oneIoTa User Guide

Revision	Date	Comment
1.0	February 8, 2016	Published User Guide
1.1	April 24, 2016	Updated with OCF name; addition of Derived Models and other additional sections

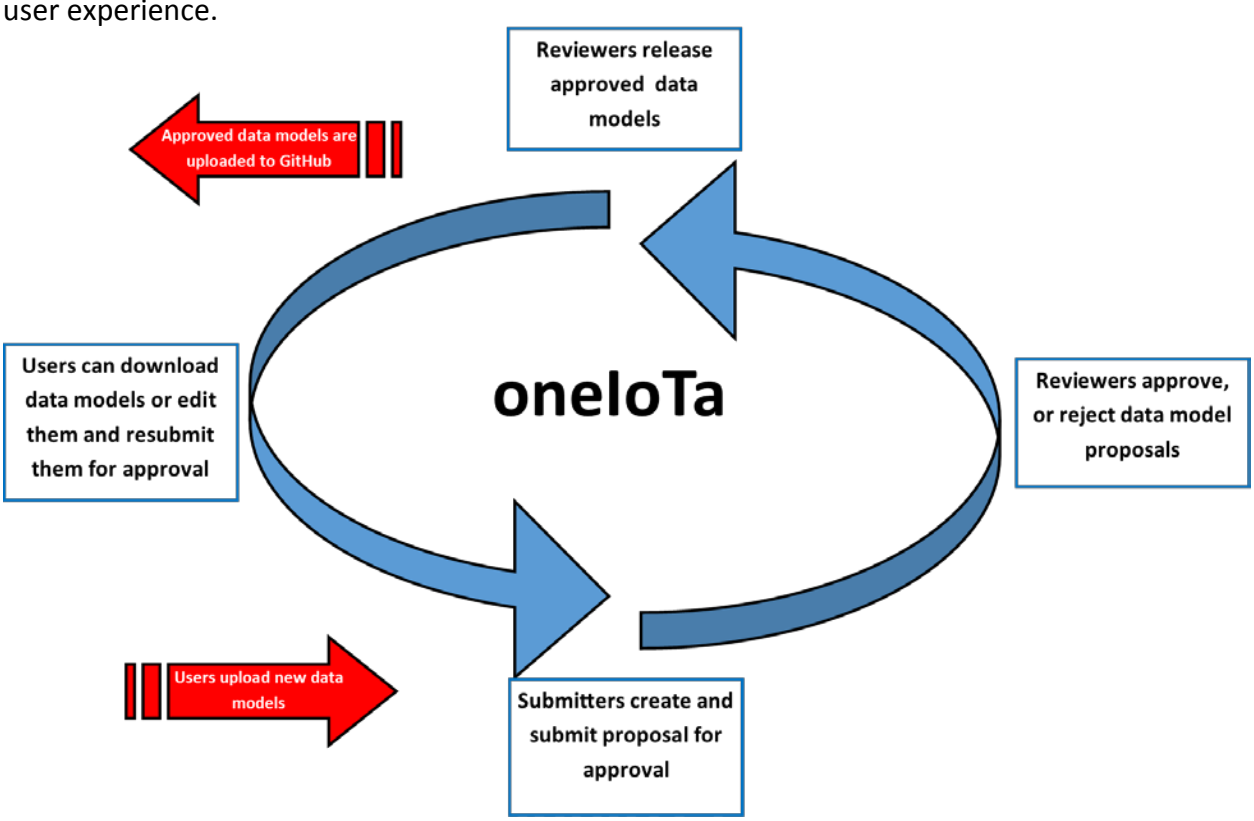
Table of Contents

Contact Information:

Open Connectivity Foundation
3855 SW 153rd Drive
Beaverton, Oregon 97003
Phone: +1.503.619.0673
Email: oneiota@openconnectivity.org

Introduction

The Open Connectivity Foundation (OCF) was established with the goal of providing a common scalable standard for the Internet of Things with a certification tool that could ensure interoperability and an open-source implementation that could accelerate implementation time. In order to meet these objectives, a RESTful architecture was developed that would limit the system to just five (CRUDN) APIs and a constructive data model that would create complex devices and systems as collections of simpler devices and resources. It was also important to ensure interoperability with other ecosystems in order to expand the market and simplify the user experience.

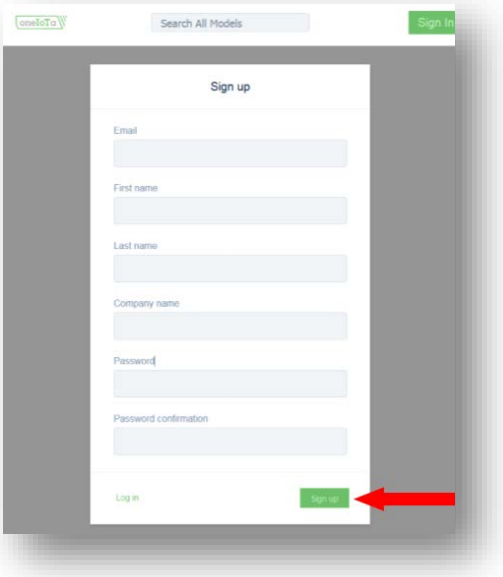


oneIoTa Overview

oneIoTa (one Internet of Things architecture) is essentially an Integrated Development Environment (IDE) that sits at the center of these requirements and delivers on this promise using OCF. It has integrated, syntax-colored, and validating text editors for the needed RAML (Restful API Modeling Language) interface definitions and the JSON (JavaScript Simple Object Notation) schemas that define each model. It supports a crowd-sourced process that allows anyone to submit model proposals, and a back-end approval process that allows multiple organizations to approve models for their particular ecosystems. Finally, it's backed by a git repository so anyone can get the models without ever using the tool if they choose to do so.

User Registration

To register for a user account on the oneIoTa site, go to http://oneiota-production.herokuapp.com/users/sign_up, enter your contact information and create a password. Your username is your email address. If you forget your password, click **Forgot your password?**



Once you click **Sign-Up**, you will be sent a verification email to the email address you entered. In the account verification email, there will be a link titled, "**Confirm my account**," which you will need to click to authenticate your user account, and gain access to the site. If you do not receive the verification email, please check your spam folder.

License Agreements

All new users will automatically receive *Viewer* status (see user status definitions below). Before downloading a released data model, you must accept the **License Agreement**. Before a user can become a *Submitter and* contribute new data models, you must accept the **Contributor Agreement**. A notification will then be sent to oneIoTa admin, who will verify your status as a *Submitter*.

User Types

There are four types of users supported in the oneIoTa tool. User rights are cumulative. The roles are listed here from most restrictive to least restrictive.

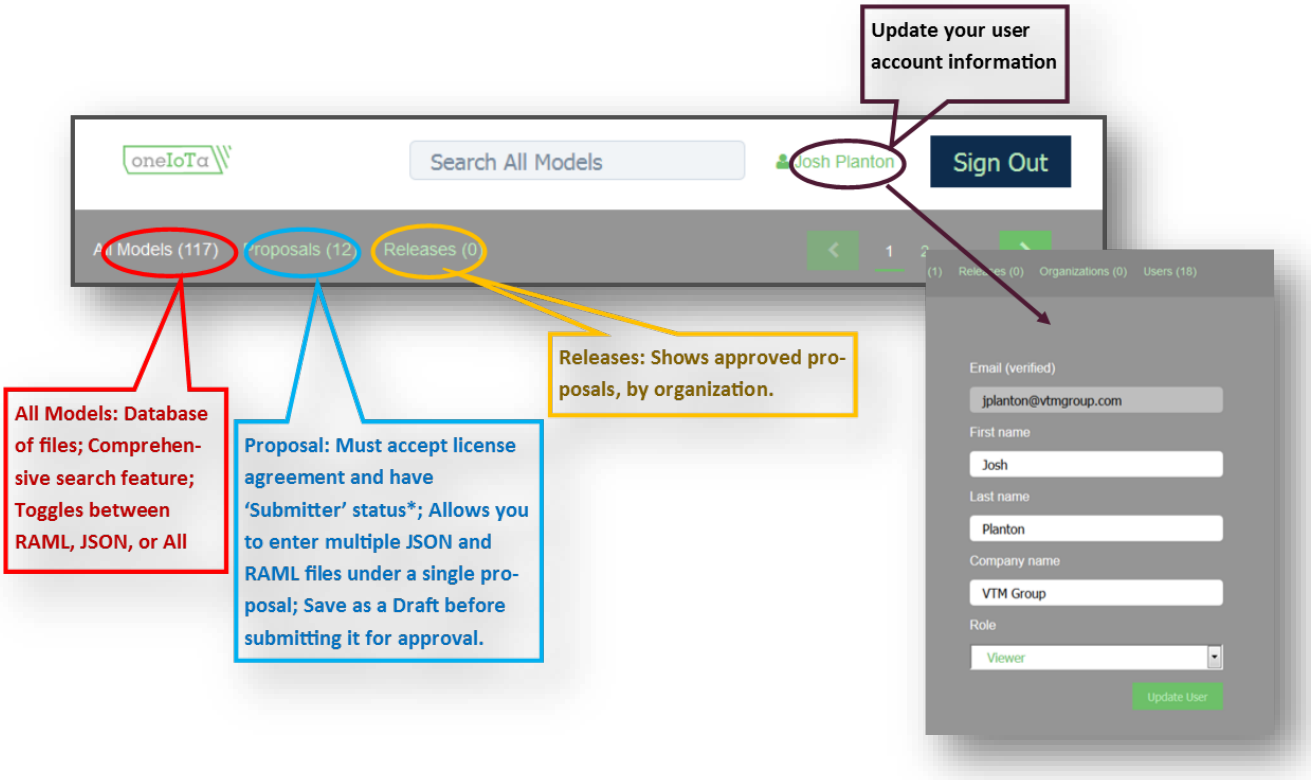
- **Viewers** – Have the ability to view and download approved and pending data models. Included files are discovered and linkable, so the device model tree can be easily traversed. The models list can be filtered by approval status and organization with the filter drop-down at the top right of the models screen.
- **Submitters** – Have agreed to the Contributors Agreement and are able to create and submit new data model proposals to OCF, or other approved organizations for their consideration. Submitters are able to save drafts of their proposal and return to them later.
- **Reviewers** – Reviewers, a.k.a "Managers," of their respective organization, have the authority to approve or reject submitted proposals based on the organization's processes, outlined within their internal procedures. Once a proposal is submitted, a reviewer can change the status of a model from "pending" to "approved." A reviewer can also tag a release. Typically, there will only be a handful of reviewers for a particular organization and they will determine their own process for reviewing models and tagging releases.
- **Admin** – Has authority to approve new user status and review proposals and all data models.

Data Model Status Types

- **Approved** – device definition is approved and will be added to the list of official devices supported by the organization
- **Pending** – device definition is of interest, but has not yet been approved, still has errors or omissions, or is otherwise not ready for approval
- **Rejected** – device definition is duplicative, non-compliant, or otherwise not of interest to the organization

Getting to Know the Site

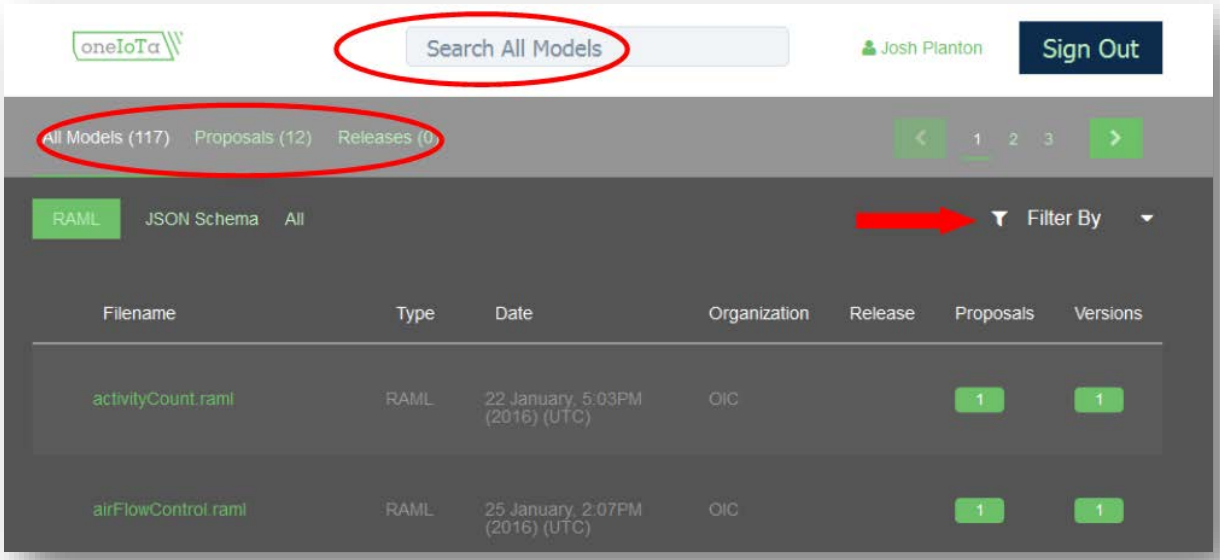
Once you login to the site you will see three main tabs across the header: *All Models*, *Proposals*, and *Releases*. In the upper right corner you will also find a link to your user profile, which will allow you to edit your user profile, and view your current user status.



All Models

The *All Models* area is a list of the individual data model files, is organized by file type (RAML, JSON Schema), and can be *Filtered By* approval status or organization. The header search feature allows users to search based on specific data model types (e.g. light, thermostat, etc.).

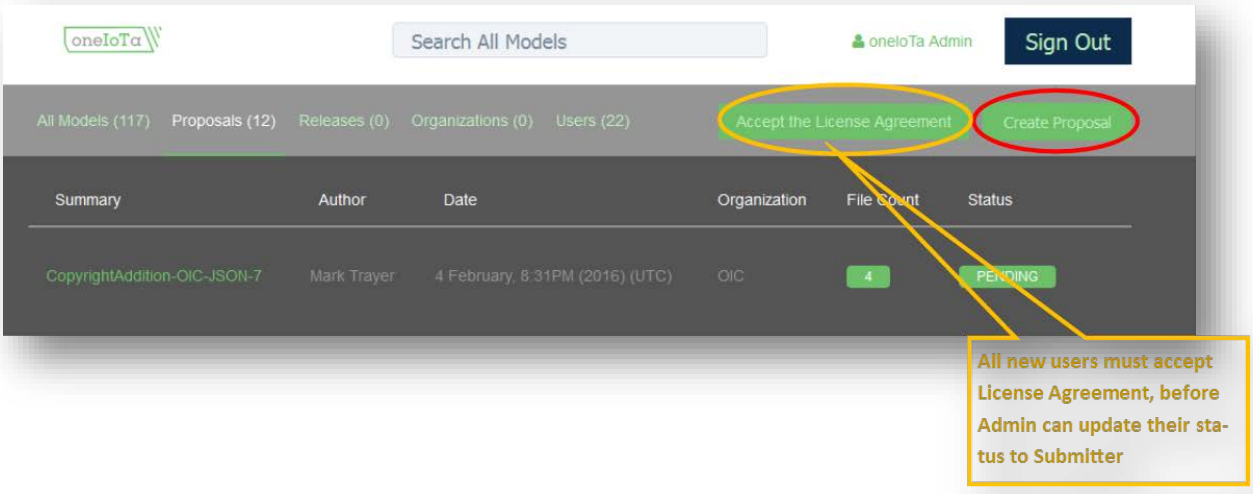
Users with *Viewer* status are able to sign on to the site and see this information, and copies of the approved RAML and JSON released files are also saved on the OCF GitHub repository: https://github.com/OpenInterConnect/IoTDataModels. Users will be able to see the specific revision numbers for each data-model in the far right column titled *Version*.



Proposals

Users with *Submitter* status will be able to login to the oneIoTa site, click on the *Proposals* area, and upload data models directly from their desktop to the site. If a user only has *Viewer* status, they must click on the *Proposals* area, and accept the **Contributor Agreement**. Admin will then update the user's status to *Submitter* (this could take 24-48 hours).

Once a user receives *Submitter* status, that user can create proposals. To do so, the user should click on the **Create Proposal** button in the upper right hand side of the screen (see below).



If you have any questions, please contact oneiota@openconnectivity.org

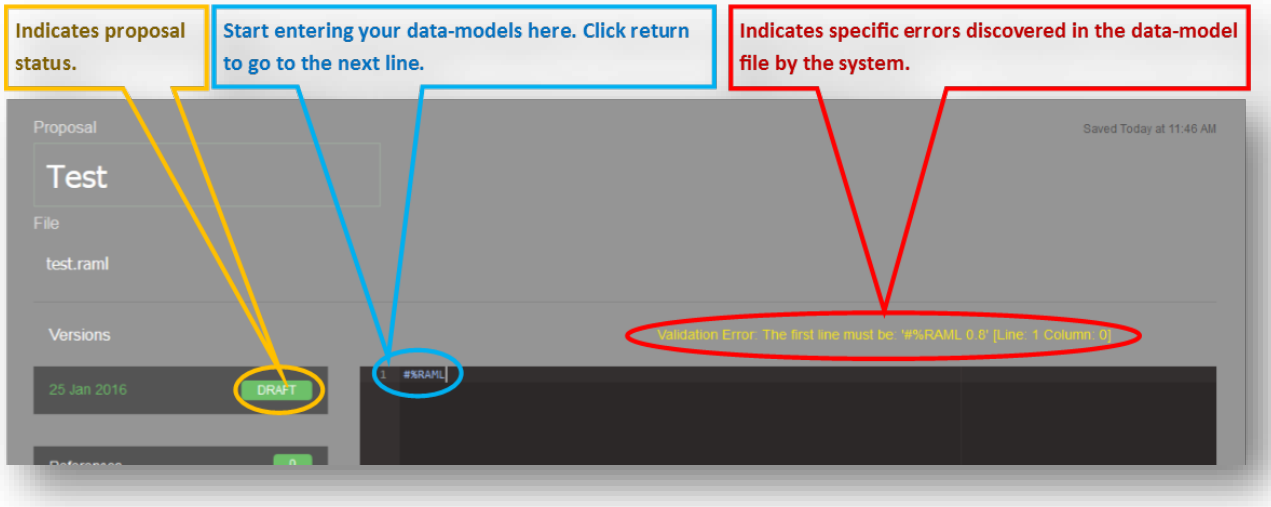
Entering Data Models:

A *Submitter* just clicks on the "Create Proposal" button to create a new model. A proposal can contain one or more models. The "Proposal Summary" is an optional description field that will eventually show up in the git repository if the proposal is approved. When a file name is added (all files must either have the file extension ".raml" or ".json"), the file is automatically created and an editing window is opened. The submitter can simply begin entering code, start with a default template, or paste in copied text. Files can also be batch-imported via an import window that allows for the selection of multiple files in a single import instance (see upload data models section below). If a file being added to or imported to a proposal already exists then oneIoTa automatically creates a new version of that file.

Tip: ***Users can, and are encouraged to, review other data models on the oneIoTa site, to understand the recommended syntax to obtain proposal approval. If you have specific questions about approved data model syntax, please email oneiota@openconnectivity.org.***

Note: *File names must end in .raml or .json for the site to allow the proposal to be created.*

Once the file has been created, you will see the proposal info at the top of the page, the file name just below that, and you will be able to start entering code into the black window to the right of the proposal information.



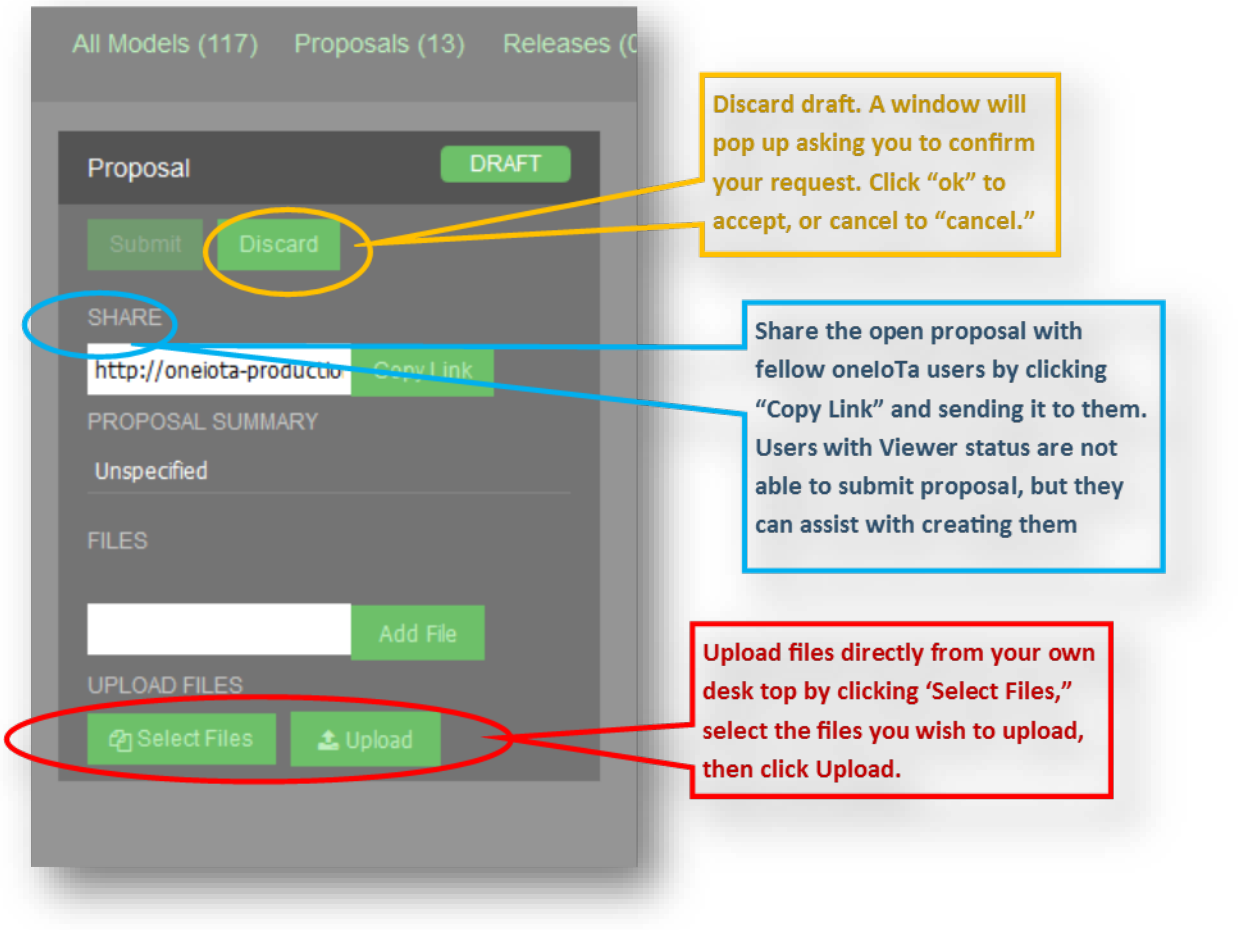
Once you start entering information into the data model entry window, users will see orange text appear just above the window. This alerts the submitter of possible errors in the data model code that the tool is detecting. The edit window understands the correct syntax of the RAML and JSON schema files, and will color-code and validate the syntax as the User enters it.

Note: *Data models will save automatically and the date/time-stamp in the upper right corner of the screen indicates the last time the file was saved.*

Uploading Data Models:

To upload files directly from your desktop, open a new proposal, click **Select Files**, which will open a dialog box on your screen, and search for the specific files. Users can filter files by 'all', '.json', '.raml', then select the files required and 'open.' Next you will need to click **Upload**, at which point all of the files will be uploaded to the proposal.

If you are working from a file you originally downloaded from the oneIoTa site[1], select the .raml or .json file with a name matching the existing .raml or .json file in the data model, click Upload, and the tool will create a new version of the existing proposal using the uploaded file.



Sharing Proposals:

Submitters can also "Share" their proposals with other users on the oneIoTa site. To do this, the *Submitter*, will need to copy the share link from the left hand navigation bar, and send it to any other users they wish to share it with. Submitters are able to share draft proposals with users,

[1] See "Download Data Models" section below

even if they only have *Viewer* status, but only *Submitters* are able to submit the proposal for *Reviewer* approval.

TIP: Submitters who share draft proposal with other users, may want to be cautious about who is editing files, as it is their responsibility as the Submitter, to make sure the proposals is compliant. It is recommended to work offline, if consulting with multiple users, then uploading all files at once to the proposal area for final review before submitting.

Submitting Proposals:

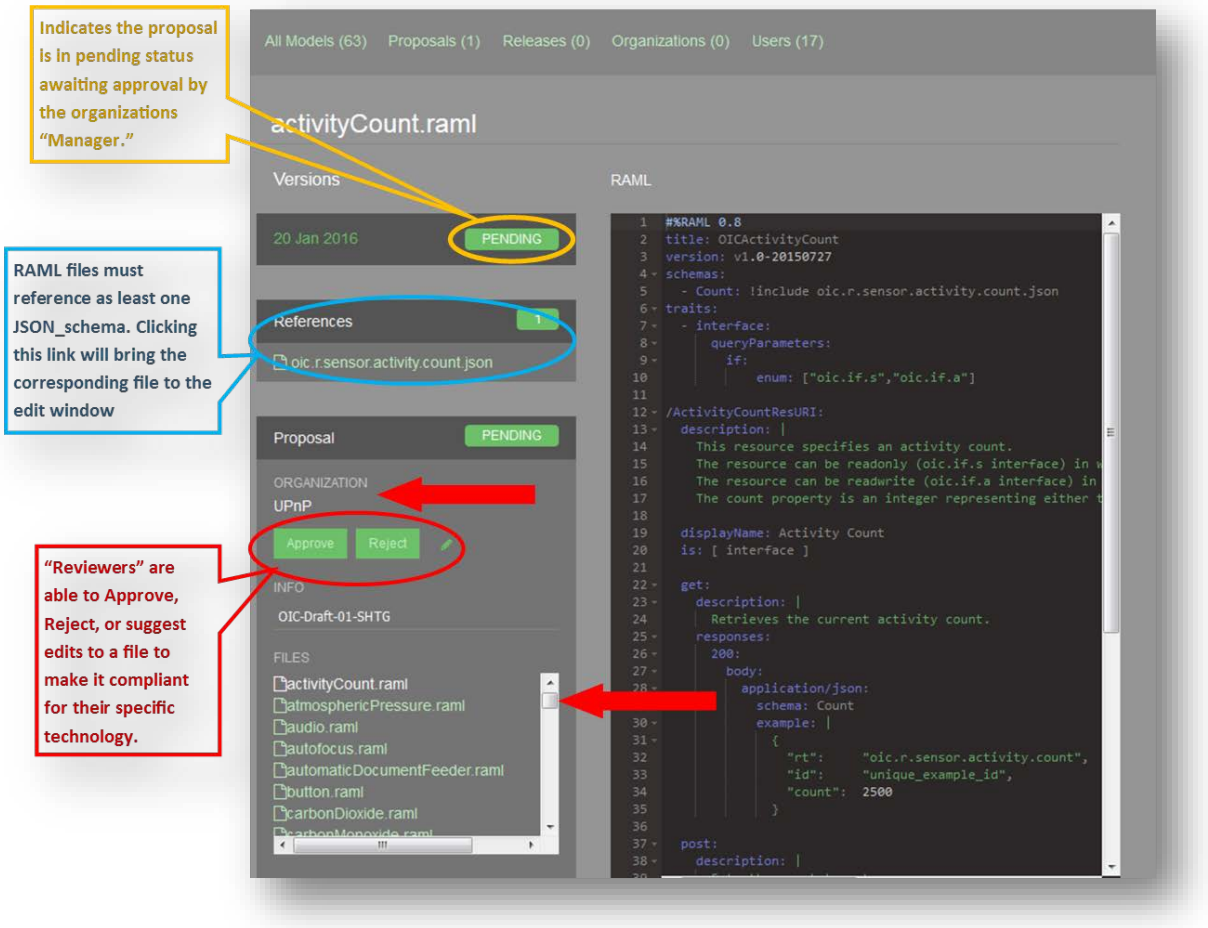
After the user is satisfied with their draft proposal, they will need to click **Submit**. Depending on the organization to which the *Submitter* is submitting the proposal, the *Reviewer* of that organization will receive an email notification that a new proposal has been submitted, and requires their review, prior to approval. Proposals can be submitted to multiple organizations at once, but all proposals must be compliant with RAML and JSON schema as determined by OCF.

Note: *OCF is the only accepting organization at the moment, so all submissions must comply with OCF specifications.*

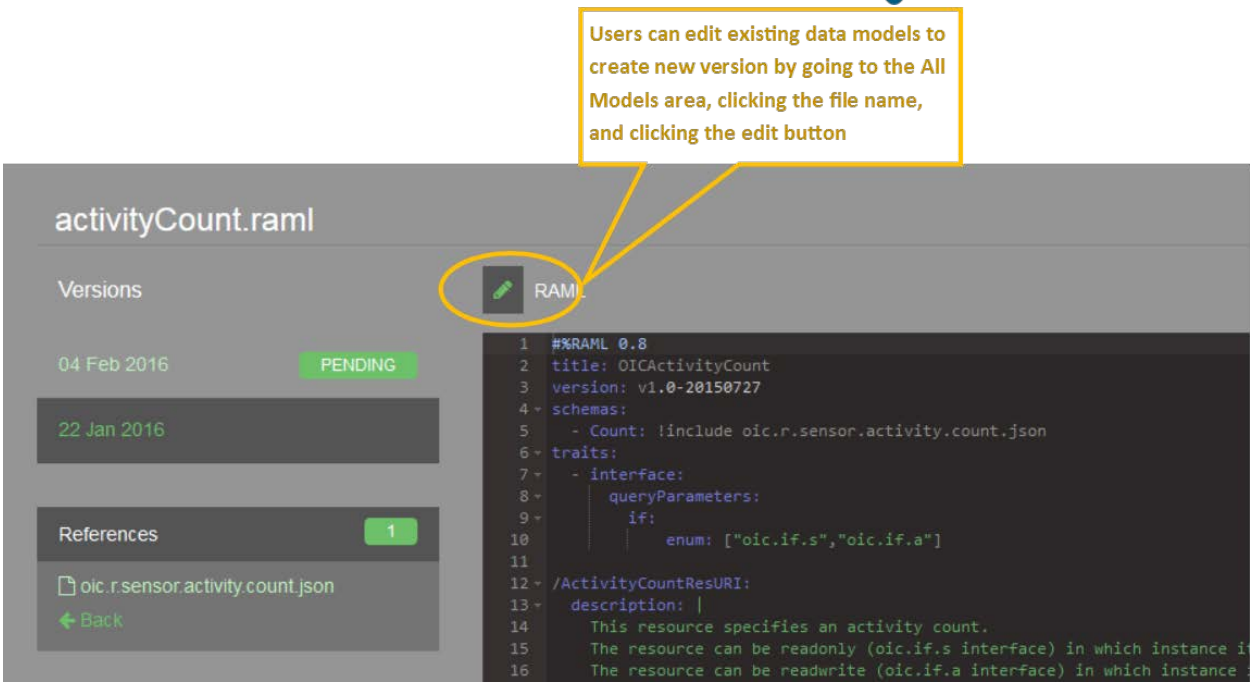
Once the proposal is submitted, the *Reviewer* of the organization will review the proposal. Within the *Reviewer's* window, you are able to see which organization the submission was submitted to, and the list of files associated with each proposal in the left-hand side bar. *Reviewers* can suggest edits within each data model file of the proposal. When a proposal is in the pending status, the *Submitter* is unable to edit it File references can be followed by clicking them in the reference box.

Reviewers can navigate through each file separately, and return to the previous file by hitting the **←Back** button in the left-hand side bar. Once a Reviewer has finished reviewing all files within the proposal, they can either select to Approve or Reject the proposal.

Please Note: *If a proposal contains several files, all files must be accepted or rejected at the SAME TIME. Each organization will be responsible for creating their own internal checks and balances to make sure data models with incorrect code are not approved for release.*



Once a *Reviewer* approves the submission, the proposal will be placed in a release queue. The *Reviewer* will then need to go to the Release area and tag the approved submission. If the *Reviewer* rejects the data model, a notification email will be sent to the submitter notifying them of the reviewer's decision, and when applicable, revision suggestions.



Approval Process

OCF's IoT Data Modeling *Reviewer* group will periodically review the pending models and determine if the proposal should be approved. If approved, no status indicator will show and the proposal will be pushed to the git repository. At that point, it can be pulled from the git repository or viewed in oneIoTa, but it can no longer be edited. It will be tagged as part of the next release unless a new proposal is created and accepted before the release is tagged.

Releases

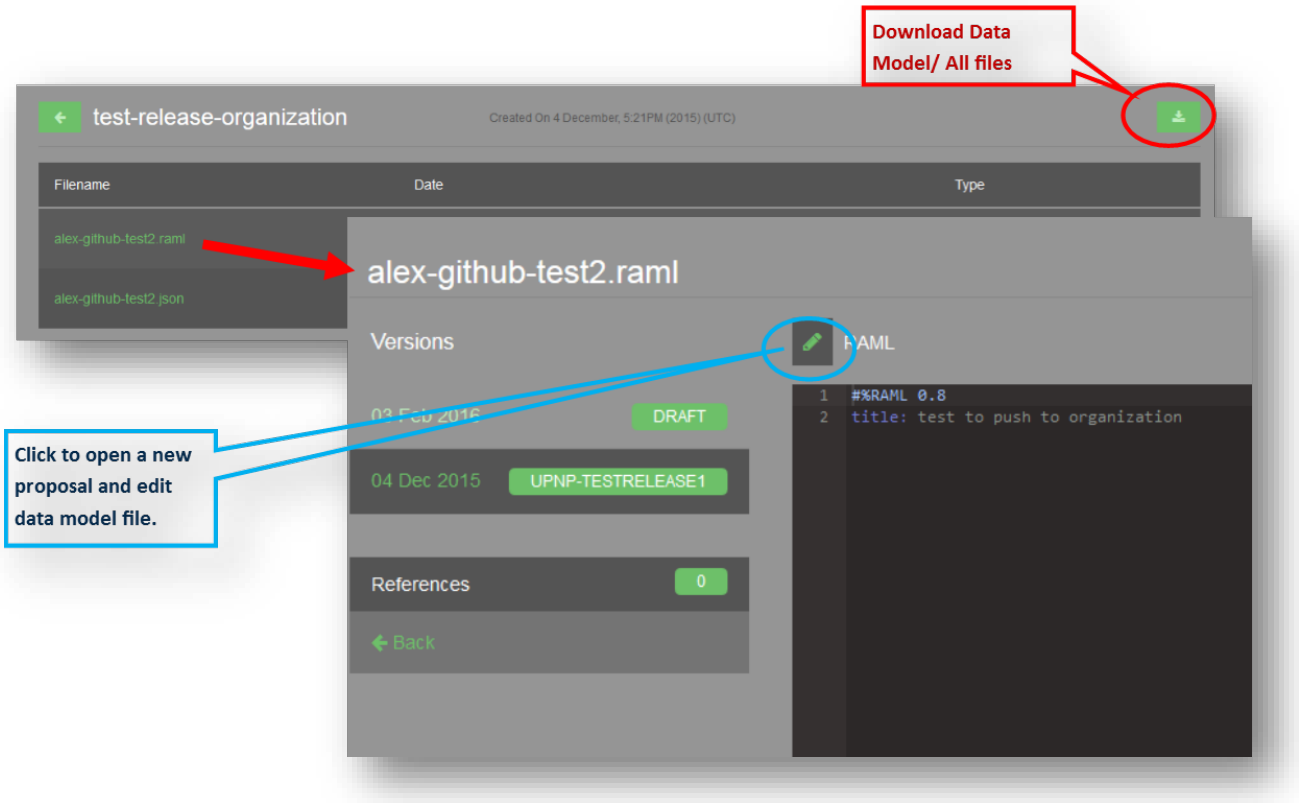
To release an approved proposal *Reviewers* must tag the proposal, using a specific naming convention as directed below, and only under their own organization's name. Once the proposal is released, it will appear in the "Release Area." Once a proposal has been approved, it will be pushed to the OCF GitHub repository, where users will be able to download it for their own use.

Note: *The "Proposal Summary" information is no longer showing as a reference field.*

Downloading and Editing Released Data Models:

User are able to edit existing data model files by downloading them directly to their own desktop or within the oneIoTa tool. To download a released data model, users must click on the released data model, then on the download button in the right hand corner. Users will then be prompted to accept the **License Agreement**. The user will need to click the download button again, after accepting the **License Agreement**, which will open a window to download the selected data model files in a zip file format.

If you have any questions, please contact oneiota@openconnectivity.org



If a user is wants to edit an existing data model file they will click on the release, then on the specific file name. Once the data model file is opened, the user can click the edit button at the top left corner of the data model code window which will open up a new draft proposal with the same data model code and file name as the prior data model file.

Git Repository

The oneIoTa tool only commits to and tags a git repository master. oneIoTa maintains a separate internal database for its functionality. The approving organization will determine which repository it wants to use. In the case of OCF, the repository is open to all for pulling the models, but contributions back to the master must be done through oneIoTa.

Derived Models

One of the major obstacles with the Internet of Things is that there are many incompatible ecosystems. While some have addressed this problem by writing various types of converters between ecosystems, this becomes hard to scale. The OCF architecture and the oneIoTa tool use OCF data models as a “common” data model and a derived version of these models to define conversions between OCF and other IoT ecosystems. This makes all derived models interoperable with OCF and all other derived models through (at most) two conversion steps. As with all OCF data models derived models can be machine-read to automatically create code stubs.

Derived models use standard JSON schema syntax. Fundamentally, derived models provide a conversion mapping between OCF data models and similar data models in another IoT ecosystem. These conversions can be very simple (as in just a property name conversion) to extremely complex (as in converting between different numbers of properties with complex mathematics). Examples of the various conversions are described below.

Simple Mapping

Simple mapping just converts between different field names. Simple mapping accounts for field mappings between ecosystems. It can be used in combination with direct mappings as well as more complex mathematical conversions. In the example below, the derived model defines the field “lightness” to map to the OCF field “brightness.” It also maps the derived ecosystem field “rgb_color” to the three OCF fields “red,” “green,” and “blue.”

```
{
   "properties": {
      "lightness": {
         "type": "number",
         "oic_conversion": {
            "oic_alias": "brightness"
         }
      },
      "rgb_color": {
         "type": "string",
         "oic_conversion": {
            "oic_alias": ["red", "blue", "green"]
         }
      }
   }
}
```

Simple Conversion

Simple conversion defines mathematical conversion between simple mappings. It does this using the two fields “from_oic” and “to_oic.” The mathematical conversion is defined in a string. The string is not validated. In this example, the field “darkness” in the derived model is converted to the field “brightness” in the OCF model by subtracting it from 255. A script that creates code stubs could read this model to identify input and output variables and use the conversion strings as comments that a coder could reference to properly implement the conversions in any programming language.

```
{
  "properties": {
    "darkness": {
      "type": "number",
      "oic_conversion": {
        "to_oic": "brightness = 255 - darkness",
        "from_oic": "darkness = 255 - brightness"
      }
    }
  }
}
```

The following example is a bit more complex and demonstrates the use of a list of strings. The derived model field “darkness_percentage” is converted to and from the OCF field brightness with two conversion steps for each direction.

```
{
  "properties": {
    "darkness_percentage": {
      "type": "number",
      "oic_conversion": {
        "to_oic": [
          "darkness = 255 * darkness_percentage",
          "brightness = 255 - darkness"
        ],
        "from_oic": [
          "darkness = 255 - brightness",
          "darkness_percentage = darkness_percentage / 255"
        ]
      }
    }
  }
}
```

Complex Conversion

Sometimes conversions are too complex to capture well in a from_oic/to_oic syntax. One example is conversion between RGB and HSV color schemes. This conversion requires intermediate variables and several conditional assignments. The example below shows how this could be done using the oic_conversions field. The from_properties and to_properties define the mappings while from_steps and to_steps define the list of conversion steps to be taken. Again, the strings defined in the steps are not validated, but are used as comments when code stubs are generated so the conversion process can be implemented in any programming language.

```
// OCF Model
{
  "oic_conversions": [
    {
      "from_properties": ["Hue", "Saturation", "Value"],
      "to_properties": ["Red", "Green", "Blue"],
      "from_oic_steps": [
        "C = V x S",
        "H_prime = H / 60 degrees",
        "X = C * (1 - |H_prime mod2 - 1|)",
        "if H is undefined: R, G, B = (0, 0, 0)",
        "if 0 < H_prime < 1: R_1, G_1, B_1 = (C, X, 0)",
        "if 1 < H_prime < 2: R_1, G_1, B_1 = (X, C, 0)",
        "if 2 < H_prime < 3: R_1, G_1, B_1 = (0, C, X)",
        "if 3 < H_prime < 4: R_1, G_1, B_1 = (0, X, C)",
        "if 4 < H_prime < 5: R_1, G_1, B_1 = (X, 0, C)",
        "if 5 < H_prime < 6: R_1, G_1, B_1 = (C, 0, X)",
        "m = V - C",
        "R, G, B = (R_1 + m, G_1 + m, B_1 + m)"
      ]
    }
  ],
  "properties": {
    "Red": {
      "type": "number"
    },
    "Green": {
      "type": "number"
    },
    "Blue": {
      "type": "number"
    }
  }
}
```

One to Many

The following example shows how you can convert from one ecosystem with a single field to a derived model with multiple fields.

```
// OCF Model
{
  "properties": {
    "rgb": {
      "type": "string"
    }
  }
}

// Derivative Model
{
  "oic_conversions": {
    "to_oic": [
      "rgb = r + b + g"
    ]
  },
  "properties": {
    "red": {
      "type": "string",
      "oic_conversions": {
        "oic_alias": "rgb",
        "from_oic": ["..."]
      }
    },
    "blue": {
      "type": "string",
      "oic_conversions": {
        "oic_alias": "rgb",
        "m_oic": ["..."]
      }
    },
    "green": {
      "type": "string",
      "oic_conversions": {
        "oic_alias": "rgb",
        "from_oic": ["..."]
      }
    }
  }
}
```

Many to One

This example shows how to convert between an OCF model with several fields to a derived model with a single field.

```
// OCF Model
{
  "properties": {
    "red": {
      "type": "string"
    },
    "blue": {
      "type": "string"
    },
    "green": {
      "type": "string"
    }
  }
}

// Derivative Model
{
  "properties": {
    "rgb": {
      "type": "string",
      "oic_conversions": {
        "oic_alias": ["red", "blue", "green"],
        "from_oic": ["rgb = red + blue + green"],
        "to_oic": [
          "red = ...",
          "blue = ...",
          "green = ..."
        ]
      }
    }
  }
}
```

Derivative Model Validation

As with common “OCF” models, derived models are validated against JSON syntax. In addition, derived models are validated to ensure they are properly referenced to an OCF model. The derived model must include all the fields of the OCF model from which it is derived. These fields consist of oic_alias and from_properties within the oic_conversoins property.

This means that if a relevant model does not yet exist in OCF, it must be proposed and accepted by OCF before a derived model can be created in some other ecosystem. This expands the

models within OCF and guarantees that the derived model with work with OCF models and all other derived models.

What's Next?

Derived models in OCF are an extremely powerful solution to interoperability between ecosystems in the Internet of Things. They are also very flexible in being able to support RESTful as well as other architectures. A prototype has been built showing interoperability between UPnP, AllSeen and Philips HUE light bulbs being adjusted in unison from a UPnP control point.

While oneIoTa is currently only populated with OCF models, derived UPnP models will soon be added. Other ecosystems interested in using oneIoTa and adding their models to the interoperable OCF ecosystem should contact OCF.