

---

# Developer Guide

---

## Table of Contents

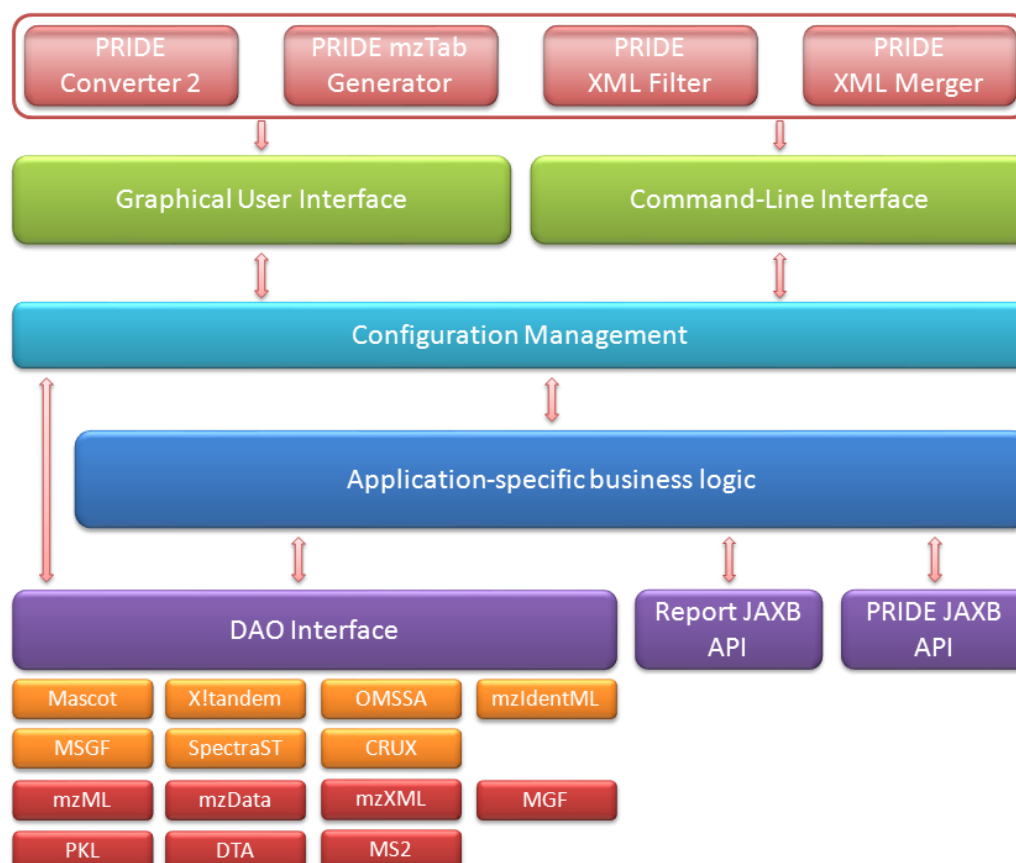
---

1. <i>PRIDE Converter 2</i> Technical Implementation	2
1.1. Technical Review	2
1.2. Source Code	3
2. How to write a <i>DAO</i> for <i>PRIDE Converter 2</i>	4
2.1. Conversion modes	4
2.2. The <i>dao-api</i> package	4
2.3. Writing a <i>DAO</i>	5
2.3.1. Maven project setup	5
2.3.2. Main <i>DAO</i> class	5
2.3.3. Adding required static method	6
2.4. The <i>DAO</i> Interface	6
2.4.1. Helper methods	7
2.4.2. Pre-scan mode methods	7
2.4.2. Conversion mode methods	10
2.4.3. Shared methods	10
3. Report File Manual Annotation Guidelines	12
3.1. <i>SearchResultIdentifier</i>	12
3.2. Metadata	12
3.2.1. Titles	12
3.2.2. Protocol	12
3.2.3. <i>MzDataDescription</i>	13
3.2.3.1 <i>cvLookup</i>	13
3.2.3.2. <i>admin</i>	13
3.2.3.3. <i>Instrument</i>	14
3.2.3.4. <i>DataProcessing</i>	15
3.2.4. <i>ExperimentalAdditional</i>	15
3.3. PTMs	15
3.4. <i>DatabaseMappings</i>	16
3.5. <i>ConfigurationOptions</i>	16
4. Generating <i>mzTab</i> files for <i>PRIDE Converter</i>	17

# 1. *PRIDE Converter 2* Technical Implementation

## 1.1. Technical Review

All the tools are written in Java and are released under the Apache 2 open source license. Figure 1 illustrates the overall design architecture.



**Figure 1.** The *PRIDE Converter 2* Technical Architecture.

At the heart of all of the applications there are 3 major modules: the PRIDE JAXB (Java Architecture for XML Binding) Application Programming Interface (API) [<http://code.google.com/p/pride-toolsuite/wiki/PRIDEXMLJAXB>], the Report file JAXB API and the Data Access Object (DAO), which is a common interface that is implemented for each supported source data format. Both the PRIDE JAXB API and the Report file JAXB API use the Java Architecture for XML Binding (JAXB, [<http://jaxb.java.net>]) to generate a java data model based on the relevant XML schema. Both also use XXIndex [<http://code.google.com/p/pride-toolsuite/wiki/XXIndex>], a library that allows very efficient retrieval of XML content from large files.

There are two major classes of data formats: those that contain spectra and identifications and those that only contain spectra. DAOs may optionally provide a set of configuration options. These options, along with any tool-specific settings that are required by the underlying business logic, are presented to the user interface in a consistent way. The GUIs are written in Java Swing and the CLIs use the Apache Commons Command-Line API.

## 1.2. Source Code

All source code is available online (<http://code.google.com/p/pride-converter-2/>) and is distributed under the terms of the Apache License, Version 2.0 (<http://www.apache.org/licenses/LICENSE-2.0.html>).

The source code SVN can be browsed online (<http://code.google.com/p/pride-converter-2/source/browse/#svn/trunk>) and contains the following modules:

Module Name	Description
converter-common	Common API for the PRIDE Converter 2
dao-api	DAO interface API
dao-crux-txt	Implementation for the CRUX DAO
dao-mascot	Implementation for the Mascot DAO
dao-msf-impl	Implementation for the Proteome Discoverer MSF DAO
dao-msgf-impl	Implementation for the MSGF DAO
dao-mzidentml	Implementation for the mzIdentML DAO
dao-mzml	Implementation for the mzML DAO
dao-mzxml	Implementation for the mzXML DAO
dao-omssa-txt	Implementation for the OMSSA DAO
dao-peaklist	Implementation for the peaklist DAOs (DTA/MS2/PKL)
dao-spectrast-xls	Implementation for the SpectraST DAO
dao-xtandem	Implementation for the X!Tandem DAO
pride-converter	Main PRIDE Converter GUI and logic
pride-mod	Automatic PTM assignment API
report-api	Report File XML JAXB API

## 2. How to write a *DAO* for PRIDE Converter 2

---

Data access objects (DAOs) in *PRIDE Converter 2* are the components used for accessing MS result files in different file formats. For every file format there must be one *DAO* supporting it. All *DAOs* implement the *DAO* interface and thereby unify the heterogeneous data coming from the various input formats and facilitating the integration of new data formats into the *PRIDE Converter 2*.

### 2.1. Conversion modes

---

*PRIDE Converter 2* works in two conversion modes: the "pre-scan" and the "convert" mode, where the first is mainly concerned about meta-data and annotation of the data and the second mode handles that actual data. Generally, the available input files do not contain all the meta-data required to create valid PRIDE XML files (and hence have to be extended by the converter).

The **pre-scan mode** is triggered first and used to collect all available meta-data from the input file to determine the level of existing annotation and the missing values that will have to be provided by the user. Most *DAO* functions are only called in *pre-scan mode*. The metadata is stored by the converter for the later conversion process.

The **conversion mode** is the mode where the actual conversion is being done. The actual conversion process uses mainly the report file (stored meta-data) generated from the *pre-scan* mode as input and only accesses the *DAO* to retrieve actual data or non-meta-data (such as spectra data and fragment ion annotations).

### 2.2. The dao-api package

---

The dao-api package contains the basic *DAO* interface as well as the *AbstractDAOImpl* class that need to be implemented / extended by every *DAO*. Furthermore, it contains several general purpose classes used by all *DAOs*.

A detailed description of the *DAO* interface can be found in the next sections below. First, a description of the general purpose classes found in the dao-api package.

#### DAOCvParams

This enum provides a list of *cvParams* used by all *DAOs*. *DAOs* should generally not contain any hard coded *cvParams* but only use the ones provided in the *DAOCvParams* enum. In case a required *cvParam* is not present in the enum it should be added by contacting a core developer of the PRIDE Converter project. The *DAOCvParams* enum provides several helper functions that automatically generate PRIDE JAXB *CvParam* objects as well as ReportFile *CvParam* objects.

#### QuantitationCvParams

To make the organization of *cvParams* a little easier, all quantitation related *cvParams* can be found in the *QuantitationCvParams* enum. The *cvParams* currently found here are temporary as they are all from the PRIDE ontology. This will be changed as soon as the required *cvParams* are added to the MS ontology.

## DAOProperty

This class is used by the *DAOs* to let the PRIDE Converter framework know which options they support. A detailed description of this class can be found at the description of the *GetSupportedProperties* function.

## Utils

The *Utils* class provides general purpose functions used in several *DAOs*. All functions of the *Utils* class are static. Detailed descriptions about the functions of the *Utils* class can be found in the corresponding JavaDoc.

## 2.3. Writing a DAO

### 2.3.1. Maven project setup

Every *DAO* is developed as a single maven project (<http://maven.apache.org/>). The project is generally called *dao-[fileformat]-impl* (for example *dao-mascot-impl* or *dao-mzidentml-impl*).

When setting up the project add the following dependencies and maven repositories to your *DAO* project's pom.xml:

```
<dependency>
  <groupId>uk.ac.ebi.pride.tools.converter</groupId>
  <artifactId>dao-api</artifactId>
  <version>1.0.1-SNAPSHOT</version>
</dependency>

<repository>
  <id>ebi-repo</id>
  <name>The EBI internal repository</name>
  <url>http://www.ebi.ac.uk/~maven/m2repo</url>
</repository>

<repository>
  <id>ebi-snapshot-repo</id>
  <name>The EBI internal snapshot repository</name>
  <url>http://www.ebi.ac.uk/~maven/m2repo_snapshots</url>
</repository>
```

### 2.3.2. Main DAO class

Every *DAO* consists of one main class that is used by the PRIDE Converter framework. This class is generally called *[fileformat]Dao* (for example *MascotDao*).

The *DAO* class **must** implement the *DAO* interface as well as extend the *AbstractDAOImpl* class.

```
public class YourDao extends AbstractDAOImpl implements DAO {
  // your functions...
}
```

Detailed descriptions about every function of the *DAO* Interface can be found in the next section.

### 2.3.3. Adding required static method

---

Every *DAO* must overwrite the *AbstractDAOImpl*'s `getSupportedProperties` function. This method returns a Collection of *DAOProperty* describing the various specific configuration options for the *DAO*. These options are automatically exposed through the command line interface as well as the GUI component.

One example of such an option is Mascot's minimum probability setting:

```
DAOProperty<Double> minProbability = new DAOProperty<Double>(  
    "min_probability", // the property's name - must only contain [A-Za-z0-9_]   
    0.05,              // the property's default value (if applicable)   
    0.0,               // the property's minimum value (if applicable)   
    1.0);              // the property's maximum value (if applicable)   
  
minProbability.setDescription(  
    "Specifies a cut-off point for protein scores, a cut-off for an Integrated error tolerant   
    search and a threshold for calculating MudPIT scores. This value represents a probability   
    threshold."  
);
```

Every *DAO* **must** provide at least one option for the user to set a minimum threshold for peptides to be reported unless the supported file format doesn't allow such a filter. The example above shows the minimum probability threshold used for Mascot results.

### 2.4. The *DAO* Interface

---

This is the main interface for format-specific parsing. Each implementation is responsible to support as much of desired functionality as possible. It is appreciated that not all formats will make the requested data items available. In such cases, the methods should return null primitives and empty collections.

Some methods are expected to return *Param* types. Wherever possible, *cvParam* objects should be used but it is acceptable to return *userParam* objects to capture information from the source files where the actual value of the parameter cannot be fully known by the *DAO* during the prescan. In effect, these *userParam* objects will act as placeholders to indicate that a proper annotation is required. It will be the responsibility of the user to inspect the report file generated and make certain that the information is correct and, if possible, convert the *userParam* data into the appropriate *cvParams*. In any case, the report formats will undergo a validation step where missing or incorrect information will be flagged to the user before the full parsing into PRIDE XML is executed.

The *DAO* must report all possible protein-to-peptide assignments. External tools will be available to update the report file based on specific protein inference algorithms.

The *DAO* interface contains optional and required functions. Optional functions may return NULL in case they are not supported by the search engines. Required functions must always return an object of the defined return type and in case the information is not available should return a sensible default value.

### 2.4.1. Helper methods

#### setConfiguration

This function is called by the PRIDE Converter framework to pass the *DAO* specific configuration setting made by the user to the *DAO*. The Properties names will be the same as set in the *DAOProperty* objects returned by *getSupportedProperties*.

#### getConfiguration

Return the current configuration as a Properties object.

### 2.4.2. Pre-scan mode methods

#### getExperimentTitle (required)

Must return some experiment title. In case no title is provided by the search engine's output file, a default title should be returned.

#### getExperimentShortLabel (optional)

The experiment short label should be used to identify the experiment internally.

#### getExperimentParams (required)

As a minimal requirement the date of search and the original MS data file format should be set. A common implementation of this function is:

```
// initialize the collection to hold the params
Param params = new Param();

Date searchDate = new Date(msec);
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd HH:mm");

params.getCvParam().add(DAOCvParams.DATE_OF_SEARCH.getParam(formatter.format(searchDate)));
params.getCvParam().add(DAOCvParams.ORIGINAL_MS_FORMAT.getParam("Mascot dat file"));

// the type of search performed can also be reported here
if (isPMF())
    params.getCvParam().add(DAOCvParams.PMF_SEARCH.getParam());
if (isMSMS())
    params.getCvParam().add(DAOCvParams.MS_MS_SEARCH.getParam());

// in this example the DAO supports the generation of FDR values
Double fdr = getFDR();
params.getCvParam().add(DAOCvParams.PEPTIDE_FDR.getParam(fdr.toString()));
```

#### getSampleName (optional)

A human readable name for the analyzed sample. This information will generally not be available in a search engine result file.

#### getSampleComment (optional)

A human readable description of the analyzed sample. This information will generally not be available in a search engine result file.

#### getSampleParams (required)

Parameters describing the analyzed sample. As a minimal requirement the sample's species should be returned.

#### getSourceFile (required)

Returns a *SourceFile* object representing the processing input file. A common implementation of this function is:

```
// initialize the return variable
SourceFile file = new SourceFile();
file.setPathToFile(sourcefile.getAbsolutePath());
file.setNameOfFile(sourcefile.getName());
file.setFileType("Mascot dat file");
return file;
```

#### getContacts (optional)

A Collection of contacts for the given experiment. This information will generally not be available in a search engine result file.

#### getInstrument (optional)

Returns parameters describing the instrument configuration used to perform the reported experiment. This information will generally not be available in a search engine result file.

#### getSoftware (required)

Should contain the search engine's name and version.

#### getProcessingMethod (optional)

Should describe processing methods used to generate the peak list. Furthermore, common search engine settings such as the used tolerance settings (parent and fragment mass tolerance) as well as possible thresholds and scoring methods should also be reported here.

#### getProtocol (optional)

Describes the experimental procedures performed during the whole experiment. This information will generally not be available in a search engine result file.

#### getReferences (optional)

A Collection of *Reference* objects describing publications presenting this experiment. This information will generally not be available in a search engine result file.

#### getSearchDatabaseName (required)



The names of the search database(s) used in the experiment. These will be written to the FASTA attributes and will be used in the FASTA section if there are multiple sequence files, the search database name will be a string-delimited concatenation of all the names. Idem for version.

This should represent the same data as in the *DatabaseMappings*, with a simplified format as single String (database names can be concatenated in cases where multiple DBs have been used). Also see *getDatabaseMappings*.

#### getSearchDatabaseVersion (required)

See *getSearchDatabaseName*.

#### getPTMs (required)

Should return a collection of PTMs representing all PTMs that are used in this search. The PTM object should at least contain the *SearchEnginePTMLabel*, whether they are fixed or variable modifications, the delta masses and the Residues.

The *SearchEnginePTMLabel* is used by the *DAO* to identify the given modification in this function and when reporting modifications for peptides. It is used by the converter framework to merge user annotated information about the modification with the one reported by the *DAO*. The *SearchEnginePTMLabel* will not be written to the final PRIDE XML file.

The Residues string specifies the amino acids as single-letter code on which the given modification was observed (f.e. "CM" for cysteine and methionine). The N-terminus should be reported as "0" and the C-terminus as "1".

#### getDatabaseMappings (required)

Should return a collection of *DatabaseMappings* that contain all search database names and versions used in this search. The *DatabaseMapping* objects should only contain the *SearchEngineDatabaseName* and *SearchEngineDatabaseVersion*. This information is the used for the user to curate the reported database name and version.

#### getSearchResultIdentifier (required)

Returns a *SearchResultIdentifier* object identifying the input file. A common implementation of this function is:

```
/**
 * sourcefile used in this example is a File object
 * representing the passed input file.
 */
// initialize the search result identifier
SearchResultIdentifier identifier = new SearchResultIdentifier();

// format the current time
SimpleDateFormat formatter = new SimpleDateFormat("yyyy-MM-dd'T'HH:mm:ss'Z'");

identifier.setSourceFilePath(sourcefile.getAbsolutePath());
identifier.setTimeCreated(formatter.format(new Date(System.currentTimeMillis())));
identifier.setHash(FileUtils.MD5Hash(sourcefile.getAbsolutePath()));

return identifier;
```

#### getCvLookup (required)

Must return a non-null list of all CV lookups used by the *DAO*. A common implementation using the PRIDE and the MS ontology looks as follows:

```
ArrayList<CV> cvs = new ArrayList<CV>();  
cvs.add(new CV("MS", "PSI Mass Spectrometry Ontology", "1.2",  
"http://psidev.cvs.sourceforge.net/viewvc/*checkout*/psidev/psi/psi-  
ms/mzML/controlledVocabulary/psi-ms.obo"));  
cvs.add(new CV("PRIDE", "PRIDE Controlled Vocabulary", "1.101", "http://ebi-  
pride.googlecode.com/svn/trunk/pride-core/schema/pride_cv.obo"));  
return cvs;
```

## 2.4.2. Conversion mode methods

### getSpectrumCount (required)

Must return a count of the number of spectra. If *onlyIdentified* is true, returns only count of identified spectra. If false, returns count of all spectra.

### getSpectrumIterator (required)

This iterator is the only method used by the PRIDE Converter framework to access the spectra in the input file. If *onlyIdentified* is set to true the returned iterator should only iterate over identified spectra.

### getSpectrumReferenceForPeptideUID (required)

Returns the spectrum reference for the given peptide. The peptide's unique identifier (*peptideUID*) is only used by the *DAO* to identify peptide objects.

### getIdentificationByUID (required)

Returns the *Identification* object identified by the passed unique identifier. The identification unique identifier (*identifierUID*) is only used by the *DAO* to keep track of identification object. This identifier is not written to the final PRIDE XML file.

## 2.4.3. Shared methods

### getIdentificationIterator

This method will return an iterator that will return individual identification objects.

In *prescan-mode* the complete Identification and Peptide objects should be returned without the peptide's fragment ion annotation. Peptide items have to contain all the PTMs.

In *conversion-mode* (= !prescanMode) Peptide and Protein objects should **NOT** contain any additional parameters and peptide PTMs should **NOT** be included. Furthermore, the different handlers should also **NOT** be invoked. Peptide Fragmentation annotations are mandatory (if applicable) in *scan mode*. The identification iterator may return null for an identification.

### setExternalSpectrumFile

Some search engines only store the identification data in the search result file. The spectra are then referenced in an additional MS data file.

Generally, a *DAO* should look for the referenced MS data file in the current working directory and the directory the search engine result is present. This function can furthermore pass a path to the MS data file and should overwrite the behaviour described before.

## 3. Report File Manual Annotation Guidelines

---

The report file is an intermediate file that is generated as part of the conversion process from the search engine result files into well-annotated and valid PRIDE XML files. The report file can either be filled-in using the *PRIDE Converter 2* GUI, or manually, using a XML editor or by writing dedicated software or scripts to supply the required information. Here are the relevant sections of the report file and the information that is required to perform a successful conversion. More information about individual fields can be found at the PRIDE submission guidelines page at <http://www.ebi.ac.uk/pride/submissionGuidelines.do> - [SubmissionForBiologistsSecondVersion-comprehensivemetadatatable](#).

### 3.1. SearchResultIdentifier

---

Automatically generated by the DAO based on the source file being converted, do not modify.

```
<SearchResultIdentifier>
  <hash>7279ea25f596fb5946f2076a9790e734</hash>
  <sourceFilePath>/home/rcote/ /19749_017.dat</sourceFilePath>
  <timeCreated>2012-06-14T16:46:10Z</timeCreated>
</SearchResultIdentifier>
```

### 3.2. Metadata

---

#### 3.2.1. Titles

---

‘Title’ (experiment title) and ‘short label’ are required fields in PRIDE. These are provided as free-text.

```
<Title></Title>
<ShortLabel></ShortLabel>
```

#### 3.2.2. Protocol

---

‘ProtocolName’ is required in PRIDE. This is free-text. ‘ProtocolSteps’ are optional, and should be populated with controlled vocabulary (CV) params\* as follows:

```
<Protocol>
  <ProtocolName>SDS-PAGE</ProtocolName>
  <ProtocolSteps>
    <StepDescription>
      <cvParam cvLabel="PRIDE" accession="PRIDE:0000039"
        name="Organelle isolation" value="Supernatant"/>
    </StepDescription>
    <StepDescription>
      <cvParam cvLabel="SEP" accession="sep:00173"
        name="sodium dodecyl sulfate polyacrylamide gel electrophoresis" value=""/>
      <cvParam cvLabel="SEP" accession="sep:00163" name="silver staining" value=""/>
      <cvParam cvLabel="SEP" accession="sep:00132" name="excised band" value=""/>
    </StepDescription>
    <StepDescription>
      <cvParam cvLabel="SEP" accession="sep:00142" name="enzyme digestion"
        value="Trypsin"/>
    </StepDescription>
    <StepDescription>
      <cvParam cvLabel="PRIDE" accession="PRIDE:0000027" name="Mass spectrometry"
        value="IonTrap XCT Ultra"/>
    </StepDescription>
  </ProtocolSteps>
</Protocol>
```

\* A lot of diversity in the details for experimental protocols can be introduced. Please, report those steps in the protocol you consider to be important. If unsure about which ontology to use, go to the OLS web page (<http://www.ebi.ac.uk/ontology-lookup/>) and search for the desired term using the 'Search in all ontologies' option.

### 3.2.3. MzDataDescription

#### 3.2.3.1 cvLookup

cvLookup tags will be automatically added by the DAOs for the CVs that they use internally. Should your pipeline use CVs that are not already listed, you will need to add 'cvLookup' items at this level.

```
<cvLookup cvLabel="MS" fullName="PSI Mass Spectrometry Ontology"
address=http://psidev.cvs.sourceforge.net/viewvc/\*checkout\*/psidev/psi/psi-
ms/mzML/controlledVocabulary/psi-ms.obo version="1.2" />
```

#### 3.2.3.2. admin

At minimum, the contact information of the primary investigator should be submitted and there is no limit on the number of contact elements that can be added to the report file. All fields of a contact element are mandatory and free-text. The 'contactInfo' should contain a valid e-mail address, at minimum.

```
<admin>
  <sampleName></sampleName>
  <sampleDescription comment="" />
  <sourceFile>
    <nameOfFile>19749_017.dat</nameOfFile>
    <pathToFile>/home/rcote/ 19749_017.dat</pathToFile>
    <fileType>Mascot dat file</fileType>
  </sourceFile>
  <contact>
    <name></name>
    <institution></institution>
    <contactInfo></contactInfo>
  </contact>
</admin>
```

The 'sampleName' is a required field in PRIDE and is free-text. 'sampleDescription' should contain, at minimum, NEWT taxonomy terms annotations for reporting the species name. For the sampleDescription field, if possible, tissue (using the BRENDA Tissue ontology, BTO, accessible at <http://obo.cvs.sourceforge.net/obo/obo/ontology/anatomy/BrendaTissue.obo>) and cell type (using the Cell Type ontology, CL, accessible at [http://obo.cvs.sourceforge.net/obo/obo/ontology/anatomy/cell\\_type/cell.obo](http://obo.cvs.sourceforge.net/obo/obo/ontology/anatomy/cell_type/cell.obo)) annotations should also be provided as 'cvParams'. Other CVs/ ontologies that can be used are:

- **GO term** (sub-cellular localization). Use the "Gene Ontology" (GO).

- **Disease**. Use the "Human Disease" ontology (DOID).

```
<sampleDescription comment="[optional]">
  <cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (HUMAN)" value=" " />
  <cvParam cvLabel="CL" accession="CL:0000738" name="leukocyte" value=" " />
  <cvParam cvLabel="DOID" accession="DOID:1240" name="leukemia" value=" " />
</sampleDescription>
```

In cases where mzTab files (<http://code.google.com/p/mztab/>) have been used to supply quantitation information, the 'sampleDescription' element will be populated with the quantitation label descriptions. Note that the value of the "Subsample X description" params is free text, while

the value of the subsample reagent params is set by the *DAO* and **must not** be modified as it will cause exceptions during the conversion process.

```
<sampleDescription comment="">
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000366" name="Contains multiple subsamples"
    value=""/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000367" name="Subsample 1 description"
    value="free-text-1"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000285" name="TMT reagent 126"
    value="subsample1"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000368" name="Subsample 2 description"
    value="free-text-2"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000286" name="TMT reagent 127"
    value="subsample2"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000369" name="Subsample 3 description"
    value="free-text-3"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000287" name="TMT reagent 128"
    value="subsample3"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000370" name="Subsample 4 description"
    value="free-text-4"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000288" name="TMT reagent 129"
    value="subsample4"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000371" name="Subsample 5 description"
    value="free-text-5"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000289" name="TMT reagent 130"
    value="subsample5"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000372" name="Subsample 6 description"
    value="free-text-6"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000290" name="TMT reagent 131"
    value="subsample6"/>
  <userParam name="Available protein quantitation fields"
    value="PRIDE:0000354,PRIDE:0000355,PRIDE:0000356,PRIDE:0000357,PRIDE:0000358,PRIDE:0000359"/>
  <userParam name="Available peptide quantitation fields"
    value="PRIDE:0000354,PRIDE:0000355,PRIDE:0000356,PRIDE:0000357,PRIDE:0000358,PRIDE:0000359"/>
</sampleDescription>
```

In cases where quantitation information is submitted and multiple subsamples are present, if you wish to add NEWT, CL, BTO or other annotations, each subsample **must be** described, using the proper subsample identifier. See example below:

```
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample1"/>
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample2"/>
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample3"/>
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample4"/>
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample5"/>
<cvParam cvLabel="NEWT" accession="9606" name="Homo sapiens (Human)" value="subsample6"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample1"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample2"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample3"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample4"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample5"/>
<cvParam cvLabel="DOID" accession="DOID:1240" name="leukaemia" value="subsample6"/>
```

### 3.2.3.3. Instrument

An 'instrumentName' is required in PRIDE and is provided as free-text. The description of the instrumentation setup, however, must use controlled vocabulary terms. We strongly recommend providing as much information as possible regarding the instrumentation. Source, analyzers and detectors are enumerations of 'cvParams'. An instrument must have at least one of each source, analyser and detector, but multiple analyzers can be annotated. It is recommended to use the Proteomics Standards Initiative (PSI)-MS CV to report this information. It is accessible at <http://psidev.cvs.sourceforge.net/viewvc/psidev/psi/psi-ms/mzML/controlledVocabulary/psi-ms.obo>

```

<instrument>
  <instrumentName>LTQ</instrumentName>
  <source>
    <cvParam cvLabel="MS" accession="MS:1000073" name="electrospray ionization"
      value=""/>
    <cvParam cvLabel="MS" accession="MS:1000398" name="nanoelectrospray" value=""/>
  </source>
  <analyzerList count="1">
    <analyzer>
      <cvParam cvLabel="MS" accession="MS:1000083" name="radial ejection linear ion
        trap" value=""/>
    </analyzer>
  </analyzerList>
  <detector>
    <cvParam cvLabel="MS" accession="MS:1000253" name="electron multiplier" value=""/>
  </detector>
  <additional/>
</instrument>

```

### 3.2.3.4. DataProcessing

The *DAO* will try and populate as much information as possible for the ‘dataProcessing’ element. All fields of the software element are required in PRIDE. Software name and version can be provided as free text. For the processing method, a enumeration of ‘cvParams’ can be used.

```

<dataProcessing>
  <software>
    <name>Matrix Science Mascot</name>
    <version>2.2.04</version>
  </software>
  <processingMethod>
    <cvParam cvLabel="PRIDE" accession="PRIDE:0000161" name="Fragment mass
      tolerance setting" value="0.4 Da"/>
    <cvParam cvLabel="PRIDE" accession="PRIDE:0000078" name="Peptide mass tolerance
      setting" value="0.4 Da"/>
    <cvParam cvLabel="PRIDE" accession="PRIDE:0000162" name="Allowed missed cleavages"
      value="3"/>
    <cvParam cvLabel="MS" accession="MS:1001316" name="Mascot:SigThreshold"
      value="0.05"/>
    <cvParam cvLabel="MS" accession="MS:1001758" name="Mascot:SigThresholdType"
      value="identity"/>
  </processingMethod>
</dataProcessing>

```

### 3.2.4. ExperimentalAdditional

The *DAO* will automatically add information about the source files and conversion software used, but anything that relates to the experiment as a whole can be added. If you use software to programmatically edit the report file, you should annotate it here.

```

<ExperimentAdditional>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000218" name="Original MS data file
    format" value="Mascot dat file"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000219" name="Date of search" value="2009-08-
    08 23:43"/>
  <cvParam cvLabel="PRIDE" accession="PRIDE:0000175" name="XML generation software"
    value="PRIDE Converter Toolsuite 2.0-SNAPSHOT-20120622"/>
  <cvParam cvLabel="MS" accession="MS:1001083" name="ms/ms search" value=""/>
</ExperimentAdditional>

```

## 3.3. PTMs

Each post-translational modification (PTM) that is reported by the *DAO* will have a PTM entry. *PRIDE Converter 2* will try and automatically map the PTMs to terms to the PSI-MOD ontology (accessible at <http://psidev.cvs.sourceforge.net/psidev/psi/mod/data/PSI-MOD.obo>). **DO NOT** edit the ‘searchEnginePTMLabel’ or ‘ModMonoDelta’ elements as this will cause exceptions during the conversion process. Compare the two PTM elements below. The top one has not been automatically mapped; the bottom one has.

You must ensure that all PTM elements contain the following tags: 'ModAccession', 'ModName', 'ModDatabase', 'ModDatabaseVersion', and 'ModMonoDelta'. This information must be correct and be mapped to entries from PSI-MOD.

```
<PTM>
  <SearchEnginePTMLabel>Phospho (S)</SearchEnginePTMLabel>
  <FixedModification>false</FixedModification>
  <ModMonoDelta>79.966324</ModMonoDelta>
  <Residues>S</Residues>
  <additional/>
</PTM>
```

```
<PTM>
  <SearchEnginePTMLabel>Oxidation (M)</SearchEnginePTMLabel>
  <ModAccession>MOD:00425</ModAccession>
  <ModName>monohydroxylated residue</ModName>
  <ModDatabase>MOD</ModDatabase>
  <ModDatabaseVersion>1.010.7</ModDatabaseVersion>
  <FixedModification>false</FixedModification>
  <ModMonoDelta>15.994919</ModMonoDelta>
  <Residues>M</Residues>
  <additional>
    <cvParam cvLabel="MOD" accession="MOD:00425" name="monohydroxylated residue"/>
  </additional>
</PTM>
```

### 3.4. DatabaseMappings

---

Database mappings are used to describe the protein sequence database used by the search engine to perform protein assignments. The 'SearchEngineDatabaseName' and 'SearchEngineDatabaseVersion' will be reported by the *DAO*, where possible. **DO NOT** edit the 'SearchEngineDatabaseName' entries as this will cause exceptions during the conversion process.

Each 'DatabaseMapping' entry must have a 'CuratedDatabaseName' and 'CuratedDatabaseVersion'. The curated database name should, ideally, be of the following values: IPI, UniProtKB, UniProtKB/Swissprot, UniProtKB/TrEMBL, Ensembl, NCBI nr database or RefSeq. The value is, however, free-text and custom information can be entered. The curated database version should either be the release version that was downloaded to generate the search database or the date the search database was build.

```
<DatabaseMapping>
  <SearchEngineDatabaseName>IPI_HUMAN</SearchEngineDatabaseName>
  <SearchEngineDatabaseVersion>3.54</SearchEngineDatabaseVersion>
  <CuratedDatabaseName>IPI</CuratedDatabaseName>
  <CuratedDatabaseVersion>2012-01-02</CuratedDatabaseVersion>
</DatabaseMapping>
```

### 3.5. ConfigurationOptions

---

Automatically generated by the *DAO* based on the options supplied (or taken by default) to generate the report, **do not** modify.

```
<ConfigurationOptions>
  <option>
    <key>homology_threshold</key>
    <value>false</value>
  </option>
  <option>
    <key>enable_protein_grouping</key>
    <value>true</value>
  </option>
</ConfigurationOptions>
```



## 4. Generating mzTab files for PRIDE Converter

---

In PRIDE Converter 2 it is possible to supply gel-based and quantitative data using mzTab files. In case the reader is not familiar with the mzTab file format, the “10-minute guide to mzTab” is a good start to get familiar with the format (<http://mztab.googlecode.com>).

When using mzTab files to provide additional information to PRIDE Converter 2 it is imperative that the mzTab file contains the exact same protein and peptide identifications that are reported by the DAOs. It is therefore recommended that you use the PRIDE mzTab Generator to create skeleton mzTab files from your search results. These will then correspond to the data expected by PRIDE Converter 2 during the conversion process.

Gel-based information can be provided using optional columns in the mzTab file’s protein section. The following optional columns are mapped to the corresponding cvParams in the generated PRIDE XML file:

opt_gel_spotidentifier*	The spot identifier the protein was identified in. This identifier should be identical to the one used in the paper / gel image.
opt_gel_identifier*	The gel identifier of the gel. This identifier should resemble the one used in f.e. the publication.
opt_gel_url	The gel's URL to use as a link.
opt_ycoord_pixel	Y coordinates in pixel of the spot in the gel
opt_xcoord_pixel	X coordinates in pixel of the spot in the gel
opt_mw	The protein's (observed) molecular weight.
opt_pi	The protein's (observed) pI.

\* Required fields for PRIDE Converter 2 to report gel-based information in the generated PRIDE XML files.

The reporting of quantitative data is natively supported by the mzTab format. PRIDE Converter 2 expects any quantitative data to follow the mzTab standard and will automatically integrated any data quantitative data provided in this format in the generated PRIDE XML file. Extensive documentation on how to correctly report quantitative data in mzTab files can be found in the mzTab file format specification at <http://mztab.googlecode.com>.

It is also possible to automatically add placeholders for quantitative values in the generated skeleton mzTab file produced by the PRIDE mzTab Generator. This option can be enabled by setting the option “generate quant fields” during the mzTab file generation. The value for this option should correspond to the number of labels used during the quantitation process (f.e. “4” for a 4-plex iTRAQ).