PSI Recommendation
PSI Quality Control Workgroup
Version 1.0.0 DRAFT v5

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

December 8, 2023

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 Organization (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 v5, December 8, 2023.

Abstract

The Human Proteome Organization (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, storage of quality decisions, visualization efforts, and easy persistence and exchange of all of the above. Importantly, the standard makes no judgment about what constitutes good quality. 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 subsection of the PSI-MS 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.

Table of 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, does not consider tool support for data capture or storage, nor provides comprehensive examples of mzQC files. Tutorials and example material are available via the Appendix and through the PSI-QC working group repository.

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

1.1 JSON file format

Unlike most previous PSI standards, which are predominantly XML-based file formats (e.g., mzML (*1*), mzIdentML (*2*, *3*), mzQuantML (*4*)), 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 requires specialized software libraries, whose usage can be complex. The JSON file format has been developed to be a lightweight and universal data interchange format. As such, JSON files have a small memory footprint and have a wide built-in support in many programming languages. 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 (see Software). Similar to qcML, a previous yet non-standardized XML-based effort to represent MS quality information, mzQC preserves the organizational structure of the data, exemplified by the hierarchical structure and representation of quality metrics through `cvTerm`-like objects. The actual value representation and use of a controlled vocabulary (CV) to specify and exchange quality metrics was also evolved and adopted to ensure machine readability and to reflect a broader base of use cases.

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 broaden the reach of mzQC within the scientific community.

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. Keys must be strings.

Syntax: `{key_1: value_1, key_2: value_2}`
Example: `{"precursor_charge": 2, "is_contaminant": true}`

- 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, {"subobject": "value_4"}, value_5]`
  Example: `[true, {"precursor_charge": 42}, "text"]`

Values can be a string in double quotes, a number, a boolean value (`true` or `false`), `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. Syntactic validation is available through the usage of JSON Schema for annotation and validation of JSON documents.

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 defines keywords, such as "MUST NOT", that apply throughout this document. Section 3 lists use cases for the mzQC format. Sections 4 and 5 outline the relationships between mzQC and controlled vocabularies or other file format specifications, respectively. Sections 6 and 7 detail the format specification. Section 8 provides information on validation of mzQC files. Section 9 provides general recommendations on using mzQC, Section 10 lists a few pending issues, and Section 11 contains a brief summary and conclusion. Sections 12, 13, 14, 15, and 16 include the list of authors, software, 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 (*5*).

In this document, we colloquially refer to the data resulting from a single mass spectrometry experiment as a "run". For example, 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 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 mzQC format is intended to be applicable for any type of biological mass spectrometry, including proteomics, metabolomics, lipidomics, etc. As such, the following use cases have been deliberately kept general to cover a wide variety of applications.

Importantly, mzQC provides an objective medium to facilitate data comparison, exchange, and verification. The goal of the mzQC format is not to *judge* the quality of the data that it describes.

Qualitative or quantitative interpretation of data encoded in mzQC files will likely be context dependent. For example, an LC-MS/MS experiment might contain some number of MS scans and some number of MS/MS scans; these can be recorded in a run-level mzQC file. To judge that an individual LC-MS/MS experiment represents an outlier for the overall experiment, however, requires the context of the other experiments. A summary of these values for several runs (*set* of runs) could be recorded in a set-level mzQC file. Additionally, run-level QC metrics and set-level QC metrics can be mixed in a single mzQC file. The mzQC file itself does not impose a judgment on an individual run or set, but it may record the judgment of an external tool that is able to take the experimental context into account.

The mzQC format is intended to:

- Report quality metrics calculated by QC tools (such as Quameter (*6*), PTX-QC (*7*), and QCloud (*8*)) coming from MS-based experiments.
- Enable the presentation of quality reports to researchers for assessment of instrument performance.
- Record longitudinal QC metrics to monitor instrument performance over time.
- Perform quality control of collections of MS experiments across individual runs, batches of samples, biological or technical conditions, studies, or laboratories.
- Store and archive QC metrics next to their originating raw MS files and derived results in public data repositories or internal laboratory information management systems (LIMS).
- Explore and select 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 Non-Conforming Mass Spectrometry 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. 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 (*9*). Experiments that lie far away from the main group of experiments may then be interrogated as potential outliers to investigate technical (data issues and sources of performance degradation) or biological factors.

3.2 Longitudinal Monitoring of Instrument Performance

QC samples can range from pure samples to complex mixtures. Each of these types of samples can be employed in a specific fashion to analyze the system performance. Common QC metrics to assess instrument performance include the number of annotated spectra, the mass accuracy, or—in proteomics—the sequence coverage. As instrument performance gradually degrades over time, such metrics should be monitored carefully and preventative tuning and calibration 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. The QC metrics stored in the mzQC format for individual experiments reflect 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, and 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 specifically tailored to the users' preferences. For example, the report can be as simple as only reporting the quality of sample runs, it can include metrics derived from multiple runs in experimental groups, or it can even be combined with instrument performance 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, (*10*) and BoxCar acquisition (*11*), 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 (proteomics) or compound (metabolomics) 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 for Multiplexing Experiments

Isobaric tags are frequently used to perform relative quantitation in clinical samples, such as the CPTAC efforts in breast and ovarian cancer (*12*, *13*). The ratio a peptide represents between a pair of samples may be compromised if one of the labeling reagents is used with lower efficiency than the others. For example, if the third of four reagents labels with 20% lower efficiency, then all proteins are likely to appear "down" in that sample. In this case, a "run" would represent the LC-MS/MS experiment, even though it multiplexes multiple samples. A quality metric might characterize the sum across all peptides for each channel's reporter ion intensities to assess any channel-specific bias. Alternatively, one might define a quality metric to report the fraction of all PSMs that lack any peak for a given isobaric tag channel. The same general principles might govern the creation of quality metrics for a metabolic labeling experiment.

3.6 Quality Control of Metabolomics Experiments

Similar to proteomics, quality control samples in metabolomics can be used to provide a mechanism to judge the quality of the dataset or assay produced and to assess the analytical variance of individual MS runs 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 for a study, placed in certain locations or randomly throughout the sample acquisition sequence, depending on the experimental setup. Blank samples are used to assess the carryover and 'leakage' from the columns and to determine the limit of detection. 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 MS run. Relevant QC metrics to monitor include the number of missing values, the distribution of peak intensities, the trend and/or drift of aggregated intensity values, and coefficients of variation of detected *m*/*z* features. 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 Controlled Vocabularies

The mzQC format describes quality control information based on the definitions formalized in controlled vocabulary terms. The use of CV terms from different vocabularies as control mechanisms and metric definitions are explained in the following.

4.1 The Quality Control Controlled Vocabulary (QC CV)

The PSI-QC controlled vocabulary subsection (QC CV) of the PSI-MS CV (Section 4.2) defines quality metrics and related supporting values. Its main purpose is describing metrics related to mass spectrometry quality control, for which it builds on established terms and definitions from chemistry, physics, and biology ontologies. Another purpose is to provide the underlying definitions and specialized metrics for data structures usable within mzQC files. Initially included in the QC CV are a collection of published and basic metrics. Special care went into clearly defining the metrics and associated values to help interpretation, use, and visualization.

The QC CV is present in the `MS:4000000–MS:4999999` namespace of the PSI-MS CV. The CV is encoded as an OBO (Open Biological and Biomedical Ontologies) file that follows the OBO specification. The OBO file format is a biology-oriented language for building ontologies, similar to the Web Ontology Language (OWL). The OBO Foundry maintains a comprehensive index of ontologies related to the life sciences.

Each metric (and CV entry request) includes 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...).
- Synonyms: OPTIONAL alternative names which may be exact matches or related (see below).
- Value type: Is the metric value a single value, an n-tuple, a table, or a matrix? What is the type of the metric (int, float, datetime, etc.)?
- Unit: OPTIONAL unit of the value, specified using an existing CV term.
- 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.

A comment is OPTIONAL, but if present it should convey information which is non-trivial and relevant for usage, implementation, or interpretation. Examples of this would be that the value computation might result in division by zero or otherwise unobtainable in certain situations. (And counter-examples, comments that should NOT be included, would be repeating the name/definition, synonyms, or development comments.)

Some restrictions apply to how CV term information is represented. The name, definition, and comment SHOULD NOT contain escaped characters, such as \", or special characters, such as backticks (`). Single quotes SHOULD be used to quote words or sentences. For example:

```
def: "The basis for the metric calculation, such as 'single run' or 'single spectrum'." [PSI:QC]
```

Example CV term:

```
[Term]
id: MS:4000059
name: number of MS1 spectra
def: "The number of MS1 events in the run." [PSI:MS]
comment: An unusual low number may indicate incomplete sampling/scan rate of the MS instrument, low sample volume and/or failed injection of a sample.
synonym: "MS1-Count" EXACT [PMID:24494671]
is_a: MS:4000003 ! single value
relationship: has_metric_category MS:4000009 ! ID free metric
relationship: has_metric_category MS:4000012 ! single run based metric
relationship: has_metric_category MS:4000021 ! MS1 metric
relationship: has_value_type xsd:int ! The allowed value-type for this CV term
relationship: has_units UO:0000189 ! count unit
```

ID

```
id: MS:4000059
```

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

Name

```
name: number of MS1 spectra
```

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 `id` and `name` must be present as reference when a term is used in mzQC or related context (see schema).

Definition

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

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: An unusual low number may indicate incomplete sampling/scan rate of the MS instrument, low sample volume and/or failed injection of a sample.
```

The comment section MAY contain calculation and interpretation details, like whether smaller or larger values are desirable. It is also RECOMMENDED to give a short explanation about how the metric can be interpreted. 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: MS:4000003 ! single value
relationship: has_value_type xsd:int ! The allowed value-type for this CV term
relationship: has_units UO:0000189 ! count unit
```

This example shows an integer single value with a count as unit (`UO:0000189`).

```
is_a: MS:4000003 ! single value
relationship: has_value_type xsd:float ! The allowed value-type for this CV term
relationship: has_value_concept STATO:0000237 ! standard deviation
relationship: has_units UO:0000221 ! dalton
```

This example shows a floating point single value that represents the standard deviation (`STATO:0000237`) with the unit 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_units` is REQUIRED, type specification using `has_value_type` is RECOMMENDED.
- **n-tuple**: An ordered list/array of length ‘n’. Unit specification using `has_units` is REQUIRED, type specification using `has_value_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 ‘n’.
- **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 where key(s) represent the column term names/accessions and the value(s) are JSON arrays of uniform value type and length.
- **table column types**: Any CV term with a single value specification can be used as a table column, specified using the `has_column` or `has_optional_column` relationships. Additionally, non-metric value terms from other vocabularies can similarly be used as long as their unit is unambiguous. The term name will be used as the column’s header. In case the term’s unit is ambiguous it is RECOMMENDED to specify the unit in the table’s CV term definition. E.g.: `MS:1000894` (retention time) can be used, but since its definition has both second and minute unit relationships, it is recommended to clarify in the table definition whether seconds or minutes are expected.
- **matrix**: Unit specification using `has_units` is REQUIRED, type specification using `has_value_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.

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.

Metric Categorization

```
relationship: has_metric_category MS:4000009 ! ID free metric
relationship: has_metric_category MS:4000012 ! single run based metric
relationship: has_metric_category MS:4000021 ! MS1 metric
```

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 the metric is derived, to which element of the instrumental setup the metric pertains, etc.) can be specified as well. It is RECOMMENDED to align the categorization of novel metrics to existing terms to facilitate consumption of related metrics.

```
relationship: has_units UO:0000010 ! second
relationship: has_column: MS:1000041 ! charge state
```

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

```
relationship: has_value_concept UO:0000191 ! fraction
```

For full semantic integration, it is RECOMMENDED to specify the value concept for automatic processing and interpretation of the value.

Additional Information

```
synonym: "MS1-Count" EXACT [PMID:24494671]
```

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).

4.1.1 Requesting new CV Terms for QC Metric Definition

As recommended by the PSI CV guidelines (*14*), the QC CV is dynamically maintained via either the psidev-qc-dev@lists.sourceforge.net mailing list or (preferably) the MS CV working group's GitHub repository using the dedicated "QC term request" issue functionality. 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.

New CV terms should be requested via the PSI-MS CV GitHub issue tracker. Upon creating a new issue, the requester has to select the "New QC Term" option. This will produce a template that will guide the requester in providing the necessary information to request their new CV term, as detailed above. If additional information or clarifications beyond the initial request are needed, the mzQC and PSI-MS working groups will work with the requester to finalize their CV term request. When all the necessary information has been provided, a new CV term will be created based on the request and added to the PSI-MS CV.

4.2 The PSI Mass Spectrometry Controlled Vocabulary (MS CV)

The PSI-MS controlled vocabulary (*14*, *15*) 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 general mass spectrometry 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. Integration of QC specific terms into the PSI-MS CV is achieved by grouping QC related terms under one common QC term ancestor (`MS:4000000`).

Example:

```
[Term]
id: MS:1001844
name: MS1 feature area
def: "Area of MS1 feature." [PSI:PI]
is_a: MS:1002735 ! feature-level quantification datatype
relationship: has_value_type xsd:double ! The allowed value-type for this CV term
```

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

4.3 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 dependencies 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 with identical prefixes. 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 or OLS.

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.

5 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. 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 it in conjunction with other formats (for example, peak files in the Mascot Generic Format (MGF)).

5.1 Universal Spectrum Identifier

The Universal Spectrum Identifier (*16*) (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, USIs can be used to uniquely refer to spectra. In the most likely use case, when reporting quality metrics for multiple spectra simultaneously represented by a tabular QC metric, either an optional USI or nativeID column is RECOMMENDED to refer to individual spectra.

5.2 mzML

mzML (*1*) 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. Universal spectrum identifiers (USIs) are the preferred mechanism to refer to individual spectra in an mzML file. Alternatively, the native spectrum identifier format (`MS:1000767`; nativeID in the MS CV) can be used.

It is also possible to represent 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 (via a `<cvParam>` element of a `<run>`) to reflect a spectrum filtering strategy. It is RECOMMENDED that in case mzQC elements are used within an mzML file, the corresponding mzQC file is referenced in the `<sourceFile>` tag of the mzML and registered in the

`<processingMethod>` object. An example of referencing an mzQC file within an mzML file can be found in the mzQC repository (see the Appendix).

5.3 mzIdentML

mzIdentML (*3*) 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>`.

5.4 mzTab and mzTab-M

mzTab (*17*) 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 supports metabolomics use cases via the mzTab-M format (*18*). mzQC files can be referenced via the `external_study_uri` metadata field in mzTab. Additionally, it might be relevant to store simple QC metrics covered by a CV term from the QC CV (via the QC terms included in the PSI-MS CV) directly in an mzTab file. For example, the mass deviation could be reported for identification and quantification data.

5.5 MAGE-TAB Proteomics

The Proteomics Sample Metadata Project describes the MAGE-TAB standard to encode metadata that captures the relationship between samples and the data generated. MAGE-TAB Proteomics makes it possible to provide rich metadata information in a consistent and structured format, with the aim to provide a mechanism for including metadata when depositing datasets to public proteomics data repositories.

As raw peak files typically only contain a very limited amount of information about the experimental design, metadata in the MAGE-TAB Proteomics format can be highly valuable to provide details on the study design. This can inform the calculation of relevant QC metrics to assess experimental quality beyond factors that are apparent from the peak files. For example, information on the biological conditions can be used to verify that proper randomization of the samples was performed, or QC metrics can be separately calculated for technical replicates and biological replicates to assess the respective sources of variability.

6 Format Specification

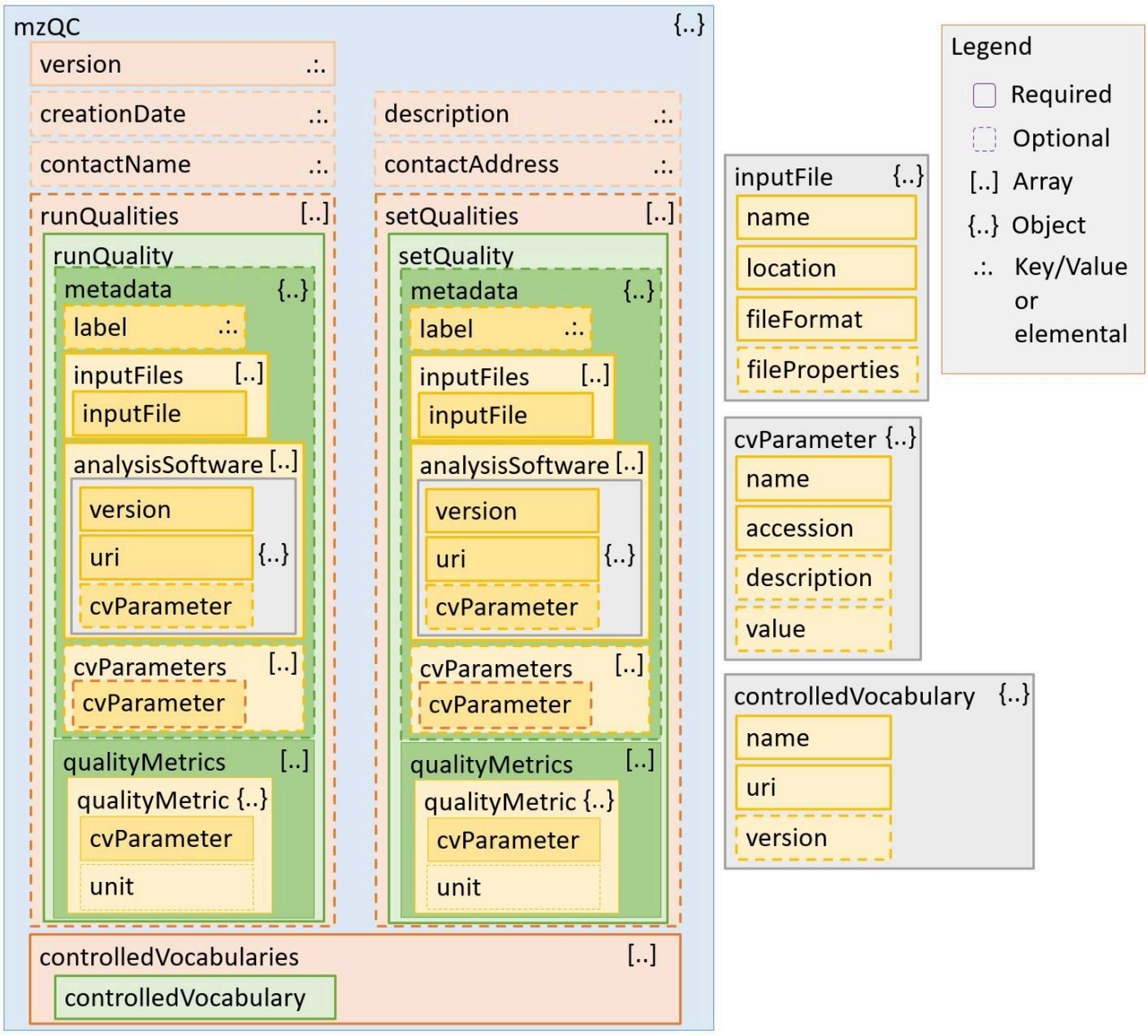

Figure 1: Diagrammatic overview of the mzQC schema.

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, the `controlledVocabularies` element contains `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. We use the notational convention to list the acceptable child elements 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.

6.1 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) .
`description`	string	false	Description and comments about the mzQC file contents.
`runQualities`	array of `runQuality`	true*	Description in section 6.3.
`setQualities`	array of `setQuality`	true*	Description in section 6.4.
`controlledVocabularies`	array of `controlledVocabulary`	true	Description in section 6.17.

*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 a json-schema compliant date-time string, as defined in RFC3339. Notably, RFC3339 is a subset of the ISO8601 format (e.g. `2019-10-29T14:40:17Z`), where the time-offset specifier at the end is mandatory (e.g. ”Z” for UTC or the specific time zone offset).

Example:

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

6.2 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 6.8.
`qualityMetrics`	array of `qualityMetric`	true	Description in section 6.15.

Example: see `runQuality` or `setQuality`.

6.3 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.

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

Example:

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

6.4 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.

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

Example:

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

6.5 runQuality

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

Type: `baseQuality`
Object definition:

Name	Type	Required	Description
`metadata`	object	true	Description in section 6.8.
`qualityMetrics`	array of `qualityMetric`	true	Description in section 6.15.

Example:

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

6.6 setQuality

Element containing `metadata` and `qualityMetrics` for a collection of related runs (“set”). It is REQUIRED that `setQuality` only contains `qualityMetrics` that describe properties that 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 6.8.
`qualityMetrics`	array of `qualityMetric`	true	Description in section 6.15.

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": [..]
    }
]
```

6.7 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:1003151",
      "name": "SHA-256",
      "value": "3522254348badf0723d5d2f406cff412ff45111ae31323a306b033147fc0cdcc"
```

}

6.8 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 of `inputFile`	true	Description in section 6.9.
`analysisSoftware`	array of `analysisSoftware`	true	Description in section 6.13.
`cvParameters`	array of `cvParameter`	false	Description in section 6.14.

`label` attributes MUST be unique within individual mzQC files. 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”), optionally including a random substring to ensure uniqueness (e.g. a random UUID).

Example:

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

6.9 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": [..]
```

6.10 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 location representing the complete file path, REQUIRED to be specified as a URI. The file URI is RECOMMENDED to be publicly accessible.
`fileFormat`	`cvParameter`	true	Description in section 6.11.
`fileProperties`	array of `cvParameter`	false	Description in section 6.12.

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": "2012-02-03T11:00:41Z"
        },
        {
```

```
            "accession": "MS:1003151",
            "name": "SHA-256",
            "value":
"3522254348badf0723d5d2f406cff412ff45111ae31323a306b033147fc0cdcc"
        }
    ]
}
```

6.11 fileFormat

Type of input file.

Type: `cvParameter`

Example:

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

6.12 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": "2012-02-03T11:00:41Z"
    },
        {
            "accession": "MS:1003151",
            "name": "SHA-256",
```

```
            "value":
"3522254348badf0723d5d2f406cff412ff45111ae31323a306b033147fc0cdcc"
        }
]
```

6.13 analysisSoftware

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

Type: array
Item types: extension of `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	Name of the software tool.
`version`	string	true	Version number of the software tool. In case no explicit software version is available, it is RECOMMENDED to use a git commit hash, software release data, or alternative specifiers.
`uri`	string (URI)	false	Publicly accessible URI of the software tool or documentation.

Example:

```
"analysisSoftware": [
    {
        "accession": "MS:1003164",
        "name": "QuaMeter IDFree",
        "version": "1.1.21070",
        "uri": "http://proteowizard.sourceforge.net/"
    }
]
```

6.14 cvParameters

OPTIONAL list of parameters 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": [..]
```

6.15 qualityMetrics

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

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

Example:

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

6.16 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 7).
`unit`	`cvParameter` / array of `cvParameter`	false	One or more controlled vocabulary elements describing the unit of the metric (if applicable — see section 7).

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. To facilitate independent consumption of an mzQC file without explicitly having to parse all listed CVs, it is REQUIRED to repeat the unit as defined in the CV. For table metrics, the `unit` MUST be an array with the order of the units matching the order of the table columns (see Section 7).

Example:

```
{
    "accession": "MS:4000059",
    "name": "Number of MS1 spectra",
    "description": "The number of MS1 events in the run.",
    "value": 7787,
    "unit": {
        "accession": "UO:0000189",
        "name": "count unit"
    }
}
```

6.17 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 or OLS.

6.18 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"
}
```

7 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 definition:

Name	CV Accession	Description	Example
`single value`	`MS:4000003`	A single string, number, or boolean value.	`"target"` or `1.0` or

			`true`
`n-tuple`	`MS:4000004`	An array of values of the same type (and unit).	`[2, 3, 5, 7, 11]`

`table`	`MS: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`	`MS: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, column]. All values in a matrix MUST have the same type.	`[[1, 2, 3],`
			` [4, 5, 6],`
			` [7, 8, 9]]`

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. In this case, the `unit` element MUST be specified as an array whose order matches the table columns. For a concrete example we refer to the companion document on CV term use (Appendix).

8 Validation of mzQC Files

Here we give a brief overview of syntactic and semantic validation requirements of mzQC files. Full validation is implemented in the pymzqc reference implementation. A living document with up-to-date validation information can be found on our website.

8.1 Syntactic Validation

With the help of the mzQC JSON schema, mzQC instances can be readily checked for syntactic schema compliance. There are a number of validators already implemented for use in various programming languages that support validation of JSON schema draft-07 from which the mzQC schema is designed.

8.2 Semantic Validation

Due to the advanced design of the mzQC JSON schema, many aspects of a valid mzQC file can already be verified via syntactic validation without the need for explicit semantic validation. There are, however, a few additional semantic rules that need to be followed to create fully compliant mzQC files.

- All used `controlledVocabulary` elements MUST reference a valid ontology.
- All used CV terms (including the quality metrics) must be present in one of the controlled vocabularies listed in the files `controlledVocabularies` element. CV terms MUST match by accession and SHOULD match by name if the versions of the controlled vocabularies are identical.
- `label` attributes in the `metadata` elements MUST be unique within its mzQC file.
- All `inputFile` elements MUST have a unique `location` attribute within their `runQuality` or `setQuality` element.
- All `qualityMetric` elements MUST be unique within their `runQuality` or `setQuality` parent.
- The value type of `qualityMetric` elements MUST match the specification in the CV (i.e. single value, n-tuple, matrix, or table).
- The value of `qualityMetric` MUST match its type and unit definition.
- All columns of table `qualityMetric` elements MUST have the same length.
- Units MUST be present in `qualityMetric` elements if they are specified in the corresponding CV terms.
- In case ID-based metrics are used, a corresponding `inputFile` element that includes the identification results MUST be present.
- For CV terms used in table column definitions, the unit SHOULD be unambiguous (i.e. specified using `has_units`).
- Multi-file metrics MUST refer to (the correct) existing `inputFile` elements.

Additionally, more detailed checks might be performed to assess the validity of specific metric values on a case-by-case basis (e.g., percentages should sum to 100%, etc.).

9 General Recommendations

9.1 Recommended File Extension

The RECOMMENDED extension for an mzQC file is ".mzqc".

9.2 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.

9.3 Encoding Non-Computable Numbers

Although the JSON format does not explicitly support encoding non-computable numbers, such as NaN or infinity, most parsing libraries unofficially support such values. To represent these values in mzQC files, we RECOMMEND to use their JavaScript representation: `NaN`, `Infinity`, and `-Infinity` (note the lack of quotation marks to avoid interpreting these values as strings). This encoding maximizes compatibility with JSON serialization and deserialization libraries in various programming languages.

9.4 Number of runQualities/setQualities 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). If automated processing of files is available in the lab, it is advisable to create one mzQC file per run (i.e. per raw file) shortly after acquisition to spot quality issues and enable fast remeasurement cycles. Once the whole batch/experiment is available, QC software can be instructed to create a new mzQC file containing metrics spanning the whole batch, i.e. compute setQuality metrics. The runQuality metrics may also be added to that file for a more comprehensive picture in downstream analysis. In general, libraries/software capable of reading/writing mzQC files should support merging of runQuality metrics of multiple mzQC files into a single mzQC file and vice versa.

9.5 Element Order in cvParameter Derived Objects

JSON does not enforce a specific order of elements within an object. However, for streaming of large mzQC files, it is RECOMMENDED that the `controlledVocabularies` element occurs before any `runQualities`/`setQualities` element(s). Additionally, for improved readability, we RECOMMEND that in `cvParameter` derived objects, if possible, “human-readable” elements, like `name` and `value`, are listed first.

9.6 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.

9.7 Development

For ongoing development, documentation, and to report issues with the format specification please see the working group's GitHub page (https://github.com/HUPO-PSI/mzQC/). For current software supporting mzQC, please see Section 13.

10 Pending Issues

10.1 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 visualized. 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.

10.2 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 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.

10.3 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.

10.4 Use of JSONPath

As with XPath for XML, JSONPath can be used for software agnostic instructions to query specific contents in JSON files. The use of JSONPath to process mzQC files comes with a few performance considerations. Since it is possible for mzQC files to include multiple `runQuality` elements in the `runQualities` list, and similarly multiple `setQuality` elements in the `setQualities` list, combinations of JSONPath queries might be necessary to retrieve specific information, leading to a higher computational load (e.g., to retrieve certain metrics above/below a threshold for specific runs or sets). Alternatively, if JSONPath efficiency is a major concern, mzQC files can be created such that they only contain a single `runQuality`/`runQuality` element.

11 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 artifacts are currently undergoing the PSI document process, which will result in a standard officially sanctioned by PSI.

12 Authors

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

Wout Bittremieux
University of Antwerp
Antwerp, Belgium
wout.bittremieux@uantwerpen.be

Nils Hoffmann
Forschungszentrum Jülich
Jülich, Germany
n.hoffmann@fz-juelich.de

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

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

Correspondence – Wout Bittremieux (wout.bittremieux@uantwerpen.be).

Additional Contributors

In addition to the authors, the following people contributed to format development, reviewed the specification document, or tested mzQC:

- Martin Eisenacher, Ruhr-University Bochum
- Juan Antonio Vizcaino, European Bioinformatics Institute (EMBL-EBI)

- Steffen Neumann, IPB Halle
- Tim Van Den Bossche, Ghent University
- Thomas Naake, European Molecular Biology Laboratory (EMBL)

We also want to thank Jinmeng Jia, Marina Pauw, Peter Crowther, Stefan Tenzer, Paul Brack, Weimin Zhu, Timo Sachsenberg, Eralp Dogu, and Axel Walter for contributing ideas and suggestions during the PSI-QC working group meetings.

13 Software

The following are the initial software implementations supporting the mzQC format.

Core libraries:

- pymzqc: Python library to process and validate mzQC files. It includes a Python object model, (de-)serialization functions, syntactic validation, and semantic validation of mzQC files.
- jmzqc: Java library to create, process, and validate mzQC files. It includes a Java object model, (de-)serialization functions, syntactic validation, and semantic validation of mzQC files.
- rmzqc: An R package for reading, validating, and writing mzQC files. It includes an R object model, (de-)serialization functions, and syntactic validation of mzQC files.

Metrics generating software:

- OpenMS: Open-source software C++ library for LC/MS data management and analyses.
- PTXQC: 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.
- Yamato / SwaMe / Prognosticator: SWATH-MS QC metrics generation tools.

An up-to-date list of software tools that support the mzQC format can be found on our website.

14 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 contact information on the PSI website).

15 Copyright Notice

16 References

1. L. Martens, M. Chambers, M. Sturm, D. Kessner, F. Levander, J. Shofstahl, W. H. Tang, A. Römpp, S. Neumann, A. D. Pizarro, L. Montecchi-Palazzi, N. Tasman, M. Coleman, F. Reisinger, P. Souda, H. Hermjakob, P.-A. Binz, E. W. Deutsch, *Mol. Cell. Proteomics*, in press, doi:10.1074/mcp.R110.000133.
2. A. R. Jones, M. Eisenacher, G. Mayer, O. Kohlbacher, J. Siepen, S. J. Hubbard, J. N. Selley, B. C. Searle, J. Shofstahl, S. L. Seymour, R. Julian, P.-A. Binz, E. W. Deutsch, H. Hermjakob, F. Reisinger, J. Griss, J. A. Vizcaíno, M. Chambers, A. Pizarro, D. Creasy, *Mol. Cell. Proteomics*, in press, doi:10.1074/mcp.M111.014381.
3. J. A. Vizcaíno, G. Mayer, S. Perkins, H. Barsnes, M. Vaudel, Y. Perez-Riverol, T. Ternent, J. Uszkoreit, M. Eisenacher, L. Fischer, J. Rappsilber, E. Netz, M. Walzer, O. Kohlbacher, A. Leitner, R. J. Chalkley, F. Ghali, S. Martínez-Bartolomé, E. W. Deutsch, A. R. Jones, The mzIdentML Data Standard Version 1.2, Supporting Advances in Proteome Informatics. *Mol. Cell. Proteomics*. **16**, 1275–1285 (2017).
4. M. Walzer, D. Qi, G. Mayer, J. Uszkoreit, M. Eisenacher, T. Sachsenberg, F. F. Gonzalez-Galarza, J. Fan, C. Bessant, E. W. Deutsch, F. Reisinger, J. A. Vizcaíno, J. A. Medina-Aunon, J. P. Albar, O. Kohlbacher, A. R. Jones, The mzQuantML data standard for mass spectrometry-based quantitative studies in proteomics. *Mol. Cell. Proteomics*. **12**, 2332–2340 (2013).
5. S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels* (RFC Editor, 1997).
6. Z.-Q. Ma, K. O. Polzin, S. Dasari, M. C. Chambers, B. Schilling, B. W. Gibson, B. Q. Tran, L. Vega-Montoto, D. C. Liebler, D. L. Tabb, QuaMeter: multivendor performance metrics for LC-MS/MS proteomics instrumentation. *Anal. Chem.* **84**, 5845–5850 (2012).
7. C. Bielow, G. Mastrobuoni, S. Kempa, Proteomics quality control: quality control software for maxquant results. *J. Proteome Res.* **15**, 777–787 (2016).
8. C. Chiva, R. Olivella, E. Borràs, G. Espadas, O. Pastor, A. Solé, E. Sabidó, QCloud: A cloud-based quality control system for mass spectrometry-based proteomics laboratories. *PLoS ONE*. **13**, e0189209 (2018).
9. X. Wang, M. C. Chambers, L. J. Vega-Montoto, D. M. Bunk, S. E. Stein, D. L. Tabb, QC metrics from CPTAC raw LC-MS/MS data interpreted through multivariate statistics. *Anal. Chem.* **86**, 2497–2509 (2014).
10. L. C. Gillet, P. Navarro, S. Tate, H. Röst, N. Selevsek, L. Reiter, R. Bonner, R. Aebersold, *Mol. Cell. Proteomics*, in press, doi:10.1074/mcp.O111.016717.
11. F. Meier, P. E. Geyer, S. Virreira Winter, J. Cox, M. Mann, BoxCar acquisition method enables single-shot proteomics at a depth of 10,000 proteins in 100 minutes. *Nat. Methods*. **15**, 440–448 (2018).

12. P. Mertins, D. R. Mani, K. V. Ruggles, M. A. Gillette, K. R. Clauser, P. Wang, X. Wang, J. W. Qiao, S. Cao, F. Petralia, E. Kawaler, F. Mundt, K. Krug, Z. Tu, J. T. Lei, M. L. Gatza, M. Wilkerson, C. M. Perou, V. Yellapantula, K. Huang, NCI CPTAC, Proteogenomics connects somatic mutations to signalling in breast cancer. *Nature*. **534**, 55–62 (2016).

13. H. Zhang, T. Liu, Z. Zhang, S. H. Payne, B. Zhang, J. E. McDermott, J.-Y. Zhou, V. A. Petyuk, L. Chen, D. Ray, S. Sun, F. Yang, L. Chen, J. Wang, P. Shah, S. W. Cha, P. Aiyetan, S. Woo, Y. Tian, M. A. Gritsenko, CPTAC Investigators, Integrated Proteogenomic Characterization of Human High-Grade Serous Ovarian Cancer. *Cell*. **166**, 755–765 (2016).

14. G. Mayer, L. Montecchi-Palazzi, D. Ovelleiro, A. R. Jones, P.-A. Binz, E. W. Deutsch, M. Chambers, M. Kallhardt, F. Levander, J. Shofstahl, S. Orchard, J. A. Vizcaíno, H. Hermjakob, C. Stephan, H. E. Meyer, M. Eisenacher, HUPO-PSI Group, The HUPO proteomics standards initiative- mass spectrometry controlled vocabulary. *Database (Oxford)*. **2013**, bat009 (2013).

15. G. Mayer, A. R. Jones, P.-A. Binz, E. W. Deutsch, S. Orchard, L. Montecchi-Palazzi, J. A. Vizcaíno, H. Hermjakob, D. Oveillero, R. Julian, C. Stephan, H. E. Meyer, M. Eisenacher, Controlled vocabularies and ontologies in proteomics: overview, principles and practice. *Biochim. Biophys. Acta*. **1844**, 98–107 (2014).

16. E. W. Deutsch, Y. Perez-Riverol, J. Carver, S. Kawano, L. Mendoza, T. Van Den Bossche, R. Gabriels, P.-A. Binz, B. Pullman, Z. Sun, J. Shofstahl, W. Bittremieux, T. Mak, J. Klein, Y. Zhu, H. Lam, J. A. Vizcaino, N. Bandeira, Universal Spectrum Identifier for mass spectra. *BioRxiv* (2020), doi:10.1101/2020.12.07.415539.

17. J. Griss, A. R. Jones, T. Sachsenberg, M. Walzer, L. Gatto, J. Hartler, G. G. Thallinger, R. M. Salek, C. Steinbeck, N. Neuhauser, J. Cox, S. Neumann, J. Fan, F. Reisinger, Q.-W. Xu, N. Del Toro, Y. Pérez-Riverol, F. Ghali, N. Bandeira, I. Xenarios, H. Hermjakob, The mzTab data exchange format: communicating mass-spectrometry-based proteomics and metabolomics experimental results to a wider audience. *Mol. Cell. Proteomics*. **13**, 2765–2775 (2014).

18. N. Hoffmann, J. Rein, T. Sachsenberg, J. Hartler, K. Haug, G. Mayer, O. Alka, S. Dayalan, J. T. M. Pearce, P. Rocca-Serra, D. Qi, M. Eisenacher, Y. Perez-Riverol, J. A. Vizcaíno, R. M. Salek, S. Neumann, A. R. Jones, mzTab-M: A Data Standard for Sharing Quantitative Results in Mass Spectrometry Metabolomics. *Anal. Chem.* **91**, 3302–3310 (2019).

17 Appendix

17.1 Examples

- individual run
- set of runs
- QC2 Sample
- metabolomics batch correction
- mzML with mzQC metrics
- USI use

Hint: You can find *worked* and *annotated versions* of these examples on our website.

17.2 Companion Documents

We have prepared a number of companion documents that explain mzQC from various use-case perspectives. You can find these here, but they are best viewed on our website.