understand the actual XML, please refer to the maintainer of your mod creation tool to inquire if they have a guide they have written for creating the necessary XML Injector snippet. If you are a maintainer of one of these mod creation tools, you are certainly encouraged to have your creation tools create these snippets automatically for your users. If you have any questions regarding this, please feel free to contact me.

The XML Injector is intended to be a standard for modders to use, so you should not include a copy of the script in your mod distribution. Instead, inform your mod users that they should download and install a copy of the XML Injector (currently hosted at Mod The Sims and maintained by the original author, Scumbumbo). Make sure your users know that installing the XML Injector is required in order for your mod to operate properly, and that they need only install one copy of the injector in their mods folder.

Your users should be encouraged to always update to the latest version of the library as soon as it is released, as it may include bug fixes or required updates for compatibility with the latest game patches. Future versions of the XML Injector library may also include additional new features (your suggestions for these are welcome) to simplify modifying game objects without scripting requirements. Be sure, if you utilize any possible new features, that you inform your users of any said minimum version requirement for the library.

As the idea is to make this library a useful standard for all modders, in the event I choose to leave The Sims 4 modding community in the future, I will select someone willing to take over the maintenance of this library and will assist them in any technical details when necessary. As the code currently only adds interactions to various affordance lists, the code is not particularly complex at this time; however, possible future features may include some more advanced coding requirements.

Ok, I think that gets the *boring* stuff out of the way, so let's move on to...

Summary of New Features in Version 2

- Mods can include two tests for proper installation and availability of XML Injector features.
  - xml_injector_minimum_version allows mods to check that a minimum version of XML Injector installed.
  - A standardized DialogDramaNode tuning can be included in a mod to test to see if XML Injector is not installed at all.
- A new object selection variant, objects_with_tag is available.
- New tunings
  - add_to_loot_actions and add_to_random_loot_actions may be used to append loot actions to LootActions and RandomWeightedLoot tunings.

- add_buffs_to_traits is used to add buffs to the buffs list for Trait tunings.
- add_object_relationships_to_objects allows adding an object relationship component to object tunings.
- add_name_component_to_objects allows adding a name component to object tunings.
- add_states_to_objects may be used to add states and state_triggers to object tuning's state components.

Coding an XML Injector Snippet

Like any of The Sims 4 XML snippets, an XML Injector snippet begins with an <I> tag which specifies the various parameters required for each snippet. You should choose an "n" (name) for the snippet and use an FNV-64 hash with the high-bit set for the "s" (instance ID) value. The remainder of the attributes of the <I> tag should remain as in the example below. This example is from the test package from the XML Injector distribution:

```
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_AddInteractionTest" s="13197251937291617972">
```

In the above example, the name of the snippet is Scumbumbo_XmlInjector_AddInteractionTest and the FNV-64 hash for this string, with the high bit set, is 13197251937291617972.

You also need to specify the Type, Group and Instance for the snippet within your package creation tool. The type should be the standard for all snippets, 0x7DF2169C. The group should be specified as 0x00000000. And the instance should be the hex equivalent of the "s" attribute used on the <I> tag (0xB7260EB7059A46B4 for the above example, as that is the hex equivalent of 13197251937291617972).

The remainder of the snippet should consist of lists of the various values you want to inject into the game object tunings.

XML Injector Version Checking

Two optional tunings may be specified to check if the installed version of XML Injector is new enough to support the features your mod requires and to specify a custom dialog if the version number requirement is not met. A default dialog will be displayed if no custom dialog is set.

xml_injector_minimum_version

The xml_injector_minimum_version tuning should be set to an integer specifying the minimum version of the XML Injector which is required to process your snippet properly. This should be set to version 2 or higher as version checking was not present in version 1 of the injector.

If the minimum version requirement is not met by the installed XML Injector, a dialog will pop up when the first lot is loaded in a player's game alerting them of the unmet requirement. A default dialog will be created if you do not require any special notification text; however, it is a good idea to set the optional dialog tuning in your own mod to offer the player additional information, such as the name of your mod and the download location for the injector, as well as text strings for other languages apart from the English text supplied by the default dialog.

version_error_dialog

The optional custom dialog is set using the version_error_dialog tuning. This is a standard game dialog tuning with the majority of the more specialized options locked from modification. The text and title will likely be the only options you will want to set. See the tuning description for the XML Injector for details on the other dialog options which may be tuned.

Note that, like any custom game dialog, you will need to provide a string table (STBL) resource in your mod package which contains the text strings you wish to display for each of the game's supported languages.

Only one alert dialog will be displayed at startup. Typically this is not a cause for concern as the first mod loaded which has the highest version requirement set will appear. To test your dialog appearance you should simply set the xml_injector_minimum_version to a value greater than the current installed version. Of course, be sure to set the value to the proper version requirement after testing and before publishing your mod!

If your mod package contains multiple XmlInjector snippets, you only need to set the version requirements and dialog in a single snippet; however, it may be a good idea to include the version tuning in each of your snippets in case you remove a snippet later in the development of your mod.

Example Version Check Tuning

```
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="XmlInjector_Version_Example" s="9759675081600351838">
  <T n="xml_injector_minimum_version">2</T>
  <V n="version_error_dialog" t="enabled">
    <U n="enabled">
      <V n="text" t="single">
        <T n="single">0xE04EC6B2<!--String: "<b><i>Custom Mod Name</i></b> requires version 2 of the XML Injector library.
                                          \n\n<font size='14'>The mod will not operate properly until you have updated
                                          your copy of the XML Injector to the latest version.</font>"--></T>
      </V>
      <V n="title" t="enabled">
        <T n="enabled">0x5CD96C46<!--String: "XML Injector Update Required"--></T>
      </V>
    </U>
  </V>
  <!-- Additional snippet tunings as required -->
</I>
```

Adding Interactions / Super Affordances

Five types of tunings may be specified in a snippet in order to add interactions to the various affordance lists of objects and to affordance list snippets.

- **add_interactions_to_objects** - Contains a list of tuples, each tuple should contain an object selection variant followed by a list of the interactions to add to those selected objects.
- **add_interactions_to_sims** - Contains a list of interactions to add to the object_sim tuning. (This is really just a shortcut to specifying an add_interactions_to_object for just the Sim game object, which would work identically well.)
- **add_interactions_to_phones** - Contains a list of interactions to add to Sim's phone panels.
- **add_interactions_to_relationship_panel** - Contains a list of interactions to add to the Sim relationship panel.
- **add_mixer_interactions** - Contains a list of tuples, each tuple should contain a list of mixer snippet references followed by a list of the mixer interactions to add to those mixers.

Only *one* of *each* of these lists may be specified in an Xml Injector snippet. For instance your snippet may contain both an add_interactions_to_objects and an add_interactions_to_sims tuning; however, it cannot contain two add_interactions_to_objects sections. To specify multiple object selection/interaction additions, you should simply use multiple tuples in the list.

add_interactions_to_objects

This section is used to add interactions to the _super_affordances list of a game object or objects, allowing that interaction to then appear on the pie menu of an object, be run autonomously, or both. The tuning should include an object selection variant used to select which object tunings should be modified. This is then followed by a _super_affordances list of those interactions to be added to the selected objects.

Object Selection Variants

Each individual tuple must contain an object_selection variant followed by a _super_affordances list of those interactions to be added to the _super_affordances list of those selected objects.

The simplest and most common object selector is to specify a list of object tuning references to be modified. For example, if you wish to select all of the game's object_trash_generic objects, you could use the following object_selection:

```
<V n="object_selection" t="object_list">
  <U n="object_list">
    <L n="object_list">
      <T>14992<!--GameObject: object_trash_generic--></T>
      <T>155762<!--GameObject: object_trash_generic_noSalvage--></T>
      <T>137444<!--GameObject: object_trash_generic_simInventory--></T>
      <T>156396<!--GameObject: object_trash_generic_small--></T>
    </L>
  </U>
</V>
```

An object_list is the most precise method of choosing which object tunings are selected, but it may not always be the easiest method. The above example could also be accomplished using the objects_matching_name variant which will select all object tunings which contain the partial_name in their object tuning name.

```
<V n="object_selection" t="objects_matching_name">
  <U n="objects_matching_name">
    <T n="partial_name">object_trash_generic</T>
  </U>
</V>
```

If care is not used to make the partial_name specific enough, this could result in object tunings that are not desired being selected. For instance, object_trash_ could be used to match all trash pile objects; however, if the trailing underscore is removed, object_trash would also match the object_trashcan tunings in addition to trash piles.

Another method of selecting object tunings to be modified is the objects_with_affordance variant which selects all objects that include a specific affordance in their _super_affordance list. This is very useful for selecting a number of objects without having to specify each individually in a list, dealing with the uncertainty of using name matching, and having to update the mod when a new expansion adds additional tunings that you would like to select.

For instance, consider the game's sofa furniture objects. There are many of these, and more get added by various expansions. Most of these contain the string object_sitSofa, but others do not. Other mods may add sofa objects which do not contain that string as well. By selecting all objects which include the standard sofa_Nap interaction in their super affordances, all sofa objects can be easily selected including those added by expansions or other mods.

```
<V n="object_selection" t="objects_with_affordance">
  <U n="objects_with_affordance">
    <T n="affordance">14305<!--SuperInteraction: sofa_Nap--></T>
  </U>
</V>
```

The last variation of the object_selection tuning is objects_with_tag which is used to select all objects in the game that have a specific tag applied to their object definition. Typically the tag used for selection would be a functionality tag, such as Func_Toy. For example, the following could be used to select all double beds for adding affordances.

```
<V n="object_selection" t="objects_with_tag">
  <U n="objects_with_tag">
    <E n="tag">Func_DoubleBed</E>
  </U>
</V>
```

Selecting objects using functionality tags is dependent on the creator of any custom content objects to have specified the proper tags in their object definition. If the player has CC objects which have invalid tagging this can result in objects not being targeted properly. If possible, you should consider using the objects_with_affordances selection variant instead.

This complete example demonstrates how all of these items are put together to add two new custom interactions with the instance numbers of 123456789 and 987654321 to all computer objects in the game, and interaction 5432198765 to the three camera objects.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test1" s="10608251930441092048">
  <L n="add_interactions_to_objects">
    <U>
      <V n="object_selection" t="objects_with_affordance">
        <U n="objects_with_affordance">
          <T n="affordance">13228<!--SimPickerInteraction: computer_Picker_Chat--></T>
        </U>
      </V>
      <L n="_super_affordances">
        <T>123456789<!--SuperInteraction: new_custom_interaction_1--></T>
        <T>987654321<!--SuperInteraction: new_custom_interaction_2--></T>
      </L>
    </U>
    <U>
      <V n="object_selection" t="object_list">
        <U n="object_list">
          <L n="object_list">
            <T>110804<!--GameObject: object_Camera_Cheap--></T>
            <T>110805<!--GameObject: object_Camera_Moderate--></T>
            <T>110806<!--GameObject: object_Camera_Expensive--></T>
          </L>
        </U>
      </V>
      <L n="_super_affordances">
        <T>5432198765<!--ImmediateSuperInteraction: new_custom_interaction_3--></T>
      </L>
    </U>
  </L>
</I>
```

add_interactions_to_sims

The remaining tunings for adding interactions are simpler as there is no need for any complexity in choosing a selection method for objects. The add_interactions_to_sims tuning will add a list of interactions to the _super_affordances of the object tuning for Sims. It's really just an easy shortcut to coding an add_interactions_to_objects where the selected object is the object_sim (instance ID 14965).

This example would add two fictional new interactions, 123456789 and 987654321 to Sims, allowing those interactions to appear on the pie menu when clicking on a Sim and/or being used autonomously with a Sim as the target.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test2" s="10608251930441092051">
  <L n="add_interactions_to_sims">
    <T>123456789<!--SuperInteraction: new_custom_interaction_1--></T>
    <T>987654321<!--SuperInteraction: new_custom_interaction_2--></T>
  </L>
</I>
```

add_interactions_to_phones

The add_interactions_to_phones tuning also modifies the object_sim tuning; however, it adds the new interactions to the _phone_affordances of Sims, allowing them to show up on a Sim's phone panel.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test3" s="10608251930441092050">
  <L n="add_interactions_to_phones">
    <T>123456789<!--SuperInteraction: new_custom_interaction_1--></T>
    <T>987654321<!--SuperInteraction: new_custom_interaction_2--></T>
  </L>
</I>
```

add_interactions_to_relationship_panel

Likewise, the add_interactions_to_relationship_panel tuning may be used to add interactions to the _relation_panel_affordances of the object_sim. These new interactions would then show up on the pie menu which appears when clicking another Sim via the relationship panel.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test4" s="10608251930441092053">
  <L n="add_interactions_to_relationship_panel">
    <T>123456789<!--SuperInteraction: new_custom_interaction_1--></T>
    <T>987654321<!--SuperInteraction: new_custom_interaction_2--></T>
  </L>
</I>
```

add_mixer_interactions

Finally, the add_mixer_interactions tuning allows adding interactions to one or more of the social mixer AffordanceList tunings, for example the friendly social mixers. These interactions would then appear as topics of conversation, or other actions, to be used when Sims are having conversations with each other.

This tuning consists of one or more tuples, each of which contains two lists: mixer_snippets to select the AffordanceList or lists to be modified, and affordances to indicate which interactions should be added.

Although, as the tuning name suggests, this is primarily used for adding social mixer interactions, it can also be used to add interactions to any of the many other AffordanceList snippets, for instance those used to determine which interactions are valid for a club interaction group, or interactions which should be marked as invalid for a Sim who is grounded.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test5" s="10608251930441092052">
  <L n="add_mixer_interactions">
    <U>
      <L n="mixer_snippets">
        <T>24508<!--AffordanceList: social_Mixers_Friendly_NonTouching--></T>
      </L>
      <L n="affordances">
        <T>123456789<!--SuperInteraction: new_custom_interaction_1--></T>
        <T>987654321<!--SuperInteraction: new_custom_interaction_2--></T>
      </L>
    </U>
  </L>
</I>
```

Adding Buffs to Traits

add_buffs_to_trait

This tuning makes it possible to add entries to the buffs list of any existing trait. The tuning consists of a trait tunable reference to identify the trait to be modified, followed by a tunable list of buffs to be added. The same options which are available in the trait tuning are usable in the buffs list, both a buff_reason and buff_type.

For example, this snippet would add a permanent "Feeling Flirty" buff to any Sims that have the romance trait. It's not meant to be realistic, but effectively demonstrates what can be done with these tunings.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test10" s="17534587827348902464">
  <L n="add_buffs_to_trait">
    <U>
      <T n="trait">27454<!--Trait: Trait_Romantic--></T>
      <L n="buffs">
        <U>
          <V n="buff_reason" t="enabled">
            <T n="enabled">0x040B0900<!--String: "(From Being Romantic)"--></T>
          </V>
          <T n="buff_type">9305<!-- Buff: Buff_Trait_FeelingFlirty--></T>
        </U>
      </L>
    </U>
  </L>
</I>
```

Caveat modtor (Let the modder beware) - Some care needs to be exercised when adding buffs to many traits. For instance, say we wanted to make all aliens flirty instead of romance Sims (yes it's possible to add to the hidden auto-assigned traits such as species and gender). This could have unintended side-effects if we added that buff to trait_OccultAlien as it would also apply to child and toddler Sims. In this specific case, it *would* be safe as Buff_Trait_FeelingFlirty does have a test on it to disallow adding the buff to any Sims younger than teen. A little thinking ahead, as always, can go a long way to avoiding potential side-effects.

Adding Loot Actions

These tunings are designed to add loot actions to tuning types. Currently two features are supported to allow adding to the loot action lists of LootActions and RandomWeightedLoot types.

add_to_loot_actions

This tuning allows adding one or more loot action tunings to the loot_actions list of an existing LootActions tuning class. Any of the game's supported loot actions may be specified, see the XML Injector tuning description (or the Actions / LootActions tdesc) to see the supported loot actions.

The loot_actions_ref tuning specifies a reference to the LootActions tuning from the game which you wish to modify, e.g. loot_Homework (instance 31590) or loot_Reaction_Jealousy (119779).

This should then be followed by a list of additional loot actions to add in the loot_actions_to_add tuning. The original loot_actions list will then have these added to the referenced tuning.

This actually sounds much more complex than it is, but the example should make it clear what is going on. Refer to the example's referenced tuning classes of the game if you are not familiar with those. This example would add a 50% chance for a happy "Did Good" buff to be applied to an actor Sim when they tip a violinist by adding the appropriate buff loot action to the loot_actions list of Give_Tip_Violin. This action will be added to the existing loot action already in Give_Tip_Violin of giving the target Sim a money_loot.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test6" s="10608251930441092053">
  <L n="add_to_loot_actions">
    <U>
      <T n="loot_actions_ref">38891<!--LootActions: Give_Tip_Violin--></T>
      <L n="loot_actions_to_add">
        <V t="buff">
          <U n="buff">
            <U n="buff">
              <V n="buff_reason" t="enabled">
                <T n="enabled">0xDB2C6673<!--String: "(From Good Trait)"--></T>
              </V>
              <T n="buff_type">35774<!--Buff: Buff_Trait_Good_DidGood--></T>
            </U>
            <U n="chance">
              <T n="base_chance">50</T>
            </U>
            <L n="tests">
              <L>
                <V t="trait">
                  <U n="trait">
                    <L n="whitelist_traits">
                      <T>27915<!--Trait: trait_Good--></T>
                    </L>
                  </U>
                </V>
              </L>
            </L>
          </U>
        </V>
      </L>
    </U>
  </L>
</I>
```

add_to_random_loot_actions

This tuning works very similarly to the add_to_loot_actions tuning described above; however, it is used to apply additional weighted loot actions to a RandomWeightedLoot tuning. A weighted loot action is simply a single loot action, combined with a weight. Any weighted loot actions you add will be added to the set of possible choices.

Again, it is hard to make this sound simple, but it is. Here's another example! This adds the possibility that a starter vegetable seed packet may contain a rare seed instead. This is done by adding a loot action variant with a small weight (the other actions in randomWeightedLoot_Garden_SeedPacket_StarterVegetable all have a weight of 1, so a value of less than 1 will make our rare seed less likely to be chosen) which redirects the loot action to perform the randomWeightedLoot_Garden_SeedPacket_Rare action instead.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test7" s="10608251930441092054">
  <T n="xml_injector_minimum_version">2</T>
  <L n="add_to_random_loot_actions">
    <U>
      <T n="random_weighted_loot_ref">188638<!--RandomWeightedLoot: randomWeightedLoot_Garden_SeedPacket_StarterVegetable--></T>
      <L n="random_loot_actions_to_add">
        <U>
          <V n="action" t="actions">
            <T n="actions">188586<!--RandomWeightedLoot: randomWeightedLoot_Garden_SeedPacket_Rare--></T>
          </V>
          <U n="weight">
            <T n="base_value">0.1</T>
          </U>
        </U>
      </L>
    </U>
  </L>
</I>
```

The result then is that when a random vegetable seed is chosen, there will be a slight chance (0.1 chance out of 5.1, since there are already 5 possible choices each with a weight of 1) of a rare seed instead.

Loot Action Recursion Test

The loot action "actions" variant type (as used above in the vegetable seed example) can cause recursion errors where a loot action calls itself, either directly or via another loot action. The game will normally not detect these issues until a loot action occurs in game, then throwing an exception error.

XML Injector tests for these recursions whenever loot actions are added to either a LootActions or RandomWeightLoot using a snippet. If any chance of recursion is detected the additional tunings are rejected and skipped (the remainder of the snippet will continue to be processed). This eliminates the possibility of an added loot action throwing exceptions during game play. Instead an error will be logged to the XmlInjector.log file (assuming you have the Sims Log Enabler, see the Troubleshooting section on page 13 for details).

For example, if in addition to the vegetable seed example above allowing vegetable seeds to possibly trigger the rare seed loot, an addition was also made to the rare seed loot to allow it to possibly trigger a vegetable seed instead -- It would then be possible for the game to choose a rare seed, which chooses a vegetable seed, which chooses a rare seed, etc. Even though in our example there would be only a slim chance of this actually occurring due to the small chance of a rare seed in a vegetable packet, the game would *always* throw an exception and reject the action (opening the seed packet would fail completely) if this chance were even remotely possible. By testing at the time of injection, this mess is avoided completely.

This kind of recursion possibility is only possible when the "actions" variant type is used allowing an endless cycle of loots to occur, so in the majority of circumstances this can be ignored; however, when using that variant type you must use care to avoid the chance of recursion. This is not unique to XML Injector, but is the case even when overriding XML loot action resources or creating new ones using normal XML overrides.

Object Component Modifications

Two components may be added to object tunings using XML Injector: name and object relationship components. In addition to the two components which can be added, the add_states_to_objects tuning is available to add state_triggers and/or states tunings to an existing state component on game objects. All three of these tuning snippet types use the same object selection methods discussed in the Object Selection Variant section on page 5.

add_name_component_to_objects

The add_name_component_to_objects tuning allows a name component to be added to selected objects in the game. The tuning consists of an object_selection variant to select one or more game object tunings to target, followed by a name_component tuning. The tuning can include any of the standard name component variables: affordance, allow_description, allow_name and templates.

This example creates a name component for any objects which have the Func_Fridge tag set on them, allowing refrigerators in the game to have a custom name set on them.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test8" s="10608251930441092055">
  <T n="xml_injector_minimum_version">2</T>
  <L n="add_name_component_to_objects">
    <U>
      <V n="object_selection" t="objects_with_tag">
        <U n="objects_with_tag">
          <E n="tag">Func_Fridge</T>
        </U>
      </V>
      <U n="name_component">
        <L n="templates">
          <U>
            <T n="template_name">0x719396BC<!--String: "Type Name Here"--></T>
          </U>
        </L>
      </U>
    </U>
  </L>
</I>
```

add_object_relationships_to_objects

This tuning allows you to add object relationship components to game objects. Much like the name component snippet, this consists of an object_selection variant, followed by an object_relationships_component tuning.

Most of the standard object relationship tunings can be added via this *with the exception of* the icon_override tuning. An object relationship's icon_override requires modification of the SimData resource for the object and those resources cannot be created or changed via Python scripting. Because of this, if an icon_override behavior is required by your mod you are better off changing the object tuning by overriding the original EA object tuning. If an icon_override is specified in your tuning snippet, an error will be logged and that setting will be ignored (the rest of the object relationship component will be created normally).

This snippet only adds the relationship component to the object. A completely modded object will also require relationship statistics, loots and interactions added or modified in order to increase or decrease a Sim's relationship to that object. If you are unfamiliar with doing this, you should take a look at one of the existing game objects (stuffed toys, for instance) to see how this would all be set up.

The following example creates an object relationship component which would allow Sim's to make friends with their outdoor trash cans. Again, this just adds the necessary component and for this example to function, XML for the custom statistic would need to be created and interactions modified or added to the trashcan to allow the relationship to change.

```
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test9" s="10608251930441092056">
  <T n="xml_injector_minimum_version">2</T>
  <L n="add_object_relationships_to_objects">
    <U>
      <V n="object_selection" t="object_list">
        <U n="object_list">
          <L n="object_list">
            <T>14991<!--GameObject: object_trashcan_outdoor--></T>
          </L>
        </U>
      </V>
      <U n="object_relationships_component">
        <T n="relationship_stat">12998889312675390570<!--Statistic: statistic_Object_TrashCanRelationship--></T>
        <V n="relationship_track_visual" t="enabled">
          <U n="enabled">
            <T n="relationship_track">16650<!--RelationshipTrack: LTR_Friendship_Main--></T>
          </U>
        </V>
      </U>
    </U>
  </L>
</I>
```

add_states_to_objects

An add_states_to_objects tuning allows additional state_triggers and states tunings to be added to the state component of game objects. All of the standard state_triggers and state tunings are supported. Note that *this does not create a state component* on a game object, there must be one that already exists in the object's XML tuning in order for this to function.

The tuning snippet consists of an object_selection variant, followed by a state_component tuning which may contain state_triggers and/or states lists.

The example below modifies bowls of chili to add a new set of states which can be set on them which controls whether or not they are extra hot and spicy. A state_triggers is also added which will tell the game to set any chili that has been made extra hot and spicy as being of outstanding quality.

Note that, just like the previous object relationship example, this only modifies the game object tuning itself. A full mod would require the addition of XML resources for the object state values and some form of interaction or loot which can be used to set the bowl of chili as spicy.

```xml
<?xml version="1.0" encoding="utf-8"?>
<I c="XmlInjector" i="snippet" m="xml_injector.snippet" n="Scumbumbo_XmlInjector_Test11" s="17534587827348902465">
  <T n="xml_injector_minimum_version">2</T>
  <L n="add_states_to_objects">
    <U>
      <V n="object_selection" t="object_list">
        <U n="object_list">
          <L n="object_list">
            <T>14906<!--GameObject: object_Food_Chili--></T>
          </L>
        </U>
      </V>
      <U n="state_component">
        <L n="states">
          <U>
            <L n="client_states">
              <U>
                <T n="key">10536816939022308551<!--ObjectStateValue: Chili_ExtraHotAndSpicy--></T>
              </U>
              <U>
                <T n="key">13107449241270223922<!--ObjectStateValue: Chili_NotExtraHotAndSpicy--></T>
              </U>
            </L>
            <V n="default_value" t="reference">
              <T n="reference">13107449241270223922<!--ObjectStateValue: Chili_NotExtraHotAndSpicy--></T>
            </V>
          </U>
        </L>
        <L n="state_triggers">
          <U>
            <L n="at_states">
              <T>10536816939022308551<!--ObjectStateValue: Chili_ExtraHotAndSpicy--></T>
            </L>
            <V n="set_states" t="Set_state_list">
              <L n="Set_state_list">
                <T>15305<!--CommodityBasedObjectStateValue: Quality_Outstanding--></T>
              </L>
            </V>
          </U>
        </L>
      </U>
    </U>
  </L>
</I>
```

XML Injector Installation Detection

A method has been developed to allow the game to display an alert dialog to players if the XML Injector is *not* installed, notifying them that they need to download and install the XML Injector. It seems a simple idea now, but when the need for this feature became apparent it seemed quite impossible to create an XML-only resource that could notify users that a required script was not installed. So special thanks to Triplis, andrew and RabidGamer on my Discord channel for their help in thinking up this method!

The feature works by creating a new DialogDramaNode tuning which is setup to display an alert dialog every 48 Sim hours. When XML Injector version 2 or higher loads into the game the drama node is then disabled automatically, so the alert will only appear if a player does not have XML Injector installed.

The reference standard version of this drama node tuning is included in the Developer's Documentation download in the XmlInjector_Detector.package file. To enable the detection alert, follow these **rules**:

1. Include the Scumbumbo_dialogDramaNode_XmlInjector_Detection drama node in your mod package, along with the required STBL entries.
2. You may copy the existing STBL resources, or create your own custom text string for the dialog.
3. **DO NOT** alter the instance ID of the drama node. It must be 15419684579670968912 (0xD5FDB96D9FDB4A50 hex) in order for the script to properly disable the alert when XML Injector loads. Changing the instance ID will result in the dialog being displayed, possibly multiple times for multiple mods, even if XML Injector is properly installed.
4. **DO NOT** change anything else in the drama node tuning apart from a custom text message, it is important that the drama node itself is standardized.

5. Version 1 of XML Injector will not disable the alert, so if you create a custom text message for the dialog be sure that it specifies that Version 2 or higher of the XML Injector is required to disable the alert.
6. If you create a custom text message please include information on where to obtain the XML Injector.
7. Since this is a standard DialogDramaNode tuning, the syntax required for it may change with future game updates. If changes are required to the tuning it will be reported on the mod download forum as well as on my Discord channel and a new reference standard will be made available as soon as possible.
8. **TEST IT!** After adding the drama node to your package, remove XML Injector from your mods temporarily and run the game. The dialog should appear shortly between 3:00 to 4:00 AM Sim's time. Add the XML Injector back to your mods and run the game again and the dialog should no longer appear.

Since a player may have multiple mods installed which include this detection snippet, only the first mod loaded will "win" the resource override and have its dialog displayed. This is a Very Good Thing™ (see rule 3 above).

Example/Test Mod

The XmlInjector_Test package included in the download is primarily intended to test that the XML Injector is working properly. It also contains examples of each of the tuning types to be used as reference. The three simple custom interactions included in the package are added to:

- The mailbox objects
- All computer objects
- Sim objects
- Phone and relationship panels
- Friendly chat mixers

See the Scumbumbo_XmlInjector_AddInteractionTest snippet (instance 0xB7260EB7059A46B4) in the XmlInjector_Test package to see how all these additions are coded into one single snippet such as would be used in a real-life larger mod.

Troubleshooting

If your interactions aren't getting applied the way you think they should be, the first step should always be to ensure that the snippet script is working properly. Make sure the XmlInjector_Test package is in your mods folder and try clicking a mailbox, computer, or any of the other methods that the test interaction is injected in the test package.

If the XML Injector isn't working at all, make sure the script is installed properly, etc. Check to see if the script has been updated by checking the download web site, particularly if the game has been updated with a new patch.

Secondly, make sure that your interactions are coded properly and getting loaded into the game in the first place. A bad interaction that is rejected by the game will not be properly loaded and the XML Injector cannot add something that isn't loaded. Your interaction may not be showing up due to a global test that is not being met, or any number of other reasons.

You can verify that the interaction is being applied to objects by using the Sims Log Enabler (available from http://modthesims.info/d/606667/sims-log-enabler.html) with the ENABLE_LOGGING_AT_STARTUP option set to True in the sims_log_enabler.py script. An XmlInjector.log file will be created in your log enabler folder containing the information on what XmlInjector tuning classes are being processed at startup, what objects get modified, and what interactions are added to those objects.

In addition, the Tuning.log file may contain relevant information on any tuning errors (mismatched or invalid tags, etc.) that are encountered while attempting to load your XML tunings. You will likely see a lot of tuning errors if you have a number of mods installed, but searching the file for the (decimal) instance ID number of your XmlInjector snippet should find any relevant tuning errors quickly.

It can help to add the interaction in the normal method of overriding the game XML in your package to test to ensure that your interaction appears properly. This would conflict with other mods, but can be quite useful for testing. Once you are sure the interaction is working normally, that override can then be removed and the XML Injector used.

If an exception occurs during processing of a snippet, the exception will be logged to the XmlInjector.log file, the exception will not appear in the LastExceptions.txt file in order to minimize any chance of interfering with user's game play. If you do see an exception occur, please send me a copy of the log file and your XmlInjector snippet so that I can attempt to determine how to avoid that error in the future.

FUTURE PLANS

New features for the XML Injector are being suggested and development of them will proceed as time permits. My number one goal is to keep it updated in the event bugs are found, or game patches cause it to no longer function properly. However, I will be happy to continue to consider any additional functionality that the modding community wants added to the snippet. Suggestions should be useful for the modding community to *eliminate the need to include scripts with their mods* with the goal of reducing the possibility of tuning mod conflicts in order to be seriously considered.

I can be contacted on Mod The Sims, or on my Discord server “Scumbumbo Werld”. The #xml_injector-tech channel on my Discord channel is a very good place to get help with the XML Injector and suggest possible new features.

- http://www.modthesims.info/member.php?u=7401825
- https://discord.gg/xV9VYuw

A GitLab project has been setup with the latest development sources. These are not official releases. They may have unstable and incomplete features and new tuning features could change dramatically or be removed at any time. The GitLab development sources would primarily be of interest to those who wish to hack at adding new features. Joining the Discord channel above to discuss modifications is an absolute must for this!

- https://gitlab.com/scumbumbo/xml-injector