Implementing the Model/View architecture in software of Brewer Network Spectrophotometer for long-term monitoring of UV radiation and ozone atmospheric content

V V Savinykh and O V Postylyakov
A.M. Obukhov Institute of Atmospheric Physics, Russian Academy of Sciences, 3 Pyzhevsky per., Moscow 119017, Russia

E-mail: amita@ifaran.ru, oleg.postylyakov@gmail.com

Abstract. Observed recovery of the ozone layer likely caused by prohibiting some ozone-depleting substances under the Montreal Protocol is observed since 1998 but still may be unstable. In this regard, the task remains to provide the homogeneity of the ozone observations in comparison with the measurements of previous decades. One of the oldest global systems providing ozone data is the network of fully automated Brewer spectrophotometers which operates since the earlier 1980s. The existing software for control Brewer spectrophotometer was created more than 35 years ago and is in need of redeveloping to maintain long-term observations in the face of changing computer platforms. A new cross-platform control software for the Brewer is under development by a group of researchers from OIAP of RAS. This software is capable for running on computers with modern multitasking operating systems (Windows, Linux, macOS), and at the same time it has one codebase. In the new Brewer control software, the Model/View architecture is implemented to separate the code of measurement data (models) from the code of their display to the user (views). The model is responsible for data management and provides an interface for reading and writing this data, and the view retrieves the necessary data from the model, reacting to its changes, and sends the data to the user, placing them properly in the GUI of application. Models are developed in the C++ programming language using the cross-platform Qt framework, and the relational database SQLite is used as a persistent storage. This software allows improving the quality and uniformity of TOC and UV radiation measurements on the Brewer spectrophotometers of the global ozone network.

1. Introduction
Ozone is one of key atmospheric gases which participates in a lot of photochemical reactions and significantly contributes to the Earth climatic system. It strongly absorbs incoming UV radiation in the stratosphere and emits in the thermal infrared. The depletion of the Earth’s ozone layer together with trends of other radiatively-active gases cause climate change [1]. In particular, the ozone layer depletion entails the high surface UV exposure, which causes aging and cancer of skin, cataracts, suppression of the immune system in humans and animals, leads to damage to plants and a decrease in their yield [2, 3]. The reduction in emissions of some ozone-depleting substances prohibited by the Montreal Protocol has likely led to recovery of the ozone layer since 1998 [4, 5]. However, the recovery of the ozone layer is uneven [5], and is also affected by the emissions of those ozone-depleting substances that are not limited by the Montreal Protocol [6]. In this regard, the tasks remain
to continue monitoring the ozone layer and to provide the homogeneity of the observations obtained in comparison with the measurements of previous decades.

One of the oldest global systems providing ozone data is the network of fully automated Brewer spectrophotometers [7, 8] operated since the earlier 1980s. It provides globally high-accuracy data on the total ozone content (TOC) [9, 10] and the vertical profiles of ozone [11, 12, 13], and includes about 80 ground stations in 40 countries worldwide now. The Brewer system [14] consists of a weatherproof spectrophotometer, azimuth tracker and tripod (figure 1). Biaxial tracking the Sun or the Moon, choice of a suitable filter, built-in wavelength calibration, and recording of measurement data are performed by internal electronics of the instrument and operating computer that allows making the round-the-clock automated observations according to in advance made schedule. Also the Brewer network provides spectral solar UV radiation [15, 16] and total column of sulphur dioxide [17]. The existing software for control Brewer spectrophotometer was created more than 35 years ago and is in need of redeveloping to continue long-term observations in the face of changing computer platforms.

Kislovodsk High-Mountain Scientific Station of A.M. Obukhov Institute of Atmospheric Physics (KHMSS) performs the long-term measurements of TOC, spectral UV radiation and the ozone vertical profiles by the Brewer MkII spectrophotometer #043 since 1989, and also carries out measurements of the total nitrogen dioxide (NO2) content [18], the total carbon monoxide (CO) content, and etc. The station is located on Shadzhatmaz Plateau in the North Caucasus (43.7N, 42.7E, and 2070 m above the sea level). Polluted air from Kislovodsk city located in 18 km northward of the station at the height of 890 m above the sea level does not reach the station, and observational conditions at the station can be considered as background for the most part of the time. The open horizon, the big number of sunny days per year, and the low-latitude observations allow to obtain a large number of the high-quality Brewer measurements of TOC not only for radiation scattered at the zenith, but also for direct solar radiation. Thus, favorable observational conditions at KHMSS allow for quality testing of the measurement data obtained by the new developed software for Brewer spectrophotometer throughout the year.

![Figure 1. The Brewer MkII spectrophotometer #043 with azimuth tracker and tripod for measurements of TOC and UV radiation at KHMSS of OIAP.](image1)

![Figure 2. The tabbed configuration dialog (view) of the new software for Brewer spectrophotometer with implementation of the Model/View architecture for macOS.](image2)

![Figure 3. The Model/View architecture separates the way that data is stored from the way that it is presented to the user [23].](image3)

The existing software for control of Brewer spectrophotometer used on the global ozone network was created for personal computers with the MS-DOS operating system [19] and has limited compatibility with modern software and hardware platforms. This software does not run on the most widespread 64-bit versions of Windows without using third-party emulators of DOS environment (e.g., DOSBox emulator [20]), and on 32-bit Windows it attempts to use all resources of CPU (or one of its cores) of the host computer without performing any useful work. To replace it the new cross-platform control software for Brewer spectrophotometer is being developed [21], which is capable of
running on host computers with modern multitasking operating systems (Windows, Linux, macOS), and at the same time it has one codebase (figure 2).

The new software implements the Model/View architecture [22] to separate the implementation of a set of classes for working with measurement data (models) from a set of classes to display this data to the application's user (views). This separation allows one to display the same data in several different views and implement new types of views without changing the underlying data structures. A Model is responsible for data management and provides the interface for reading and writing this data, and a View retrieves the necessary data from the Model, reacting to its changes, and sends the data to a user, placing them properly in the graphical user interface of an application [23] (figure 3). Each piece of information that can be obtained via a model is represented by a Model index that is a reference to item of data. By supplying model indices to the model, the view can retrieve items of data from the data source. A Delegate renders the items of data in a view; when an item is edited, the delegate communicates with the model directly using model indices. Models, views, and delegates communicate with each other using signals (methods for forwarding messages about events that occurred) and slots (functions that are called in response to particular signals). Signals from the model inform the view about changes to the data held by the data source; signals from the view provide information about the user's interaction with the items being displayed; and signals from the delegate are used during editing to tell the model and view about the state of the editor.

To separate the new software's engine from the user interface, to reduce coupling between the different parts of its code and make it modular, the core of the application's logic that realizes the Model pattern is dedicated in the separate module named "brewer-core". It has implemented as a cross-platform shared library on the C++14 (ISO/IEC 14882:2014 standard) object-oriented programming language [24] using the Qt framework [25]. Qt is a cross-platform application framework for creating classic and embedded applications that run on various software and hardware platforms with little or no change in the underlying codebase, while still being a native application with native capabilities and speed. The "brewer-core" library uses the embedded SQLite [26] relational database management system (DBMS) contained in a programming library as a persistent storage. SQLite is not a client-server DBMS, it implements transactional SQL database engine that does not have a separate server process and configuration files and is embedded into the target application. SQLite reads and writes directly to ordinary files on the disk; a complete SQL database with multiple tables, indices, triggers and views is contained in a single file, the format of which is cross-platform and is supported by various 32/64-bit operating systems.

All measurements and sets of instrumental constants of the Brewer have their own implementation of the Model pattern. It consists of a separate class describing the data structure (data class), its corresponding class for accessing the SQL persistent storage (database class), and a class that interacts with persistent storage (custom model class) and supplies the data contained in it to the view. Herewith, the custom model makes the SQL storage available to the user interface (the views) through a single entry point from multiple locations using the common database manager class that opens the connection to the database (database layer class) and lends it to all database classes [27]. The user interface module (e.g., "brewer-view") will depend on the "brewer-core" library (the software's engine) and use all the features offered by the model.

2. Defining data classes
The development of each custom model begins with the definition of the data classes for the Brewer's measurement data and its instrumental constants required for the correct operation of the application database layer. The data classes describe the spectrophotometers themselves (Brewer class), their specifications (Specification class), locations (Location class), measurements (Measurement class), as well as other information entities of the model. They include fields that characterize the state of an entity, a constructor to initialize them, and methods for reading and writing these fields. This is a fragment of the Brewer class declaration:
// brewer.h
...
class BREWERCORESHARED_EXPORT Brewer
{
  public:
    explicit Brewer(const QString &brewer_number = ""); // constructor

    int getId() const; // gets database ID
    void setId(int value); // sets database ID
    QString getNumber() const; // gets Brewer serial number
    void setNumber(const QString &value); // sets Brewer serial number

  ...
  private:
    int id; // database table ID
    QString number; // Brewer serial number
    int specificationId; // Specifications table ID
    int locationId; // Locations table ID
};

The Brewer data class contains an id variable that matches a unique identifier in the corresponding database table and a number variable that is the serial number of the Brewer spectrophotometer. The specificationId and locationId fields connect the Brewer class with the Specification and Location classes through their id fields and define relationships between the corresponding database tables (Brewers, Specifications and Locations) by a foreign key in the model database. This is a fragment of the Specification class declaration:

// specification.h
...
class BREWERCORESHARED_EXPORT Specification
{
  public:
    explicit Specification(const QString &brewer_type = ""); // constructor

    int getId() const; // gets database ID
    void setId(int value); // sets database ID
    QString getBrewerType() const; // gets Brewer type
    void setBrewerType(const QString &value); // sets Brewer type

  ...
  private:
    int id; // database table ID
    QString BrewerType; // Brewer type
    double hgCalibration = 0.0; // mercury-calibration (nm)
    double ozoneSlit0 = 0.0; // ozone Hg slit (nm)
    double ozoneSlit1 = 0.0; // ozone slit 1 (nm)
    double ozoneSlit2 = 0.0; // ozone slit 2 (nm)
};

The BREWERCORESHARED_EXPORT macro in the class declaration defined earlier in the software code means that the model is implemented as a cross-platform shared library and provides access to its symbols of functions, variables and classes for potential clients (views) on different software platforms (Windows, Linux or macOS). Implementation of the Brewer class:

// brewer.cpp
...
Brewer::Brewer(const QString &brewer_number) : // constructor
  id { -1 },
  number { brewer_number },
...
All other data classes of the model (Specification, Location, Measurement, etc.) have similar implementations.

3. Implementing database layer

SQLite DBMS is supported in Qt using SQL database driver, which is included in the framework, and is great for use in the Brewer software: a very simple database scheme, very few or no concurrent transactions, and a single-purpose database. Similarly, in the case of recording measurements of the Brewer spectrophotometer, the database layer of an application no needs for complex queries and a complex transactional model, also no need to spawn a system service because the database is stored in a single file and does not need to be accessed by multiple applications.

The persistent storage (database) of the new Brewer software is available from different parts of the application, it has a single entry point implemented as a singleton pattern (a protected constructor and a static function returning a reference to a class instance), which ensures the creation of a single instance of this class. The instance() static function realizes the "unique access point" and returns a reference to the DatabaseManager class. The visibility of the DatabaseManager() constructor is protected, so only DatabaseManager class and its child classes will be allowed to call this constructor providing one instance of the above class:

// database_manager.h
...
class DatabaseManager
{
public:
    static DatabaseManager &instance(); // reference to the class instance
...
    const BrewerDao brewerDao; // Brewers table access
    const SpecificationDao specificationDao; // Specifications table access
...
protected:
    // protected constructor
    explicit DatabaseManager(const QString &path = filename);

private:
    std::unique_ptr<QSqlDatabase> database; // database connection
};

The DatabaseManager class opens the connection to the database in the database field and lends it to database tables access classes (BrewerDao, SpecificationDao, etc.); this field is declared as a smart pointer [24], which prevents memory leaks in the event of a failure when accessing database tables. The open fields of brewerDao, specificationDao and others provide access to the corresponding database tables (Brewers, Specifications, etc.); the const qualifier prevents them from being unintentionally modified because these fields have the public visibility to be accessed by clients of the DatabaseManager class. Implementation of the DatabaseManager class:

// database_manager.cpp
DatabaseManager &DatabaseManager::instance()
{
    static DatabaseManager singleton; // reference to the class instance
    return singleton;
}

DatabaseManager::DatabaseManager(const QString &path) : // loading SQLite driver
database{new QSqlDatabase{QSqlDatabase::addDatabase("QSQLITE")}},
brewerDao{"database"}, // and initializing access to tables
specificationDao{"database"}, // of Brewers, Specifications, etc.
...
{database->setDatabaseName(path); // sets database filename
    bool open_status = database->open(); // opens the database connection
    if(open_status != true)
        emit error(tr("Unable to connect to the Brewer database."));
    brewerDao.init(); // creates the Brewers table
    specificationDao.init(); // creates the Specifications table
    ...
}

To enable the model to interact with the database, each of data classes has a dedicated database class for accessing to corresponding tables (BrewerDao, SpecificationDao classes, etc., DAO – data access object) that implements basic functions of a persistent storage: creating, reading, updating and deleting measurement data. A fragment of the database class declaration for storing an information about Brewer spectrophotometers looks like this:

    // brewer_dao.h
    ...
    class BREWERCORESHAREDCORE_EXPORT BrewerDao
    {
    public:
        explicit BrewerDao(QSqlDatabase &data_base); // constructor
        void init() const; // creating Brewers table
        bool addBrewer(Brewer &brewer) const; // adding a new Brewer
        bool removeBrewer(int id) const; // removing the Brewer
        bool updateBrewer(const Brewer &brewer) const; // updating the Brewer
        std::unique_ptr<std::vector<std::unique_ptr<Brewer>>> brewers() const;
    ...
    private:
        QSqlDatabase &database; // database connection
    };

The database field stores a reference to the database connection that will be used for all SQL queries done by the BrewerDao class. The init() function aims to create (using the appropriate SQL query) the Brewers table in the database (if it does not exist) with an information about spectrophotometers; its fields correspond to the fields of the Brewer data class (Id, Number fields, etc.). The addBrewer() function adds information about the new spectrophotometer to the database, the removeBrewer() and updateBrewer() functions remove and update the spectrophotometer's data accordingly, and the brewers() function returns a list with descriptions of all Brewers as a vector of smart pointers and is used to populate the custom model with data. This is a fragment of the BrewerDao class implementation:

    // brewer_dao.cpp
    ...
    // SQL query to create the Brewers table
    const QString BREWERS_CREATE { "CREATE TABLE IF NOT EXISTS Brewers ("
"Id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, "
"Number TEXT NOT NULL UNIQUE CHECK (CAST(Number AS INTEGER) > 0), "
"SpecificationId INTEGER NOT NULL REFERENCES Specifications(Id), "
"LocationId INTEGER NOT NULL REFERENCES Locations(Id));"

// SQL query to insert a new Brewer to the database
const QString BREWER_INSERT { "INSERT INTO Brewers ("Number, SpecificationId, "
"LocationId) VALUES (:Number, :SpecificationId, :LocationId);" );

BrewerDao::BrewerDao(QSqlDatabase &data_base) : database { data_base } { }
void BrewerDao::init() const
{
  if (!database.tables().contains("Brewers")) {
    QSqlQuery query { database }; // makes SQL query to create
    bool exec_status = query.exec(BREWERS_CREATE); // executes the SQL query to create
    if (exec_status != true) // create Brewers table
      emit error(tr("Unable to create the Brewers table."));
  }
}

bool BrewerDao::addBrewer(Brewer &brewer) const
{
  QSqlQuery query { database }; // makes SQL query to add
  query.prepare(BREWER_INSERT); // makes of SQL query to add
  query.bindValue("Number", brewer.getNumber()); // add entry for new Brewer
  query.bindValue("SpecificationId", brewer.getSpecificationId());
  query.bindValue("LocationId", brewer.getLocationId());

  bool exec_status = query.exec(); // executes the SQL query
  if (exec_status != true)
    emit error(tr("Unable to add the Brewer."));
  else
    brewer.setId(query.lastInsertId().toInt()); // sets database table ID

  return exec_status;
}

... The others database classes are built similarly. In total, the database classes create and manage about 30 tables that store about 800 pieces of information with measurement data, Brewer specifications, instrument constants, and so on.

4. Implementing model classes
When creating custom models for the new Brewer control software, the abstract class of the Qt for one-dimensional lists of QAbstractListModel was used as the basis, its capabilities were extended so that the final model could work with database tables. This is how the custom model class is implemented to manage data of the Brewers table:

// brewer_model.h
...

class BREWERCORESHAREDEXPORT BrewerModel : public QAbstractListModel
{
  public:
  ...
    explicit BrewerModel(QObject *parent = Q_NULLPTR); // constructor

    QModelIndex addBrewer(const Brewer &brewer); // adding a Brewer to model
    int rowCount(const QModelIndex &parent = QModelIndex { }); const override;
    QVariant data(const QModelIndex &index, int role) const override;

...
bool setData(const QModelIndex &index, const QVariant &val, int role) override;
bool removeRows(int row, int count, const QModelIndex &parent) override;
...
private:
bool isIndexValid(const QModelIndex &index) const; // checking the model index
DatabaseManager &databaseManager; // reference to the database
std::unique_ptr<std::vector<std::unique_ptr<Brewer>>> brewers; // buffer
};

In the Model/View scheme, the custom model communicates with the data layer through the databaseManager field; it is a link to the database. The brewers vector acts as a buffer that avoids too often accessing the Brewers table of the database, it reduces the load on the disk subsystem because the buffer is located in the computer's RAM. The model base class of the QAbstractListModel is extended with two new methods: the addBrewer() function adds information about a new Brewer to the custom model, and the isIndexValid() function checks the validity of the model index that is used to find the data in the model. Model indices provide temporary references to pieces of information, and can be used to retrieve or modify data via the model. Because models may reorganize their internal structures from time to time, model indices may become invalid, and should not be stored. The rest of the custom model's methods are overridden virtual functions of the base class.

In the class constructor implementation, the databaseManager field is initialized with the DatabaseManager singleton address, and the brewers vector is populated with the content of the Brewers table, while the internal mechanism of the custom model matches its contents to the database state. The overridden rowCount() function returns the number of elements in the custom model, and the overridden data() function returns a specific piece of information about the data to display (e.g., spectrophotometer number or specification). The overridden setData() function updates specific data in the model (e.g., location), and the overridden removeRows() function removes a record of the specific Brewer from the custom model. The role parameter in the data() and setData() functions indicates which particular data elements (e.g., specification or location) need to be retrieved by the view, it associates each data element to a tag for the view to know what category of data is displayed or edited. A fragment of the BrewerModel class implementation looks like this:

```cpp
// brewer_model.cpp
...
BrewerModel::BrewerModel(QObject *parent) :
QAbstractListModel { parent },
databaseManager { DatabaseManager::instance() }, // initializing singleton
brewers { dbManager.brewerDao.brewers() } // getting Brewers table
{ // contents
QModelIndex BrewerModel::addBrewer(const Brewer &brewer)
{std::unique_ptr<Brewer> new_brewer { new Brewer { brewer } };if (!databaseManager.brewerDao.addBrewer(*new_brewer)) return QModelIndex { }; // adding a new Brewer return QModelIndex { }; // to the custom model
int row = rowCount(); // updating the model, informing beginInsertRows(QModelIndex { }, row, row); // the view about this event, and brewers->push_back(move(new_brewer)); // adding a new Brewer to the buffer endInsertRows();
return index(row, 0);
}
QVariant BrewerModel::data(const QModelIndex &index, int role) const
{if (!isIndexValid(index)) // checking index validity
```
...}

5. Conclusions
A group of researchers from OIAP of RAS is developing new cross-platform control software for Brewer spectrophotometer to replace existing MS-DOS operating software for the Brewer used on the global ozone network. This software is capable of running on computers with modern multitasking operating systems (Windows, Linux, macOS), and at the same time it has one codebase. Debugging and testing of the new software is performed using the Brewer MkII #043 at the KHMSS of OIAP. The developed database of the new software for storing measurement data and instrumental constants consists of more than 30 tables and stores about 800 pieces of information. For most database tables within the Model/View architecture, the classes that implement the interface of the Model pattern have been developed (i.e. the methods of these classes were developed). The implementation of the Model pattern for each database table consists of the auxiliary data class and database class of access to the persistent storage, the class of the custom model itself based on the abstract model class for one-dimensional lists of Qt framework, as well as the common for all tables the database layer class. The database layer class uses the embedded SQLite relational DBMS as a persistent storage. The model of the new software is implemented using the cross-platform Qt application development framework as a cross-platform shared library on the object-oriented programming language C++14 (ISO/IEC 14882:2014) with the use of new language features (lambda functions, smart pointers, etc.). This software will allow to improve the quality and uniformity of measurements of TOC and UV radiation on Brewer spectrophotometers (that are widely used on the global ozone network) and will enable to maintain the integrity and consistency of the measurement data through sets of the integrity constraints applied to the database tables (primary keys, referential integrity, check constraints, etc.) [28].

Acknowledgments
The work was supported by the Russian Science Foundation with grant № 16-17-10275.

References
The work was supported by the Russian Science Foundation with grant № 16-17-10275.

[1] Pachauri R K et al. 2014 IPCC, 2014: Climate Change 2014: Synthesis Report. Contribution of Working Groups I, II and III to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change ed L A Meyer (Geneva: IPCC) p 151
[2] Kerr J B and McElroy C T 1993 Evidence for large upward trends of ultraviolet-B radiation linked to ozone depletion Science 262(5136) 1032–34
[3] Kerr J B and Fioletov V E 2010 Surface ultraviolet radiation Atmosphere-Ocean 46(1) 159–84
[4] Strahan S E and Douglass A R 2018 Decline in antarctic ozone depletion and lower stratospheric chlorine determined from aura microwave limb sounder observations Geophys. Res. Lett. 45 382–90
[5] Ball W T, Alsing J, Mortlock D J, Staehelin J, Haigh J D, Peter T, Tummon F, Stübi R, Stenke A, Anderson J, Bourassa A, Davis S M, Degenstein D, Frith S, Froidevaux L, Roth C, Sofieva V, Wang R, Wild J, Yu P, Ziemke J R and Rozanov E V 2018 Evidence for a
continuous decline in lower stratospheric ozone offsetting ozone layer recovery Atmos. Chem. Phys. 18 1379–94

[6] Oram D E, Ashfold M J, Laube J C, Gooch L J, Humphrey S, Sturges W T, Leedham-Elvidge E, Forster G L, Harris N R P, Mead M I, Samah A A, Phang S M, Ou-Yang C-F, Lin N-H, Wang J-L, Baker A K, Brennikmeijer C A M and Sherry D 2017 A growing threat to the ozone layer from short-lived anthropogenic chlorocarbons Atmos. Chem. Phys. 17 11929–41

[7] Kerr J B, McElroy C T, Wardle D I, Olafson R A and Evans W F J 1985 The automated Brewer spectrophotometer Atmospheric Ozone ed C S Zerefos and A Ghazi (Dordrecht: Springer) pp 396-401

[8] Fioletov V E, Kerr J B, McElroy C T, Wardle D I, Savastiouk V and Grajnar T S 2005 The Brewer reference triad Geophys. Res. Lett. 32 L20805

[9] Kerr J B, McElroy C T and Olafson R A 1981 Measurements of ozone with the Brewer ozone spectrophotometer Proc. Quad. Int. Ozone Symp. (Boulder: IAMAP) pp 74–79

[10] Dorokhov V, Yushkov V, Makshtas A, Ivlev G, Tereb N, Savinykh V, Shepelev D, Nakajima H, McElroy C T, Tarasick D, Goutail F, Pommereau J-P and Pazmino A 2015 Brewer, SAOZ and ozone sonde observations in Siberia Atmosphere-Ocean 53(1), 14-18

[11] McElroy C T, Mateer C L, Kerr J B and Wardle D I 1989 Umkehr observations made with the Brewer ozone spectrophotometer Proc. of the Quad. Ozone Symp. 1988 (Göttingen) ed R D Bojkov and P Fabian (Hampton: Deepak Publishing) p 725

[12] Elansky N F, Mitin I V and Postylyakov O V 1999 Maximum accuracy of Umkehr measurements of vertical ozone profiles Izvestiya - Atmospheric and Oceanic Physics 35(1) 65-77

[13] Postylyakov O V, Belikov I B, Elansky N F and Elokhov A S 2006 Observations of the ozone and nitrogen dioxide profiles in the TROICA-4 experiment Adv. Space Res. 37(12) 2231–37

[14] Kipp & Zonen Inc. 2015 Brewer MkIII Spectrophotometer operator’s manual, Rev F (Delft: Kipp & Zonen B.V.) p 123

[15] Bais A F, Zerefos C S and McElroy C T 1996 Solar UVB measurements with the double- and single-monochromator Brewer ozone spectrophotometers Geophys. Res. Lett. 23(8) 833–36

[16] Fioletov V E, Kerr J B and Wardle D I 1997 The relationship between total ozone and spectral UV irradiance from Brewer observations and its use for derivation of total ozone from UV measurements Geophys. Res. Lett. 24(23) 2997–3000

[17] Zerefos C S et al. 2017 Detecting volcanic sulfur dioxide plumes in the Northern Hemisphere using the Brewer spectrophotometers, other networks, and satellite observations Atmos. Chem. Phys. 17 551–74

[18] Borovskii A N, Arabov A Ya, Golitsyn G S, Gruzdev A N, Elanskii N F, Elokhov A S, Mokhov I I, Savinykh V V, Senik I A and Timazhev A V 2016 Variations of total nitrogen oxide content in the atmosphere over the North Caucasus Russ. Meteorol. Hydrol. 41 93–103

[19] IOS Inc. 2017 Brewer operating software http://www.io3.ca/Download/brewer_v4.10.zip

[20] The DOSBox Team 2016 DOSBox project https://www.dosbox.com/wiki/

[21] Savinykh V V and Skornyakov V Yu 2016 New cross-platform control software for Brewer spectrophotometer Proc. SPIE 10035, 100355P

[22] Summerfield M 2010 Advanced Qt programming: creating great software with C++ and Qt 4 (Upper Saddle River: Prentice Hall) p 553

[23] The Qt Company 2018 Qt Documentation http://doc.qt.io/qt-5/model-view-programming.html

[24] Stroustrup B 2013 The C++ programming language, 4th edition (Upper Saddle River: Addison-Wesley Professional) p 1368

[25] Blanchette J and Summerfield M 2008 C++ GUI Programming with Qt 4, 2nd Edition (Upper Saddle River: Prentice Hall) p 752

[26] Kreibich J A 2010 Using SQLite (Sebastopol: O’Reilly) p 525

[27] Lazar G and Penea R 2016 Mastering Qt 5 (Birmingham: Pactk Publishing) p 526

[28] SQLite Team 2018 SQLite Documentation https://sqlite.org/lang_createtable.html#constraints