School of Information Technology and Mathematical Sciences

COMP 3023 Software Development with C++

Individual Project

~~Networked~~ Asset Manager

Introduction

This document describes the Individual variant of the Networked Asset Manager Group Project in response to the issues around group organisation. Each group may choose to continue the Group Project with the alterations described in the accompanying Group Project Amendment document. Alternatively, your group may disband and each student may work on the Individual project instead. You will need to refer to the original specification for the bulk of the requirements, this document contains only the changes over the original specification.

If any parts of this specification appear unclear, please ensure you seek clarification.

Networking

There is **no need** to complete a networking component for the Group Project. That is, the following sections (and their subsections) of the original document can be disregarded:

- Networking
- Messages

Therefore, the following classes **do not** have to be implemented **nor** included in the design (UML diagram) and any existing version of them can be ignored or deleted:

- `NetworkedAssetRegister`
- `NetworkManager`
- `MessageFactory`
- `PlainTextMessageFactory`
- `Message`
- `RequestMessage`
- `ResponseMessage`

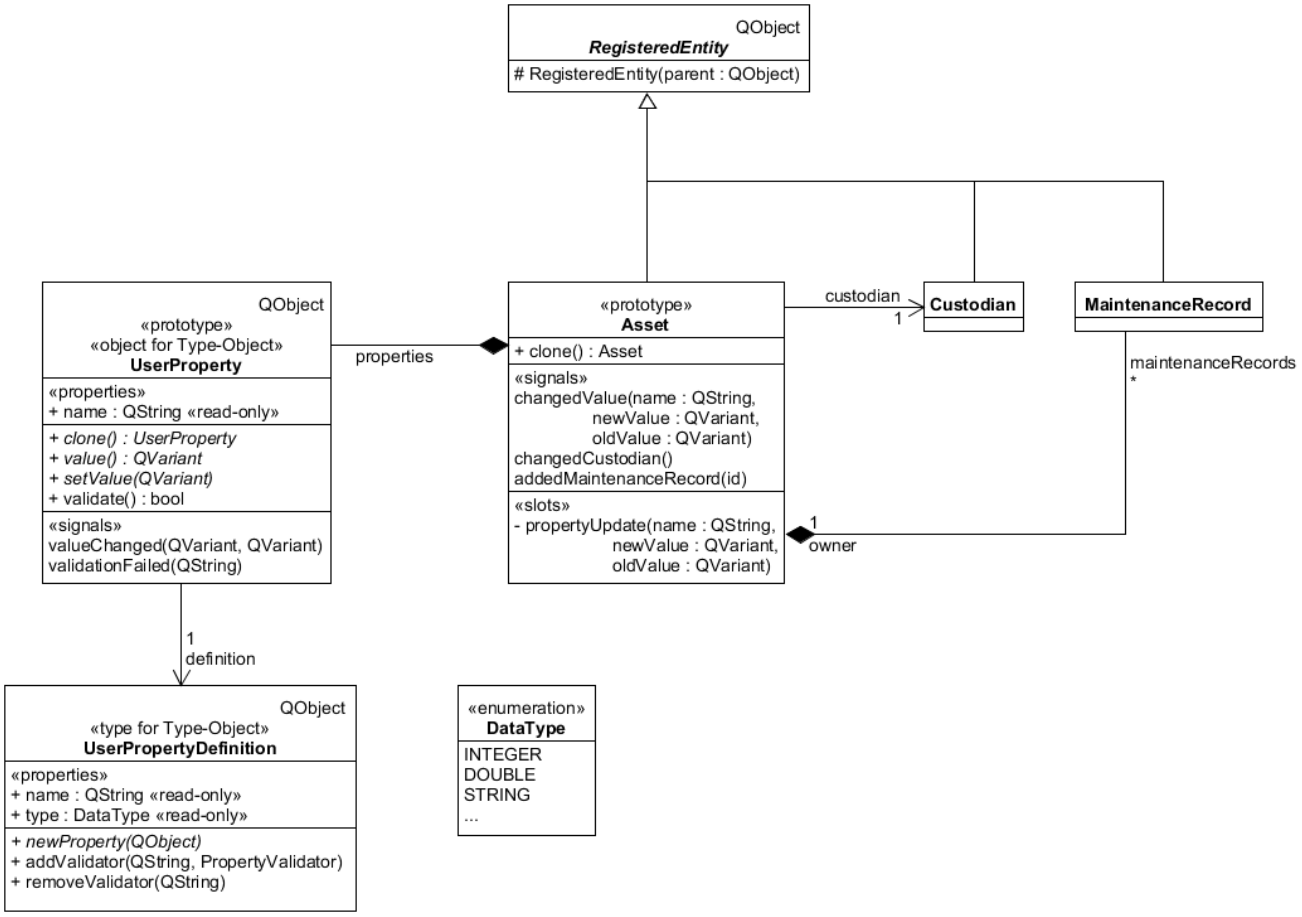
Assets and their Properties

To reduce the scope of the application, the `AssetType` class will be removed. All references to `AssetType` from the original Group Project can be disregarded and the class **does not** have to be implemented **nor** appear in the design (UML diagram). This requires some rearrangement of the core structure as `UserPropertyDefinitions` will no longer be present on the `AssetType`. The new structure is illustrated below.

The consequences of this change are as follows:

- There is only one instance of the Type-Object pattern (`UserPropertyDefinition`/`UserProperty`).
- When a `UserProperty` is destroyed its associated `UserPropertyDefinition` must also be destroyed unless it is still referenced by another `UserProperty` (which may occur when as `Asset` it cloned).
- There is no need to create user interfaces for `AssetType`, i.e., **no** *AssetTypes List View*, **no** *AssetType Details View*, etc.

- The user needs to be able copy an existing `Asset` so that they do not have to re-add the same user properties over and over. See section GUI Changes below.
- When updating a `UserPropertyDefinition`, it must only change for the current Asset. This can be achieved by simply creating a new `UserPropertyDefinition` and replace the existing one. Other `UserProperties` referencing the old `UserPropertyDefinition` must remain unaffected.



GUI Changes

The following views are **not** required:

- *View Asset Types*
- View/Edit Asset Types

Additionally, the *Main Menu/Dashboard* does **not** need to provide access to the removed views.

As `AssetTypes` are not included in this variant of the project the cloning of an `Asset` cannot be performed by an `AssetType`. Therefore, the user **must** be able to ***copy*** an `Asset` (i.e., clone it) to simplify the creation of new `Asset`s with existing sets of properties and values. This functionality must be available from the *Asset Details View* and the *Assets List View*. Selecting the *copy* option **must** take the user to the editable details of the ***new*** Asset object, which will only be stored in the register if the user confirms the edit.

When specifying the `UserProperties` of `Assets`, the user must be presented with selections for the type, precision, and validators (minimum, maximum, and enumeration) of the property, which will be stored in the `UserProperty`'s associated `definition`.

To support saving the register content to file, the *Login Screen* must contain a field to specify the file to load. The user must be able to select the file location through a File selection dialog. After the singleton `AssetRegister` instance has been retrieved, the specified file will be loaded into the register.

When the application is closed, the user must be prompted (Yes/No) to save the database. If the user selects 'Yes' a File selection dialog will be displayed for the user to select the location and name of the file.

Optional: the file from which the entities were loaded may be kept in memory to facilitate quick saving (i.e., set the initial value of the File selection dialog) to the same file at application close.

Optional: the user may be able to periodically save the stored entities to file without exiting the application.

Storing Assets to File

In lieu of the networking component, the Asset Manager must be able to store its database of `Assets`, `Custodians`, and `MaintenanceRecords` to file when the application closes and load the file at application start-up. The file format will be a simple, plain-text format (described below) and all entities will be stored in a single file. To ensure correct resolution of dependent entities, the entities must be stored in the following order:

- Custodians
- Assets
- MaintenanceRecords

File Format

The following describes the data format of the offline storage file. It requires that the objects be serialised into a plain-text format and be deserialised on application load. You must determine an appropriate method of serialisation and deserialization: for example, you may use QTextStream and overloaded input/output operators (`>>` and `<<`, respectively) to read and write entities from/to text and subsequently converted into a QByteArray (see QString::toUtf8()) for consistent storage. Note: if you take this approach you will need to overload the operators for `QTextStream` not `std::istream`/`std::ostream`; there are some differences between them and the manipulators they support, so you will need to read the relevant documentation.

The output file will contain ***one entity per line***. Each line will be structured as a list of **fields separated by pipe '|' symbols**. To protect against additional pipe symbols breaking the format, fields that contain user-entered text must be encoded using *URL percent encoding*, which is an encoding style used by URLs that encode unfriendly characters using a percent '%' sign followed by a pair of hex digits. Refer to QByteArray::toPercentEncoding() and QByteArray::fromPercentEncoding().

The following sections describe the format for each type of stored entity.

Custodian

The data format for serialising `Custodian` objects is as follows (ignore line breaks in the specification, serialisation is one entity per line):

```
Custodian|<id>|<lastEditedBy:%>|<lastEditTime:iso8601>|<name:%>|<department>|<phoneNumber>
```

Where fields in angle brackets denote attributes of `Custodian`, with special instructions included after a colon ':'. For example, <name:%> indicates that the `name` attribute should be percent encoded and written in that position. The instruction, 'iso8601' instruction for date/time data fields indicates it should be serialised in ISO 8601 format (see QDateTime::toString and QDateTime::fromString).

For example, an instance of `Custodian` may be serialised as:

```
Custodian|cust001|matt|2018-09-14T00:00:00+09:30|Johnny|Maintenance|0880001111
```

Asset

The data format for `Assets` is as follows (ignore line breaks in the specification, serialisation is one entity per line):

```
Asset|<id>|<lastEditedBy:%>|<lastEditTime:iso8601>|<serialNo:%>|<brand:%>|<model:%>|
<purchasePrice>|<purchaseDate:iso8601>|[disposalDate:iso8601]|<custodian.id>|
<properties.size>[|<properties:separated by '|'>]
```

If `properties.size()` is zero ('0'), no `UserProperty` will be output and there will be no trailing pipe '|' symbol.

Each `UserProperty` is serialised as:

```
<name:%>=<value:%>|<type: 'int', 'double', 'string'>[|precision=<precision>]|min=<min
validator>,max=<max validator>,enum=<enum validator>
```

Where <precision> is the precision (number of decimal places) of a double and is only present if the type is 'double' (the square brackets indicate that the entire field is optional and should only be present if a precision value is specified), <min validator> is the value of the minimum value validator if present (otherwise empty), <max validator> is the value of the maximum value validator, and <enum validator> is a colon separated list of the enumeration values (percent encoded). The `type`, `precision`, `min`, `max`, and `enum` values come from the `UserPropertyDefinition` associated with the `UserProperty`. For example,

```
Asset|asset001|matt|2018-09-14T00:00:00+09:30|x23-zz0|ACME%20Co.|x23series%20Pump|100.22|
2018-09-14T00:00:00+09:30||cust001|1|flow=2.5|double|min=,max=,enum=1.0:2.5

Asset|asset002|matt|2018-09-14T00:00:00+09:30|P-101|Pumps%27R%27Us||50.00|
2018-09-14T00:00:00+09:30|2018-09-15T00:00:00+09:30|cust001|0

Asset|asset003|matt|2018-09-14T00:00:00+09:30|Pump3|Pumps%27R%27Us|x23series|50.00|2018-09-
14T00:00:00+09:30|2018-09-15T00:00:00+09:30|cust001|1|flow=2.50|precision=2|min=,max=,enum=
```

Note: %27 is an apostrophe ' character

Note: the blank line separates distinct entities for readability, it should not occur in an output file.

Note: When percent encoding the enumeration values, do not percent encode the colon separators. This allows the values themselves to contain colons (as they will be percent encoded) while maintaining the separator (which will not be encoded).

MaintenanceRecord

`MaintenanceRecords` are serialised as follows (ignore line breaks in the specification, serialisation is one entity per line):

```
MaintenancRecord|<id>|<lastEditedBy:%>|<lastEditTime:iso8601>|<owner.id>|<timestamp:iso8601>
|<performedBy:%>|<maintenanceType>
```

For example:

```
MaintenanceRecord|rec001|matt|2018-09-14T00:00:00+09:30|asset001|
2018-09-20T00:00:00+09:30|maintenance guy 01|Inspection
```

Submission Details

You must submit a ***single ZIP*** archive file containing the following:

- UML diagram (in PDF or PNG format)
- Your Qt Creator project folder including all source files required to compile and run your application, your Doxygen config file, and the version control directory (i.e., the '.git' folder)

The submitted ZIP file **must not** include any generated files, such as object files and executables, or extraneous files, such as the '*.pro.user' file.

Once you have created the zip file according to the specifications, you are required to upload it to the ***Assessment Item 3: Project*** submission via ***LearnOnline***. The deadline for submission is **2 November 2018 11:59 PM**. After which time the submission will be considered late and attract a penalty as detailed in the course outline.