SessionItem

Introduction

SessionItem class is a base element to build a hierarchical structure representing all the data of the application running. SessionItem can contain an arbitrary amount of basic data types, and can be a parent for other SessionItem's.

The tree of SessionItem objects can be built programmatically via SessionItem API, or be reconstructed from persistent content (XML and JSON files, for example).

While being an end leaf in some ramified hierarchy the SessionItem often plays a role of a single editable/displayable entity. For example, a SessionItem can be seen as an aggregate of information necessary to display/edit a single integer number 42 in the context of some view. Then, it will carry:

- An integer number with the value set to 42.
- A collection of appearance flags, stating if the value is visible, is read-only, should be shown as disabled (grayed out), and so on.
- Other auxiliary information (tooltips to be shown, allowed limits to change, and similar).

SessionItem doesn't have any dependencies on any particular GUI library, and doesn't have any signaling capabilities.

The data of SessionItem

The data carried by SessionItem is always associated with the role - a unique integer number defining the context in which the data has to be used. They both came in pairs, and the item can have multiple data/roles defined.

```
// currently supported elementary data types
using variant_t = std::variant<std::monostate, bool, int, double, std::string,
std::vector<double>>;

// convenience type
using datarole_t = std::pair<variant_t, int>;

// collection of predefined roles
namespace DataRole
{
const int kIdentifier = 0;  //!< item unique identifier
const int kData = 1;        //!< main data role
const int kDisplay = 2;     //!< display name
const int kAppearance = 3;  //!< appearance flag
}
```

In the snipped below, the data is set and then accessed for two roles, the display role holding a label and the data role, holding the value.

```
SessionItem item;
item.setData(42, kData);
item.setData("Width [nm]", kDisplay)

auto number = item.data<int>(kData);
auto label = item.data<std::string>(kDisplay);
```

Related files

- `variant.h` contains definitions of `variant_t` and `datarole_t` data types. Check it for all suported elementary data types.
- `mvvm_types.h` defines constants and enums. Check it to see current roles, or appearance flags.
- `sessionitemdata.h` contains the definition of `SessionItemData` class. It is a member of `SessionItem` and carries all the logic related to item's data. Most of methods of `SessionItemData` are replicated by `SessionItem`.
- `sessionitemdata.test.cpp` contains unit tests of `SessionItemData` and can be used as an API usage example.

Inheriting from `SessionItem`

The `SessionItem` class type name is stored in a string variable and can be accessed via the `modelType()` method:

```
SessionItem item;
std::cout << item.modelType() << std::endl;
>>> "SessionItem"
```

This name is used during item serialization/deserialization and during undo/redo operations to create objects of the correct type in item factories (explained in `sessionmodel.md`).

To inherit from `SessionItem` the new unique name has to be provided in the constructore of the derived class. It is convenient to make this name identical to the class name itself:

```
class SegmentItem : publis SessionItem
{
public:
  const static std::string Type = "SegmentItem";
  SegmentItem() : SessionItem(Type) {}
}
```

Related files

- `itemfactory.h` contains `ItemFactory` class definition. It is used in the context of `SessionModel` for user class registration.

Links

Martin Fowler, Presentation Model Martin Folwer, GUI architectures