KovaCommunity

# Overview

At its core, KovaCommunity is a set of classes designed to abstract control over a community presentation. It is based on a Model|View|Controller relationship:

**Model:** The class mirroring the data coming in from the given API urls set in CommunityController. These models are automatically generated from fields that exist in the JSON response using Newtonsoft’s JSON plugin. For each deserialized JSON input, the plugin will attempt to create a matching Model object with the JSON object’s properties.

**View:** The class responsible for Unity visualization. This is comprised of at least a GameObject but can encapsulate child objects within the script as well.

**Controller:** For classes with complex data->visualization transformations, an intermediary of a Controller is introduced. The role of a controller is to bridge the gap between the data in the model and the format needed for the visualization class as well as respond to actions coming from the View. This is only currently used for Community, which also implements some controller-like functionality for Lots and Models.

## Setup

1. Set the community RID string in the Inspector under the root Community object. This is what is pulled in order to generate the correct API urls for Kova.
2. In Community>Models, add each desired model named according to ModelRID from Kova. These should have the ModelView component attached, and the ModelRID there should also be set appropriately.
3. In Community>Lots, add each desired lot named according to LotID from Kova. Each lot should have a LotView component attached. Set the LotRID accordingly.
4. Under each lot object should be GameObjects associated with the realtime view for the given lot. Camera Anchor and Model Anchor serve as placeholders that will contain their given Camera/Model for the lot. Set the fields in the lot object to point to these objects appropriately. Under Labels are two TextMesh objects – DisplayName and Status. These are currently used to display the lot’s RID and status data, respectively. Set the lot’s fields for these appropriately as well. ButtonObj should point to the Space object, which will be explained next.
5. Each Space object represents the actual 3D lot area for a given lot. It should have the correct mesh, position, and orientation. This object should have the LotButtonView component attached. Set the Lot field of the LotButtonView component in the Inspector to the parent lot.

Done! Repeat for as many lots as desired. Modify the View scripts as desired in order to display other fields from the associated Model. Please refer to the KC-integrated Turnberry map for a live reference.