Journal of proteome research

Technical Note

pubs.acs.org/jpr

jqcML: An Open-Source Java API for Mass Spectrometry Quality Control Data in the qcML Format

Wout Bittremieux,[†,‡] Pieter Kelchtermans,[§,⊥,∥] Dirk Valkenborg,[⊥,#,▽] Lennart Martens,[§,∥] and Kris Laukens*[,†,‡]

[†]Department of Mathematics and Computer Science, University of Antwerp, Middelheimlaan 1, B-2020 Antwerp, Belgium

[‡]Biomedical Informatics Research Center Antwerp (biomina), University of Antwerp/Antwerp University Hospital, Wilrijkstraat 10, B-2650 Antwerp, Belgium

[§]Department of Medical Protein Research, VIB, Albert Baertsoenkaai 3, B-9000 Ghent, Belgium

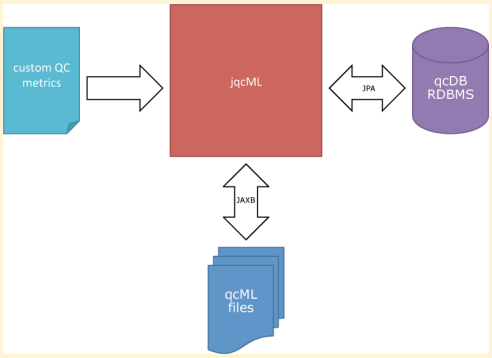
[∥]Department of Biochemistry, Faculty of Medicine and Health Sciences, Ghent University, Albert Baertsoenkaai 3, B-9000 Ghent, Belgium

[⊥]Flemish Institute for Technological Research (VITO), Boeretang 200, B-2400 Mol, Belgium

[#]CFP-CeProMa, University of Antwerp, Groenenborgerlaan 171, B-2020 Antwerp, Belgium

[▽]I-BioStat, Hasselt University, Agoralaan - Building D, B-3590 Diepenbeek, Belgium

ABSTRACT: The awareness that systematic quality control is an essential factor to enable the growth of proteomics into a mature analytical discipline has increased over the past few years. To this aim, a controlled vocabulary and document structure have recently been proposed by Walzer et al. to store and disseminate quality-control metrics for mass-spectrometry-based proteomics experiments, called qcML. To facilitate the adoption of this standardized quality control routine, we introduce jqcML, a Java application programming interface (API) for the qcML data format. First, jqcML provides a complete object model to represent qcML data. Second, jqcML provides the ability to read, write, and work in a uniform manner with qcML data from different sources, including the XML-based qcML file format and the relational database qcDB. Interaction with the XML-based file format is obtained through the Java Architecture for XML Binding (JAXB), while generic database functionality is obtained by the Java Persistence API (JPA). jqcML is released as open-source software under the permissive Apache 2.0 license and can be downloaded from https://bitbucket.org/proteinspector/jqcml.



KEYWORDS: *mass spectrometry, proteomics, quality control, qcML, Java, API, open source, XML, qcDB, database*

INTRODUCTION

Over the past few years, the awareness has risen that in order for mass-spectrometry-based proteomics to advance and evolve into a mature analytical field, the need for systematic quality control is paramount.[1] As a result, a lot of research has been conducted to identify quality metrics that are able to capture the operational characteristics of a mass spectrometer.[2] These efforts have enabled a shift in focus toward a "quality by design" paradigm, as illustrated by the extensive and recent review by Tabb[3] that summarizes the current state of quality-control research in proteomics.

The defined quality-control metrics provide valuable information and can play an important role in allowing researchers to assess the accuracy of the results and to subsequently correctly interpret the experimental results. However, there are still some limiting factors that prevent the widespread adoption of these metrics as an inherent quality control component in mass-spectrometry-based proteome analysis. More specifically, there are two main issues prevalent.[4] First, data storage and communication of this new type of information is not yet standardized, hampering the dissemination of quality-control data alongside experimental data. Second, the quality-control metrics are heterogeneous by nature and can be generated by a variety of different software tools, each currently with their own content and definitions.

The qcML format was recently developed to address both issues.[4] It comprises an XML-based data standard and associated controlled vocabularies (CVs) for storing various types of performance metrics, along with applicable metadata about the experiments. In addition, an equivalent relational database structure, called qcDB, has been developed to complement the XML-based file format. Bioinformatics tools that support the qcML format are already under way. For example, the freely available OpenMS software[5] is able to

Received: December 20, 2013
Published: June 6, 2014

dx.doi.org/10.1021/pr401274z | J. Proteome Res. 2014, 13, 3484−3487

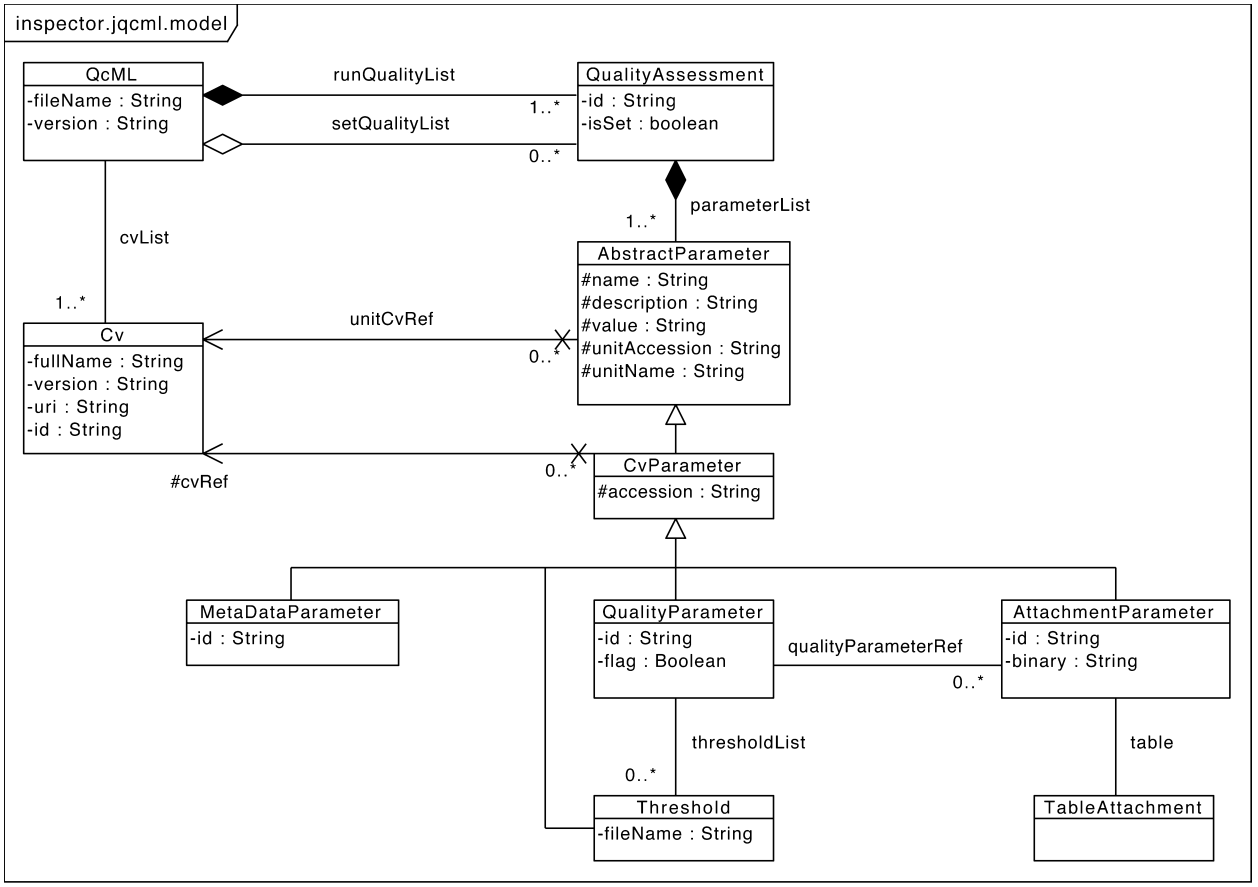


Figure 1. Class diagram highlighting the key components of the jqcML object model for qcML version 0.0.8.

calculate a variety of quality control metrics and stores these in XML-based qcML files.

Yet the success of any such standards rests in large part on the existence of software libraries in popular programming languages that allow easy read and write access to the standard format. We therefore present jqcML, a fully operational, production-grade Java application programming interface (API) for working with qcML data. Written in Java, the jqcML API is inherently platform-independent. Because the qcML standard was only recently finalized, jqcML is the first stand-alone software library to work with qcML data. Nevertheless, jqcML already supports the full capabilities of the qcML format. Built to be used in demanding circumstances, jqcML provides a complete object model to interpret and manipulate qcML data while retaining a minimal memory footprint and without sacrificing the overall speed of data access. Furthermore, jqcML is able to interact in a uniform and transparent way with both the XML-based qcML files and the qcDB relational database. This dual access enables the user to transparently work with qcML data from both sources, without requiring any changes in the data processing code.

EXPERIMENTAL SECTION

The objective of jqcML is two-fold. First, it provides a complete object model to represent qcML data. This object model is accessible through an intuitive API. Second, jqcML provides the ability to work in a uniform manner with qcML data from several sources, such as the XML-based file format or the relational database qcDB. However, a common focus for both of these objectives is simplicity and ease of use. Care has been taken to provide a complete yet straightforward API to assist other developers, making use of jqcML, as much as possible. For example, when reading or writing qcML files, the content of these files can automatically be validated against the XML schema for the latest qcML version.

Object Model

The data defined by the qcML data format are represented by a complete object model consisting of simple Java classes. It is of note that the qcML data format is less complex than related standard formats, such as mzML.[6] The object model intends to reflect this relative simplicity while at the same time providing expressive access to the data. The information specified by the qcML standard can be represented by the object model and includes basic quality metrics as well as tabular attachments or more complex attachments (such as binary strings). Figure 1 highlights the key components of the jqcML object model.

Data Processing

As previously mentioned, jqcML is capable of reading and writing XML-based qcML files but also fully supports reading from and writing to the qcDB relational database. The interaction with both types of data sources is generalized through common interfaces, allowing the underlying data access implementation to be abstracted from the user. This generalized object model thus allows the user to handle the different data sources in a uniform manner, making it trivial to switch between several different data sources. A simplified schema of the workflow is shown in Figure 2.

Similar to other XML-based proteomics standards, the XML-based qcML file structure is intended to exchange quality-

dx.doi.org/10.1021/pr401274z | J. Proteome Res. 2014, 13, 3484−3487

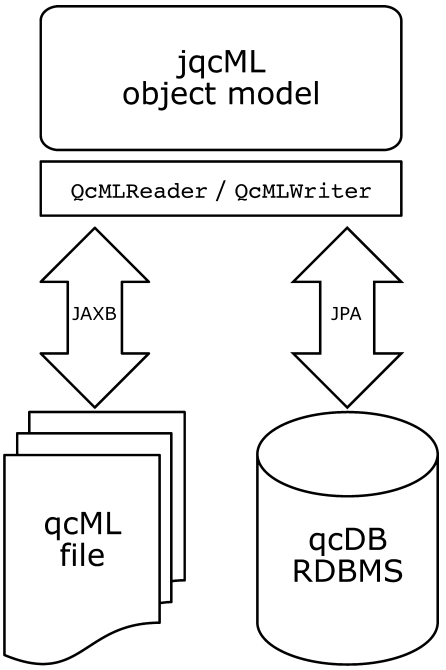


Figure 2. Simplified representation of the jqcML architecture. Through the use of the common QcMLReader and QcMLWriter interfaces, jqcML is able to work with qcML data from several sources in a uniform way.

control data between multiple users or organizations, while the qcDB relational database will most likely be used within a single organization to store large amounts of quality-control data over time and to perform data retrieval and analysis operations across extensive collections of qcML data. To ease the translation of quality-control data from one format to the other, jqcML also supports the easy conversion of data between both representations. The key implementation aspects of jqcML for the XML and relational database formats are discussed next.

XML-Based File Format. To interact with the XML-based qcML files, the Java Architecture for XML Binding (JAXB)[7] is used in the form of the EclipseLink MOXy implementation.[8] The interaction between the object model and the XML-based file format is handled through the mapping of the formal XML schema definition of qcML to specific elements of the jqcML object model. This mapping allows for an automatic translation between the object model and the XML structure, which enables both reading of qcML files into jqcML Java objects as well as exporting jqcML objects to qcML files.

Because a qcML file can contain data from several different runs, it can, in principle, become arbitrarily large. When working with such very large files, precautions should be taken to guarantee that there is sufficient memory available to process the file. Therefore, jqcML is designed to elegantly handle very large qcML files while controlling the memory requirements. For this purpose, jqcML exploits an XML indexer component, xxIndex,[9,10] that allows the qcML file to be accessed like an indexed random access file. This fragmented data access prevents the reading of the complete qcML file into memory, allowing jqcML to read even very large qcML files while retaining a minimal memory footprint. The XML indexer approach also enables the user to retrieve only the data of a single experiment as an iterative procedure, such that only a subset of the full qcML file is kept in main memory at once.

The XML-based file structure makes use of internal references to cross-link various elements, for example, to link an annotation parameter to the corresponding entry in a controlled vocabulary. When only retrieving an individual element, as explained previously, these references may initially not refer to existing Java objects because only part of the qcML file is read into main memory at once. However, the use of xxIndex enables jqcML to resolve these references automatically as well. A complete object model is thus always accessible, even when reading only a section of a qcML file.

Relational Database Structure. In addition to the XML-based file format, the qcML specification includes an entity-relationship model as a reference implementation of a qcDB relational database to store qcML data. To interact with a qcDB, the Java Persistence API (JPA)[11] is used, and, in particular, the EclipseLink JPA implementation.[12] By analogy to the way JAXB is used for interacting with XML-based qcML files, JPA is used to interact with a relational database by creating a mapping between the object model and the database model.

By using JPA, the need for low-level operations, such as SQL queries, can be avoided in the code. Instead of having to manage the various tables in the database and other technicalities, operations can be defined on the level of the object model. An additional advantage of defining operations on a higher level is that the user does not need to focus on implementation details, such as adapting to the specific SQL syntax of the chosen relational database engine.

Retrieving or storing of quality-control data in a qcDB is implemented by executing the corresponding JPA persistence methods, while more advanced queries can easily be performed using the Java Persistence Query Language (JPQL).[13] Equivalent to the JAXB access layer previously discussed, the JPA implementation supports fragmented data retrieval of the quality-control information based on an iterative procedure.

jqcML currently fully supports both the MySQL and SQLite database engines. However, other relational database systems can easily be included as well. The only requirement is the availability of a suitable Java driver for the specific type of database. When initializing a new system, it is even possible to let jqcML create the required tables in the database.

■ CONCLUSIONS

Several standard data formats have been proposed by the Human Proteome Organization's Proteomics Standards Initiative (HUPO-PSI) and have been embraced by the community, in no small part because they enable the uniform sharing of data. For this purpose, the availability of production-grade libraries to read and write these standard data formats is crucial, and Java has proven to be a popular programming language to maximize the usage of these tools. Examples include the jmzML API[14] for the mzML format,[6] the jTraML API[15] for the TraML format,[16] the jmzIdentML API[17] and jmzReader tool[18] for the mzIdentML format,[19] and the jmzQuantML API[20] for the mzQuantML format.[21] Similarly, jqcML fills this void for the qcML format.

jqcML presents a complete and easy-to-use Java API to manipulate qcML data. As such, it will provide an invaluable tool for the adoption of the qcML standard. Furthermore, jqcML can prove to be useful for both producers and users of qcML data. Producers can use jqcML to directly export their data in compliance with the qcML format or convert between their existing output results and the qcML format. Meanwhile,

dx.doi.org/10.1021/pr401274z | J. Proteome Res. 2014, 13, 3484−3487

end users will have out-of-the-box access to qcML data, without having to worry about implementation details. This universal applicability is reinforced by the fact that as a pure Java API jqcML is completely platform-independent.

jqcML is freely available and is released as open source under the permissive Apache 2.0 license. The binaries, source code, and documentation can be downloaded from the project Web site at https://bitbucket.org/proteinspector/jqcml.

AUTHOR INFORMATION

Corresponding Author

*E-mail: kris.laukens@uantwerpen.be. Phone: +32 (0) 3 265 33 10. Fax: +32 (0) 3 265 37 77.

Notes

The authors declare no competing financial interest.

ACKNOWLEDGMENTS

This work was supported by SBO grant "InSPECtor" (120025) of the Flemish agency for Innovation by Science and Technology (IWT).

REFERENCES

(1) Martens, L. Bringing proteomics into the clinic: The need for the field to finally take itself seriously. *Proteomics: Clin. Appl.* **2013**, *7*, 388−391.

(2) Rudnick, P. A.; et al. Performance metrics for liquid chromatography-tandem mass spectrometry systems in proteomics analyses. *Mol. Cell. Proteomics* **2010**, *9*, 225−241.

(3) Tabb, D. L. Quality assessment for clinical proteomics. *Clin. Biochem.* **2013**, *46*, 411−420.

(4) Walzer, M.; et al. qcML: An exchange format for quality control metrics from mass spectrometry experiments. *Mol. Cell. Proteomics* **2014**, DOI: 10.1074/mcp.M113.035907.

(5) Sturm, M.; Bertsch, A.; Gröpl, C.; Hildebrandt, A.; Hussong, R.; Lange, E.; Pfeifer, N.; Schulz-Trieglaff, O.; Zerck, A.; Reinert, K.; Kohlbacher, O. OpenMS −An open-source software framework for mass spectrometry. *BMC Bioinf.* **2008**, *9*, 163.

(6) Martens, L.; et al. mzML—a community standard for mass spectrometry data. *Mol. Cell. Proteomics* **2011**, *10*, R110.000133−R110.000133.

(7) Java Architecture for XML Binding (JAXB). https://jaxb.java.net/ (accessed 12/12/2013).

(8) EclipseLink MOXy. http://www.eclipse.org/eclipselink/moxy.php (accessed 12/12/2013).

(9) Montecchi-Palazzi, L.; Kerrien, S.; Reisinger, F.; Aranda, B.; Jones, A. R.; Martens, L.; Hermjakob, H. The PSI semantic validator: A framework to check MIAPE compliance of proteomics data. *Proteomics* **2009**, *9*, 5112−5119.

(10) XXIndex - a library to index XML files for random access. http://code.google.com/p/pride-toolsuite/wiki/XXIndex (accessed 12/12/2013).

(11) Java Persistence API. http://jcp.org/en/jsr/detail?id=317 (accessed 12/12/2013).

(12) EclipseLink JPA. http://www.eclipse.org/eclipselink/jpa.php (accessed 12/12/2013).

(13) Enterprise JavaBeans 3.0. https://jcp.org/en/jsr/detail?id=220 (accessed 12/12/2013).

(14) Côté, R. G.; Reisinger, F.; Martens, L. jmzML, an open-source Java API for mzML, the PSI standard for MS data. *Proteomics* **2010**, *10*, 1332−1335.

(15) Helsens, K.; Brusniak, M.-Y.; Deutsch, E.; Moritz, R. L.; Martens, L. jTraML: An open source Java API for TraML, the PSI standard for sharing SRM transitions. *J. Proteome Res.* **2011**, *10*, 5260−5263.

(16) Deutsch, E. W.; Chambers, M.; Neumann, S.; Levander, F.; Binz, P.-A.; Shofstahl, J.; Campbell, D. S.; Mendoza, L.; Ovelleiro, D.; Helsens, K.; Martens, L.; Aebersold, R.; Moritz, R. L.; Brusniak, M.-Y. TraML—A Standard Format for Exchange of Selected Reaction Monitoring Transition Lists. *Mol. Cell. Proteomics* **2012**, *11*, R111.015040−R111.015040.

(17) Reisinger, F.; Krishna, R.; Ghali, F.; Ros, D.; Hermjakob, H.; Vizcano, J. A.; Jones, A. R. jmzIdentML API: A Java interface to the mzIdentML standard for peptide and protein identification data. *Proteomics* **2012**, *12*, 790−794.

(18) Griss, J.; Reisinger, F.; Hermjakob, H.; Vizcano, J. A. jmzReader: A Java parser library to process and visualize multiple text and XML-based mass spectrometry data formats. *Proteomics* **2012**, *12*, 795−798.

(19) Jones, A. R.; et al. The mzIdentML data standard for mass spectrometry-based proteomics results. *Mol. Cell. Proteomics* **2012**, *11*, M111.014381−M111.014381.

(20) Qi, D.; Krishna, R.; Jones, A. R. The jmzQuantML programming interface and validator for the mzQuantML data standard. *Proteomics* **2014**, *14*, 685−688.

(21) Walzer, M.; et al. The mzQuantML data standard for mass spectrometry-based quantitative studies in proteomics. *Mol. Cell. Proteomics* **2013**, *12*, 2332−2340.