PSI Recommendation
PSI Quality Control Workgroup
Version 1.0.0 DRAFT

Mathias Walzer, European Bioinformatics Institute
Wout Bittremieux, University of California San Diego
Julian Uszkoreit, Ruhr University Bochum
Chris Bielow, Freie Universität Berlin
Reza Salek, The International Agency for Research on Cancer, France
David L. Tabb, Stellenbosch University

December 13, 2020

mzQC: Reporting and exchange format for mass spectrometry quality control data

Document Status

This document presents a specification of the mzQC data format developed by members of the Human Proteome Organisation (HUPO) Proteomics Standards Initiative (PSI) Quality Control (QC) Working Group. Distribution is unlimited.

Document Version

The current version of this document is 1.0.0 DRAFT, December 13, 2020.

Abstract

The Human Proteome Organisation (HUPO) Proteomics Standards Initiative (PSI) defines community standards for data representation in biological mass spectrometry, including proteomics, metabolomics, and lipidomics, to facilitate data comparison, exchange, and verification. The Quality Control Working Group develops standards and recommendations to describe the quality of mass spectrometry data and related analysis results.
This document defines the mzQC file format to report and exchange quality-related information for a mass spectrometry experiment, associated analysis results, or collections thereof. The mzQC specification defines a simple yet versatile file format with a hierarchical structure to store quality metrics, thereby providing support for general quality control, (automated) decision making, visualisation efforts, and easy persistence and exchange of all of the above. The format and its specification are realized in the widespread JavaScript Object Notation (JSON) that can be easily implemented in software to produce or consume mzQC files. The mzQC format is complemented by the Quality Control Controlled Vocabulary (QC CV), which includes formal definitions of relevant quality metrics. The combination of the clear, human-readable syntax of the mzQC format and the rich semantic information associated with quality metrics stored in the QC CV provides powerful mechanisms to interpret, store, and enable reuse of quality control data.

Contents

1 Introduction

This document systematically describes how the mzQC file format can be used to store quality-related information from MS-based experiments. It is a specification, not a tutorial. As such, the presentation of technical details is deliberately direct. The role of the text is to describe the mzQC file format and justify design decisions. The document does not discuss how mzQC should be used in practice, consider tool support for data capture or storage, or provide comprehensive examples of mzQC files. Tutorials and example material are available via the Appendix and through the PSI-QC working group repository.

1.1 Background (from XML to JSON)

Unlike most previous PSI standards, which are predominantly XML-based file formats (Martens et al. 2011, Jones et al. 2012, Walzer et al. 2013), mzQC is defined using JSON syntax. A disadvantage of XML-based file formats is that XML is quite verbose, adding substantial formatting overhead to the already large data volume of mass spectrometry files. Additionally, producing or consuming an XML file is not natively supported in most programming languages and instead requires specialized software libraries, whose usage can be complex. In contrast, JSON has been developed to be a lightweight and universal data interchange format. As such, JSON files have a small memory footprint and there is wide built-in support for JSON in many programming languages. JSON makes use of two data structures: key–value pairs and ordered lists of values. JSON files are easy to understand, which explains their emergence as a replacement for XML in many systems. For example, it has become the *de facto* encoding language of (web) APIs, where easy-to-understand, lightweight data interchange formats are adopted most frequently. Functionally, JSON can be deployed for the same kind of data interchange purposes as XML. By specifying mzQC using a JSON syntax, we can leverage the broader availability of efficient libraries and simpler implementations while adhering to the proven design principles of PSI formats and maintaining the necessary level of compatibility. This lowers the technical threshold for mzQC adoption and will increase the reach of mzQC through an extended use case basis.

JSON data structures are built using two structures:

- An (unordered) collection of key–value pairs, generally called an **object**.
  An object begins with a left brace `{` and ends with a right brace `}`. Each name is followed by a colon and the key–value pairs are separated by a comma.
  Syntax: `{key_1: value_1, key_2: value_2}`
  Example: `{"apple-count": 5, "delivery-date": "1970-01-01"}`
- An (ordered) list of values, generally called an **array** or list.
  An array begins with a left bracket `[` and ends with a right bracket `]`. Values are separated by a comma. Value types can be mixed.
  Syntax: `[value_3, value_4, value_5]`
  Example: `[true, {"subobject": "yes"}, 42]`

Values can be a string in double quotes, a number, `true` or `false` or `null`, an object, or an array. These structures can be nested; for example, multidimensional tables can be represented as arrays of arrays. Strings and values are like the respective C or Java structures. [json.org]

1.1.1 Resources

Users new to JSON can find detailed information via the following resources:

- JSON website
- formal JSON specification

1.2 Document Structure

The remainder of this document is structured as follows. Section 2 describes a variety of conventions for the notation that apply throughout the document. Section 3 lists use cases for the mzQC format. Section 4 outlines the relationships between mzQC and other file format specifications. Sections 5 and 6 detail the format specification. Section 7 provides general recommendations on using mzQC, section 8 lists a few pending issues, and section 9 contains a brief summary and conclusion. Sections 10, 11, 12, 13, and 14 include the list of authors, contributors, intellectual property statement, copyright notice, and references, respectively. An appendix introduces tutorial material and companion documents explaining various aspects of the format and its use cases.

2 Notational Conventions

The keywords “MUST,” “MUST NOT,” “REQUIRED,” “SHALL,” “SHALL NOT,” “SHOULD,” “SHOULD NOT,” “RECOMMENDED,” “MAY,” and “OPTIONAL” are to be interpreted as described in RFC 2119 (Bradner 1997).

In this document, we colloquially refer to the data resulting from a single mass spectrometry experiment as a “run”. A run may represent a shotgun LC-MS experiment containing tens of thousands of MS/MS scans. Alternatively, it might originate from a DIA experiment quantifying thousands of peptides, or from a Selected Reaction Monitoring (SRM) experiment spanning hundreds of transitions. The mzQC format is also intended to represent mass spectrometry quality when liquid chromatography is not directly coupled; a run may also represent a set of spectra resulting from a plate of MALDI-TOF data or from a GC-MS experiment. In all of these cases, an mzQC file will typically contain quality metrics for a collection of multiple spectra. The mzQC specification is equally applicable, however, to describe the quality (for example, the identifiability) of a single mass spectrum. The specification also extends to quality representation for “sets” of runs, either as groupings informed by the experimental design, longitudinal data, or differentiated by instrument, batch, site, etc.

3 mzQC Use Cases

The following use cases have driven the development of the mzQC format and have been used to define its scope for the initial release.

The mzQC format is intended for:

- Reporting quality metrics calculated by QC tools (such as Quameter (Ma et al. 2012), PTX-QC (Bielow et al. 2016), QCloud (Chiva et al. 2018)) coming from MS-based experiments.
- Presentation of quality reports to researchers for assessment of instrument performance.
- Recording longitudinal QC metrics to monitor instrument health over time.

- Performing quality control of collections of MS experiments across batches, biological or technical conditions, studies, or laboratories.
- Storing and archiving QC metrics next to their originating raw MS files and derived results in public data repositories or internal laboratory information management systems (LIMS).
- Exploring and selecting datasets within public data repositories based on desired data characteristics.
- Provide QC metrics as input for visualization in reports and dashboards.

Several use cases in which mzQC files can be used are described in more detail below.

3.1 Identifying Low-Performing MS Experiments Using Outlier Detection

When working with a set of mass spectrometry experiments, researchers frequently want to determine whether the collection of files are of consistent quality; this is a prerequisite for detecting differences between cohorts, for example. High-level quality metrics for each MS experiment (such as the number of identified spectra, peptides, and proteins; the mass calibration error; the width of chromatographic peaks; etc.) can be generated using QC software and exported to individual mzQC files. Subsequently, these separate mzQC files can be combined into a single mzQC for further unified analysis. An outlier detection algorithm can be used to project the experiments in a single principal components analysis (Wang et al. 2014). Experiments that fall far away from the main group of experiments may then be interrogated as outliers to investigate potential data issues and sources of performance degradation.

3.2 Longitudinal Monitoring of Instrument Performance

As instrument performance gradually degrades over time, data should be monitored carefully and preventative tuning needs to be performed regularly to avoid data quality issues. Longitudinal performance tracking is essential to assess whether the instrument is still within its operational parameters and to schedule timely interventions. QC metrics can be stored in the mzQC format after individual experiments are performed to record the instrument performance at a specific point in time. As such, system suitability test results can be consulted prior to data acquisition, and this information can be used to contextualize the experimental data quality. Additionally, QC metrics for multiple experiments, covering a longer time span, can be compared across multiple mzQC files. QC metrics can be consistently queried from multiple mzQC files corresponding to individual experiments or can be combined in a single mzQC file for easy analysis of instrument performance over time. Additionally, the information stored within the mzQC file(s) can be used to assess the longitudinal data quality across batches within an experiment or for inter-lab quality assessments (given that the sample content originates from the same biological resource and the data was acquired over different timepoints).

3.3 Producing Audience-Targeted Quality Reports

Most MS experiments conducted are intended to be shared with other parties such as a specific collaborator, customer, or a broader audience, either immediately or at a later point. Examples include data acquisition by a core facility or an external collaborator, or data submission to a public repository,

which is often required prior to publication in a scientific journal. Downstream use of the data can be facilitated if it is characterized by an accompanying quality report, particularly if the report targets a specific audience or a certain downstream use case. By including the metrics relevant for the envisioned audience(s) in an mzQC file, quality reports can be tailored to collaborators, core facility customers, and general data consumers like data repository users. The report can be as simple as only reporting the quality of sample runs, it can include metrics derived for multiple runs in experimental groups, or it can even be combined with instrument health information from longitudinal monitoring before and during sample run acquisitions.

3.4 Assisting in Novel Instrument Method Development

Novel instrument methods, such as data-independent acquisition (DIA) (Gillet et al. 2012) and BoxCar acquisition (Meier et al. 2018), are being developed to expand the robustness, depth, and coverage of MS experiments. For DIA experiments, simple constant-window strategies can mean that some isolation windows are crowded with peptides or fragments while others have relatively low density. Quality metrics can guide researchers to variable-size windows that are more uniform in peptide density, leading to better subsequent identification through spectral libraries and improved quantitative precision. Novel software tools to characterize DIA experiments and report their metrics via mzQC are currently already in development.

3.5 Quality Control of Metabolomics Experiments

Similar to proteomics, in metabolomics quality control samples can be used to provide a mechanism to judge the quality of the dataset or assay produced and to assess the analytical variance of the data and instrument variance over time. Several different types of quality samples can be used within an experimental setup. A common QC sample is a pooled sample, for which a small aliquot of each biological sample in the study set is mixed together. Several different pooled QC samples might be used throughout the study, placed in certain locations or randomly throughout the run, depending on the experimental setup. Blank samples are used to assess the carryover and ‘leakage’ from the columns. Often labs can use a commercially available QC sample, for example, NIST SRM 1950 human serum, or alternatively, create a stock pooled QC sample that differs from the one created using a mixture of the samples within the current experiment to measure the longitudinal variance of the instrument performance across different batches. Such quality metrics can be used for cross-study and cross-lab comparison. Another type of QC for metabolomics are synthetic cocktails that include multiple representatives from different classes of metabolites expected in a study. Overall, the QC sample measurements provide a qualitative and quantitative representation of the entire collection of samples in a study. An mzQC document can capture such quality indicators, to be fed directly into analysis software, for visualization purposes, for pre-analysis data cleaning, or to create a study quality report.

4 Relationship to Other Data Standard Specifications

The mzQC format describes quality control information from MS-based experiments and is therefore dependent on data in different formats. The mzQC specification tries to minimize the replication of information contained in other PSI formats and uses concepts of and references to related specifications.

4.1 The Quality Control Markup Language (qcML)

The predecessor to the mzQC format, the qcML format (Walzer et al. 2014), is based on a data representation in XML, which constitutes the major difference between the formats. While JSON has a large feature intersection with XML, its simpler structure and widespread language support will facilitate implementation of mzQC producing and consuming software, as exemplified by the various software tools that already support mzQC compared to limited community uptake of qcML. The organizational structure of the data is largely preserved between qcML and mzQC, like the hierarchical structure and representation of quality metrics through cvTerm-like objects. An important difference between the qcML format and the mzQC format is that complex data types, such as lists or tables of QC metrics can be represented natively in the more versatile mzQC format while their support in qcML is limited and cumbersome. Additionally, lessons learned from the use of qcML led to a refinement of the actual value representation of quality metrics in the mzQC format. The use of a CV to specify and exchange quality metrics was also adopted and evolved to ensure machine readability and to reflect a broader base of use cases.

4.2 The Quality Control Controlled Vocabulary (QC CV)

The PSI-QC controlled vocabulary is intended to provide terms for the definition of quality metrics and related supporting values. The CV has been generated initially with a collection of published and basic metrics. Further care went into the definition of the metrics and the involved values to help interpretation, use, and visualization. The vocabulary builds on established terms and definitions from chemistry, physics, and biology ontologies in references and term relations. The main purpose of the QC CV lies in the definition of metrics related to mass spectrometry quality control. Additionally, another purpose is to provide the underlying definitions for the data structures usable within mzQC files and specific metrics.

As recommended by the PSI CV guidelines (Mayer et al. 2013), psi-qccv.obo should be dynamically maintained via either the psidev-qc-dev@lists.sourceforge.net mailing list or (preferably) the QC working group's GitHub repository (https://github.com/HUPO-PSI/mzQC/issues). This allows any user to request new terms in a transparent manner and in agreement with the community involved. Changes and new entries can be requested and discussed within a dedicated issue list, where a template is available to guide new entry definitions. Once a consensus is reached among the community the new terms are added within a few business days.

Ultimately, we envision that the growth of the QC CV will slow, at which point it will be desirable to merge it with the MS CV (see below). We therefore follow the same principles of CV maintenance as the MS CV to avoid incompatibilities, which are likely to occur in case of duplications, redefinitions, and insufficient referencing.

Online reference: https://github.com/HUPO-PSI/mzQC/tree/master/cv

4.2.1 CV Term Creation for QC Metric Definition

The CV contains entries for metrics that can be recorded in the mzQC files. Even if the mzQC format allows storing any metric information, the CV makes it possible to interpret the actual values.

Each metric (and CV entry request) MUST include the following information:

- Name: A (short) string describing your metric.
- Definition: A longer description. This MUST include information about how the metric should be represented in an mzQC file.
- Comment: OPTIONAL details on how the metric should be interpreted (e.g. is a higher value better, can it only be interpreted relative to...).
- Metric type: Is the metric type a single value, an n-tuple, a table, or a matrix?
- Categorization: A categorization can OPTIONALLY be supplied. Examples are whether the metric depends on spectrum, peptide, protein, or metabolite identifications; or to describe the metric context.

Example CV term:

```
[Term]
id: QC:4000059
name: MS1 count
def: "The number of MS1 events in the run." [PSI:QC]
comment: This comment is for illustration purposes.
is_a: QC:4000003 ! single value
is_a: QC:4000010 ! ID free
is_a: QC:4000023 ! MS1 metric
property_value: has_units STATO:0000047
relationship: has_relation MS:1000579 ! MS1 spectrum
relationship: has_relation QC:4000013 ! QC metric relation: one run
```

ID

```
id: QC:4000059
```

Each term MUST have a unique ID, specified as `QC:XXXXXXX`. Metric IDs are immutable and not reusable (e.g. for redefinition), and will be assigned upon inclusion or redefinition.

Name

```
name: MS1 count
```

Each term MUST have a human-readable name. The name SHOULD be informative, SHOULD consist of maximum 100 characters, and SHOULD only consist of alphanumeric 7-bit ASCII characters, spaces, and punctuation marks (`[\-_,\.]`).

Both the ID and the name for each term will be given in the mzQC files as well when the term is used.

Definition

```
def: "The number of MS1 events in the run." [PSI:QC]
```

The definition SHOULD consist of a short explanation of the term and how it should be stored in the mzQC file. The description SHOULD also provide aid in interpreting the values. The definition section SHOULD NOT contain calculation or interpretation details, but rather it should explain the purpose, requirements, and scope of the metric.

Comment

```
comment: This comment is for illustration purposes.
```

The comment section SHOULD contain calculation and interpretation details, like whether smaller or bigger values are desirable. It is also RECOMMENDED to give a short explanation about how the metric works. If the metric calculation is not obvious, the calculation is RECOMMENDED to be briefly described in common terms. For published metrics, it is also RECOMMENDED to refer to the corresponding code.

Value Type And Unit

```
is_a: QC:4000003 ! single value
property_value: has_units UO:0000010
property_value: has_type NCIT:C25330
```

A single value metric with as unit the duration (`NCIT:C25330`) in seconds (`UO:0000010`), for example, the run (retention time) duration.

```
is_a: QC:4000003 ! single value
property_value: has_units UO:0000221
property_value: has_type STATO:0000237
```

A single value metric with as unit the standard deviation (`STATO:0000237`) in Dalton (`UO:0000221`), for example, the standard deviation of the distribution of precursor mass errors of identified spectra.

Each term that reports a value MUST indicate the corresponding value type using an `is_a` relation. Different value types are possible: single value, n-tuple, table, or matrix. A value must be associated with a unit, see Metric Categorization. Depending on the value type, different additional categorization is REQUIRED.

- **single value:** Unit specification using `has_unit` is REQUIRED, type specification using `has_type` is RECOMMENDED.
- **n-tuple**: Unit specification using `has_unit` is REQUIRED, type specification using `has_type` is RECOMMENDED. Units and types (optional) MUST be uniform for all values. An n-tuple is represented by a JSON array, which implicitly defines its length.
- **table**: A table MUST have one or more columns defined using `has_column` and MAY have optional columns defined using `has_optional_column`. A table is represented using a JSON key–value object with as key(s) the column term names/accessions and as value(s) JSON arrays of uniform value type and length.
- **table column type definitions**: Unit specification using `has_unit` is REQUIRED, type specification using `has_type` is RECOMMENDED. The term name will be used as the column's header.
- **matrix**: Unit specification using `has_unit` is REQUIRED, type specification using `has_type` is RECOMMENDED. Units and types (optional) MUST be uniform for all values. A matrix is represented by a JSON array of JSON arrays where the inner arrays MUST be of uniform length, which implicitly defines the matrix dimensions.

Metric Categorization

```
is_a: QC:4000010 ! ID free
```

```
is_a: QC:4000023 ! MS1 metric
relationship: has_relation MS:1000579 ! MS1 spectrum
relationship: has_relation QC:4000013 ! QC metric relation: one run
```

Different types of categorization can be assigned to CV terms. First, it is RECOMMENDED to specify whether a metric requires identification information to be computed (ID based) or not (ID free). Second, additional categories to describe the metric context (from which data is the metric is derived, to which element of the instrumental setup does the metric pertain, etc.) can be specified as well. It is RECOMMEND to align the categorization of novel metrics to existing terms to facilitate consumption of related metrics.

```
property_value: has_units UO:0000010
property_value: has_column QC:4000117
```

If the metric term has an associated value, its unit MUST be defined using the `property_value` tag. "Single", "n-tuple", and "matrix" type values MUST be assigned a single, uniform unit type with `has_units`. For "table" type values, one or more `has_column_type`/`has_optional_column_type` specifications MUST be associated with the table. These implicitly define the column units through the `has_units` attributes of the corresponding column definitions.

```
property_value: has_type STATO:0000237
```

For full semantic integration, it is RECOMMENDED to specify the value type for automatic processing and interpretation of the value. It is RECOMMENDED to source value types from STATO.

Additional Information

```
synonym: "MS2-4A" NARROW []
```

In case of reimplementing, renaming, or redefining a metric, it is RECOMMENDED to also add synonym attributes with either the name or ID of the initial metric. It is not required for the initial metric to be included in any controlled vocabulary, but the name SHOULD be unambiguous and recognizable (e.g. from the source publication). Synonyms can be "RELATED" (the defined metric is similar, but not the same as what is connected with the synonym name), "NARROW" (the metric's values can be identically interpreted as in the meaning of the synonym metric, however, definition and calculation may somewhat differ), "EXACT" (the defined metric is basically a result of renaming).

4.3 The PSI Mass Spectrometry Controlled Vocabulary (MS CV)

The PSI-MS controlled vocabulary (Mayer et al. 2013; Mayer et al. 2014) has been generated by software vendors and academic groups working in the area of mass spectrometry and proteome informatics and is a rich source of terms allowing for the annotation of mzQC files. Some terms describe attributes that must be coupled with a numerical value and optionally a unit for that value. Terms that require a value are denoted by having a `value-type` reference in the CV itself. Terms that need to be qualified with units are denoted with a `has_units` relationship in the CV itself.

Example:

```
[Term]
id: MS:1001117
name: theoretical mass
def: "The theoretical mass of the molecule (e.g. the peptide sequence and its
modifications)." [PSI:PI]
is_a: MS:1001105 ! peptide sequence-level identification attribute
relationship: has_units UO:0000221 ! dalton
```

Online reference: https://github.com/HUPO-PSI/psi-ms-CV

4.4 Other Controlled Vocabularies

Due to the wide availability of alternative ontologies, these can be used to define terms to represent quality metrics as well. However, the requirements enforced for QC CV terms may not be met in all cases and caution needs to be exercised not to introduce value interpretation ambiguities. We acknowledge the existence of overlaps in the concepts defined in openly available bio-ontologies, and hence only suggest precedence of choice if possible. It is desirable to integrate strictly quality control related concepts in the QC CV, even if relevant terms might already be available elsewhere. Including all relevant terms in the QC CV will help to reduce dependence on outside projects and ensures that a proper definition is available for all terms included in mzQC files. However, the flexible design of the mzQC format does not preclude the inclusion of third party or custom CVs besides the QC CV. Note that when custom CVs are included the user should make sure that there are no namespace clashes between different CVs. For CVs mentioned in this document and other widely established vocabularies this can be assumed to be the case, but it can be verified with the help of CV lookup services such as ontobee.org.

In general, units SHOULD be sourced from the Units of Measurement Ontology (UO), if available, otherwise from the Statistical Methods Ontology (STATO) or others as necessary. Protein modifications SHOULD be sourced from Unimod or PSI-MOD where possible.

4.5 mzML

mzML (Martens et al. 2011; http://www.psidev.info/mzml) is the PSI standard for capturing mass spectra/peak lists resulting from an MS experiment. As one of the main sources of raw spectral data from which quality metrics may be computed, mzML has a strong relationship to mzQC. In general, mzQC files will refer to mzML files as the input for metric calculation rather than the converse. To refer to individual spectra in an mzML file, the native spectrum identifier format (MS:1000767; nativeID in the MS CV) is the preferred strategy. However, it will also be possible to refer to mzQC assessments from within an mzML file, since metric entries in mzQC and QC CV terms may be used to annotate spectra (as a `<cvParam>` element of a `<spectrum>` object) or even an entire mzML file (likely via a `<cvParam>` element of a `<processingMethod>` object) to reflect a spectrum filtering strategy. It is RECOMMENDED that mzQC should be used in conjunction with PSI standards such as mzML, mzTab, or mzIdentML, although it is also possible to use mzQC with other peak list formats (for example, such as the Mascot Generic Format (MGF)).

4.6 mzIdentML

mzIdentML (Vizcaíno et al. 2017; http://www.psidev.info/mzidentml) is the PSI standard for capturing peptide and protein identification data. As with mzML, the connection from mzQC to mzIdentML is strong, as input data to calculate identification-based quality metrics has to be sourced from identification file formats. Additionally, similar to mzML, mzQC information can be referred to in mzIdentML files via `<cvParam>` elements of `<SpectrumIdentificationItem>`, `<SpectrumIdentificationResult>`, and `<SpectrumIdentificationList>`.

4.7 mzTab

mzTab (Griss et al. 2014; https://github.com/HUPO-PSI/mztab) is the combined 'lightweight supplement' PSI standard to report identification and quantification results in a tabular text format. It is easy to parse and contains the essential information required to evaluate results. mzTab bridges the gap between a pure text summary and machine-only readable formats. Its design allows a comprehensive summarization of processed MS results, in addition to referring to more details present in XML-based standard formats. Due to the minimally required comprehensiveness, its content values represent a sufficient base to calculate basic to intermediate quality metrics. mzTab also has limited support for metabolomics via the mzTab-M format (Hoffmann et al. 2019).

4.8 Universal Spectrum Identifier

The Universal Spectrum Identifier (USI; http://www.psidev.info/usi) describes a virtual path to locate a spectrum within a public MS dataset available via ProteomeXchange. When storing quality metrics related to individual spectra, besides the aforementioned nativeID format, USIs can be used to uniquely refer to spectra.

5 Format Specification

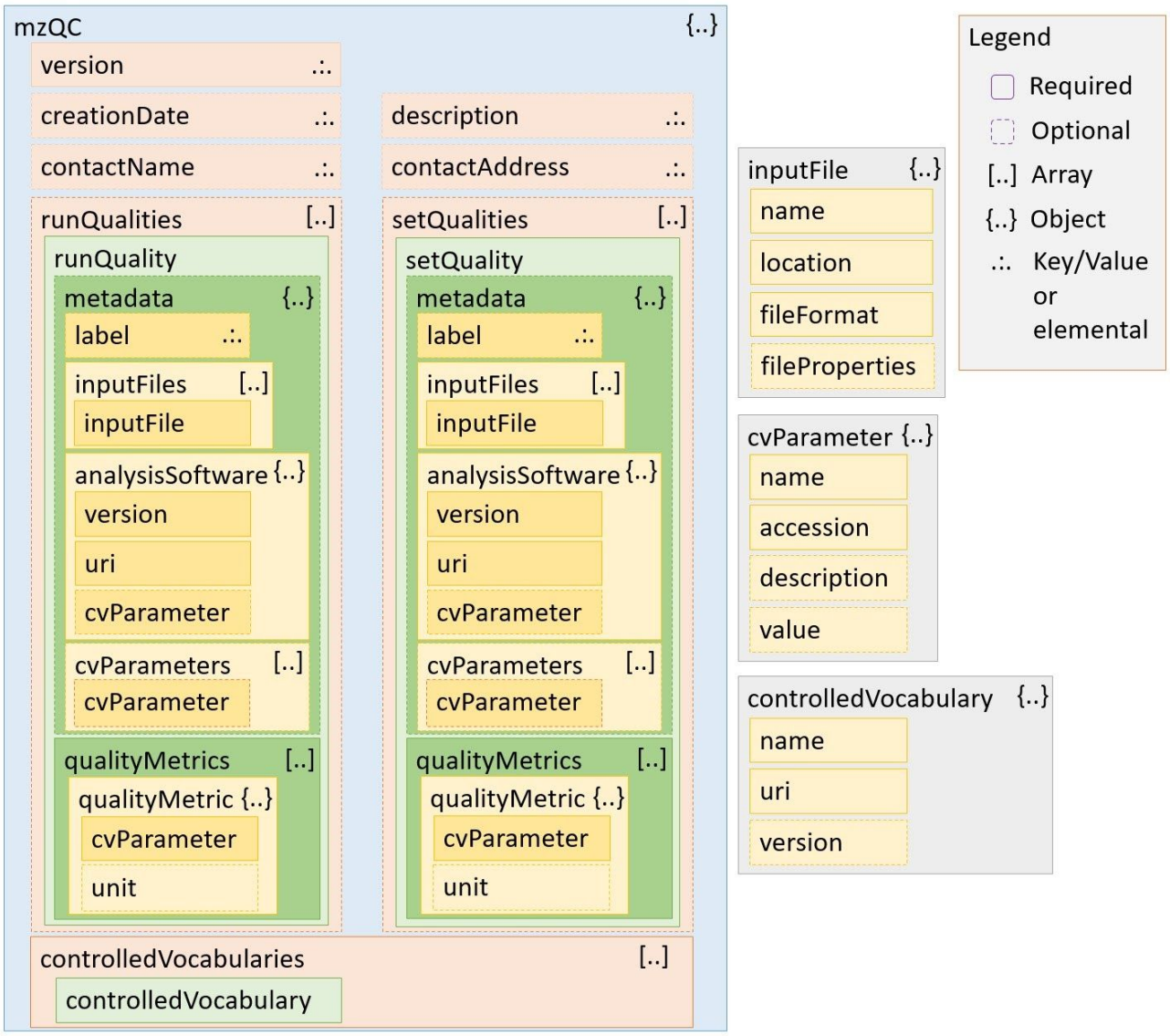

Figure 1: Diagrammatic overview of the mzQC schema.

5.1 Schema Sections

The mzQC format can accommodate multiple metrics, possibly from multiple MS runs. Several dedicated (and named) data structures (arrays) are available for this purpose. These are named as the plural of the elements they provide space for, for example, `controlledVocabularies` contains the `controlledVocabulary` objects. In the following, the schema elements are described in outer-to-inner order, starting with the `mzQC` root element (Figure 1). If elements are allowed in multiple places in this

hierarchy, they are listed only once at the first valid occurrence. All objects described are mandatory and of single occurrence, unless stated otherwise. Arrays need to contain at least one element or MUST otherwise be omitted. The acceptable child elements are listed under the "item types" enumeration. The naming convention followed for all schema elements is camelCase (starting lower case) to denote a type. The examples given are sometimes abbreviated for conciseness where putting the example into context would result in verbosity. Symbols and characters used are the same as seen in the legend of Figure 1.

We recommend consulting the introductory companion documents listed in the appendix for examples of complete mzQC files and to get more familiar with the schema.

5.1.1 mzQC

Root element of an mzQC file. It MUST enclose the listed elements and MUST be the sole root element in an mzQC file.

Type: object
Object definition:

Name	Type	Required	Description
`version`	string	true	Version of the mzQC format.
`creationDate`	date-time	true	Creation date of the mzQC file.
`contactName`	string	false	Name of the operator/creator of this mzQC file.
`contactAddress`	string	false	Contact address (mail/e-mail or phone)
`readMe`	string	false	Description and comments about the mzQC file contents.
`runQualities`	array	true*	Description in section 5.13.
`setQualities`	array	true*	Description in section 5.1.4.
`controlledVocabularies`	array	true	Description in section 5.1.17.

*At least one of `runQualities` or `setQualities` MUST be present.

The `version` string MUST be in the format "major.minor.patch", each separated by a period. The creation date string MUST be in ISO8601 format, for example, `2019-10-29T14:40:17`.

Example:

```
"mzQC": {
    "version": "1.0.0",
    "creationDate": "2019-10-29T14:40:17",
    "runQualities": [..],
    "controlledVocabularies": [..]
}
```

5.1.2 baseQuality

Base element from which both `runQuality` and `setQuality` elements are derived. `baseQuality` is an abstract element; only its derived `runQuality` and `setQuality` elements SHALL be used in a valid mzQC document.

Type: object
Object definition:

Name	Type	Required	Description
`metadata`	object	true	Description in section 5.1.7.
`qualityMetrics`	array	true	Description in section 5.1.15.

Example: see `runQuality` or `setQuality`.

5.1.3 runQualities

OPTIONAL list of `runQuality` elements. However, it is REQUIRED that at least one of `runQualities` or `setQualities` is present. If specified, the `runQualities` list MUST contain at least one `runQuality` element.
Although it is possible to include multiple `runQuality` elements in the `runQualities` list, a consideration for using JSONPath is that querying of individual `runQuality` elements can become less efficient.

Type: array
Item types: `runQuality`
Min/max: (1, -)

Example:

```
"runQualities": [..]
```

5.1.4 setQualities

OPTIONAL list of `setQuality` elements. However, it is REQUIRED that at least one of `runQualities` or `setQualities` is present. If specified, the `setQualities` list MUST contain at least one `setQuality` element.
Although it is possible to include multiple `setQuality` elements in the `setQualities` list, a consideration for using JSONPath is that querying of individual `setQuality` elements can become less efficient.

Type: array
Item types: `setQuality`
Min/max: (1, -)

Example:

```
"setQualities": [..]
```

5.1.5 runQuality

Element containing `metadata` and `qualityMetrics` for a single run.

Type: `baseQuality`
Object definition:

Name	Type	Required	Description
`metadata`	object	true	Description in section 5.17.
`qualityMetrics`	array	true	Description in section 5.1.15.

Example:

```
runQualities: [
    {
        "metadata": {
            "label": "file_1_4h",
            "inputFiles": [..],
            "analysisSoftware": [..]
        },
        "qualityMetrics": [..]
    }
]
```

5.1.6 setQuality

Element containing `metadata` and `qualityMetrics` for a collection of related runs ("set"). It is REQUIRED that `setQuality` only contains `qualityMetrics` which describe properties which apply to the set as a whole (as opposed to a list of values where each value can be attributed to a single run in isolation and would thus rather be stored as `runQuality`).

Type: `baseQuality`
Object definition:

Name	Type	Required	Description
`metadata`	object	true	Description in section 5.1.7.
`qualityMetrics`	array	true	Description in section 5.1.15.

It is REQUIRED that all `qualityMetrics` contained in the `setQuality` are derived from the same set of input files. For example, if `qualityMetric` m_1 contained in `setQuality` s_1 is derived from runs r_1, r_2, and r_3, then `qualityMetric` m_2 contained in s_1 MUST also be derived from runs r_1, r_2, and r_3. If this is not the case, m_2 MUST be contained in a different `setQuality` s_2.

Example:

```
setQualities: [
    {
        "metadata": {
            "label": "groupA",
            "inputFiles": [..],
            "analysisSoftware": [..]
        },
        "qualityMetrics": [..]
    }
]
```

5.1.7 cvParameter

Base element for a term that is defined in a controlled vocabulary, with OPTIONAL value.

Type: object
Object definition:

Name	Type	Required	Description
`accession`	string	true	Accession number identifying the term within its controlled vocabulary (pattern: `^[A-Z]+:[A-Z0-9]+$`).
`name`	string	true	Name of the controlled vocabulary term describing the parameter.
`description`	string	false	Definition of the controlled vocabulary term.
`value`	any	false	Value of the parameter.

The `description` attribute MAY be omitted, especially in larger documents to reduce their size, since the CV term definition can be easily retrieved from the respective CV via the `accession`.

Example:

```
{
    "accession": "MS:1000569",
    "name": "SHA-1",
    "value": "76de62feccaaaadb608e89d897db57135e39ad87"
}
```

5.1.8 metadata

Metadata describing the *runQuality* or *setQuality* to which it belongs.

Type: object
Object definition:

Name	Type	Required	Description
`label`	string	true	Unique name for the run (for `runQuality`) or set (for `setQuality`).
`inputFiles`	array	true	Description in section 5.1.9.
`analysisSoftware`	array	true	Description in section 5.1.13.
`cvParameters`	array	false	Description in section 5.1.14.

We RECOMMEND that the `label` is informative, for example so that it can be used to label values in a figure. Relevant information that the `label` can convey relates to the experimental design, the base name of the input file(s) to which the QC metrics correspond (for a *runQuality*), or a descriptive label of a grouping (for a *setQuality*, e.g. “timepoint4h”).

Example:

```
"metadata": {
    "label": "groupA",
    "inputFiles": [..],
    "analysisSoftware": [..],
    "cvParameters": [..]
}
```

5.1.9 inputFiles

List of input files from which the QC metrics have been generated. At least one input file MUST be present and it is RECOMMENDED that this `inputFile` corresponds to a raw or peak file.

Type: array
Item types: `inputFile`
Min/max: (1, -)

Example:

```
"inputFiles": [..]
```

5.1.10 inputFile

Input file used to generate the QC metrics. We RECOMMEND that only meta information about the files used are stored here.

Type: object
Object definition:

Name	Type	Required	Description
`name`	string	true	The `name` MUST uniquely match to a `location` (specified below) listed in the mzQC file.
`location`	string (URI)	true	Unique file location, REQUIRED to be specified as a URI. The file URI is RECOMMENDED to be publicly accessible.
`fileFormat`	`cvParameter`	true	Description in section 5.1.11.
`fileProperties`	array	false	Description in section 5.1.12.

We RECOMMEND to use a short, yet descriptive, `name`, such as the base file name (without file extension), which may lend itself as a label on a plot.

Example:

```
{
    "name": "H2-1-1",
    "location": "ftp://massive.ucsd.edu/MSV000081205/ccms_peak/RAWs/H2-1-1.mzML",
    "fileFormat": {
        "accession": "MS:1000584",
        "name": "mzML format"
    },
    "fileProperties": [
        {
            "accession": "MS:1000747",
            "name": "completion time",
            "value": "2012-02-03T11:00:41"
        },
        {
            "accession": "MS:1000569",
            "name": "SHA-1",
            "value": "76de62feccaaaadb608e89d897db57135e39ad87"
        }
    ]
}
```

5.1.11 fileFormat

Type of input file.

Type: `cvParameter`

Example:

```
"fileFormat": {
    "accession": "MS:1000584",
    "name": "mzML format"
}
```

5.1.12 fileProperties

Detailed information of the input file.

RECOMMENDED properties and their CV accessions are:

- Completion time of the input file (MS:1000747).
- Checksum of the input file (any child of: MS:1000561 ! data file checksum type).

Type: array
Item types: `cvParameter`
Min/max: (1, -)

Example:

```
"fileProperties": [
    {
        "accession": "MS:1000747",
        "name": "completion time",
        "value": "2012-02-03T11:00:41"
    },
    {
        "accession": "MS:1000569",
        "name": "SHA-1",
        "value": "76de62feccaaaadb608e89d897db57135e39ad87"
    }
]
```

5.1.13 analysisSoftware

Software tool(s) used to generate the QC metrics.

Type: array
Item types: `cvParameter`
Min/max: (1, -)
Object definition:

Name	Type	Required	Description
`accession`	string	true	Accession number identifying the term within its controlled vocabulary (pattern: `^[A-Z]+:[A-Z0-9]+$`).
`name`	string	true	Name of the controlled vocabulary term describing the software tool.
`description`	string	false	Definition of the controlled vocabulary term.
`value`	any	false	Value of the software tool.
version	string	true	Version number of the software tool.
url	string (URI)	true	Publicly accessible URI of the software tool or documentation.

Example:

```
"analysisSoftware": [
    {
        "accession": "MS:1000752",
        "name": "TOPP software",
        "version": "2.4",
        "uri": "https://www.openms.de/"
      }
]
```

5.1.14 cvParameters

OPTIONAL list of `cvParameter` elements containing additional metadata about its parent `runQuality`/`setQuality`. If specified, the `cvParameters` list MUST contain at least one `cvParameter` element.

Type: array
Item types: `cvParameter`
Min/max: (1, -)

Example:

```
"cvParameters": [..]
```

5.1.15 qualityMetrics

The collection of `qualityMetrics` for a particular `runQuality` or `setQuality`.

Type: array
Item types: `qualityMetric`
Min/max: (1, -)

Example:

```
"qualityMetrics": [..]
```

5.1.16 qualityMetric

Element containing the value and description of a QC metric defined in a controlled vocabulary.

Type: `cvParameter`
Object definition:

Name	Type	Required	Description
`accession`	string	true	Accession number identifying the term within its controlled vocabulary (pattern: `^[A-Z]+:[A-Z0-9]+$`).
`name`	string	true	Name of the controlled vocabulary element describing the metric.
`description`	string	false	Definition of the controlled vocabulary term.
`value`	Single value, n-tuple, table, matrix	false	Value of the metric (see section 6).
`unit`	`cvParameter` / array	false	One or more controlled vocabulary elements describing the unit of the metric (if applicable — see section 6).

The `description` attribute MAY be omitted, especially in larger documents to reduce their size, since the CV term definition can be easily retrieved from the respective CV via the `accession`. We RECOMMEND to include the description only if added verbosity is expected to help (human) readability. In any case, the description text MUST NOT be altered from the term definition. If a `qualityMetric` does not take a value, both the `value` and `unit` attributes MUST NOT be present. For a more independent mzQC file that needs to be *consumed* without the presence of the whole CV parsed, it is REQUIRED to repeat the unit as defined in the CV.

Example:

```
{
    "accession": "QC:4000059",
    "name": "MS1 count",
    "description": "The number of MS1 events in the run.",
    "value": 7787,
    "unit": {
        "accession": "STATO:0000047",
        "name": "count"
    }
}
```

5.1.17 controlledVocabularies

Collection of controlled vocabulary elements used to refer to the source of the used CV terms in the `qualityMetric` objects (and others).

Type: array
Item types: `controlledVocabulary`
Min/max: (1, -)

Example:

```
"controlledVocabularies": [..]
```

All CVs containing terms used in an mzQC document MUST be referenced in the `controlledVocabularies` element. To allow direct JSONPath queries of mzQC documents, there MUST NOT be any namespace clashes between the used CVs. For CVs mentioned in this document and other widely established vocabularies this can be assumed to be the case, but it can be verified with the help of CV lookup services such as ontobee.org.

5.1.18 controlledVocabulary

Element describing a controlled vocabulary used to refer to the source of the used CV terms in `qualityMetric` objects (and others).

Type: object
Object definition:

Name	Type	Required	Description
`name`	string	true	Full name of the controlled vocabulary.
`uri`	string (URI)	true	Publicly accessible URI of the controlled vocabulary.
`version`	string	false	Version of the controlled vocabulary.

The `uri` is RECOMMENDED to be a public URL. In cases where a code repository is referenced, the `uri` MUST be stable, i.e. refer to either a specific commit (for that version) or a (direct) release link of the file.

Example:

```
{
    "name": "Proteomics Standards Initiative Mass Spectrometry Ontology",
    "uri": "https://raw.githubusercontent.com/HUPO-PSI/psi-ms-CV/
5e02c611b8392301ce00e8d7bb3a4d63848ef8b3/psi-ms.obo",
    "version": "4.1.45"
}
```

6 cvParameter Values For Metrics

Values of `qualityMetrics` are defined by their corresponding controlled vocabulary term. They can be one of four different types.

Type: string / number / boolean / array / object
Object definition:

Name	CV Accession	Description	Example
single value	QC:4000003	A single string, number, or boolean value.	`"target"` or `1.0` or `true`
n-tuple	QC:4000004	An array of values of the same type (and unit).	`[2, 3, 5, 7, 11]`
table	QC:4000006	An object with each column a key–value pair of the form `column name: [array of values]`. All columns MUST have equal length. Values in the table MAY have different types in each column (in contrast to a matrix). The actual structure of the table is defined by the CV term.	`{ score: [0, 1, 1], intensity: [0, 13, 9] }`
matrix	QC:4000007	An array of child arrays with values. All child arrays MUST have the same length and represent a row of the matrix, i.e. the matrix elements are indexed as [row,	`[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

		column]. All values in a matrix MUST have the same type.	

For most metric types only a single unit is needed (single value, n-tuple, matrix). In contrast, the table metric type can have different units for each individual column. For a concrete example we refer to the companion document on CV term use (Appendix).

7 General Recommendations

7.1 Element Order in cvParameter Derived Objects

JSON does not enforce a specific order of elements within an object. However, for improved readability, we RECOMMEND that if possible, "human-readable" elements, like `name` and `value`, are listed first.

7.2 Metadata

Depending on the use case to which mzQC is applied, there can be supplemental data that is not strictly metadata but is still considered relevant. We RECOMMEND that only meta information about the input files that were used to compute the QC metrics is stored in the `metadata` section of an mzQC file. If a piece of data does not fit into any `metadata` section, for example, such as the instrument type, it can be considered a QC metric to a lesser degree and should be stored in the respective `qualityMetrics` section.

7.3 Compression

We have attempted to remove redundancy from the mzQC format wherever possible during its design. As a result typical mzQC files are not expected to grow overly large, however, due to the diverse nature of quality metric values and because much of the specification contains long form names to maintain human readability, there is ample opportunity to improve storage efficiency through compression. Compression of mzQC files comes at little overhead as text-based formats are inherently ideal targets for compression and common compression libraries are freely available on all computing platforms. Therefore, if a low memory footprint is desired, we RECOMMEND making use of a free compression algorithm, such as gzip.

7.4 Number of runQualities in mzQC Files

The mzQC specification allows storing multiple `runQuality` or `setQuality` elements in a single mzQC file. The main purpose hereof is to group QC information from multiple runs for the subsequent calculation of `setQuality` metrics (or similarly, higher level `setQuality` metrics in a hierarchical experimental set-up). However, to facilitate mzQC file processing and storage, we RECOMMEND to

restrict individual mzQC files to single `runQuality`/`setQuality` elements when possible. An example of the benefits of such a policy can be found in the companion document on the core facility use case.

7.5 Development

For ongoing development, documentation, and to report issues with the format specification please see the working group's GitHub issues page (https://github.com/HUPO-PSI/mzQC/issues).

8 Pending Issues

8.1 Inclusion of Graphics

It can be desirable to include graphical metric plots when mzQC files are used to create quality reports or for archival purposes, as metric values can be much more intuitive to grasp when visualised. However, producing such plots is highly system dependent and often requires additional software of specific versions to be installed. To circumvent these limitations, metric plots can be directly stored in an mzQC file as a custom `qualityMetric` element. We RECOMMEND the metric value to be the base64 encoded string of the plot image file and we RECOMMEND to use a custom metric with a descriptive and unambiguous name that makes it clear from which QC metric(s) the plot is derived and that it is an encoded image.

Further work is needed to make this system work reliably between different methods of mzQC use, for which the respective standardization details are deferred to a later release of the mzQC standard. This includes considerations on the image formats encoded and controlled vocabulary additions for graphical metrics. Until then we RECOMMEND to only use this system in custom applications at your own risk.

8.2 Referencing

Due to the (few) limitations of the JSON format, intra- and inter-file referencing of `qualityMetric` elements is left to custom mechanisms. The specification of such a mechanism requires considerable effort to benefit only a few applications and is hence deferred to a later release of the mzQC standard. We RECOMMEND using a concept partially outlined in section 8.4 to only store a single `runQuality` element per mzQC file and replicate `qualityMetric` elements in-context if necessary. An example where this would be the case is when a `setQuality` uses multiple QC metrics from multiple MS runs. For good practice, the base `runQuality` elements should be replicated in an mzQC file that reports such a `setQuality`, making the references implicit.

8.3 Custom Thresholds and Flagging

In the current version, specifying thresholds for QC metrics or flagging metrics, runs, or sets of runs as "quality unfulfilled" is not directly supported. Once this is supported via the appropriate CV terms, an mzQC file may be enriched in a second pass with thresholds and flags together with a rationale as to

why the "quality unfulfilled" flag was set. Note that the mzQC format does not prescribe any thresholds denoting acceptable quality itself, this is left to the generating and interpreting software.

9 Conclusions

This document contains the specifications for using the mzQC format to represent quality control metrics in the context of biological mass spectrometry. This specification constitutes a proposal for a standard from the Proteomics Standards Initiative. These artefacts are currently undergoing the PSI document process, which will result in a standard officially sanctioned by PSI.

10 Authors

Chris Bielow
Freie Universität Berlin
Berlin, Germany
chris.bielow@fu-berlin.de

Wout Bittremieux
University of California San Diego
La Jolla, CA, USA
wbittremieux@health.ucsd.edu

David L. Tabb
Stellenbosch University
Cape Town, South Africa
dtabb1973@gmail.com

Julian Uszkoreit
Ruhr University Bochum
Bochum, Germany
julian.uszkoreit@rub.de

Mathias Walzer
European Bioinformatics Institute (EMBL-EBI)
Hinxton, United Kingdom
walzer@ebi.ac.uk

Reza Salek
The International Agency for Research on Cancer
Lyon, France
r7salek@gmail.com

Correspondence – Mathias Walzer (walzer@ebi.ac.uk).

11 Contributors and Software

In addition to the authors, the following people have contributed to the model development, gave feedback, or tested the mzQC format: Martin Eisenacher, Juan Antonio Vizcaino.

11.1 Implementations

Initial software implementations supporting the mzQC format are:

- mzqc-pylib: Python library to process and validate mzQC files. It includes a Python object model, (de-)serialisation functions, syntactic validation, and semantic validation of mzQC files.
- PTX-QC: An R package for creation of QC reports from MaxQuant results and OpenMS mzTab files
- QCCalculator: Python tool for base QC metric calculation from mzML, mzIdentML, and MaxQuant input files.

An up-to-date list of software tools that support the mzQC format can be found on https://github.com/HUPO-PSI/mzQC/blob/master/software.md.

12 Intellectual Property Statement

The PSI takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Copies of claims of rights made available for publication and any assurances of licenses to be made available or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the PSI Chair.

The PSI invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to practice this recommendation. Please address the information to the PSI Chair (see contacts information at PSI website).

13 Copyright Notice

14 References

Bielow, C., Mastrobuoni, G. and Kempa, S. 2016. Proteomics quality control: quality control software for maxquant results. *Journal of Proteome Research* 15(3), pp. 777–787.

Bradner, S. 1997. *Key words for use in RFCs to Indicate Requirement Levels*. RFC Editor.

Chiva, C., Olivella, R., Borràs, E., et al. 2018. QCloud: A cloud-based quality control system for mass spectrometry-based proteomics laboratories. *Plos One* 13(1), p. e0189209.

Gillet, L.C., Navarro, P., Tate, S., et al. 2012. Targeted data extraction of the MS/MS spectra generated by data-independent acquisition: a new concept for consistent and accurate proteome analysis. *Molecular & Cellular Proteomics* 11(6), p. O111.016717.

Griss, J., Jones, A.R., Sachsenberg, T., et al. 2014. The mzTab data exchange format: communicating mass-spectrometry-based proteomics and metabolomics experimental results to a wider audience. *Molecular & Cellular Proteomics* 13(10), pp. 2765–2775.

Hoffmann, N., Rein, J., Sachsenberg, T., et al. 2019. mzTab-M: A Data Standard for Sharing Quantitative Results in Mass Spectrometry Metabolomics. *Analytical Chemistry* 91(5), pp. 3302–3310.

Jones, A.R., Eisenacher, M., Mayer, G., et al. 2012. The mzIdentML data standard for mass spectrometry-based proteomics results. *Molecular & Cellular Proteomics* 11(7), p. M111.014381.

Martens, L., Chambers, M., Sturm, M., et al. 2011. mzML--a community standard for mass spectrometry data. *Molecular & Cellular Proteomics* 10(1), p. R110.000133.

Mayer, G., Jones, A.R., Binz, P.-A., et al. 2014. Controlled vocabularies and ontologies in proteomics: overview, principles and practice. *Biochimica et Biophysica Acta* 1844(1 Pt A), pp. 98–107.

Mayer, G., Montecchi-Palazzi, L., Ovelleiro, D., et al. 2013. The HUPO proteomics standards initiative- mass spectrometry controlled vocabulary. *Database: the Journal of Biological Databases and Curation* 2013, p. bat009.

Ma, Z.-Q., Polzin, K.O., Dasari, S., et al. 2012. QuaMeter: multivendor performance metrics for LC-MS/MS proteomics instrumentation. *Analytical Chemistry* 84(14), pp. 5845–5850.

Meier, F., Geyer, P.E., Virreira Winter, S., Cox, J. and Mann, M. 2018. BoxCar acquisition method enables single-shot proteomics at a depth of 10,000 proteins in 100 minutes. *Nature Methods* 15(6), pp.

440–448.

Vizcaíno, J.A., Mayer, G., Perkins, S., et al. 2017. The mzIdentML Data Standard Version 1.2, Supporting Advances in Proteome Informatics. *Molecular & Cellular Proteomics* 16(7), pp. 1275–1285.

Walzer, M., Pernas, L.E., Nasso, S., et al. 2014. qcML: an exchange format for quality control metrics from mass spectrometry experiments. *Molecular & Cellular Proteomics* 13(8), pp. 1905–1913.

Walzer, M., Qi, D., Mayer, G., et al. 2013. The mzQuantML data standard for mass spectrometry-based quantitative studies in proteomics. *Molecular & Cellular Proteomics* 12(8), pp. 2332–2340.

Wang, X., Chambers, M.C., Vega-Montoto, L.J., Bunk, D.M., Stein, S.E. and Tabb, D.L. 2014. QC metrics from CPTAC raw LC-MS/MS data interpreted through multivariate statistics. *Analytical Chemistry* 86(5), pp. 2497–2509.

15 Appendix

15.1 Examples

- single run
- QC2 Sample
- multiple runs
- metabolomics batch correction

15.2 Companion Documents

- mzQC: a “common currency” for quality control of biological mass spectrometry
- mzQC with multiple experiments
- mzQC format for analytical chemists
- cv guide for mzQC