Bio-Formats

Release 6.14.0

The Open Microscopy Environment

Jul 05, 2023
## CONTENTS

1 About Bio-Formats .................................................. 3
   1.1 License, copyright, and citation ............................ 3
   1.2 Help .......................................................... 4
   1.3 Bio-Formats versions ......................................... 4
   1.4 Why Java? .................................................... 5
   1.5 Bio-Formats metadata processing ............................ 5

2 User Information .................................................... 7
   2.1 Using Bio-Formats with ImageJ and Fiji ..................... 7
   2.2 Command line tools .......................................... 20
   2.3 OMERO ....................................................... 33
   2.4 Image server applications .................................. 34
   2.5 Libraries and scripting applications ...................... 36
   2.6 Numerical data processing applications .................. 37
   2.7 Visualization and analysis applications .................. 41

3 Developer Documentation ........................................ 47
   3.1 Introduction to Bio-Formats ............................... 47
   3.2 Using Bio-Formats as a Java library ....................... 56
   3.3 Using Bio-Formats as a native C++ library ............... 82
   3.4 Contributing to Bio-Formats ................................ 82

4 Formats ...................................................................... 99
   4.1 Dataset Structure Table ..................................... 99
   4.2 Supported Formats ............................................ 103
   4.3 Summary of supported metadata fields ................... 219
   4.4 Grouping files using a pattern file ....................... 466
   4.5 Additional reader and writer options ..................... 467

5 Version history ................................................... 469
   5.1 6.14.0 (2023 July) ........................................... 469
   5.2 6.13.0 (2023 April) .......................................... 470
   5.3 6.12.0 (2023 February) ..................................... 472
   5.4 6.11.1 (2022 December) .................................... 473
   5.5 6.11.0 (2022 October) ...................................... 474
   5.6 6.10.1 (2022 August) ....................................... 475
   5.7 6.10.0 (2022 May) ........................................... 477
   5.8 6.9.1 (2022 January) ........................................ 477
   5.9 6.9.0 (2022 February) ...................................... 478
   5.10 6.8.1 (2022 January) ........................................ 478
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.65</td>
<td>5.0.7 (2015 February 5)</td>
<td>531</td>
</tr>
<tr>
<td>5.66</td>
<td>5.0.6 (2014 November 11)</td>
<td>531</td>
</tr>
<tr>
<td>5.67</td>
<td>5.0.5 (2014 September 23)</td>
<td>531</td>
</tr>
<tr>
<td>5.68</td>
<td>5.0.4 (2014 September 3)</td>
<td>531</td>
</tr>
<tr>
<td>5.69</td>
<td>5.0.3 (2014 August 7)</td>
<td>531</td>
</tr>
<tr>
<td>5.70</td>
<td>5.0.2 (2014 May 28)</td>
<td>532</td>
</tr>
<tr>
<td>5.71</td>
<td>5.0.1 (2014 Apr 7)</td>
<td>532</td>
</tr>
<tr>
<td>5.72</td>
<td>5.0.0 (2014 Feb 25)</td>
<td>533</td>
</tr>
<tr>
<td>5.73</td>
<td>5.0.0-RC1 (2013 Dec 19)</td>
<td>533</td>
</tr>
<tr>
<td>5.74</td>
<td>5.0.0-beta1 (2013 June 20)</td>
<td>533</td>
</tr>
<tr>
<td>5.75</td>
<td>4.4.10 (2014 Jan 15)</td>
<td>534</td>
</tr>
<tr>
<td>5.76</td>
<td>4.4.9 (2013 Oct 16)</td>
<td>534</td>
</tr>
<tr>
<td>5.77</td>
<td>4.4.8 (2013 May 2)</td>
<td>534</td>
</tr>
<tr>
<td>5.78</td>
<td>4.4.7 (2013 April 25)</td>
<td>534</td>
</tr>
<tr>
<td>5.79</td>
<td>4.4.6 (2013 February 11)</td>
<td>534</td>
</tr>
<tr>
<td>5.80</td>
<td>4.4.5 (2012 November 13)</td>
<td>535</td>
</tr>
<tr>
<td>5.81</td>
<td>4.4.4 (2012 September 24)</td>
<td>535</td>
</tr>
<tr>
<td>5.82</td>
<td>4.4.2 (2012 August 22)</td>
<td>535</td>
</tr>
<tr>
<td>5.83</td>
<td>4.4.1 (2012 July 20)</td>
<td>535</td>
</tr>
<tr>
<td>5.84</td>
<td>4.4.0 (2012 July 13)</td>
<td>535</td>
</tr>
<tr>
<td>5.85</td>
<td>4.3.3 (2011 October 18)</td>
<td>535</td>
</tr>
<tr>
<td>5.86</td>
<td>4.3.2 (2011 September 15)</td>
<td>536</td>
</tr>
<tr>
<td>5.87</td>
<td>4.3.1 (2011 July 8)</td>
<td>536</td>
</tr>
<tr>
<td>5.88</td>
<td>4.3.0 (2011 June 14)</td>
<td>536</td>
</tr>
<tr>
<td>5.89</td>
<td>4.2.2 (2010 December 6)</td>
<td>537</td>
</tr>
<tr>
<td>5.90</td>
<td>4.2.1 (2010 November 12)</td>
<td>537</td>
</tr>
<tr>
<td>5.91</td>
<td>4.2.0 (2010 July 9)</td>
<td>538</td>
</tr>
<tr>
<td>5.92</td>
<td>4.1.1 (2009 December 3)</td>
<td>538</td>
</tr>
<tr>
<td>5.93</td>
<td>4.0.1 (2009 June 1)</td>
<td>539</td>
</tr>
<tr>
<td>5.94</td>
<td>4.0.0 (2009 March 3)</td>
<td>539</td>
</tr>
<tr>
<td>5.95</td>
<td>2008 December 29</td>
<td>539</td>
</tr>
<tr>
<td>5.96</td>
<td>2008 August 30</td>
<td>540</td>
</tr>
<tr>
<td>5.97</td>
<td>2008 July 1</td>
<td>540</td>
</tr>
<tr>
<td>5.98</td>
<td>2008 April 17</td>
<td>541</td>
</tr>
<tr>
<td>5.99</td>
<td>2008 Feb 12</td>
<td>541</td>
</tr>
<tr>
<td>5.100</td>
<td>2008 Feb 8</td>
<td>541</td>
</tr>
<tr>
<td>5.101</td>
<td>2008 Feb 1</td>
<td>542</td>
</tr>
<tr>
<td>5.102</td>
<td>2007 Dec 28</td>
<td>542</td>
</tr>
<tr>
<td>5.103</td>
<td>2007 Dec 1</td>
<td>543</td>
</tr>
<tr>
<td>5.104</td>
<td>2007 Nov 21</td>
<td>543</td>
</tr>
<tr>
<td>5.105</td>
<td>2007 Oct 17</td>
<td>543</td>
</tr>
<tr>
<td>5.106</td>
<td>2007 Oct 1</td>
<td>544</td>
</tr>
<tr>
<td>5.107</td>
<td>2007 Sept 11</td>
<td>544</td>
</tr>
<tr>
<td>5.108</td>
<td>2007 Aug 1</td>
<td>544</td>
</tr>
<tr>
<td>5.109</td>
<td>2007 July 16</td>
<td>545</td>
</tr>
<tr>
<td>5.110</td>
<td>2007 July 2</td>
<td>545</td>
</tr>
<tr>
<td>5.111</td>
<td>2007 June 18</td>
<td>545</td>
</tr>
<tr>
<td>5.112</td>
<td>2007 June 6</td>
<td>545</td>
</tr>
<tr>
<td>5.113</td>
<td>2007 May 24</td>
<td>545</td>
</tr>
<tr>
<td>5.114</td>
<td>2007 May 2</td>
<td>546</td>
</tr>
<tr>
<td>5.115</td>
<td>2007 Mar 16</td>
<td>546</td>
</tr>
<tr>
<td>5.116</td>
<td>2007 Mar 7</td>
<td>546</td>
</tr>
<tr>
<td>5.117</td>
<td>2007 Feb 28</td>
<td>547</td>
</tr>
<tr>
<td>5.118</td>
<td>2007 Feb 26</td>
<td>547</td>
</tr>
</tbody>
</table>
The following documentation is split into four parts. About Bio-Formats explains the goal of the software, discusses how it processes metadata, and provides other useful information such as version history and how to report bugs. User Information focuses on how to use Bio-Formats as a plugin for ImageJ and Fiji, and also gives details of other software packages which can use Bio-Formats to read and write microscopy formats. Developer Documentation covers more in-depth information on using Bio-Formats as a Java library and how to interface from non-Java codes. Finally, Formats is a guide to all the file formats currently supported by Bio-Formats.

- About Bio-Formats
- User Information
- Developer Documentation
- Formats
- Version history
- Reporting a bug

Bio-Formats 6.14.0 requires Java 8 or above and uses the June 2016 release of the OME Model.

**Bio-Formats is a community project and we welcome your input.** You can find guidance on Reporting a bug and contact us via the forums. Further information about how the OME team works and how you can contribute to our projects is in the Contributing Developer Documentation.
Bio-Formats is a standalone Java library for reading and writing life sciences image file formats. It is capable of parsing both pixels and metadata for a large number of formats, as well as writing to several formats.

The primary goal of Bio-Formats is to facilitate the exchange of microscopy data between different software packages and organizations. It achieves this by converting proprietary microscopy data into an open standard called the OME data model, particularly into the OME-TIFF file format.

We believe the standardization of microscopy metadata to a common structure is of vital importance to the community. You may find LOCI's article on open source software in science of interest.

1.1 License, copyright, and citation

Licensing information for Bio-Formats is described on the OME licensing page. Unless otherwise noted, the current copyright statement is:

Copyright (C) 2005 - 2018 Open Microscopy Environment:

- Board of Regents of the University of Wisconsin-Madison
- Glencoe Software, Inc.
- University of Dundee

Bio-Formats can be cited in publications as follows:


PMID:20513764

Additional citation information can be found on the main OME citation page.
1.2 Help

There is a guide for reporting bugs here.

For help relating to opening images in ImageJ or FIJI or when using the command line tools, refer to the users documentation. You can also find tips on common issues with specific formats on the pages linked from the supported formats table.

Please contact us if you have any questions or problems with Bio-Formats not addressed by referring to the documentation.

Other places where questions are commonly asked and/or bugs are reported include:
- OME Forums
- OME Trac
- ome-devel mailing list (searchable using google with 'site:lists.openmicroscopy.org.uk')
- ome-users mailing list (searchable using google with 'site:lists.openmicroscopy.org.uk')
- ImageJ mailing list
- Fiji GitHub Issues
- Confocal microscopy mailing list

1.3 Bio-Formats versions

Since Bio-Formats 5.1.3, Bio-Formats is decoupled from OMERO with its own release schedule rather than being updated whenever a new version of OMERO is released. This change allows for more frequent releases to get fixes out to the community faster. See the version history for a list of changes in each release.

1.3.1 Versioning policy

The following set of rules describe the current versioning policy using RFC 2119.

The Bio-Formats API follows strict semantic versioning since Bio-Formats 5.4.0 i.e.:

- The version number MUST take the form X.Y.Z where X, Y, and Z are non-negative integers, and MUST NOT contain leading zeroes. X is the major version, Y is the minor version and Z is the patch version.
- The patch version Z MUST be incremented if only backwards-compatible bug fixes are introduced. A bug fix is defined as an internal change that fixes incorrect behavior.
- The minor version Y MUST be incremented if new, backwards-compatible functionality is introduced to the public API.
- The major version X MUST be incremented when backwards-incompatible changes are introduced to the public API.
- Either the minor version or the major version MUST be incremented if the version of a non-OME/external dependency is updated.

Serialization functionality was implemented as a ReaderWrapper called Memoizer in Bio-Formats 5.0.0 and is exposed to the community via a public API. Since Bio-Format 5.4.0,:

- The minor version Y MUST be incremented if changes are introduced that are not backwards compatible with regard to serialization.
• Serialized memo files written with a previous minor version MAY not be readable by later versions and MAY need to be rewritten.
• Consumers with code relying on Bio-Formats caching stability SHOULD not upgrade the minor version of Bio-Formats version for now.

For format reader fixes and additions, the policy should read as follows:
• The minor version Y MUST be incremented when a new file format reader is introduced.
• The minor version Y MUST be incremented when a non backwards-compatible format bug fix is introduced, e.g. a fix that modifies the core metadata of existing files.
• The patch version Z MUST be incremented if only backwards-compatible format bug fixes are introduced.

See this GitHub issue for further details.

1.4 Why Java?

From a practical perspective, Bio-Formats is written in Java because it is cross-platform and widely used, with a vast array of libraries for handling common programming tasks. Java is one of the easiest languages from which to deploy cross-platform software. In contrast to C++, which has a large number of complex platform issues to consider, and Python, which leans heavily on C and C++ for many of its components (e.g., NumPy and SciPy), Java code is compiled one time into platform-independent byte code, which can be deployed as is to all supported platforms. And despite this enormous flexibility, Java manages to provide time performance nearly equal to C++, often better in the case of I/O operations (see further discussion on the comparative speed of Java on the LOCI site).

There are also historical reasons associated with the fact that the project grew out of work on the VisAD Java component library. You can read more about the origins of Bio-Formats on the LOCI Bio-Formats homepage.

1.5 Bio-Formats metadata processing

Pixels in microscopy are almost always very straightforward, stored on evenly spaced rectangular grids. It is the metadata (details about the acquisition, experiment, user, and other information) that can be complex. Using the OME data model enables applications to support a single metadata format, rather than the multitude of proprietary formats available today.

Every file format has a distinct set of metadata, stored differently. Bio-Formats processes and converts each format’s metadata structures into a standard form called the OME data model, according to the OME-XML specification. We have defined an open exchange format called OME-TIFF that stores its metadata as OME-XML. Any software package that supports OME-TIFF is also compatible with the dozens of formats listed on the Bio-Formats page, because Bio-Formats can convert your files to OME-TIFF format.

To facilitate support of OME-XML, we have created a library in Java for reading and writing OME-XML metadata.

There are three types of metadata in Bio-Formats, which we call core metadata, original metadata, and OME metadata.

1. **Core metadata** only includes things necessary to understand the basic structure of the pixels: image resolution; number of focal planes, time points, channels, and other dimensional axes; byte order; dimension order; color arrangement (RGB, indexed color or separate channels); and thumbnail resolution.

2. **Original metadata** is information specific to a particular file format. These fields are key/value pairs in the original format, with no guarantee of cross-format naming consistency or compatibility. Nomenclature often differs between formats, as each vendor is free to use their own terminology.

3. **OME metadata** is information from #1 and #2 converted by Bio-Formats into the OME data model. **Performing this conversion is the primary purpose of Bio-Formats.** Bio-Formats uses its ability to convert proprietary
metadata into OME-XML as part of its integration with the OME and OMERO servers—essentially, they are able to populate their databases in a structured way because Bio-Formats sorts the metadata into the proper places. This conversion is nowhere near complete or bug free, but we are constantly working to improve it. We would greatly appreciate any and all input from users concerning missing or improperly converted metadata fields.
2.1 Using Bio-Formats with ImageJ and Fiji

The following sections explain the features of Bio-Formats and how to use it within ImageJ and Fiji:

2.1.1 ImageJ overview

ImageJ is an image processing and analysis application written in Java, widely used in the life sciences fields, with an extensible plugin infrastructure. You can use Bio-Formats as a plugin for ImageJ to read and write images in the formats it supports.

Installation

Download bioformats_package.jar and drop it into your ImageJ/plugins folder. Next time you run ImageJ, a new Bio-Formats submenu with several plugins will appear in the Plugins menu, including the Bio-Formats Importer and Bio-Formats Exporter.

Usage

The Bio-Formats Importer plugin can display image stacks in several ways:

- In a standard ImageJ window (including as a hyperstack)
- With Joachim Walter’s Image5D plugin (if installed)
- With Rainer Heintzmann’s View5D plugin (if installed)

ImageJ v1.37 and later automatically (via HandleExtraFileTypes) calls the Bio-Formats logic, if installed, as needed when a file is opened within ImageJ, i.e. when using File → Open instead of explicitly choosing Plugins → Bio-Formats → Bio-Formats Importer from the menu.

For a more detailed description of each plugin, see the Bio-Formats page of the ImageJ wiki.
Upgrading

To upgrade, just overwrite the old `bioformats_package.jar` with the latest one.

You can also upgrade the Bio-Formats plugin directly from ImageJ. Select `Plugins → Bio-Formats → Update Bio-Formats Plugins` from the ImageJ menu, then select which release you would like to use. You will then need to restart ImageJ to complete the upgrade process.

Macros and plugins

Bio-Formats is fully scriptable in a macro, and callable from a plugin. To use in a macro, use the Macro Recorder to record a call to the Bio-Formats Importer with the desired options. You can also perform more targeted metadata queries using the Bio-Formats macro extensions.

Here are some example ImageJ macros and plugins that use Bio-Formats to get you started:

- `basicMetadata.txt` - A macro that uses the Bio-Formats macro extensions to print the chosen file’s basic dimensional parameters to the Log.
- `planeTimings.txt` - A macro that uses the Bio-Formats macro extensions to print the chosen file’s plane timings to the Log.
- `recursiveTiffConvert.txt` - A macro for recursively converting files to TIFF using Bio-Formats.
- `bfOpenAsHyperstack.txt` - This macro from Wayne Rasband opens a file as a hyperstack using only the Bio-Formats macro extensions (without calling the Bio-Formats Importer plugin).
- `zvi2HyperStack.txt` - This macro from Sebastien Huart reads in a ZVI file using Bio-Formats, synthesizes the LUT using emission wavelength metadata, and displays the result as a hyperstack.
- `dvSplitTimePoints.txt` - This macro from Sebastien Huart splits timepoints/channels on all DV files in a folder.
- `batchTiffConvert.txt` - This macro converts all files in a directory to TIFF using the Bio-Formats macro extensions.
- `Read_Image` - A simple plugin that demonstrates how to use Bio-Formats to read files into ImageJ.
- `Mass_Importer` - A simple plugin that demonstrates how to open all image files in a directory using Bio-Formats, grouping files with similar names to avoiding opening the same dataset more than once.

Usage tips

- “How do I make the options window go away?” is a common question. There are a few ways to do this:
  - To disable the options window only for files in a specific format, select `Plugins > Bio-Formats > Bio-Formats Plugins Configuration`, then pick the format from the list and make sure the “Windowless” option is checked.
  - To avoid the options window entirely, use the `Plugins > Bio-Formats > Bio-Formats Windowless Importer` menu item to import files.
  - Open files by calling the Bio-Formats importer plugin from a macro.

- A common cause of problems having multiple copies of `bioformats_package.jar` in your ImageJ plugins folder, or a copy of `bioformats_package.jar` and a copy of `formats-gpl.jar`. It is often difficult to determine for sure that this is the problem - the only error message that pretty much guarantees it is a `NoSuchMethodException`. If you downloaded the latest version and whatever error message or odd behavior you are seeing has been reported as fixed, it is worth removing all copies of `bioformats_package.jar` (or any other Bio-Formats jars) and download a fresh version.
• The Bio-Formats Exporter plugin’s file chooser will automatically add the first listed file extension to the file name if a specific file format is selected in the Files of Type box (e.g. .ome.tif for OME-TIFF). This can prevent BigTIFF and OME BigTIFF files from being created, as the .btf or .ome.btf file extension will be overwritten. To ensure that the desired extension is used, select All files or All supported file types in the Files of type box, as an extension will not be automatically added in those cases.

• Saving an open image using Bio-Formats must be done via Plugins > Bio-Formats > Bio-Formats Exporter or the corresponding macro code. File > Save and File > Save As… do not use Bio-Formats. In particular, using File > Save As… to save a TIFF will result in an ImageJ-specific TIFF being written. While Bio-Formats can read ImageJ TIFFs, other software may not; see TIFF (Tagged Image File Format) for additional information.

2.1.2 Fiji overview

Fiji is an image processing package. It can be described as a distribution of ImageJ together with Java, Java 3D and a lot of plugins organized into a coherent menu structure. Fiji compares to ImageJ as Ubuntu compares to Linux.

Fiji works with Bio-Formats out of the box, because it comes bundled with the Bio-Formats ImageJ plugins.

The Fiji documentation has been combined with the ImageJ wiki; for further details on Bio-Formats in Fiji, see the Bio-Formats ImageJ page.

Upgrading

Upgrading Bio-Formats within Fiji is as simple as invoking the “Update Fiji” command from the Help menu. By default, Fiji even automatically checks for updates every time it is launched, so you will always be notified when new versions of Bio-Formats (or any other bundled plugin) are available.

Manual upgrade

Manually updating your Fiji installation should not be necessary but if you need to do so, the steps are detailed below. Note that although we assume you will be upgrading to the latest release version, all previous versions of Bio-Formats are available from http://downloads.openmicroscopy.org/bio-formats/ so you can revert to an earlier version using this guide if you need to.

1) Fiji must first be fully updated
2) Close Fiji
3) Open the Fiji installation folder (typically named ‘Fiji.app’)
4) Remove bio-formats_plugins.jar from the ‘plugins’ sub-folder
5) Remove all of the .jars from the ‘jars/bio-formats’ sub-folder:
   • ome-jai.jar
   • formats-gpl.jar
   • ome-common.jar
   • turbojpeg.jar
   • ome-xml.jar
   • formats-bsd.jar
   • ome-poi.jar
   • specification.jar
• mdbtools-java.jar
• metakit.jar
• formats-api.jar

6) Download bio-formats_plugins.jar (from the latest release http://downloads.openmicroscopy.org/bio-formats/) and place it in the ‘plugins’ sub-folder

7) Download each of the following (from the latest release http://downloads.openmicroscopy.org/bio-formats/) and place them in the ‘jars/bio-formats’ sub-folder:
   • ome-jai.jar
   • formats-gpl.jar
   • ome-common.jar
   • turbojpeg.jar
   • ome-xml.jar
   • formats-bsd.jar
   • ome-poi.jar
   • specification.jar
   • mdbtools-java.jar
   • metakit.jar
   • formats-api.jar

8) To Check Version of Bio-Formats Select Help > About Plugins > Bio-Formats Plugins... Check that the version of Bio-Formats matches the freshly downloaded version.

9) Start Fiji and open any Image file using Plugins > Bio-Formats > Bio-Formats Importer

**Note:** It is vital to perform all of those steps in order; omitting even one will cause a problem. In particular, make sure that the old files are fully removed; it is not sufficient to add the new files to any sub-directory without removing the old files first.

### 2.1.3 Bio-Formats features in ImageJ and Fiji

When you select Bio-Formats under the Plugin menu, you will see the following features:

- The **Bio-Formats Importer** is a plugin for loading images into ImageJ or Fiji. It can read over 140 proprietary life sciences formats and standardizes their acquisition metadata into the common OME data model. It will also extract and set basic metadata values such as spatial calibration if they are available in the file.

- The **Bio-Formats Exporter** is a plugin for exporting data to disk. It can save to the open OME-TIFF file format, as well as several movie formats (e.g. QuickTime, AVI) and graphics formats (e.g. PNG, JPEG).

- The **Bio-Formats Remote Importer** is a plugin for importing data from a remote URL. It is likely to be less robust than working with files on disk, so we recommend downloading your data to disk and using the regular Bio-Formats Importer whenever possible.

- The **Bio-Formats Windowless Importer** is a version of the Bio-Formats Importer plugin that runs with the last used settings to avoid any additional dialogs beyond the file chooser. If you always use the same import settings, you may wish to use the windowless importer to save time (Learn more here).
• The **Bio-Formats Macro Extensions** plugin prints out the set of commands that can be used to create macro extensions. The commands and the instructions for using them are printed to the ImageJ log window.

• The **Stack Slicer** plugin is a helper plugin used by the Bio-Formats Importer. It can also be used to split a stack across channels, focal planes or time points.

• The **Bio-Formats Plugins Configuration** dialog is a useful way to configure the behavior of the Bio-Formats plugin or each file format. The general tab allows you to configure features of the Bio-Formats plugin such as the display of the slice label (see *Bio-Formats plugin configuration options*). The Formats tab lists supported file formats and toggles each format on or off, which is useful if your file is detected as the wrong format. It also toggles whether each format bypasses the importer options dialog through the “Windowless” checkbox. You can also configure any specific option for each format (see *Additional reader and writer options*). The Libraries tab provides a list of available helper libraries used by Bio-Formats.

• The **Bio-Formats Plugins Shortcut Window** opens a small window with a quick-launch button for each plugin. Dragging and dropping files onto the shortcut window opens them quickly using the **Bio-Formats Importer** plugin.

• The **Update Bio-Formats Plugins** command will check for updates to the plugins. We recommend you update to the newest Trunk build as soon as you think you may have discovered a bug.

### 2.1.4 Installing Bio-Formats in ImageJ

**Note:** Since FIJI is essentially ImageJ with plugins like Bio-Formats already built in, people who install Fiji can skip this section.

If you are also using the OMERO plugin for ImageJ, you may find the set-up guide on the new user help site useful for getting you started with both plugins at the same time.

Once you download and install ImageJ, you can install the Bio-Formats plugin by going to the Bio-Formats download page and saving the `bioformats_package.jar` to the Plugins directory within ImageJ.

![Plugin Directory for ImageJ](image)

Fig. 1: Plugin Directory for ImageJ: Where in ImageJ’s file structure you should place the file once you downloaded it.

You may have to quit and restart ImageJ. Once you restart it, you will find Bio-Formats in the Bio-Formats option under the Plugins menu:
You are now ready to start using Bio-Formats.

2.1.5 Using Bio-Formats to load images into ImageJ

This section will explain how to use Bio-Formats to import files into ImageJ and how to use the settings on the Bio-Formats Import Options screen.

Opening files

There are three ways you can open a file using Bio-Formats:

1. Select the Bio-Formats Importer under the Bio-Formats plugins menu.
2. Drag and drop it onto the Bio-Formats Plugins Shortcut window.
3. Use the Open command in the File menu.

Unless you used the Bio-Formats Plugins Configuration dialog to open the file type windowlessly, you know you used Bio-Formats to open a file when you see a screen like this:
If you used the File > Open command and did not see the Bio-Formats Import Options screen, ImageJ/Fiji probably used another plugin instead of Bio-Formats to open the file. If this happens and you want to open a file using Bio-Formats, use one of the other two methods instead.

**Opening files windowlessly**

When you open a file with Bio-Formats, the Import Options Screen automatically recalls the settings you last used to open a file with that specific format (e.g. JPG, TIF, LSM, etc.). If you always choose the same options whenever you open files in a specific format, you can save yourself time by bypassing the Bio-Formats Import Options screen. You can accomplish this two ways:

1. You can select the **Bio-Formats Windowless Importer**, located in the Bio-Formats menu under ImageJ’s Plugin menu. When you select this option, Bio-Formats will import the file using the same settings you used the last time you imported a file with the same format.

2. If you invariably use the same settings when you open files in a specific format, you can always bypass the Import Options Screen by changing the settings in the **Bio-Formats Plugins Configuration** option, which is also located in the Bio-Formats menu under ImageJ’s Plugin menu.

Once you select this option, select the file format you are interested in from the list on the left side of the screen. Check both the **Enabled** and **Windowless** boxes. Once you do this, whenever you open a file using the Bio-Formats Windowless Importer, the Bio-Formats Importer, or the drag-and-drop method described in the previous section, the file will always open the same way using the last setting used.

Please note that if you want to change any of the import settings once you enable this windowless option, you will have to go back to the Bio-Formats Plugins Configuration screen, unselect the windowless option, open a file using the regular Bio-Formats Importer, select your settings, and re-select the windowless option.
**Group files with similar names**

**Note:** The functionality described below is also available outside ImageJ, by using a pattern file to tell Bio-Formats how to group the files. See *Grouping files using a pattern file* for more information.

One of the most important features of Bio-Formats is to combine multiple files from a data set into one coherent, multi-dimensional image.

To demonstrate how to use the *Group files with similar names* feature, you can use the *dub* data set available under LOCI’s *Sample Data* page. You will notice that it is a large dataset: each of the 85 files shows the specimen at 33 optical sections along the z-plane at a specific time.

If you open just one file in ImageJ/Fiji using the *Bio-Formats Importer*, you will get an image incorporating three dimensions (x, y, z). However, if you select *Group files with similar names* from the Bio-Formats Import Options screen, you will be able to create a 4-D image (x, y, z, and t) incorporating the 85 files.

After clicking OK, you will see a screen like this:

![Image Description](image.png)

This screen allows you to select which files within the 85-file cluster to use to create that 4-D image. Some information will be pre-populated in the fields. Unless you want to change the settings in that field, there is no need to change or delete it. If you click OK at this point, you will load all 85 files.

However, you can specify which files you want to open by adjusting the “axis information”, the file “name contains”, or the “pattern” sections. Even though there are three options, you only need to need to make changes to one of them. Since Bio-Format’s precedence for processing data is from top to bottom, only the uppermost section that you made changes to will be used. If you change multiple boxes, any information you enter into lower boxes will be ignored.

To return to the example involving the *dub* data set, suppose you want to open the first image and only every fifth image afterwards (i.e. *dub01*, *dub06*, *dub11* … *dub81*). This would give you 17 images. There are different ways to accomplish this:

You can use the *Axis Settings* only when your files are numbered in sequential order and you want to open only a subset of the files that have similar names. Since the *dub* data set is numbered sequentially, you can use this feature.

**Axis 1 number of images** refers to the total number of images you want to open. Since you want to view 17 images, enter 17. **Axis 1 axis first image** specifies which image in the set you want to be the first. Since you want to start with *dub01*, enter 1 in that box. You also want to view only every fifth image, so enter 5 in the **Axis 1 axis increment** box.

The **File name contains** box should be used if all of the files that you want to open have common text. This is
especially useful when the files are not numbered. For example, if you have “Image_Red.tif”, “Image_Green.tif”, and “Image_Blue.tif” you could enter “Image_” in the box to group them all.

To continue the example involving the dub data set, you cannot use the file name contains box to open every fifth image. However, if you only wanted to open dub10 thorough dub19, you could enter “dub1” in the file name contains box.

The pattern box can be used to do either of the options listed above or much more. This box can accept a single file name like “dub01.pic”. It can also contain a pattern that use “<” and “>” to specify what numbers or text the file names contain.

There are three basic forms to the “< >” blocks:

- Text enumeration - “Image_<Red,Green,Blue>.tif” is the pattern for Image_Red.tif, Image_Green.tif, Image_Blue.tif. (Note that the order you in which you enter the file names is the order in which they will be loaded.)

- Number range - “dub<1-85>.pic” is the pattern for “dub1.pic”, “dub2.pic”, “dub3.pic” … “dub85.pic”.

- Number range with step - “dub<1-85:5>.pic” is the pattern for “dub1.pic”, “dub6.pic”, “dub11.pic”, “dub16.pic” … “dub85.pic”.

It can also accept a Java regular expression.

**Autoscale**

Autoscale helps increase the brightness and contrast of an image by adjusting the range of light intensity within an image to match the range of possible display values. Note that Autoscale does not change your data. It just changes how it is displayed.

Each pixel in an image has a numerical value ascribed to it to describe its intensity. The bit depth—the number of possible values—depends on the number of bits used in the image. Eight bits, for example, gives 256 values to express intensity where 0 is completely black, 255 is completely white, and 1 through 254 display increasingly lighter shades of grey.

ImageJ can collect the intensity information about each pixel from an image or stack and create a histogram (you can see it by selecting Histogram under the Analyze menu). Here is the histogram of a one particular image:
Notice that the histogram heavily skews left. Even though there are 256 possible values, only 0 through 125 are being used.

Autoscale adjusts the image so the smallest and largest number in that image or stack’s histogram become the darkest and brightest settings. For this image, pixels with the intensity of 125 will be displayed in pure white. The other values will be adjusted too to help show contrast between values that were too insignificant to see before.

Here is one image Bio-Formats imported with and without using Autoscale:
Autoscale readjusts the image based on the highest value in the entire data set. This means if the highest value in your dataset is close to maximum display value, Autoscale’s adjusting may be undetectable to the eye.

ImageJ/Fiji also has its own tools for adjusting the image, which are available by selecting Brightness/Contrast, which is under the Adjust option in the Image menu.
2.1.6 Managing memory in ImageJ/Fiji using Bio-Formats

When dealing with a large stack of images, you may receive a warning like this:

![Warning message](image.png)

This means the allotted memory is less than what Bio-Formats needs to load all the images. If you have a very large data set, you may have to:

- Crop the view area
- Open only a subset of images
- Use Virtual Stack
- Increase ImageJ/Fiji’s memory.

If your files contain JPEG or JPEG-2000 images, you may see this memory warning even if your file size is smaller than the amount of allocated memory. This is because compressed images like JPEG need to be decompressed into memory before being displayed and require more memory than their file size suggests. If you are having this issue, try utilizing one of the memory management tools below.

**Cropping the view area**

**Crop on Import** is useful if your images are very large and you are only interested in one specific section of the stack you are importing. If you select this feature, you will see a screen where you can enter the height and width (in pixels) of the part of image you want to see. Note that these measurements are from the top left corner of the image.

**Opening only a subset of images**

The **Specify Range for Each Series** option is useful for viewing a portion of a data set where all the plane images are encapsulated into one file (e.g. the Zeiss LSM format). If your file has a large quantity of images, you can specify which channels, Z-planes, and times you want to load.

**Use Virtual Stack**

**Virtual Stack** conserves memory by not loading specific images until necessary. Virtual Stack does not contain a buffer and may produce choppy animations.
Increasing ImageJ/Fiji’s memory

Finally, you can also increase the amount of the computer memory devoted to ImageJ/Fiji by selecting Memory & Threads under the Edit menu.

Generally, allocating more than 75% of the computer’s total memory will cause ImageJ/Fiji to become slow and unstable.

Please note that, unlike the other two features, ImageJ/Fiji itself provides this feature and not Bio-Formats. You can find out more about this feature by looking at ImageJ’s documentation.
2.1.7 Bio-Formats plugin configuration options

The Bio-Formats plugin can be configured by opening the Bio-Formats Plugins Configuration dialog from the Plugin menu. The General tab allows for configuration of features for the plugin such as upgrade checking or modifying the slice label pattern.

Configuring the slice label pattern

The slice label is the text displayed at the top of the image window in ImageJ. The values displayed here can be modified by configuring the slice label pattern.

The list of available parameters for configuration is as follows:

- \%s
  series index
- \%n
  series name
- \%c
  channel index
- \%w
  channel name
- \%z
  Z index
- \%t
  T index
- \%A
  acquisition timestamp

Each index value will be 1-based rather than 0-based. The index will be displayed along with the total dimension count and with a prefix for the particular dimension. For example using \%c for channel index will result in the display c:1/3.

See also:

Additional reader and writer options

2.2 Command line tools

The Bio-Formats Command line tools (bftools.zip) provide a complete package for carrying out a variety of tasks:

2.2.1 Command line tools

There are several scripts for using Bio-Formats on the command line.
Displaying images and metadata

The `showinf` command line tool can be used to show the images and metadata contained in a file. If no options are specified, `showinf` displays a summary of available options.

To simply display images:

```
showinf /path/to/file
```

All of the images in the first ‘series’ (or 5 dimensional stack) will be opened and displayed in a simple image viewer. The number of series, image dimensions, and other basic metadata will be printed to the console.

- `-noflat`
  Do not flatten resolutions into individual series:

```
showinf -noflat /path/to/file
```

- `-series SERIES`
  Displays a different series, for example the second one:

```
showinf -series 1 /path/to/file
```

Note that series numbers begin with 0.

- `-omexml`
  Displays the OME-XML metadata for a file on the console:

```
showinf -omexml /path/to/file
```

- `-nopix`
  Image reading can be suppressed if only the metadata is needed:

```
showinf -nopix /path/to/file
```

- `-option KEY VALUE`
  Passes options expressed as key/value pairs:

```
showinf -option key value /path/to/file
```

  e.g. additional reader options, see Additional reader and writer options:

```
showinf -option leicalif.old_physical_size true /path/to/file
```

New in version 5.3.0.

- `-range START END`
  A subset of images can also be opened instead of the entire stack, by specifying the start and end plane indices (inclusive):

```
showinf -range 0 0 /path/to/file
```

That opens only the first image in first series in the file.

- `-crop X,Y,WIDTH,HEIGHT`
  For very large images, it may also be useful to open a small tile from the image instead of reading everything into memory. To open the upper-left-most 512×512 tile from the images:
The parameter to \texttt{-crop} is of the format \texttt{x,y,width,height}. The (x, y) coordinate (0, 0) is the upper-left corner of the image; \texttt{x + width} must be less than or equal to the image width and \texttt{y + height} must be less than or equal to the image height.

\textit{-no-upgrade}

By default, \texttt{showinf} will check for a new version of Bio-Formats. This can take several seconds (especially on a slow internet connection); to save time, the update check can be disabled:

\begin{verbatim}
showinf -no-upgrade /path/to/file
\end{verbatim}

\textit{-novalid}

Similarly, if OME-XML is displayed then it will automatically be validated. On slow or missing internet connections, this can take some time, and so can be disabled:

\begin{verbatim}
showinf -novalid /path/to/file
\end{verbatim}

\textit{-nocore}

Most output can be suppressed:

\begin{verbatim}
showinf -nocore /path/to/file
\end{verbatim}

\textit{-omexml-only}

Displays the OME-XML alone:

\begin{verbatim}
showinf -omexml-only /path/to/file
\end{verbatim}

This is particularly helpful when there are hundreds or thousands of series.

\textit{-debug}

Enables debugging output if more information is needed:

\begin{verbatim}
showinf -debug /path/to/file
\end{verbatim}

\textit{-fast}

Displays an image as quickly as possible. This is achieved by converting the raw data into a 8 bit RGB image:

\begin{verbatim}
showinf -fast /path/to/file
\end{verbatim}

\textbf{Note:} Due to the data conversion to a RGB image, using this option results in a loss of precision.

\textit{-autoscale}

Adjusts the display range to the minimum and maximum pixel values:

\begin{verbatim}
showinf -autoscale /path/to/file
\end{verbatim}

\textbf{Note:} This option automatically sets the \texttt{-fast} option and suffers from the same limitations.
-cache
Caches the reader under the same directory as the input file after initialization:

```
showinf -cache /path/to/file
```

-cache-dir DIR
Specifies the base directory under which the reader should be cached:

```
showinf -cache-dir /tmp/cachedir /path/to/file
```

-swap DIMENSIONORDER
overrides the default input dimension order:

```
showinf -swap XYZTC /path/to/file
```

-format FORMAT
Specifies the reader to be used for opening the specified file. The utility will look for a reader named loci.formats.in.<FORMAT>Reader. If the reader does not exist or no \-format option is passed, the file will be opened with loci.formats.in.ImageReader:

```
showinf -format APNG test.png
```

-fill UNSIGNED_BYTE
Sets the fill value to use for undefined pixels. Valid values are 0-255, with 0 representing a black pixel and 255 representing a white pixel. The value set here will be applied to all bytes in an undefined pixel, e.g. setting to 128 (0x80) on a uint16 image will result in a pixel value of 32896 (0x8080). The default fill value is typically 0, but some readers may have different defaults.

New in version 6.13.0.

Converting a file to different format

The bfconvert command line tool can be used to convert files between supported formats.

bfconvert with no options displays a summary of available options.

To convert a file to single output file (e.g. TIFF):

```
bfconvert /path/to/input output.tiff
```

The output file format is determined by the extension of the output file, e.g. .tifffor TIFF files, .ome.tifffor OME-TIFF, .png for PNG.

-option KEY VALUE
Passes options expressed as key/value pairs:

```
bfconvert -option key value /path/to/input /path/to/output
```

e.g. additional writer options, see Additional reader and writer options:

```
bfconvert -option ometiff.companion converted.companion.ome input.fake converted.→ome.tiff
```

New in version 5.4.0.
-noflat
Do not flatten resolutions into individual series. This option is mandatory to read images with pyramidal levels using the sub-resolution API and generate an output image with sub-resolutions. As of Bio-Formats 6.0.0, only the OME-TIFF output format properly supports this option:

```
bfconvert -noflat /path/to/input output-first-series.ome.tiff
```
New in version 6.0.0.

-series SERIES
All images in the input file are converted by default. To convert only one series:

```
bfconvert -series 0 /path/to/input output-first-series.tiff
```

-timepoint TIMEPOINT
To convert only one timepoint:

```
bfconvert -timepoint 0 /path/to/input output-first-timepoint.tiff
```

-channel CHANNEL
To convert only one channel:

```
bfconvert -channel 0 /path/to/input output-first-channel.tiff
```

-z Z
To convert only one Z section:

```
bfconvert -z 0 /path/to/input output-first-z.tiff
```

-range START END
To convert images between certain indices (inclusive):

```
bfconvert -range 0 2 /path/to/input output-first-3-images.tiff
```

-tilex TILEX, -tiley TILEY
All images larger than 4096×4096 will be saved as a set of tiles if the output format supports doing so. The default tile size is determined by the input format, and can be overridden like this:

```
bfconvert -tilex 512 -tiley 512 /path/to/input output-512x512-tiles.tiff
```

-tilex is the width in pixels of each tile; -tiley is the height in pixels of each tile. The last row and column of tiles may be slightly smaller if the image width and height are not multiples of the specified tile width and height. Note that specifying -tilex and -tiley will cause tiles to be written even if the image is smaller than 4096×4096.

Also note that the specified tile size will affect performance. If large amounts of data are being processed, it is a good idea to try converting a single tile with a few different tile sizes using the -crop option. This gives an idea of what the most performant size will be.

-crop X,Y,WIDTH,HEIGHT
For very large images, it may also be useful to convert a small tile from the image instead of reading everything into memory. To convert the upper-left-most 512×512 tile from the images:

```
bfconvert -crop 0,0,512,512 /path/to/file output-512x512-crop.tiff
```
The parameter to -crop is of the format \(x, y, width, height\). The \((x, y)\) coordinate \((0, 0)\) is the upper-left corner of the image; \(x + width\) must be less than or equal to the image width and \(y + height\) must be less than or equal to the image height.

Images can also be written to multiple files by specifying a pattern string in the output file. For example, to write one series, timepoint, channel, and Z section per file:

```
bfconvert /path/to/input output_series_%s_Z%z_C%c_T%t.tif
```

\(\%s\) is the series index, \(\%z\) is the Z section index, \(\%c\) is the channel index, and \(\%t\) is the timepoint index (all indices begin at 0).

For large images in particular, it can also be useful to write each tile to a separate file:

```
bfconvert -tilex 512 -tiley 512 /path/to/input output_tile_%x_%y_%m.jpg
```

\(\%x\) is the row index of the tile, \(\%y\) is the column index of the tile, and \(\%m\) is the overall tile index. As above, all indices begin at 0. Note that if \(\%x\) or \(\%y\) is included in the file name pattern, then the other must be included too. The only exception is if \(\%m\) was also included in the pattern.

Note for Windows Users: The command interpreter for batch files needs the \(\%\) characters to be doubled in order to process the sequencing variables correctly. So in Windows, the above example would read:

```
bfconvert /path/to/input output_series_%%s_Z%%z_C%%c_T%%t.tif
```

-\(\text{compression}\) COMPRESSION

By default, all images will be written uncompressed. Supported compression modes vary based upon the output format, but when multiple modes are available the compression can be changed using the -\(\text{compression}\) option. For example, to use LZW compression in a TIFF file:

```
bfconvert -compression LZW /path/to/input output-lzw.tiff
```

-\(\text{overwrite}\)

If the specified output file already exists, bfconvert will prompt to overwrite the file. When running bfconvert non-interactively, it may be useful to always allow bfconvert to overwrite the output file:

```
bfconvert -overwrite /path/to/input /path/to/output
```

-\(\text{noverwrite}\)

To always exit without overwriting:

```
bfconvert -noverwrite /path/to/input /path/to/output
```

-\(\text{nolookup}\)

To disable the conversion of lookup tables, leaving the output file without any lookup tables:

```
bfconvert -nolookup /path/to/input /path/to/output
```

New in version 5.2.1.

-\(\text{bigtiff}\)

This option forces the writing of a BigTiff file:

```
bfconvert -bigtiff /path/to/input output.ome.tiff
```
New in version 5.1.2.

The `-bigtiff` option is not necessary if a BigTiff extension is used for the output file, e.g.:

`bfconvert /path/to/input output.ome.btf`

**-nobigtiff**

This option disables the automatic switching to BigTiff based upon the number of pixel bytes (TIFF files larger than 4GB):

`bfconvert -nobigtiff /path/to/input output.ome.tiff`

New in version 6.4.0.

Using the `-nobigtiff` will disable writing BigTiff when the output format is less than 4GB. It will not be able to write standard Tiff files greater than 4GB. An example of when it might be used would be when converting using a compression codec that reduces the size of the output file, e.g.:

`bfconvert -nobigtiff -compression LZW /path/to/input output.ome.btf`

**-padded**

This option is used alongside a pattern string when writing an image to multiple files. When set this will enforce zero padding on the filename indexes set in the provided pattern string:

`bfconvert /path/to/input output_xy%sz%zc%ct%t.ome.tif -padded`

New in version 5.2.2.

**-pyramid-resolutions** RESOLUTIONS

**-pyramid-scale** SCALE

When using `-noflat` by default, each series of the converted file will contain the same number of resolutions as in the input file. The `-pyramid-resolutions` option allows to set the number of expected resolutions in the output file for each series. If the target number of resolutions is greater than the actual number of sub-resolutions present in the input file, additional pyramidal levels will be calculated using the downsampling factor specified by the `-pyramid-scale` option:

`bfconvert -noflat -pyramid-resolutions 4 -pyramid-scale 2 /path/to/input out.ome.tiff`

New in version 6.0.0.

**-cache**

This option will cache the initialized reader under the same directory as the input file after initialization:

`bfconvert -cache /path/to/input output.ome.tiff`

New in version 6.2.0.

**-cache-dir** DIRECTORY

This option is to be used in conjunction with `-cache`. When used it specifies the directory to store the cached initialized reader. If unspecified, the cached reader will be stored under the same folder as the image file:

`bfconvert -cache-dir /path/to/store/cached/reader /path/to/input output.ome.tiff`

New in version 6.2.0.
-no-sas
Do not preserve the OME-XML StructuredAnnotation elements:

```bash
bfconvert -no-sas /path/to/input output.ome.tiff
```

New in version 6.2.0.

-no-sequential
Do not assume that planes are written in sequential order:

```bash
bfconvert -no-sequential /path/to/input output.ome.tiff
```

New in version 6.8.0.

-swapp DIMENSIONORDER
Overrides the default input dimension order:

```bash
bfconvert -swap XYZTC /path/to/input output.ome.tiff
```

New in version 6.9.0.

-fill U N S I G N E D _B Y T E
Sets the fill value to use for undefined pixels. Valid values are 0-255, with 0 representing a black pixel and 255 representing a white pixel. The value set here will be applied to all bytes in an undefined pixel, e.g. setting to 128 (0x80) on a uint16 image will result in a pixel value of 32896 (0x8080). The default fill value is typically 0, but some readers may have different defaults.

New in version 6.13.0.

Validating XML in an OME-TIFF

The XML stored in an OME-TIFF file can be validated using the command line tools.

Both the `tiffcomment` and `xmlvalid` commands are used; `tiffcomment` extracts the XML from the file and `xmlvalid` validates the XML and prints any errors to the console.

For example:

```bash
xmlvalid /path/to/file.ome.tiff
```

and

```bash
xmlvalid /path/to/file.xml
```

will perform validation for an ome.xml file and xml file, respectively.

Also:

```bash
tiffcomment "/path/to/file.ome.tiff" | xmlvalid
```

will perform the extraction and validation all at once.

Typical successful output of `xmlvalid` is:

```bash
[~/Work/bftools]$ ./xmlvalid sample.ome
Parsing schema path
http://www.openmicroscopy.org/Schemas/OME/2010-06/ome.xsd
```

(continues on next page)
Validating sample.ome
No validation errors found.
[~/Work/bftools]$

If any errors are found they are reported. When correcting errors it is usually best to work from the top of the file as errors higher up can cause extra errors further down. In this example the output shows 3 errors but there are only 2 mistakes in the file:

If the XML is found to have validation errors, the **tiffcomment** command can be used to overwrite the XML in the OME-TIFF file with corrected XML. The XML can be displayed in an editor window:

```
tiffcomment -edit /path/to/file.ome.tif
```

or the new XML can be read from a file:

```
tiffcomment -set new-comment.xml /path/to/file.ome.tif
```

**Editing XML in an OME-TIFF**

To edit the XML in an OME-TIFF file you can use **tiffcomment**, one of the Bio-Formats tools.

**Note:** The **tiffcomment** tool requires that the *ImageDescription* tag is present in the TIFF file and will error otherwise.

To use the built in editor run:

```
tiffcomment -edit sample.ome.tif
```

To extract or view the XML run:

```
tiffcomment sample.ome.tif
```

To inject replacement XML into a file run:

```
tiffcomment -set 'newmetadata.xml' sample.ome.tif
```
**List formats by domain**

Each supported file format has one or more imaging domains associated with it. To print the list of formats associated with each imaging domain:

```plaintext
domainlist
```

The command does not accept any arguments. The known image domains are defined by:

- `ASTRONOMY_DOMAIN`
- `EM_DOMAIN`
- `FLIM_DOMAIN`
- `GEL_DOMAIN`
- `GRAPHICS_DOMAIN`
- `HCS_DOMAIN`
- `HISTOLOGY_DOMAIN`
- `LM_DOMAIN`
- `MEDICAL_DOMAIN`
- `SEM_DOMAIN`
- `SPM_DOMAIN`
- `UNKNOWN_DOMAIN`

**List supported file formats**

A detailed list of supported formats can be displayed using the `formatlist` command.

The default behavior is to print a plain-text list of formats:

```plaintext
formatlist
```

- `-txt`
  Prints the list of formats as plain-text:
  ```plaintext
  formatlist -txt
  ```

- `-html`
  Prints the list of formats as HTML:
  ```plaintext
  formatlist -html
  ```

- `-xml`
  Prints the list of formats as XML:
  ```plaintext
  formatlist -xml
  ```

- `-help`
  Displays the usage information:
Display file in ImageJ

Files can be displayed from the command line in ImageJ. The Bio-Formats importer plugin for ImageJ is used to open the file.

The command takes a single argument:

```bash
ijview /file/to/open
```

If the input file is not specified, ImageJ will show a file chooser window.

The Bio-Formats import options window will then appear, after which the image(s) will be displayed.

If the `BF_DEVEL` environment variable is set, the ImageJ jar `<jars/ij.jar>` must be included in the classpath.

Format XML data

The `xmlindent` command formats and adds indenting to XML so that it is easier to read. Indenting is currently set to 3 spaces.

If an XML file name is not specified, the XML to indent will be read from standard output. Otherwise, one or more file names can be specified:

```bash
xmlindent /path/to/xml
xmlindent /path/to/first-xml /path/to/second-xml
```

The formatted XML from each file will be printed in the order in which the files were specified.

By default, extra whitespace may be added to CDATA elements. To preserve the contents of CDATA elements:

```bash
xmlindent -valid /path/to/xml
```

Create a high-content screen for testing

The `mkfake` command creates a high-content screen for testing. The image data will be meaningless, but it allows testing of screen, plate, and well metadata without having to find appropriately-sized screens from real acquisitions.

If no arguments are specified, `mkfake` prints usage information.

To create a single screen with default plate dimensions:

```bash
mkfake default-screen.fake
```

This will create a directory that represents one screen with a single plate containing one well, one field, and one acquisition of the plate (see PlateAcquisition).

```bash
-mkfake -plates PLATES
```

To change the number of plates in the screen:

```bash
mkfake -plates 3 three-plates.fake
```
Bio-Formats, Release 6.14.0

-**runs** RUNS

To change the number of acquisitions for each plate:

```
mkfake -runs 4 four-plate-acquisitions.fake
```

-**rows** ROWS

To change the number of rows of wells in each plate:

```
mkfake -rows 8 eight-row-plate.fake
```

-**columns** COLUMNS

To change the number of columns of wells in each plate:

```
mkfake -columns 12 twelve-column-plate.fake
```

-**fields** FIELDS

To change the number of fields per well:

```
mkfake -fields 2 two-field-plate.fake
```

It is often most useful to use the arguments together to create a realistic screen, for example:

```
mkfake -rows 16 -columns 24 -plates 2 -fields 3 two-384-well-plates.fake
```

-**debug** DEBUG

As with other command line tools, debugging output can be enabled if necessary:

```
mkfake -debug debug-screen.fake
```

**Installation**

Download bftools.zip, unzip it into a new folder.

---

**Note:** As of Bio-Formats 5.0.0, this zip now contains the bundled jar and you no longer need to download bioformats_package.jar separately.

The zip file contains both Unix scripts and Windows batch files.

**Tools available**

Currently available tools include:

-**showinf**

  Prints information about a given image file to the console, and displays the image itself in the Bio-Formats image viewer (see Displaying images and metadata for more information).

-**ijview**

  Displays the given image file in ImageJ using the Bio-Formats Importer plugin. See Display file in ImageJ for details.

-**bfconvert**

  Converts an image file from one format to another. Bio-Formats must support writing to the output file (see Converting a file to different format for more information).
formatlist
   Displays a list of supported file formats in HTML, plaintext or XML. See List supported file formats for details.

xmlindent
   A simple XML prettifier similar to xmllint --format but more robust in that it attempts to produce output
   regardless of syntax errors in the XML. See Format XML data for details.

xmlvalid
   A command-line XML validation tool, useful for checking an OME-XML document for compliance with the
   OME-XML schema. See Validating XML in an OME-TIFF for details.

tiffcomment
   Dumps the comment from the given TIFF file’s first IFD entry; useful for examining the OME-XML block in an
   OME-TIFF file (also see Editing XML in an OME-TIFF).

domainlist
   Displays a list of imaging domains and the supported formats associated with each domain. See List formats by
domain for more information.

mkfake
   Creates a “fake” high-content screen with configurable dimensions. This is useful for testing how HCS metadata
   is handled, without requiring real image data from an acquired screen. See Create a high-content screen for
testing for more information.

Some of these tools also work in combination, for example Validating XML in an OME-TIFF uses both tiffcomment
and xmlvalid.

Running any of these commands without any arguments will print usage information to help you. When run with the
-version argument, showinf and bfconvert will display the version of Bio-Formats that is being used (version
number, build date, and Git commit reference).

Command-line environment

A set of environment variables can be passed to all command-line utilities:

**BF_CP**
   Extra directories to be added to the autodetected command-line classpath e.g. for external reader JARs. Default: empty.

**BF_FLAGS**
   Additional flags to be sent to the JVM. Default: empty.

**BF_MAX_MEM**
   Maximum heap size to be allocated to the JVM. Default: 512m.

**BF_PROFILE**
   Enable profiling - see Profiling for more information. Default: off.

**BF_PROFILE_DEPTH**
   Maximum profiling depth if profiling is activated. Default: 30.
Using the tools directly from source

Firstly, obtain a copy of the sources and build them (see Obtaining and building Bio-Formats). You can configure the scripts to use your source tree instead of bioformats_package.jar in the same directory by following these steps:

1. Point your CLASSPATH to the checked-out directory and the JAR files in the jar folder.
   - E.g. on Windows with Java 1.8 or later, if you have checked out the source at C:\code\bio-formats, set your CLASSPATH environment variable to the value C:\code\bio-formats\jar\*;C:\code\bio-formats. You can access the environment variable configuration area by right-clicking on My Computer, choosing Properties, Advanced tab, Environment Variables button.

2. Compile the source with ant compile.

3. Set the BF_DEVEL environment variable to any value (the variable just needs to be defined).

Version checker

If you run bftools outside of the OMERO environment, you may encounter an issue with the automatic version checker causing a tool to crash when trying to connect to upgrade.openmicroscopy.org.uk. The error message will look something like this:

```
Failed to compare version numbers
java.io.IOException: Server returned HTTP response code: 400 for URL: http://upgrade.openmicroscopy.org.uk?version=4.4.8;os.name=Linux;os.version=2.6.32-358.6.2.el6.x86_64;os.arch=amd64;java.runtime.version=1.6.0_24-b24;java.vm.vendor=Sun+Microsystems+Inc.;bioformats.caller=Bio-Formats+utilities
```

To avoid this issue, call the tool with the -no-upgrade parameter.

Profiling

For debugging errors or investigating performance issues, it can be useful to use profiling tools while running Bio-Formats. The command-line tools can invoke the HPROF agent library to profile Heap and CPU usage. Setting the BF_PROFILE environment variable allows to turn profiling on, e.g.:

```
BF_PROFILE=true showinf -nopix -no-upgrade myfile
```

See also:

Additional reader and writer options

2.3 OMERO

OMERO 5 uses Bio-Formats to read original files from over 140 file formats. Please refer to the OMERO documentation for further information.

Many other software packages can use Bio-Formats to read and write microscopy formats (click on the package for further details):
2.4 Image server applications

2.4.1 BISQUE

The BISQUE (Bio-Image Semantic Query User Environment) Database, developed at the Center for Bio-Image Informatics at UCSB, was developed for the exchange and exploration of biological images. The Bisque system supports several areas useful for imaging researchers from image capture to image analysis and querying. The bisque system is centered around a database of images and metadata. Search and comparison of datasets by image data and content is supported. Novel semantic analyses are integrated into the system allowing high level semantic queries and comparison of image content.

Bisque integrates with Bio-Formats by calling the *showinf command line tool*.

2.4.2 OME Server

The OME server is a set of software that interacts with a database to manage images, image metadata, image analysis and analysis results. The OME system is capable of leveraging Bio-Formats to import files.

**Please note** - the OME server is no longer maintained and has now been superseded by the [OMERO server](https://www.openmicroscopy.org/omero/). Support for the OME server has been entirely removed in the 5.0.0 version of Bio-Formats; the following instructions can still be used with the 4.4.x versions.

**Installation**

For OME Perl v2.6.1 and later, the command line installer automatically downloads the latest `loci_tools.jar` and places it in the proper location. This location is configurable, but is `/OME/java/loci_tools.jar` by default.

For a list of what was recognized for a particular import into the OME server, go to the Image details page in the web interface, and click the “Image import” link in the upper right hand box.

Bio-Formats is capable of parsing original metadata for supported formats, and standardizes what it can into the OME data model. For the rest, it expresses the metadata in OME terms as key/value pairs using an OriginalMetadata custom semantic type. However, this latter method of metadata representation is of limited utility, as it is not a full conversion into the OME data model.

Bio-Formats is enabled in OME v2.6.1 for all formats except:

- OME-TIFF
- Metamorph HTD
- Deltavision DV
- Metamorph STK
- Bio-Rad PIC
- Zeiss LSM
- TIFF
- BMP
- DICOM
- OME-XML
The above formats have their own Perl importers that override Bio-Formats, meaning that Bio-Formats is not used to process them by default. However, you can override this behavior (except for Metamorph HTD, which Bio-Formats does not support) by editing an OME database configuration value:

% psql ome

To see the current file format reader list:

```sql
ome=# select value from configuration where name='import_formats';
value
--------------------------------------------------------------------------------
[ 'OME::ImportEngine::OMETIFFreader', 'OME::ImportEngine::MetamorphHTDFormat',
 'OME::ImportEngine::DVreader', 'OME::ImportEngine::STKreader',
 'OME::ImportEngine::BioradReader', 'OME::ImportEngine::LSMreader',
 'OME::ImportEngine::TIFFreader', 'OME::ImportEngine::BMPreader',
 'OME::ImportEngine::DICOMreader', 'OME::ImportEngine::XMLreader',
 'OME::ImportEngine::BioFormats' ]
(1 row)
```

To remove extraneous readers from the list:

```sql
ome=# update configuration set value=['\OME::ImportEngine::MetamorphHTDFormat\',
 '\OME::ImportEngine::XMLreader\', '\OME::ImportEngine::BioFormats\']
where name='import_formats';
UPDATE 1
ome=# select value from configuration where name='import_formats';
value
--------------------------------------------------------------------------------
[ 'OME::ImportEngine::MetamorphHTDFormat', 'OME::ImportEngine::XMLreader',
 'OME::ImportEngine::BioFormats' ]
(1 row)
```

To reset things back to how they were:

```sql
ome=# update configuration set value=['\OME::ImportEngine::OMETIFFreader\',
 '\OME::ImportEngine::MetamorphHTDFormat\', '\OME::ImportEngine::DVreader\',
 '\OME::ImportEngine::STKreader\', '\OME::ImportEngine::BioradReader\',
 '\OME::ImportEngine::LSMreader\', '\OME::ImportEngine::TIFFreader\',
 '\OME::ImportEngine::BMPreader\', '\OME::ImportEngine::DICOMreader\',
 '\OME::ImportEngine::XMLreader\', '\OME::ImportEngine::BioFormats\']
where name='import_formats';
```

Lastly, please note that Li-Cor L2D files cannot be imported into an OME server. Since the OME perl server has been discontinued, we have no plans to fix this limitation.

### Upgrading

OME server is not supported by Bio-Formats versions 5.0.0 and above. To take advantage of more recent improvements to Bio-Formats, you must switch to OMERO server.

2.4. Image server applications
Source Code

The source code for the Bio-Formats integration with OME server spans three languages, using piped system calls in both directions to communicate, with imported pixels written to OMEIS pixels files. The relevant source files are:

- OmeisImporter.java – omebf Java command line tool
- BioFormats.pm – Perl module for OME Bio-Formats importer
- omeis.c – OMEIS C functions for Bio-Formats (search for “bioformats” case insensitively to find relevant sections)

2.5 Libraries and scripting applications

2.5.1 FARSIGHT

FARSIGHT is a collection of modules for image analysis created by LOCI’s collaborators at the University of Houston. These open source modules are built on the ITK library and thus can take advantage of ITK’s support for Bio-Formats to process otherwise unsupported image formats.

The principal FARSIGHT module that benefits from Bio-Formats is the Nucleus Editor, though in principle any FARSIGHT-based code that reads image formats via the standard ITK mechanism will be able to leverage Bio-Formats.

See also:

FARSIGHT Downloads page
FARSIGHT HowToBuild tutorial

2.5.2 i3dcore

i3dcore, also known as the CBIA 3D image representation library, is a 3D image processing library developed at the Centre for Biomedical Image Analysis. Together with i3dalgo and i4dcore, i3dcore forms a continuously developed templated cross-platform C++ suite of libraries for multidimensional image processing and analysis.

i3dcore is capable of reading images with Bio-Formats using Java for C++ (java4cpp).

2.5.3 ImgLib

ImgLib2 is a multidimensional image processing library. It provides a general mechanism for writing image analysis algorithms, without writing case logic for bit depth, or worrying about the source of the pixel data (arrays in memory, files on disk, etc.).

The SCIFIO project provides an ImgOpener utility class for reading data into ImgLib2 data structures using Bio-Formats.
2.5.4 ITK

The Insight Toolkit (ITK) is an open-source, cross-platform system that provides developers with an extensive suite of software tools for image analysis. Developed through extreme programming methodologies, ITK employs leading-edge algorithms for registering and segmenting multidimensional data.

ITK provides an ImageIO plug-in structure that works via discovery through a dependency injection scheme. This allows a program built on ITK to load plug-ins for reading and writing different image types without actually linking to the ImageIO libraries required for those types. Such encapsulation automatically grants two major boons: firstly, programs can be easily extended just by virtue of using ITK (developers do not have to specifically accommodate or anticipate what plug-ins may be used). Secondly, the architecture provides a distribution method for open source software, like Bio-Formats, which have licenses that might otherwise exclude them from being used with other software suites.

The SCIFIO ImageIO plugin provides an ITK imageIO base that uses Bio-Formats to read and write supported life sciences file formats. This plugin allows any program built on ITK to read any of the image types supported by Bio-Formats.

2.5.5 Qu for MATLAB

Qu for MATLAB is a MATLAB toolbox for the visualization and analysis of multi-channel 4-dimensional datasets targeted to the field of biomedical imaging, developed by Aaron Ponti.

• Uses Bio-Formats to read files
• Open source software available under the Mozilla Public License

See also:
Qu for MATLAB download page

2.6 Numerical data processing applications

2.6.1 GNU Octave

GNU Octave is a high-level interpreted language, primarily intended for numerical computations. Being an array programming language, it is naturally suited for image processing and handling of N dimensional datasets. Octave is distributed under the terms of the GNU General Public License.

The Octave language is MATLAB compatible so that programs are easily portable. Indeed, the Octave bioformats package is exactly the same as MATLAB’s, the only difference being the installation steps.

Requirements

The bioformats package requires Octave version 4.0.0 or later with support for java:

```
$ octave
>> OCTAVE_VERSION
ans = 4.0.0
>> usejava ("jvm")
ans = 1
```
Bio-Formats, Release 6.14.0

Installation

1. Download bioformats_package.jar and place it somewhere sensible for your system (in Linux, this will probably be /usr/local/share/java or ~/.local/share/java for a system-wide or user installation respectively).
2. Add bioformats_package.jar to Octave’s static javaclasspath (see Octave’s documentation).
3. Download the Octave package.
4. Start octave and install the package with:

   ```octave
   >> pkg install path-to-bioformats-octave-version.tar.gz
   ```

Usage

Usage instructions are the same as MATLAB. The only difference is that you need to explicitly load the package. This is done by running at the Octave prompt:

```octave
>> pkg load bioformats
```  

Upgrading

To use a newer version of Bio-Formats, repeat the install instructions. Do not follow the MATLAB instructions.

2.6.2 IDL

IDL (Interactive Data Language) is a popular data visualization and analysis platform used for interactive processing of large amounts of data including images.

IDL possesses the ability to interact with Java applications via its IDL-Java bridge. Karsten Rodenacker has written a script that uses Bio-Formats to read in image files to IDL.

Installation

Download the ij_read_bio_formats.pro script from Karsten Rodenacker’s IDL goodies (?) web site. See the comments at the top of the script for installation instructions and caveats.

Upgrading

To use a newer version of Bio-Formats, overwrite the requisite JAR files with the newer version and restart IDL.

2.6.3 KNIME

KNIME (Konstanz Information Miner) is a user-friendly and comprehensive open-source data integration, processing, analysis, and exploration platform. KNIME supports image import using Bio-Formats using the KNIME Image Processing (a.k.a. KNIP) plugin.
2.6.4 MATLAB

MATLAB is a high-level language and interactive environment that facilitates rapid development of algorithms for performing computationally intensive tasks.

Calling Bio-Formats from MATLAB is fairly straightforward, since MATLAB has built-in interoperability with Java. We have created a toolbox for reading and writing image files. Note the minimum recommended MATLAB version is R2017b.

**Note:** It is possible to run Bio-Formats 6 on earlier MATLAB versions using a JVM version 8 or greater although using a different JVM than the one shipped with MATLAB can affect other functionalities. Please refer to the MATLAB Answers for more information.

**Installation**

Download the MATLAB toolbox from the Bio-Formats downloads page. Unzip `bfmatlab.zip` and add the unzipped `bfmatlab` folder to your MATLAB path.

**Usage**

Please see *Using Bio-Formats in MATLAB* for usage instructions. If you intend to extend the existing .m files, please also see the developer page for more information on how to use Bio-Formats in general.

**Performance**

In our tests (MATLAB R14 vs. java 1.6.0_20), the script executes at approximately half the speed of our `showinf` command line tool, due to overhead from copying arrays.

**Troubleshooting**

If you encounter an error trying to open JPEG-2000 data in MATLAB but the file will open e.g. in Fiji using Bio-Formats, it may be due to conflicting versions of JAI ImageIO in different JARs. As discussed on the component page, JAI ImageIO is no longer maintained and you will likely need to remove the conflicting JAR(s) as a workaround.

**Upgrading**

To use a newer version of Bio-Formats, overwrite the content of the `bfmatlab` folder with the newer version of the toolbox and restart MATLAB.

**Alternative scripts**

Several other groups have developed their own MATLAB scripts that use Bio-Formats, including the following:

- https://github.com/pramukta/bf-tools
- `imread` for multiple life science image file formats
2.6.5 VisAD

The VisAD visualization toolkit is a Java component library for interactive and collaborative visualization and analysis of numerical data. VisAD uses Bio-Formats to read many image formats, notably TIFF.

Installation

The visad.jar file has Bio-Formats bundled inside, so no further installation is necessary.

Upgrading

It should be possible to use a newer version of Bio-Formats by putting the latest bioformats_package.jar or formats-gpl.jar before visad.jar in the class path. Alternately, you can create a “VisAD Lite” using the make lite command from VisAD source, and use the resultant visad-lite.jar, which is a stripped down version of VisAD without sample applications or Bio-Formats bundled in.

2.6.6 R

OME does not currently provide an R implementation for Bio-Formats. However, there are options provided by the community:

Bioconductor RBioFormats

Bioconductor is a project to develop, support, and disseminate free open source software that facilitates rigorous and reproducible analysis of data from current and emerging biological assays. Most Bioconductor components are distributed as R packages, this includes RBioFormats. RBioFormats is an R package which interfaces the OME Bio-Formats Java library to allow reading of proprietary microscopy image data and metadata.

Installation

To install this package, start R (version “4.3”) and enter:

```r
if (!require("Bioconductor", quietly = TRUE))
  install.packages("Bioconductor")

# The following initializes usage of Biocon
devel
BiocManager::install(version='devel')

BiocManager::install("RBioFormats")
```
Documentation

To view documentation for the version of this package installed in your system, start R and enter:

```r
browseVignettes("RBioFormats")
```

See also:

RBioFormats
Website for the RBioFormats package

Using Bio-Formats in R
Section of the developer documentation describing the R wrapper for Bio-Formats used by RBioFormats

2.7 Visualization and analysis applications

2.7.1 Bitplane Imaris

Imaris is Bitplane’s core scientific software module that delivers all the necessary functionality for data visualization, analysis, segmentation and interpretation of 3D and 4D microscopy datasets. Combining speed, precision and ease-of-use, Imaris provides a complete set of features for working with three- and four-dimensional multi-channel images of any size, from a few megabytes to multiple gigabytes in size.

As of version 7.2, Imaris integrates with Fiji overview, which includes Bio-Formats. See this page for a detailed list of Imaris’ features.

2.7.2 CellProfiler

CellProfiler—developed by the Broad Institute Imaging Platform—is free open-source software designed to enable biologists without training in computer vision or programming to quantitatively measure phenotypes from thousands of images automatically. CellProfiler uses Bio-Formats to read images from disk, as well as write movies.

Installation

The CellProfiler distribution comes with Bio-Formats included, so no further installation is necessary.

Upgrading

It should be possible to use a newer version of Bio-Formats by replacing the bundled `bioformats_package.jar` with a newer version.

- For example, on Mac OS X, Ctrl+click the CellProfiler icon, choose Show Package Contents, and replace the following file:
  - `Contents/Resources/bioformats/jars/bioformats_package.jar`

See also:

CellProfiler
Website of the CellProfiler software

Using Bio-Formats in Python
Section of the developer documentation describing the Python wrapper for Bio-Formats used by CellProfiler
2.7.3 Comstat2

Comstat2 is a Java-based computer program for the analysis and treatment of biofilm images in 3D. It is the Master’s project of Martin Vorregaard.

Comstat2 uses the Bio-Formats Importer plugin for ImageJ to read files in TIFF and Leica LIF formats.

2.7.4 Endrov

Endrov (EV) is a multi-purpose image analysis program developed by the Thomas Burglin group at Karolinska Institute, Department of Biosciences and Nutrition.

Installation

The EV distribution comes bundled with the core Bio-Formats library (bio-formats.jar), so no further installation is necessary.

Upgrading

It should be possible to use a newer version of Bio-Formats by downloading the latest formats-gpl.jar and putting it into the libs folder of the EV distribution, overwriting the old file.

You could also include some optional libraries, to add support for additional formats, if desired.

2.7.5 FocalPoint

FocalPoint is an image browser, similar to Windows Explorer or other file manager application, specifically designed to work with more complex image types. FocalPoint uses Bio-Formats to generate thumbnails for some formats.

Installation

FocalPoint is bundled with Bio-Formats, so no further installation is necessary.

Upgrading

It should be possible to use a newer version of Bio-Formats by overwriting the old loci_tools.jar within the FocalPoint distribution. For Mac OS X, you will have to control click the FocalPoint program icon, choose “Show Package Contents” and navigate into Contents/Resources/Java to find the loci_tools.jar file.

2.7.6 Graphic Converter

Graphic Converter is a Mac OS application for opening, editing, and organizing photos. Versions 6.4.1 and later use Bio-Formats to open all file formats supported by Bio-Formats.
2.7.7 Icy

Icy is an open-source image analysis and visualization software package that combines a user-friendly graphical interface with the ability to write scripts and plugins that can be uploaded to a centralized website. It uses Bio-Formats internally to read images and acquisition metadata, so no further installation is necessary.

2.7.8 Iqm

Iqm is an image processing application written in Java. It is mainly constructed around the Java JAI library and furthermore it incorporates the functionality of the popular ImageJ image processing software.

Because qm integrates with ImageJ, it can take advantage of the Bio-Formats ImageJ plugin to read image data.

2.7.9 Micro-Manager

Micro-Manager is a software framework for implementing advanced and novel imaging procedures, extending functionality, customization and rapid development of specialized imaging applications.

Micro-Manager offers the functionality for saving the acquired images in TIFF/OM-E-TIFF format. Based on the mode of saving and the configuration settings, the acquired image can be saved with or without a companion file (*metadata.txt):
Bio-Formats, Release 6.14.0

<table>
<thead>
<tr>
<th>Micro-Manager saving option</th>
<th>Micro-Manager saving format</th>
<th>Companion file</th>
<th>Bio-Formats reading</th>
<th>Reader users</th>
</tr>
</thead>
<tbody>
<tr>
<td>Save as separate image files</td>
<td>TIFF</td>
<td>Yes</td>
<td>Full support</td>
<td>Micromanager-Reader</td>
</tr>
<tr>
<td>Save as image stack file</td>
<td>OME-TIFF</td>
<td>No</td>
<td>Pixel data plus minimal metadata*</td>
<td>OMETiffReader</td>
</tr>
<tr>
<td>Save as image stack file</td>
<td>OME-TIFF</td>
<td>Yes**</td>
<td>Full Support</td>
<td>Micromanager-Reader</td>
</tr>
</tbody>
</table>

* Not all acquisition metadata is converted to OME-XML.
** A small change in the acquisition side facilitates better handling of the metadata from the Bio-Formats side: Tools → Options… and then select “Create metadata.txt file with Image Stack Files” in the text box.

See also:
Micro-Manager User's Guide - Files on Disk

2.7.10 MIPAV

The MIPAV (Medical Image Processing, Analysis, and Visualization) application—developed at the Center for Information Technology at the National Institutes of Health—enables quantitative analysis and visualization of medical images of numerous modalities such as PET, MRI, CT, or microscopy. You can use Bio-Formats as a plugin for MIPAV to read images in the formats it supports.

Installation

Follow these steps to install the Bio-Formats plugin for MIPAV:

1. Download bioformats_package.jar and drop it into your MIPAV folder.
2. Download the plugin source code into your user mipav/plugins folder.
3. From the command line, compile the plugin with:
   ```
   cd mipav/plugins
   javac -cp $MIPAV:$MIPAV/bioformats\_package.jar \
   PlugInBioFormatsImporter.java
   ```
4. where $MIPAV is the location of your MIPAV installation.
5. Add bioformats_package.jar to MIPAV’s class path:
   - How to do so depends on your platform.
   - E.g., in Mac OS X, edit the mipav.app/Contents/Info.plist file.

See the readme file for more information.

To upgrade, just overwrite the old bioformats_package.jar with the latest one. You may want to download the latest version of MIPAV first, to take advantage of new features and bug-fixes.
2.7.11 QuPath

QuPath is an open-source application for whole slide image analysis and visualization. QuPath can be configured to use Bio-Formats by installing the QuPath Bio-Formats extension.

2.7.12 Vaa3D

Vaa3D, developed by the Peng Lab at the HHMI Janelia Farm Research Campus, is a handy, fast, and versatile 3D/4D/5D Image Visualization & Analysis System for Bioimages & Surface Objects.

Vaa3D can use Bio-Formats via the Bio-Formats C++ bindings to read images.

2.7.13 VisBio

VisBio is a biological visualization tool designed for easy visualization and analysis of multidimensional image data. VisBio uses Bio-Formats to import files as the Bio-Formats library originally grew out of our efforts to continually expand the file format support within VisBio.

Installation

VisBio is bundled with Bio-Formats, so no further installation is necessary.

Upgrading

It should be possible to use a newer version of Bio-Formats by overwriting the old bio-formats.jar and optional libraries within the VisBio distribution. For Mac OS X, you will have to control click the VisBio program icon, choose “Show Package Contents” and navigate into Contents/Resources/Java to find the JAR files.

2.7.14 XuvTools

XuvTools is automated 3D stitching software for biomedical image data. As of release 1.8.0, XuvTools uses Bio-Formats to read image data.
The following sections describe various things that are useful to know when working with Bio-Formats. It is recommended that you obtain the Bio-Formats source by following the directions in the Source code section. Referring to the Javadocs as you read over these pages should help, as the notes will make more sense when you see the API.

For a complete list of supported formats, see the Bio-Formats supported formats table.

For a few working examples of how to use Bio-Formats, see these Github pages and the bio-formats-examples repository.

3.1 Introduction to Bio-Formats

3.1.1 Overview for developers

From the rest of the Bio-Formats developer documentation one may piece together a correct and useful understanding of what Bio-Formats does and how it does it. This section gives a high-level tour of these technical details, for those new to working on Bio-Formats itself, making it easier to understand how the information from the other sections fits into the big picture.

Terms and concepts

Bio-Formats can read image data from files for many formats, and can write image data to files for some formats. An image may have many two-dimensional “planes” of pixel intensity values. Each pixel on a plane is identified by its \( x \), \( y \) values. Planes within an image may be identified by various dimensions including \( z \) (third spatial dimension), \( c \) (channel, e.g. wavelength) or \( t \) (time). Planes may be divided into tiles, which are rectangular subsections of a plane; this is helpful in handling very large planes. A file (or set of related files) on disk may contain multiple images: each image is identified by a unique series number.

An image is more than a set of planes: it also has metadata. Bio-Formats distinguishes core metadata, such as the \( x \), \( y \), \( z \), \( c \), \( t \) dimensions of the image, from format-specific original metadata, e.g. information about the microscope and its settings, which is represented as a dictionary of values indexed by unique keys. Metadata apply to the image data as a whole, or separately to specific series within it.

Bio-Formats is able to translate the above metadata into a further form, OME metadata. The translation may be partial or incomplete, but remains very useful for allowing the metadata of images from different file formats to be used and compared in a common format defined by the OME data model.
Bio-Formats, Release 6.14.0

Implementation

Bio-Formats is primarily a Java project. It can be used from MATLAB and there is also a separate C++ implementation (OME Files C++). The source code is available for download and sometimes the user community contributes code back into Bio-Formats by opening a pull request on GitHub. Bio-Formats is built from source with Ant or Maven and some of the Bio-Formats source code is generated from other files during the build process. The resulting JARs corresponding to official Bio-Formats releases are available for download.

Readers and writers for different image file formats are implemented in separate Java classes. Readers for related formats may reflect that relationship in the Java class hierarchy. Simple standalone command-line tools are provided with Bio-Formats, but it is more commonly used as a third-party library by other applications. Various examples show how one may use Bio-Formats in different ways in writing a new application that reads or writes image data. A common pattern is to initialize a reader based on the image data’s primary file, then query that reader for the metadata and planes of interest.

The set of readers is easily modified. The readers.txt file lists the readers to try in determining an image file’s format, and there are many useful classes and methods among the Bio-Formats Java code to assist in writing new readers and writers.

3.1.2 Obtaining and building Bio-Formats

Note: Bio-Formats requires Java 8 or above

Source code

The source code for this Bio-Formats release is available from the downloads site. This release and the latest Bio-Formats source code are also available from the Git repository. This may be accessed using the repository path:

```
git@github.com:openmicroscopy/bioformats.git
```

More information about Git and client downloads are available from the Git project website. You can also browse the Bio-Formats source on GitHub

Note: Windows users must set git to use `core.autocrlf=input` to ensure that Bio-Formats uses LF rather than CRLF line endings, otherwise the build will fail (Genshi can't process code templates with CRLF line endings, leading to broken sources being generated). This can be set globally in the registry when installing msysgit or by editing `etc/gitconfig` in the git installation directory. Annoyingly, these settings appear to override per-user and per-repository configuration values, requiring these to be set globally.

Lastly, you can browse the Bio-Formats Javadocs online, or generate them yourself using the “docs” Ant target.
Source code structure

The Bio-Formats code is divided into several projects. Core components are located in subfolders of the components folder, with some components further classified into components/forks, depending on the nature of the project. See the Component overview for more information, including associated build targets for each component.

Each project has a corresponding Maven POM file, which can be used to work with the project in your favorite IDE, or from the command line, once you have cloned the source.

Building from source

Instructions for several popular options follow. In all cases, make sure that the prerequisites are installed before you begin.

If you are interested in working on the Bio-Formats source code itself, you can load it into your favorite IDE, or develop with your favorite text editor.

NetBeans

NetBeans comes with Maven support built in. To import the Bio-Formats source, perform the following steps:

1. Select File → Open Project from the menu - choose the top-level path to bioformats.git and click Open Project
2. In the ‘Projects’ tab on the left-hand side, expand the ‘Bio-Formats projects’ entry - you should now have a series of folders including ‘Other Sources’, ‘Modules’ and ‘Dependencies’.
3. Expand the ‘Modules’ folder to give a list of components and then double-click the desired project(s) to work with them.

Alternately, you can clone the source directly from NetBeans into a project by selecting Team → Git → Clone Other… from the menu.
Eclipse

Eclipse uses the “Maven Integration for Eclipse” (m2e) plugin to work with Maven projects. It is more flexible than Eclipse’s built-in project management because m2e transparently converts between project dependencies and JAR dependencies (stored in the Maven repository in ~/.m2/repository) on the build path, depending on which projects are currently open.

We recommend using Eclipse 4.3 (Kepler) or later, specifically - “Eclipse IDE for Java developers”. It comes with m2e installed (http://eclipse.org/downloads/compare.php?release=kepler).

You can import the Bio-Formats source by choosing File → Import → Existing Maven Projects from the menu and browsing to the top-level folder of your Bio-Formats working copy. Alternatively, run the Eclipse Maven target with mvn eclipse:eclipse to create the Eclipse project files, then use File → Import → Existing Maven Projects.

IntelliJ IDEA

IntelliJ IDEA can build Bio-Formats using either Maven or Ant. Go to File → Open... and browse to the top-level folder of your Bio-Formats working copy. Select Build → Build Project or select the Maven tab on the right of the screen.

Command line

If you prefer developing code with a text editor such as vim or emacs, you can use the Ant or Maven command line tools to compile Bio-Formats. The Bio-Formats source tree provides parallel build systems for both Ant and Maven, so you can use either one to build the code.

For a list of Ant targets, run:

```
ant -p
```

In general, ant jars or ant tools is the correct command.

When using Maven, Bio-Formats is configured to run the “install” target by default, so all JARs will be copied into your local Maven repository in ~/.m2/repository. Simply run:

```
mvn
```

With either Ant or Maven, you can use similar commands in any subproject folder to build just that component.

3.1.3 Component overview

The Bio-Formats code repository is divided up into separate components.

The Ant targets to build each component from the repository root are noted in the component descriptions below. Unless otherwise noted, each component can also be built with Maven by running mvn in the component’s subdirectory. The Maven module name for each component (as it is shown in most IDEs) is also noted in parenthesis.
Core components

The most commonly used and actively modified components.

- formats-api
- formats-bsd
- formats-gpl

Internal testing components

These components are used heavily during continuous integration testing, but are less relevant for active development work.

- test-suite

Forks of existing projects

- jai
- turbojpeg

All components

bio-formats-plugins (Bio-Formats Plugins for ImageJ):

Ant: jar-bio-formats-plugins

Everything pertaining to the Bio-Formats plugins for ImageJ lives in this component. Note that when built, this component produces bio-formats_plugins.jar (instead of bio-formats-plugins.jar) to be in keeping with ImageJ plugin naming conventions.

bio-formats-tools (Bio-Formats command line tools):

Ant: jar-bio-formats-tools

The classes that implement the showinf, bfconvert, and mktake command line tools are contained in this component. Note that this is built with the jar-bio-formats-tools Ant target, and not the tools target (which is the Ant equivalent of bundles).

bundles (bioformats_package bundle, LOCI Tools bundle):

Ant: tools

This is only needed by the Maven build system, and is used to aggregate all of the individual .jar files into bioformats_package.jar. There should not be any code here, just build system files.

OME JAI (deprecated):

This is a fork of JAI ImageIO. JAI ImageIO is no longer maintained; the most active fork is jai-imageio-core on GitHub. JAI provides support for decoding YCbCr JPEG-2000 data. This is primarily needed for reading images from histology/pathology formats in formats-gpl. There are no dependencies on other components.

The status of this component means that you may encounter errors due to conflicting JARs e.g. between Bio-Formats and other toolboxes within Fiji or MATLAB, especially when trying to open JPEG-2000 data. In this case, you will need to remove the conflicting JAR(s) as a workaround.

forks/turbojpeg (libjpeg-turbo Java bindings):

Ant: jar-turbojpeg

3.1. Introduction to Bio-Formats
This is a fork of libjpeg-turbo. There are not any real code changes, but having this as a separate component allows us to package the libjpeg-turbo Java API together with all of the required binaries into a single .jar file using native-lib-loader. There are no dependencies on other components.

**formats-api (Bio-Formats API):**

*Ant: jar-formats-api*

This defines all of the high level interfaces and abstract classes for reading and writing files. There are no file format readers or writers actually implemented in this component, but it does contain the majority of the API that defines Bio-Formats. `formats-bsd` and `formats-gpl` implement this API to provide file format readers and writers. `ome-common` and `ome-xml` are both required as part of the interface definitions.

**formats-bsd (BSD Bio-Formats readers and writers):**

*Ant: jar-formats-bsd*

This contains readers and writers for formats which have a publicly available specification, e.g. TIFF. Everything in the component is BSD-licensed.

**formats-gpl (Bio-Formats library):**

*Ant: jar-formats-gpl*

The majority of the file format readers and some file format writers are contained in this component. Everything in the component is GPL-licensed (in contrast with `formats-bsd`). Most file formats represented in this component do not have a publicly available specification.

**test-suite (Bio-Formats testing framework):**

*Ant: jar-tests*

All tests that operate on files from our data repository (i.e. integration tests) are included in this component. These tests are primarily run by the continuous integration jobs, and verify that there are no regressions in reading images or metadata.

**External components**

The following have been decoupled from the Bio-Formats code repository and are now available as separate build dependencies:

- Bio-Formats examples
- Bio-Formats documentation
- Metakit
- OME Common
- OME Codecs
- OME MDB Tools (Java)
- OME Apache Jakarta POI
- JXRlib

Decoupled OME data model components:

- OME-XML
- Specification
Bio-Formats examples

Usage examples for Bio-Formats with Maven and Gradle.

Bio-Formats documentation

The Sphinx source repository for this manual.

OME Common (Java / C++):

Provides I/O classes that unify reading from files on disk, streams or files in memory, compressed streams, and non-file URLs. The primary entry points are Location, RandomAccessInputStream (for reading), and RandomAccessOutputStream (for writing).

In addition to I/O, there are several classes to assist in working with XML (XMLTools), date/timestamps (DateTools), logging configuration (DebugTools), and byte arithmetic (DataTools).

OME Codecs:

Provides classes for encoding and decoding compressed data for a variety of compression formats. *ome-common* is a required dependency for I/O and service loading.

OME MDB Tools (Java port):

This is a fork of the mdbtools-java project. There are numerous bug fixes, as well as changes to reduce the memory required for large files. There are no dependencies on other components.

OME Apache Jakarta POI:

This is a fork of Apache POI, which allows reading of Microsoft OLE document files. We have made substantial changes to support files larger than 2GB and reduce the amount of memory required to open a file. I/O is also handled by classes from *ome-common*, which allows OLE files to be read from memory.

Metakit Java library:

Java implementation of the Metakit database specification. This uses classes from *ome-common* and is used by *formats-gpl*, but is otherwise independent of the main Bio-Formats API.

OME-XML Java library:

This component contains classes that represent the OME-XML schema. Some classes are committed to the Git repository, but the majority are generated at build time by using XSD-FU to parse the *OME-XML schema files*. Classes from this component are used by Bio-Formats to read and write OME-XML, but they can also be used independently.

Model specification:

All released and in-progress OME-XML schema files are contained in this component. The specification component is also the location of all XSLT stylesheets for converting between OME-XML schema versions, as well as example OME-XML files in each of the released schema versions.

Stubs:

Luratech LuraWave stubs and MIPAV stubs.

This component provides empty classes that mirror third-party dependencies which are required at compile time but cannot be included in the build system (usually due to licensing issues). The build succeeds since required class names are present with the correct method signatures; the end user is then expected to replace the stub .jar files at runtime.

JXRlib:

This component contains the Java bindings for jxrlib, an open source implementation of the JPEG-XR image format standard.

3.1. Introduction to Bio-Formats
3.1.4 Reading files

Basic file reading

Bio-Formats provides several methods for retrieving data from files in an arbitrary (supported) format. These methods fall into three categories: raw pixels, core metadata, and format-specific metadata. All methods described here are present and documented in loci.formats.IFormatReader. In general, it is recommended that you read files using an instance of loci.formats.ImageReader. While it is possible to work with readers for a specific format, ImageReader contains additional logic to automatically detect the format of a file and delegate subsequent calls to the appropriate reader.

Prior to retrieving pixels or metadata, it is necessary to call `setFile(java.lang.String)` on the reader instance, passing in the name of the file to read. Some formats allow multiple series (5D image stacks) per file; in this case you may wish to call `setSeries(int)` to change which series is being read.

Raw pixels are always retrieved one plane at a time. Planes are returned as raw byte arrays, using one of the `openBytes` methods.

Core metadata is the general term for anything that might be needed to work with the planes in a file. A list of core metadata fields is given in the table below together with the appropriate accessor method:

<table>
<thead>
<tr>
<th>Core metadata field</th>
<th>API method</th>
</tr>
</thead>
<tbody>
<tr>
<td>image width</td>
<td>getSizeX()</td>
</tr>
<tr>
<td>image height</td>
<td>getSizeY()</td>
</tr>
<tr>
<td>number of series per file</td>
<td>getSeriesCount()</td>
</tr>
<tr>
<td>total number of images per series</td>
<td>getImageCount()</td>
</tr>
<tr>
<td>number of slices in the current series</td>
<td>getSizeZ()</td>
</tr>
<tr>
<td>number of timepoints in the current series</td>
<td>getSizeT()</td>
</tr>
<tr>
<td>number of actual channels in the current series</td>
<td>getSizeC()</td>
</tr>
<tr>
<td>number of channels per image</td>
<td>getRGBChannelCount()</td>
</tr>
<tr>
<td>the ordering of the images within the current series</td>
<td>getDimensionOrder()</td>
</tr>
<tr>
<td>whether each image is RGB</td>
<td>isRGB()</td>
</tr>
<tr>
<td>whether the pixel bytes are in little-endian order</td>
<td>isLittleEndian()</td>
</tr>
<tr>
<td>whether the channels in an image are interleaved</td>
<td>isInterleaved()</td>
</tr>
<tr>
<td>the type of pixel data in this file</td>
<td>getPixelType()</td>
</tr>
</tbody>
</table>

All file formats are guaranteed to accurately report core metadata.

Bio-Formats also converts and stores additional information which can be stored and retrieved from the OME-XML Metadata. These fields can be accessed in a similar way to the core metadata above. An example of such values would be the physical size of dimensions X, Y and Z. The accessor methods for these properties return a `Length` object which contains both the value and unit of the dimension. These lengths can also be converted to other units using `value(ome.units.unit.Unit)` An example of reading and converting these physical sizes values can be found in ReadPhysicalSize.java.

Format-specific metadata refers to any other data specified in the file - this includes acquisition and hardware parameters, among other things. This data is stored internally in a `java.util.Hashtable`, and can be accessed in one of two ways: individual values can be retrieved by calling `getMetadataValue(java.lang.String)`, which gets the value of the specified key. Note that the keys in this Hashtable are different for each format, hence the name “format-specific metadata”.

See Bio-Formats metadata processing for more information on the metadata capabilities that Bio-Formats provides.

See also:

IFormatReader

Source code of the loci.formats.IFormatReader interface
OrthogonalReader.java
Example of reading XZ and YZ image planes from a file

File reading extras

The previous section described how to read pixels as they are stored in the file. However, the native format is not necessarily convenient, so Bio-Formats provides a few extras to make file reading more flexible.

- The loci.formats.ReaderWrapper API that implements loci.formats.IFormatReader allows to define “wrapper” readers that take a reader in the constructor, and manipulate the results somehow, for convenience. Using them is similar to the java.io InputStream/OutputStream model: just layer whichever functionality you need by nesting the wrappers.

The table below summarizes a few wrapper readers of interest:

<table>
<thead>
<tr>
<th>Wrapper reader</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>loci.formats.BufferedImageReader</td>
<td>Allows pixel data to be returned as BufferedImages instead of raw byte arrays</td>
</tr>
<tr>
<td>loci.formats.FileStitcher</td>
<td>Uses advanced pattern matching heuristics to group files that belong to the same dataset</td>
</tr>
<tr>
<td>loci.formats.ChannelSeparator</td>
<td>Makes sure that all planes are grayscale - RGB images are split into 3 separate grayscale images</td>
</tr>
<tr>
<td>loci.formats.ChannelMerger</td>
<td>Merges grayscale images to RGB if the number of channels is greater than 1</td>
</tr>
<tr>
<td>loci.formats.ChannelFiller</td>
<td>Converts indexed color images to RGB images</td>
</tr>
<tr>
<td>loci.formats.MinMaxCalculator</td>
<td>Provides an API for retrieving the minimum and maximum pixel values for each channel</td>
</tr>
<tr>
<td>loci.formats.DimensionSwapper</td>
<td>Provides an API for changing the dimension order of a file</td>
</tr>
<tr>
<td>loci.formats.Memoizer</td>
<td>Caches the state of the reader into a memoization file</td>
</tr>
</tbody>
</table>

- loci.formats.ImageTools and loci.formats.gui.AWTImageTools provide a number of methods for manipulating BufferedImages and primitive type arrays. In particular, there are methods to split and merge channels in a BufferedImage/array, as well as converting to a specific data type (e.g. convert short data to byte data).

Troubleshooting

- Importing multi-file formats (Leica LEI, PerkinElmer, FV1000 OIF, ICS, and Prairie TIFF, to name a few) can fail if any of the files are renamed. There are “best guess” heuristics in these readers, but they are not guaranteed to work in general. So please do not rename files in these formats.

- If you are working on a Macintosh, make sure that the data and resource forks of your image files are stored together. Bio-Formats does not handle separated forks (the native QuickTime reader tries, but usually fails).

- Bio-Formats file readers are not thread-safe. If files are read within a parallelized environment, a new reader must be fully initialized in each parallel session. See Improving reading performance about ways to improve file reading performance in multi-threaded mode.
3.1.5 Writing files

The loci.formats.IFormatWriter API is very similar to the reader API, in that files are written one plane at time (rather than all at once).

The file formats which can be written using Bio-Formats are marked in the supported formats table with a green tick in the 'export' column. These include, but are not limited to:

- TIFF (uncompressed, LZW, JPEG, or JPEG-2000)
- OME-TIFF (uncompressed, LZW, JPEG, or JPEG-2000)
- JPEG
- PNG
- AVI (uncompressed)
- QuickTime (uncompressed is supported natively; additional codecs use QTJava)
- Encapsulated PostScript (EPS)
- OME-XML (not recommended)

All writers allow the output file to be changed before the last plane has been written. This allows you to write to any number of output files using the same writer and output settings (compression, frames per second, etc.), and is especially useful for formats that do not support multiple images per file.

See also:

IFormatWriter
Source code of the loci.formats.IFormatWriter interface

loci.formats.tools.ImageConverter
Source code of the loci.formats.tools.ImageConverter class

Further details on exporting raw pixel data to OME-TIFF files
Examples of OME-TIFF writing

3.2 Using Bio-Formats as a Java library

3.2.1 Using Bio-Formats as a Java library

Bio-Formats as a Maven, Gradle or Ivy dependency

All released .jar artifacts are published to and can be obtained from the OME Artifactory. The “Client Settings” section of the Artifactory main page provides example code snippets for inclusion into your Gradle, Maven or Ivy project, which will enable the use of this repository.

Changed in version 6.12.0.

As of Bio-Formats 6.12.0 consumers will have to directly declare a dependency for their chosen SLF4J logging framework. See Logging for further details.

Examples of getting started with Bio-Formats using Maven or Gradle are given in the https://github.com/ome/bio-formats-examples repository. OMERO uses Ivy to manage its Java dependencies including Bio-Formats.

Note: In order to retrieve the NetCDF dependency of Bio-Formats, it is necessary to configure the Unidata releases repository in addition to Maven Central and the OME artifactory in your https://github.com/ome/
Bio-Formats as a Java library

Alternatively Bio-Formats can be used by including its component jar files. You can download formats-gpl.jar to use it as a library. Just add formats-gpl.jar to your CLASSPATH or build path. You will also need ome-common.jar for common I/O functions, ome-xml.jar for metadata standardization, and a SLF4J binding for Logging.

Dependencies

The complete list of third-party dependencies for formats-gpl.jar is as follows:

<table>
<thead>
<tr>
<th>Package</th>
<th>Maven name</th>
</tr>
</thead>
<tbody>
<tr>
<td>JHDF5 v19.04.0</td>
<td>cisld:jhdf5:19.04.0</td>
</tr>
<tr>
<td>cisld base v18.09.0</td>
<td>cisld:base:18.09.0</td>
</tr>
<tr>
<td>commons-io v2.7</td>
<td>commons-io:commons-io:2.7</td>
</tr>
<tr>
<td>commons-lang3 v3.10</td>
<td>org.apache.commons:commons-lang3:3.10</td>
</tr>
<tr>
<td>XMP Library for Java v6.1.11</td>
<td>com.adobe.xmp:xmpcore:6.1.11</td>
</tr>
<tr>
<td>JCommander v1.27</td>
<td>com.beust:jcommander:1.27</td>
</tr>
<tr>
<td>metadata-extractor v2.18.0</td>
<td>com.drewnoakes:metadata-extractor:2.18.0</td>
</tr>
<tr>
<td>aircompressor v0.21</td>
<td>io.airlift:aircompressor:0.21</td>
</tr>
<tr>
<td>Kryo v4.0.2</td>
<td>com.esotericsoftware.kryo:kryo:4.0.2</td>
</tr>
<tr>
<td>reflectasm v1.11.3</td>
<td>com.esotericsoftware:reflectasm:1.11.3</td>
</tr>
<tr>
<td>asm v5.0.4</td>
<td>org.ow2.asm:asm:5.0.4</td>
</tr>
<tr>
<td>MinLog v1.3</td>
<td>com.esotericsoftware.minlog:minlog:1.3</td>
</tr>
<tr>
<td>Guava v29.0</td>
<td>com.google.guava:guava:29.0-jre</td>
</tr>
<tr>
<td>JGoodies Common v1.7.0</td>
<td>com.jgoodies:jgoodies-common:1.7.0</td>
</tr>
<tr>
<td>JGoodies Forms v1.7.2</td>
<td>com.jgoodies:jgoodies-forms:1.7.2</td>
</tr>
<tr>
<td>Commons Lang v2.6</td>
<td>commons-lang:commons-lang:2.6</td>
</tr>
<tr>
<td>Commons Logging v1.2</td>
<td>commons-logging:commons-logging:1.2</td>
</tr>
<tr>
<td>NetCDF-Java Library v5.3.3</td>
<td>edu.ucar:cdm:5.3.3</td>
</tr>
<tr>
<td>Joda time v2.2</td>
<td>joda-time:joda-time:2.2</td>
</tr>
<tr>
<td>JUnit v4.10</td>
<td>junit:junit:4.10</td>
</tr>
<tr>
<td>BeanShell v2.0b4</td>
<td>org.beanshell:bsh:2.0b4</td>
</tr>
<tr>
<td>Hamcrest Core v1.2.1</td>
<td>org.jgoodies:hamcrest-core:1.2.1</td>
</tr>
<tr>
<td>Objenesis v2.5.1</td>
<td>org.objenesis:objenesis:2.5.1</td>
</tr>
<tr>
<td>Perf4J v0.9.16</td>
<td>org.perf4j:perf4j:0.9.16</td>
</tr>
<tr>
<td>Native Library Loader v2.1.4</td>
<td>org.scijava:native-lib-loader:2.1.4</td>
</tr>
<tr>
<td>SLF4J API v1.7.30</td>
<td>org.slf4j:slf4j-api:1.7.30</td>
</tr>
<tr>
<td>TestNG v6.8</td>
<td>org.testng:testng:6.8</td>
</tr>
<tr>
<td>SnakeYAML v1.32</td>
<td>org.yaml:snakeyaml:1.32</td>
</tr>
<tr>
<td>Woolz v1.4.0</td>
<td>woolz:JWlz:1.4.0</td>
</tr>
<tr>
<td>Xalan Java Serializer v2.7.2</td>
<td>xalan:serializer:2.7.2</td>
</tr>
<tr>
<td>Xalan Java v2.7.2</td>
<td>xalan:xalan:2.7.2</td>
</tr>
<tr>
<td>Xerces2 Java Parser v2.12.2</td>
<td>xerces:xercesImpl:2.12.2</td>
</tr>
<tr>
<td>XML Commons External Components XML APIs v1.4.01</td>
<td>xml-apis:xml-apis:1.4.01</td>
</tr>
<tr>
<td>minio v5.0.2</td>
<td>io.minio:minio:5.0.2</td>
</tr>
</tbody>
</table>

3.2. Using Bio-Formats as a Java library
### Table 1 – continued from previous page

<table>
<thead>
<tr>
<th>Package</th>
<th>Maven name</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>http-client v1.20.0</td>
<td>com.google.http-client:google-http-client:1.20.0</td>
<td></td>
</tr>
<tr>
<td>xpp3 v1.1.4c</td>
<td>xpp3:xpp3:1.1.4c</td>
<td></td>
</tr>
<tr>
<td>okhttp3 v3.7.0</td>
<td>com.squareup.okhttp3:okhttp3:3.7.0</td>
<td></td>
</tr>
<tr>
<td>okio v1.12.0</td>
<td>com.squareup.okio:okio:1.12.0</td>
<td></td>
</tr>
<tr>
<td>jaxb v2.3.0</td>
<td>java.xml.bind:jaxb-api:2.3.0</td>
<td></td>
</tr>
<tr>
<td>json v20090211</td>
<td>org.json:json:20090211</td>
<td></td>
</tr>
<tr>
<td>sqlite v3.28.0</td>
<td>org.xerial.sqlite-jdbc:3.28.0</td>
<td></td>
</tr>
<tr>
<td>failureaccess v1.0.1</td>
<td>com.google.guava:failureaccess:1.0.1</td>
<td></td>
</tr>
<tr>
<td>listenablefuture v9999.0</td>
<td>com.google.guava:listenablefuture:9999.0-*</td>
<td></td>
</tr>
<tr>
<td>jsr305 v3.0.2</td>
<td>com.google.code.findbugs:jsr305:3.0.2</td>
<td></td>
</tr>
<tr>
<td>checker-qual v2.11.1</td>
<td>org.checkerframework:checker-qual:2.11.1</td>
<td></td>
</tr>
<tr>
<td>errorprone v2.3.4</td>
<td>com.google.errorprone:error_prone_annotations:2.3.4</td>
<td></td>
</tr>
<tr>
<td>j2objc-annotations v1.3</td>
<td>com.google.j2objc:j2objc-annotations:1.3</td>
<td></td>
</tr>
<tr>
<td>httpservices v5.3.3</td>
<td>edu.ucar:httpservices:5.3.3</td>
<td></td>
</tr>
<tr>
<td>httpclient v4.5.9</td>
<td>org.apache.httpcomponents:httpclient:4.5.9</td>
<td></td>
</tr>
<tr>
<td>commons-codec v1.11</td>
<td>commons-codec:commons-codec:1.11</td>
<td></td>
</tr>
<tr>
<td>httpmime v4.5.9</td>
<td>org.apache.httpcomponents:httpmime:4.5.9</td>
<td></td>
</tr>
<tr>
<td>c3p0 v0.9.5.3</td>
<td>com.mchange:c3p0:0.9.5.3</td>
<td></td>
</tr>
<tr>
<td>mchange-commons-java v0.2.15</td>
<td>com.mchange:mchange-commons-java:0.2.15</td>
<td></td>
</tr>
</tbody>
</table>

The below are optional dependencies:

<table>
<thead>
<tr>
<th>Package</th>
<th>Maven name</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logback Classic v1.2.9</td>
<td>ch.qos.logback:logback-classic:1.2.9</td>
<td>Eclipse Public License v1.0</td>
</tr>
<tr>
<td>Logback Core v1.2.9</td>
<td>ch.qos.logback:logback-core:1.2.9</td>
<td>Eclipse Public License v1.0</td>
</tr>
</tbody>
</table>

As described in Versioning policy, the minor version number of a Bio-Formats release will always be increased if the version of a non-OME/external dependency is bumped.

Dependency lists for any component can be generated by checking out the code as described in Source code and then running:

```
cd components/$COMPONENT_NAME
mvn dependency:tree
```

### Examples of usage

**File reading and performance:**

**MultiFileExample** - Simple example of how to open multiple files simultaneously.

**ParallelRead** - Reads all files in given directory in parallel, using a separate thread for each.

**ReadWriteInMemory** - Tests the Bio-Formats I/O logic to and from byte arrays in memory.

**OrthogonalReader** - Reads image data in XZ and YZ order.
File writing:

MinimumWriter - A command line utility demonstrating the minimum amount of metadata needed to write a file.
FileExport - Write a file in any supported output format.
TiledExport - Shows how to convert a file one tile at a time, instead of one plane at a time (needed for very large images).
FileExportSPW - Write a file with plate (OME SPW) metadata.

File compression:

makeLZW - Converts the given image file to an LZW-compressed TIFF.

Metadata extract/print:

GetPhysicalMetadata - Uses Bio-Formats to extract some basic standardized (format-independent) metadata.
ReadPhysicalSize - Reads physical size information uses the units API to display in micrometers.
ImageInfo - A more involved command line utility for thoroughly reading an input file, printing some information about it, and displaying the pixels onscreen using the Bio-Formats viewer.
PrintTimestamps - A command line example demonstrating how to extract timestamps from a file.
PrintLensNA - Uses Bio-Formats to extract lens numerical aperture in a format-independent manner from a dataset.
PrintROIs - A simple example of how to retrieve ROI data parsed from a file.
SubResolutionExample - Demonstration of the sub-resolution API.

Metadata add/edit:

EditImageName - Edits the given file’s image name (but does not save back to disk).
EditTiffComment - Allows raw user TIFF comment editing for the given TIFF files.
writeMapAnnotations - Example method to write MapAnnotations to the ome-xml.
CommentSurgery - Edits a TIFF ImageDescription comment, particularly the OME-XML comment found in OME-TIFF files.

Image converters:

ImageConverter - A simple command line tool for converting between formats.
FileConvert - Converts a file in any supported format to any supported output format.
ConvertToOmeTiff - Converts the given files to OME-TIFF format.
WritePreCompressedPlanes - Writes the pixels from a set of JPEG files to a single TIFF. The pixel data is used as-is, so no decompression or re-compression is performed.
GeneratePyramidResolutions - Convert a file containing a single large image to a pyramid OME-TIFF.
TiledReaderWriter - Convert a file to OME-TIFF one tile at a time.
OverlappedTiledWriter - Convert a file to OME-TIFF one tile at a time, when the image size is not a multiple of the tile size.

3.2. Using Bio-Formats as a Java library
SimpleTiledWriter - Convert a file to OME-TIFF using automatic tiling.

ImageJ plugins:

Simple_Read - A simple ImageJ plugin demonstrating how to use Bio-Formats to read files into ImageJ (see ImageJ overview).

Read_Image - An ImageJ plugin that uses Bio-Formats to build up an image stack, reading image planes one by one (see ImageJ overview).

Mass_Importer - A simple plugin for ImageJ that demonstrates how to open all image files in a directory using Bio-Formats, grouping files with similar names to avoiding opening the same dataset more than once (see ImageJ overview).

Image processing utilities:

SewTiffs - Stitches the first plane from a collection of TIFFs into a single file.

SumPlanes - Sums together the image planes from the given file, and saves the result to a 16-bit TIFF.

A Note on Java Web Start (bioformats_package.jar vs. formats-gpl.jar)

To use Bio-Formats with your Java Web Start application, we recommend using formats-gpl.jar rather than bioformats_package.jar—the latter is merely a bundle of formats-gpl.jar plus all its optional dependencies.

The bioformats_package.jar bundle is intended as a convenience (e.g. to simplify installation as an ImageJ plugin), but is by no means the only solution for developers. We recommend using formats-gpl.jar as a separate entity depending on your needs as a developer.

The bundle is quite large because we have added support for several formats that need large helper libraries (e.g. Imaris' HDF-based format). However, these additional libraries are optional; Bio-Formats has been coded using reflection so that it can both compile and run without them.

When deploying a JNLP-based application, using bioformats_package.jar directly is not the best approach, since every time Bio-Formats is updated, the server would need to feed another 15+ MB JAR file to the client. Rather, Web Start is a case where you should keep the JARs separate, since JNLP was designed to make management of JAR dependencies trivial for the end user. By keeping formats-gpl.jar and the optional dependencies separate, only a <1 MB JAR needs to be updated when formats-gpl.jar changes.

As a developer, you have the option of packaging formats-gpl.jar with as many or as few optional libraries as you wish, to cut down on file size as needed. You are free to make whatever kind of "stripped down" version you require. You could even build a custom formats-gpl.jar that excludes certain classes, if you like.

3.2.2 Units of measurement

Since Bio-Formats 5.1 and the adoption of the 2015-01 OME Data Model, the data model and the corresponding Bio-Formats model and metadata APIs have added support for units of measurement. Previously, the units for various properties such as the physical size of an image, stage position, confocal pinhole size, light wavelengths etc. were fixed in the model. This was however somewhat inflexible, and not appropriate for imaging modalities at widely different scales. The solution to this was to add a unit of measurement to each of these properties. The image size, for example, was previously specified to be stored in micrometers but may now be specified in any SI length unit of choice, or one of the supported non-SI length units. This permits the preservation of the unit used by a proprietary file format or used at acquisition time, for example nanometers, millimeters, meters, or inches or thousandths of an inch could be used instead.
At the OME-XML level, the properties continue to use the old attribute names. They are supplemented by an additional attribute with a Unit suffix, for example the PhysicalSizeX attribute and its companion PhysicalSizeXUnit attribute.

At the API level, two classes are used:

- **Unit<T>**: represents a unit system for a given dimension such as length, pressure or time.

- **Quantity**: represents a value and unit in a given unit system; this is subclassed for each of the supported dimensions such as Length, Pressure etc. For example the Length class could represent the value and unit of 5.3 µm and the Pressure class 956 mbar.

All of the model and metadata APIs pass Quantity objects in place of raw numerical values. Updating your code will require replacing the use of raw values with quantities. Where your code needs to deal with the quantity in a specific unit, for example µm, you will need to perform an explicit unit conversion to transform the value to the required unit.

The three situations you will need to deal with are:

- getting a quantity from a get method in the API
- converting a quantity to a desired unit
- setting a quantity with a set method in the API (possibly also requiring the creation of a quantity)

Examples of how to use units and quantities for these purposes are shown in the sections **Reading files** (ReadPhysicalSize example which uses getPixelsPhysicalSize and also demonstrates unit conversion) and **Further details on exporting raw pixel data to OME-TIFF files** (setPixelsPhysicalSize).

### 3.2.3 Exporting files using Bio-Formats

This guide pertains to version 4.2 and later.

**Basic conversion**

The first thing we need to do is set up a reader:

```java
// create a reader that will automatically handle any supported format
IFormatReader reader = new ImageReader();
// tell the reader where to store the metadata from the dataset
MetadataStore metadata;

try {
    ServiceFactory factory = new ServiceFactory();
    OMEXMLService service = factory.getInstance(OMEXMLService.class);
    metadata = service.createOMEXMLMetadata();
} catch (DependencyException exc) {
    throw new FormatException("Could not create OME-XML store.", exc);
} catch (ServiceException exc) {
    throw new FormatException("Could not create OME-XML store.", exc);
}
reader.setMetadataStore(metadata);
```

(continues on next page)
// initialize the dataset
reader.setId("/path/to/file");

Now, we set up our writer:

```java
// create a writer that will automatically handle any supported output format
IFormatWriter writer = new ImageWriter();
// give the writer a MetadataRetrieve object, which encapsulates all of the
// dimension information for the dataset (among many other things)
OMEXMLService service = factory.getInstance(OMEXMLService.class);
writer.setMetadataRetrieve(service.asRetrieve(reader.getMetadataStore()));
// initialize the writer
writer.setId("/path/to/output/file");
```

Note that the extension of the file name passed to ‘writer.setId(...)’ determines the file format of the exported file.

Now that everything is set up, we can start writing planes:

```java
for (int series=0; series<reader.getSeriesCount(); series++) {
    reader.setSeries(series);
    writer.setSeries(series);
    for (int image=0; image<reader.getImageCount(); image++) {
        writer.saveBytes(image, reader.openBytes(image));
    }
}
```

Finally, make sure to close both the reader and the writer. Failure to do so can cause:

- file handle leaks
- memory leaks
- truncated output files

Fortunately, closing the files is very easy:

```java
reader.close();
writer.close();
```

### Converting to multiple files

The recommended method of converting to multiple files is to use a single IFormatWriter, like so:

```java
// you should have set up a reader as in the first example
ImageWriter writer = new ImageWriter();
OMEXMLService service = factory.getInstance(OMEXMLService.class);
writer.setMetadataRetrieve(service.asRetrieve(reader.getMetadataStore()));
// replace this with your own filename definitions
// in this example, we're going to write half of the planes to one file
// and half of the planes to another file
String[] outputFiles =
    new String[] {"/path/to/file/1.tif", "/path/to/file/2.tif"};
writer.setId(outputFiles[0]);
```

(continues on next page)
The advantage here is that the relationship between the files is preserved when converting to formats that support multi-
file datasets internally (namely OME-TIFF). If you are only converting to graphics formats (e.g. JPEG, AVI, MOV),
then you could also use a separate IFormatWriter for each file, like this:

```java
OMEXMLService service = factory.getInstance(OMEXMLService.class);
// again, you should have set up a reader already
String[] outputFiles = new String[] {"/path/to/file/1.avi", "/path/to/file/2.avi"};
int planesPerFile = reader.getImageCount() / outputFiles.length;
for (int file=0; file<outputFiles.length; file++) {
    ImageWriter writer = new ImageWriter();
    writer.setMetadataRetrieve(service.asRetrieve(reader.getMetadataStore()));
    writer.setId(outputFiles[file]);
    for (int image=0; image<planesPerFile; image++) {
        int index = file * planesPerFile + image;
        writer.saveBytes(image, reader.openBytes(index));
    }
    writer.close();
}
```

**Known issues**

List of Trac tickets

### 3.2.4 Further details on exporting raw pixel data to OME-TIFF files

This document explains how to export pixel data to OME-TIFF using Bio-Formats version 4.2 and later.

Working example code is provided in `FileExport.java` - code from that class is referenced here in part. You will
need to have `bioformats_package.jar` in your Java CLASSPATH in order to compile `FileExport.java`.

The first thing that must happen is we must create the object that stores OME-XML metadata. This is done as follows:

```java
ServiceFactory factory = new ServiceFactory();
OMEXMLService service = factory.getInstance(OMEXMLService.class);
IMetadata omexml = service.createOMEXLMetadata();
```

The ‘omexml’ object can now be used in our code to store OME-XML metadata, and by the file format writer to retrieve
OME-XML metadata.
Now that we have somewhere to put metadata, we need to populate as much metadata as we can. The minimum amount of metadata required is:

- endianness of the pixel data
- the order in which dimensions are stored
- the bit depth of the pixel data
- the number of channels
- the number of timepoints
- the number of Z sections
- the width (in pixels) of an image
- the height (in pixels) of an image
- the number of samples per channel (3 for RGB images, 1 otherwise)

We populate that metadata as follows:

```java
omexml.setImageID("Image:0", 0);
omexml.setPixelsID("Pixels:0", 0);

// specify that the pixel data is stored in big-endian order
// replace 'TRUE' with 'FALSE' to specify little-endian order
omexml.setPixelsBinDataBigEndian(Boolean.TRUE, 0, 0);

omexml.setPixelsDimensionOrder(DimensionOrder.XYCZT, 0);
omexml.setPixelsType(PixelType.UINT16, 0);
omexml.setPixelsSizeX(new PositiveInteger(width), 0);
omexml.setPixelsSizeY(new PositiveInteger(height), 0);
omexml.setPixelsSizeZ(new PositiveInteger(zSectionCount), 0);
omexml.setPixelsSizeC(new PositiveInteger(channelCount * samplesPerChannel), 0);
omexml.setPixelsSizeT(new PositiveInteger(timepointCount), 0);

for (int channel=0; channel<channelCount; channel++) {
    omexml.setChannelID("Channel:0:" + channel, 0, channel);
    omexml.setChannelSamplesPerPixel(new PositiveInteger(samplesPerChannel), 0, channel);
}

Unit<Length> unit = UNITS.MICROMETER;
Length physicalSizeX = new Length(1.0, unit);
Length physicalSizeY = new Length(1.5, unit);
Length physicalSizeZ = new Length(2, unit);
omexml.setPixelsPhysicalSizeX(physicalSizeX, 0);
omexml.setPixelsPhysicalSizeY(physicalSizeY, 0);
omexml.setPixelsPhysicalSizeZ(physicalSizeZ, 0);
```

There is much more metadata that can be stored; please see the Javadoc for loci.formats.meta.MetadataStore for a complete list.

Now that we have defined all of the metadata, we need to create a file writer:

```java
ImageWriter writer = new ImageWriter();
```
Now we must associate the ‘omexml’ object with the file writer:

```java
writer.setMetadataRetrieve(omexml);
```

The writer now knows to retrieve any metadata that it needs from ‘omexml’.

We now tell the writer which file it should write to:

```java
writer.setId("output-file.ome.tiff");
```

It is critical that the filename given to the writer ends with ‘.ome.tiff’ or ‘.ome.tif’, as it is the filename extension that determines which format will be written.

Now that everything is set up, we can save the image data. This is done plane by plane, and we assume that the pixel data is stored in a 2D byte array ‘pixelData’:

```java
int sizeC = omexml.getPixelsSizeC(0).getValue();
int sizeZ = omexml.getPixelsSizeZ(0).getValue();
int sizeT = omexml.getPixelsSizeT(0).getValue();
int samplesPerChannel = omexml.getChannelSamplesPerPixel(0).getValue();
sizeC /= samplesPerChannel;
int imageCount = sizeC * sizeZ * sizeT;
for (int image=0; image<imageCount; image++) {
    writer.saveBytes(image, pixelData[image]);
}
```

Finally, we must tell the writer that we are finished, so that the output file can be properly closed:

```java
writer.close();
```

There should now be a complete OME-TIFF file at whichever path was specified above.

### 3.2.5 Tiled reading and writing in Bio-Formats

#### Reading tiled images

The reading of tiled images is straightforward and can be done in much the same way as reading a full image. In this case, to read an individual tile, we pass to the reader parameters for the x and y coordinates of the tile to read and the height and width of tile desired.

```java
byte[] tile = reader.openBytes(image, tileX, tileY, tileSizeX, tileSizeY);
```

For TIFF-based readers, if the image has been written using tiles, then the tile width and height used can be found as below. These values can then be used with the above command to read the correct tiles individually.

```java
IFD tileIFd = reader.getIFDs().get(0);
int tileHeight = tileIFd.getIFDIntValue(IFD.TILE_LENGTH);
int tileWidth = tileIFd.getIFDIntValue(IFD.TILE_WIDTH);
```
Introduction to tiled writing

Tiled writing is currently supported for TIFF-based formats. To set up an image writer to use tiling the following 2 API functions are provided:

```java
public int setTileSizeX(int tileSize) throws FormatException
public int setTileSizeY(int tileSize) throws FormatException
```

Each function takes in an integer parameter for the desired tile size. As not all tile sizes are supported, the image writer will round the requested value to the nearest supported tile size. The return value will contain the actual tiling size which will be used by the writer.

To find out the tiling size currently being used at any point there are 2 further API functions to get the current tile size for a writer. If tiling is not being used or is not supported then the full image height and width will be returned.

```java
public int getTileSizeX() throws FormatException
public int getTileSizeY() throws FormatException
```

The tiling parameters for writers must be set after the image metadata is set. An example of initializing a writer for tiling is shown below.

```java
// set up the writer and associate it with the output file
writer = new OMETiffWriter();
writer.setMetadataRetrieve(omexml);
writer.setInterleaved(reader.isInterleaved());

// set the tile size height and width for writing
this.tileSizeX = writer.setTileSizeX(tileSizeX);
this.tileSizeY = writer.setTileSizeY(tileSizeY);

writer.setId(outputFile);
```

Simple tiled writing

The simplest way to write a tiled image is to set the tiling parameters on your image writer as above and have the writer automatically handle the tiling. Once the tile sizes have been set you may simply read and write your image files as normal.

```java
// set up the writer and associate it with the output file
writer = new OMETiffWriter();
writer.setMetadataRetrieve(omexml);
writer.setInterleaved(reader.isInterleaved());

// set the tile size height and width for writing
this.tileSizeX = writer.setTileSizeX(tileSizeX);
this.tileSizeY = writer.setTileSizeY(tileSizeY);

writer.setId(outputFile);
```

Full working example code is provided in `SimpleTiledWriter.java` - code from that class is referenced here in part. You will need to have `bioformats_package.jar` in your Java CLASSPATH in order to compile `SimpleTiledWriter.java`. 
Reading and writing using tiling

For the most efficient reading and writing of tiles you may instead wish to read in and write out the individual image tiles one at a time.

To do this you can set up the reader and writer as in the previous example above. In this case, when setting the tile height and width used it is important to store the return values which will be the valid tile size used by the writer.

```java
// set the tile height and width and store the actual values used by the writer
int tileSizeX = writer.setTileSizeX(tileSizeX);
int tileSizeY = writer.setTileSizeY(tileSizeY);
```

This time for each image you must determined the number of tiles using the actual tile height and width being used.

```java
int width = reader.getSizeX();
int height = reader.getSizeY();

// Determined the number of tiles to read and write
int nXTiles = width / tileSizeX;
int nYTiles = height / tileSizeY;
if (nXTiles * tileSizeX != width) nXTiles++;
if (nYTiles * tileSizeY != height) nYTiles++;
```

Now each tile can be read and written individually.

```java
for (int y=0; y<nYTiles; y++) {
    for (int x=0; x<nXTiles; x++) {
        // The x and y coordinates for the current tile
        int tileX = x * tileSizeX;
        int tileY = y * tileSizeY;

        // Read tiles from the input file and write them to the output OME-Tiff
        buf = reader.openBytes(image, tileX, tileY, tileSizeX, tileSizeY);
        writer.saveBytes(image, buf, tileX, tileY, tileSizeX, tileSizeY);
    }
}
```

Full working example code is provided in TiledReaderWriter.java - code from that class is referenced here in part. You will need to have bioformats_package.jar in your Java CLASSPATH in order to compile TiledReaderWriter.java.

Tiles which overlap image size

It may not always be the case that the tile size divides evenly into the image height and width. In these scenarios in which the last column or row of tiles overlaps with the image boundary, a smaller tile is instead read or written for the final row or column. If a full sized tile with padding is written then a loci.formats.FormatException will be thrown for Invalid tile size.

To deal with this we can modify the previous tiled writing example to check if the tile size will overlap the image boundaries. If it will not overlap then the full tile size is used, if it does then the tile size for that tile is modified to reflect the partial tile.

```java
// If the last tile row or column overlaps the image size then only a partial tile
```
Bio-Formats, Release 6.14.0

(continued from previous page)

```java
// is read or written. The tile size used is adjusted to account for any overlap.
int effTileSizeX = (tileX + tileSizeX) < width ? tileSizeX : width - tileX;
int effTileSizeY = (tileY + tileSizeY) < height ? tileSizeY : height - tileY;

// Read tiles from the input file and write them to the output OME-Tiff
buf = reader.openBytes(image, tileX, tileY, effTileSizeX, effTileSizeY);
writer.saveBytes(image, buf, tileX, tileY, effTileSizeX, effTileSizeY);
```

Full working example code is provided in `OverlappedTiledWriter.java` - code from that class is referenced here in part. You will need to have `bioformats_package.jar` in your Java CLASSPATH in order to compile `OverlappedTiledWriter.java`.

### 3.2.6 Working with whole slide images

Bio-Formats supports many whole slide image formats, but effectively working with this type of data requires some extra considerations. Each file contains images representing one or more physical slides, with each slide typically stored at multiple resolutions. The width and height of a full resolution whole slide image often exceed 100,000 pixels, so the uncompressed image size may be several gigabytes. This means that only part of the full resolution image can be accessed at any given time.

JPEG or JPEG-2000 compression is typically used such that the size on disk is often less than 100MB. Most whole slide formats split each image into many small tiles of $1024 \times 1024$ pixels or smaller and compress each tile independently, though some (e.g. Hamamatsu ndpi) compress the whole image at once. Many supported whole slide formats are based upon TIFF, with vendor-specific extensions for metadata or tile storage. Notable exceptions include Zeiss CZI and cellSens VSI.

The original full resolution image and its resolutions are collectively referred to as an image pyramid. File formats which support image pyramids are noted by the **Pyramid** column in the supported formats table.

By default, `openBytes` will load from the full resolution image in the first pyramid stored in the file. Each resolution of each pyramid is stored as a separate series, and can be accessed by calling `setSeries` prior to retrieving pixel data.

There are additional API methods that can be used to make pyramids easier to work with. These can be enabled by calling `setFlattenedResolutions(false)` prior to `setId`.

After `setFlattenedResolutions(false)`, each series represents an entire image pyramid and not just a single resolution. Calling `setSeries(...)` then skips over all other resolutions in the same pyramid, to either the next pyramid (if multiple pyramids are stored), or the thumbnail or barcode image (if present). To access the smaller resolutions in the pyramid, use the `getResolutionCount()` and `setResolution(int)` methods.

Most formats only store one pyramid per fileset, but some (e.g. cellSens VSI) allow multiple pyramids. Almost all formats allow a thumbnail, slide overview, and/or slide barcode image. Bio-Formats always stores these images as separate series, after all of the pyramids. Be careful to check the pixel type for the extra images, as the type and channel count will often differ from that of the pyramid(s).

For an example of how to use the pyramid resolution API, see [https://github.com/ome/bio-formats-examples/blob/master/src/main/java/SubResolutionExample.java](https://github.com/ome/bio-formats-examples/blob/master/src/main/java/SubResolutionExample.java).

Bio-Formats also provides some visibility into how the tiles are stored via the `getOptimalTileWidth()` and `getOptimalTileHeight()` methods. This is a suggestion of the size of tiles to be passed to `openBytes(int, byte[], int, int, int, int)` in order to minimize the number of tile decompressions. In most cases, and especially for the largest resolution, the whole image can’t be loaded at once. The amount of memory allocated is not a factor in being able to load the whole image, as no more than 2GB of pixel data can be stored in a single byte array and most full resolution images will exceed this limit.
### 3.2.7 Pyramids in OME-TIFF

Bio-Formats 6.0.0 and later can read and write image pyramids in the **OME-TIFF** format. Reading OME-TIFF pyramids uses the same API as described above. Writing OME-TIFF pyramids requires the resolution dimensions to be specified in an `IPyramidStore` object. `GeneratePyramidResolutions` shows a simple example of how to do this.

The `bfconvert` command line tool will also automatically write pyramids if the input file has at least one pyramid, the output format is OME-TIFF, and the `bfconvert -noflat` option is specified.

### 3.2.8 Internal OMERO pyramid format

For files that contain very large images and are not in a format that supports pyramids, OMERO will generate its own image pyramid to improve visualization performance. Bio-Formats can read these generated pyramids, but cannot currently write them outside of OMERO. For details of how to read image pyramids with Bio-Formats, see **Working with whole slide images**.

OMERO handles pyramid generation automatically for files that do not already have a stored pyramid, use a supported pixel type, and have images that exceed a specific XY size. The default XY size threshold is 3192×3192, but this can be configured in OMERO if necessary. Common formats for which a pyramid will be generated include **Gatan DM3**, **MRC**, and **TIFF**. Dedicated whole slide imaging formats such as **SVS** typically contain their own image pyramid, in which case an OMERO pyramid will not be generated.

For further information, see the **OMERO pyramid specification**.

### 3.2.9 In-memory reading and writing in Bio-Formats

Bio-Formats readers and writers are traditionally used to handle image files from disk. However it is also possible to achieve reading and writing of files from in-memory sources. This is handled by mapping the in-memory data to a file location using `Location.mapFile()`.

```java
Location.mapFile(fileName, byteArrayHandle);
```

This file location is not created on disk but rather maps internally to the in-memory data provided.

#### Reading file from memory

In order for Bio-Formats to read a file from memory it must be available in a byte array. For this example an input file is read from disk into a byte array.

```java
// read in entire file
System.out.println("Reading file into memory from disk...");
File inputFile = new File(path);
int fileSize = (int) inputFile.length();
DataInputStream in = new DataInputStream(new FileInputStream(inputFile));
byte[] inBytes = new byte[fileSize];
in.readFully(inBytes);
System.out.println(fileSize + " bytes read.");
```

This data can now be handled by a Bio-Formats reader. This is achieved by providing a mapping from the in-memory data to a suitable filename which will be used by the reader. The filename used must have the same suffix as the original data type.
Once the in-memory data has been mapped to a suitable filename the data can be handled by the reader as normal.

Writing to memory

To use a writer to output to memory rather than an output file a similar process is required. First a mapping is created between a suitable output filename and the in-memory data.

The mapped filename can now be passed to initialize the writer as standard.

The data can then be written to memory using the same read and write loop which would normally be used to write a file to disk.

```java
byte[] plane = null;
for (int i=0; i<imageCount; i++) {
    if (plane == null) {
        // allow reader to allocate a new byte array
        plane = reader.openBytes(i);
    } else {
        // reuse previously allocated byte array
        reader.openBytes(i, plane);
    }
}
```
writer.saveBytes(i, plane);
    System.out.print(".");
}
reader.close();
writer.close();
System.out.println();
byte[] outBytes = outputFile.getBytes();
outputFile.close();

If desired the data written to memory can then be flushed to disk and written to an output file location.

// flush output byte array to disk
System.out.println();
System.out.println("Flushing image data to disk...");
File outFile = new File(fileName + ".ome.tif");
DataOutputStream out = new DataOutputStream(new FileOutputStream(outFile));
out.write(outBytes);
out.close();

See also:
ReadWriteInMemory.java - Full source code which is referenced here in part. You will need to have bioformats_package.jar in your Java CLASSPATH in order to compile ReadWriteInMemory.java.

3.2.10 Logging

Logging frameworks

Bio-Formats uses SLF4J as a logging API. SLF4J is a facade and needs to be bound to a logging framework at deployment time. Two underlying logging frameworks are currently supported by Bio-Formats. Bio-Formats does not impose any specific SLF4J binding on downstream consumers as per the recommended SLF4J best practices. Some recommended bindings are as follows:

• logback is the recommended framework and natively implements the SLF4J API. A logback version of 1.2.x is recommended as Bio-Formats has a hard dependency for slf4j-api 1.7.30. This means that logback versions 1.3.x and higher are not yet supported in Bio-Formats as they rely on slf4j-api 2.x.

• log4j is the other logging framework supported by Bio-Formats and is primarily used in the MATLAB environment.

Examples of declaring these dependencies using Maven or Gradle are given in the Bio-Formats examples repository.

Initialization

The DebugTools class contains a series of framework-agnostic methods for the initialization and control of the logging system. This class uses reflection to detect the underlying logging framework and delegate the method calls to either Log4jTools or LogbackTools.

The main methods are described below:

• DebugTools.enableLogging() will initialize the underlying logging framework. This call will result in a no-op if logging has been initialized either via a binding-specific configuration file (see logback configuration) or via a prior call to DebugTools.enableLogging().
Bio-Formats, Release 6.14.0

• **DebugTools.enableLogging**(level) will initialize the logging framework under the same conditions as described above and set the root logger level if the initialization was successful.

• **DebugTools.setRootLevel**(level) will override the level of the root logger independently of how the logging system was initialized.

• **DebugTools.enableIJLogging**(logback-only) will add an ImageJ-specific appender to the root logger.

Changed in version 5.2.0: Prior to Bio-Formats 5.2.0, **DebugTools.enableLogging**(level) unconditionally set the logging and root logger level. Use **DebugTools.setRootLevel**(level) to restore this behavior.

### 3.2.11 Converting files from FV1000 OIB/OIF to OME-TIFF

This document explains how to convert a file from FV1000 OIB/OIF to OME-TIFF using Bio-Formats version 4.2 and later.

Working example code is provided in `FileConvert.java` - code from that class is referenced here in part. You will need to have `bioformats_package.jar` in your Java CLASSPATH in order to compile `FileConvert.java`.

The first thing that must happen is we must create the object that stores OME-XML metadata. This is done as follows:

```java
ServiceFactory factory = new ServiceFactory();
OMEXMLService service = factory.getInstance(OMEXMLService.class);
IMetadata omexml = service.createOMEXMLMetadata();
```

The `omexml` object can now be used by both a file format reader and a file format writer for storing and retrieving OME-XML metadata.

Now that have somewhere to put metadata, we need to create a file reader and writer:

```java
ImageReader reader = new ImageReader();
ImageWriter writer = new ImageWriter();
```

Now we must associate the `omexml` object with the file reader and writer:

```java
reader.setMetadataStore(omexml);
writer.setMetadataRetrieve(omexml);
```

The reader now knows to store all of the metadata that it parses into `omexml`, and the writer knows to retrieve any metadata that it needs from `omexml`.

We now tell the reader and writer which files will be read from and written to, respectively:

```java
reader.setId("input-file.oib");
writer.setId("output-file.ome.tiff");
```

It is critical that the filename given to the writer ends with ".ome.tiff" or ".ome.tif", as it is the filename extension that determines which format will be written.

Now that everything is set up, we can convert the image data. This is done plane by plane:

```java
for (int series=0; series<reader.getSeriesCount(); series++) {
    reader.setSeries(series);
    writer.setSeries(series);
    byte[] plane = new byte[FormatTools.getPlaneSize(reader)];
    for (int image=0; image<reader.getImageCount(); image++) {
        // code...
    }
}
```
reader.openBytes(image, plane);
writer.saveBytes(image, plane);
}
}

The body of the outer ‘for’ loop may also be replaced with the following:

reader.setSeries(series);
writer.setSeries(series);

for (int image=0; image<reader.getImageCount(); image++) {
byte[] plane = reader.openBytes(image);
writer.saveBytes(image, plane);
}

But note that this will be a little slower.

Finally, we must tell the reader and writer that we are finished, so that the input and output files can be properly closed:

reader.close();
writer.close();

There should now be a complete OME-TIFF file at whichever path was specified above.

### 3.2.12 Using Bio-Formats in MATLAB

This section assumes that you have installed the MATLAB toolbox as instructed in the MATLAB user information page. Note the minimum recommended MATLAB version is R2017b.

As described in Using Java Libraries, every installation of MATLAB includes a JVM (Java Virtual Machine) allowing use of the Java API and third-party Java libraries. All the helper functions included in the MATLAB toolbox make use of the Bio-Formats Java API. Please refer to the Javadoc for more information.

**Increasing JVM memory settings**

The default JVM settings in MATLAB can result in java.lang.OutOfMemoryError: Java heap space exceptions when opening large image files using Bio-Formats. Information about the Java heap space usage in MATLAB can be retrieved using:

```java
java.lang.Runtime.getRuntime.maxMemory
```

**MATLAB Preferences (R2010a+)**

Default JVM settings can be increased by Java Heap Memory Preferences of MATLAB (R2010a onwards) using Preferences → General → Java Heap Memory. We recommend to use 512 MB in general, but you don’t need to decrease the value if the default value is bigger than 512 MB.
However, the maximum value allowed in the Preferences GUI (typically set to 25% of Physical Memory, which you can ask by the memory MATLAB command) can still be too small for your purpose. In that case, you can directly edit the `matlab.prf` file in the folder specified by the `prefdir` MATLAB function (e.g., `'C:\Users\xxxxxx\AppData\Roaming\MathWorks\MATLAB\R2018b'`). Find the parameter `JavaMemHeapMax` in the file and increase the number that follows the capital `I` (in MB) to increase the maximum Java heap memory size. The change will be reflected by the Preferences as above.

```
JavaMemHeapMax=I25000
```

You will see a warning message as above.

```
MATLAB General Java Heap Memory Preferences
Java Heap Size
The heap size determines how much memory is available for Java objects. Increasing this value may help you avoid "OutOfMemoryError: Java heap space" errors.

Current value 25,000 MB. Default value 768 MB.

! The setting is outside the recommended range.
You should change it a value between 128 MB and 8,168 MB.
```

Alternatively, this can also be done by creating a `java.opts` file in the startup directory and overriding the default memory settings (see this page for more information). We recommend using `-Xmx512m` (meaning 512 MB) in your `java.opts` file. Calling:

```
bfCheckJavaMemory()
```

will also throw a warning if the runtime memory is lower than the recommended value.

If errors of type `java.lang.OutOfMemoryError: PermGen space` are thrown while using Bio-Formats with the Java bundled with MATLAB, you may try to increase the default values of `-XX:MaxPermSize` and `-XX:PermSize` via the `java.opts` file.
Opening an image file

The first thing to do is initialize a file with the `bfopen` function:

```matlab
data = bfopen('path/to/data/file');
```

This function returns an \( n \)-by-4 cell array, where \( n \) is the number of series in the dataset. If \( s \) is the series index between 1 and \( n \):

- The `data{s, 1}` element is an \( m \)-by-2 cell array, where \( m \) is the number of planes in the \( s \)-th series. If \( t \) is the plane index between 1 and \( m \):
  - The `data{s, 1}{t, 1}` element contains the pixel data for the \( t \)-th plane in the \( s \)-th series.
  - The `data{s, 1}{t, 2}` element contains the label for the \( t \)-th plane in the \( s \)-th series.
- The `data{s, 2}` element contains original metadata key/value pairs that apply to the \( s \)-th series.
- The `data{s, 3}` element contains color lookup tables for each plane in the \( s \)-th series.
- The `data{s, 4}` element contains a standardized OME metadata structure, which is the same regardless of the input file format, and contains common metadata values such as physical pixel sizes - see OME metadata below for examples.

Accessing planes

Here is an example of how to unwrap specific image planes for easy access:

```matlab
seriesCount = size(data, 1);
series1 = data{1, 1};
series2 = data{2, 1};
series3 = data{3, 1};
metadataList = data{1, 2};
% etc
series1_planeCount = size(series1, 1);
series1_plane1 = series1{1, 1};
series1_label1 = series1{1, 2};
series1_plane2 = series1{2, 1};
series1_label2 = series1{2, 2};
series1_plane3 = series1{3, 1};
series1_label3 = series1{3, 2};
```
Displaying images

If you want to display one of the images, you can do so as follows:

```matlab
series1_colorMap1 = data{1, 3}{1, 1};
figure('Name', series1_label1);
if isempty(series1_colorMap1)
    colormap(gray);
else
    colormap(series1_colorMap1);
end
imagesc(series1_plane1);
```

This will display the first image of the first series with its associated color map (if present). If you would prefer not to apply the color maps associated with each image, simply comment out the calls to `colormap`.

If you have the image processing toolbox, you could instead use:

```matlab
imshow(series1_plane1, []);
```

You can also create an animated movie (assumes 8-bit unsigned data):

```matlab
cmap = gray(256);
for p = 1 : size(series1, 1)
    M(p) = im2frame(uint8(series1{p, 1}), cmap);
end
if feature('ShowFigureWindows')
    movie(M);
end
```

Retrieving metadata

There are two kinds of metadata:

- **Original metadata** is a set of key/value pairs specific to the input format of the data. It is stored in the `data{s, 2}` element of the data structure returned by `bfopen`.

- **OME metadata** is a standardized metadata structure, which is the same regardless of input file format. It is stored in the `data{s, 4}` element of the data structure returned by `bfopen`, and contains common metadata values such as physical pixel sizes, instrument settings, and much more. See the OME Model and Formats documentation for full details.

Original metadata

To retrieve the metadata value for specific keys:

```matlab
% Query some metadata fields (keys are format-dependent)
metadata = data{1, 2};
subject = metadata.get('Subject');
title = metadata.get('Title');
```

To print out all of the metadata key/value pairs for the first series:
metadataKeys = metadata.keySet().iterator();
for i=1:metadata.size()
    key = metadataKeys.nextElement();
    value = metadata.get(key);
    fprintf('%s = %s\n', key, value)
end

**OMET metadata**

Conversion of metadata to the OME standard is one of Bio-Formats’ primary features. The OME metadata is always stored the same way, regardless of input file format.

To access physical voxel and stack sizes of the data:

```plaintext
omeMeta = data{1, 4};
stackSizeX = omeMeta.getPixelsSizeX(0).getValue(); % image width, pixels
stackSizeY = omeMeta.getPixelsSizeY(0).getValue(); % image height, pixels
stackSizeZ = omeMeta.getPixelsSizeZ(0).getValue(); % number of Z slices

voxelSizeXdefaultValue = omeMeta.getPixelsPhysicalSizeX(0).getValue(); % returns _value in default unit_
voxelSizeXdefaultUnit = omeMeta.getPixelsPhysicalSizeX(0).unit().getSymbol(); % returns _the default unit type_
voxelSizeX = omeMeta.getPixelsPhysicalSizeX(0).getValue(ome.units.UNITS.MICROMETER); % in µm
voxelSizeXdouble = voxelSizeX.doubleValue(); % numeric value represented by this object after conversion to type double

voxelSizeY = omeMeta.getPixelsPhysicalSizeY(0).getValue(ome.units.UNITS.MICROMETER); % in µm
voxelSizeYdouble = voxelSizeY.doubleValue(); % numeric value represented by this object after conversion to type double

voxelSizeZ = omeMeta.getPixelsPhysicalSizeZ(0).getValue(ome.units.UNITS.MICROMETER); % in µm
voxelSizeZdouble = voxelSizeZ.doubleValue(); % numeric value represented by this object after conversion to type double
```

For more information about the methods to retrieve the metadata, see the MetadataRetrieve Javadoc page.

To convert the OME metadata into a string, use the `dumpXML()` method:

```plaintext
omeXML = char(omeMeta.dumpXML());
```

**Changing the logging level**

By default, `bfopen` uses `bInitLogging` to initialize the logging system at the `WARN` level. To change the root logging level, use the `DebugTools` methods as described in the `Logging` section.

```plaintext
% Set the logging level to DEBUG
loci.common.DebugTools.setRootLevel('DEBUG');
```
Reading from an image file

The main inconvenience of the `bfopen.m` function is that it loads all the content of an image regardless of its size. To access the file reader without loading all the data, use the low-level `bfGetReader.m` function:

```matlab
caller = bfGetReader('path/to/data/file');
```

You can then access the OME metadata using the `getMetadataStore()` method:

```matlab
omeMeta = caller.getMetadataStore();
```

Individual planes can be queried using the `bfGetPlane.m` function:

```matlab
series1_plane1 = bfGetPlane(caller, 1);
```

To switch between series in a multi-image file, use the `setSeries(int)` method. To retrieve a plane given a set of \((z, c, t)\) coordinates, these coordinates must be linearized first using `getIndex(int, int, int)`

```matlab
% Read plane from series iSeries at Z, C, T coordinates (iZ, iC, iT)
% All indices are expected to be 1-based
caller.setSeries(iSeries - 1);
iPlane = caller.getIndex(iZ - 1, iC - 1, iT - 1) + 1;
I = bfGetPlane(caller, iPlane);
```

Saving files

The basic code for saving a 5D array into an OME-TIFF file is located in the `bfsave.m` function. For instance, the following code will save a single image of 64 pixels by 64 pixels with 8 unsigned bits per pixels:

```matlab
plane = zeros(64, 64, 'uint8');
bfsave(plane, 'single-plane.ome.tiff');
```

And the following code snippet will produce an image of 64 pixels by 64 pixels with 2 channels and 2 timepoints:

```matlab
plane = zeros(64, 64, 1, 2, 2, 'uint8');
bfsave(plane, 'multiple-planes.ome.tiff');
```

By default, `bfsave` will create a minimal OME-XML metadata object containing basic information such as the pixel dimensions, the dimension order and the pixel type. To customize the OME metadata, it is possible to create a metadata object from the input array using `createMinimalOMEXMLMetadata.m`, add custom metadata and pass this object directly to `bfsave`:

```matlab
plane = zeros(64, 64, 1, 2, 2, 'uint8');
metadata = createMinimalOMEXMLMetadata(plane);
pixelSize = ome.units.quantity.Length(java.lang.Double(.05), ome.units.UNITS.MICROMETER);
metadata.setPixelsPhysicalSizeX(pixelSize, 0);
metadata.setPixelsPhysicalSizeY(pixelSize, 0);
pixelSizeZ = ome.units.quantity.Length(java.lang.Double(.2), ome.units.UNITS.MICROMETER);
metadata.setPixelsPhysicalSizeZ(pixelSizeZ, 0);
bfsave(plane, 'metadata.ome.tiff', 'metadata', metadata);
```

The dimension order of the file on disk can be specified in two ways:

- either by passing an OME-XML metadata option as a key/value pair as shown above
• or as an optional positional argument of bfsave

If a metadata object is passed to bfsave, its dimension order stored internally will take precedence.

For more information about the methods to store the metadata, see the MetadataStore Javadoc page.

**Improving reading performance**

Initializing a Bio-Formats reader can consume substantial time and memory. Most of the initialization time is spent in the setId(java.lang.String) call. Various factors can impact the performance of this step including the file size, the amount of metadata in the image and also the file format itself.

One solution to improve reading performance is to use Bio-Formats memoization functionalities with the loci.formats.Memoizer reader wrapper. By essence, the speedup gained from memoization will only happen after the first initialization of the reader for a particular file.

The simplest way to make use the Memoizer functionalities in MATLAB is illustrated by the following example:

```matlab
% Construct an empty Bio-Formats reader
r = bfGetReader();
% Decorate the reader with the Memoizer wrapper
r = loci.formats.Memoizer(r);
% Initialize the reader with an input file
% If the call is longer than a minimal time, the initialized reader will
% be cached in a file under the same directory as the initial file
% name .large_file.bfmemo
r.setId(pathToFile);
% Perform work using the reader
r.close()
% If the reader has been cached in the call above, re-initializing the
% reader will use the memo file and complete much faster especially for
% large data
r.setId(pathToFile);
% Perform additional work
r.close()
```

If the time required to call setId(java.lang.String) method is larger than DEFAULT_MINIMUM_ELAPSED or the minimum value passed in the constructor, the initialized reader will be cached in a memo file under the same folder as the input file. Any subsequent call to setId() with a reader decorated by the Memoizer on the same input file will load the reader from the memo file instead of performing a full reader initialization.

More constructors are described in the Memoizer javadocs allowing to control the minimal initialization time required before caching the reader and/or to define a root directory under which the reader should be cached.

As Bio-Formats is not thread-safe, reader memoization offers a new solution to increase reading performance when doing parallel work. For instance, the following example shows how to combine memoization and MATLAB parfor to do work on a single file in a parallel loop:
% Construct a Bio-Formats reader decorated with the Memoizer wrapper
r = loci.formats.Memoizer(bfGetReader(), 0);
% Initialize the reader with an input file to cache the reader
r.setId(pathToFile);
% Close reader
r.close()

nWorkers = 4;

% Enter parallel loop
parfor i = 1 : nWorkers
    % Initialize logging at INFO level
    bfInitLogging('INFO');
    % Initialize a new reader per worker as Bio-Formats is not thread safe
    r2 = javaObject('loci.formats.Memoizer', bfGetReader(), 0);
    % Initialization should use the memo file cached before entering the
    % parallel loop
    r2.setId(pathToFile);

    % Perform work

    % Close the reader
    r2.close()
end

3.2.13 Using Bio-Formats in Python

OME does not currently provide a Python implementation for Bio-Formats. However, there are several options you can use to read images from Python via Bio-Formats:

AICSImageIO

The AICSImageIO project includes support for Bio-Formats:

```python
from aicsimageio import AICSImage
cells = AICSImage('/path/to/my/cells.ome.tif')

import napari
napari.view_image(cells.xarray_data)
napari.run()
```
**PyImageJ**

The PyImageJ project enables use of ImageJ2, which includes the SCIFIO library, which wraps Bio-Formats. In this way, you can open Bio-Formats-supported formats as NumPy arrays:

```plaintext
import imagej
ij = imagej.init('sc.fiji:fi ji')
jcells = ij.io().open('/path/to/my/cells.ome.tif')
cells = ij.py.from_java(jcells)

import napari
napari.view_image(cells)
napari.run()
```

**python-bioformats**

The CellProfiler project has implemented a Python wrapper around Bio-Formats used by the CellProfiler software which can be installed using pip:

```plaintext
pip install python-bioformats
```

See also:

- [https://pypi.org/project/python-bioformats](https://pypi.org/project/python-bioformats)
  - Source code of the CellProfiler Python wrapper for Bio-Formats

### 3.2.14 Using Bio-Formats in R

OME does not currently provide an R implementation for Bio-Formats. However, there are options provided by the community:

**Bioconductor RBioFormats**

The RBioFormats project from Bioconductor provides an R package which interfaces the OME Bio-Formats Java library to allow reading of proprietary microscopy image data and metadata:

```plaintext
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")

# The following initializes usage of Bioc devel
BiocManager::install(version='devel')

BiocManager::install("RBioFormats")

# To view documentation for the version of this package installed in your system
browseVignettes("RBioFormats")
```

For further details and examples of how to use the package, please see the API documentation
3.2.15 Interfacing with Bio-Formats from non-Java code

Bio-Formats is written in Java, and is easiest to use with other Java code. However, it is possible to call Bio-Formats from a program written in another language. But how to do so depends on your program’s needs.

Technologically, there are two broad categories of solutions: in-process approaches, and inter-process communication.

For details, see LOCI’s article, Interfacing from non-Java code.

Example in-process solution: Bio-Formats JACE C++ bindings (note that this is a legacy project and no longer actively maintained).

See also:

Additional reader and writer options

3.3 Using Bio-Formats as a native C++ library

Note: See the OME-Files C++ downloads page for more information.

3.4 Contributing to Bio-Formats

3.4.1 Code formatting

Note, these guidelines do not cover:

• third-party code imported into the source tree, which is covered by the guidelines for the upstream projects
• released schema files which would require re-releasing if changed by reindenting

All languages

• Use spaces to indent; do not ever use tabs

Java

All Java code is formatted with:

• an indentation size of two spaces
• braces use the Java variant of K&R style
**XML**

All XML code is formatted with:

- an indentation size of two spaces
- attributes on multiple lines aligned vertically after the element name.

### 3.4.2 Testing code changes

**Automated tests**

The Bio-Formats testing framework component contains most of the infrastructure to run automated tests against the data repository.

After checking out source code and building all the JAR files (see *Obtaining and building Bio-Formats*), switch to the test-suite component and run the tests using the `ant test-automated` target:

```bash
$ cd components/test-suite
$ ant -Dtestng.directory=$DATA/metamorph -Dtestng.configDirectory=$CONFIG/metamorph test-automated
```

where `$DATA` is the path to the full data repository and `$CONFIG` is the path to the configuration repository.

Multiple options can be passed to the `ant test-automated` target by setting the `testng.${option}` option via the command line. Useful options are described below.

- **testng.directory**
  Mandatory option. Specifies the root of the data directory to be tested:

  ```bash
  $ ant -Dtestng.directory=$DATA/metamorph -Dtestng.configDirectory=$CONFIG/metamorph test-automated
  ```

  On Windows, the arguments to the test command must be quoted:

  ```bash
  > ant "-Dtestng.directory=$DATA\metamorph" "-Dtestng.configDirectory=$CONFIG\metamorph" test-automated
  ```

- **testng.configDirectory**
  Mandatory option. Specifies the root of the directory containing the configuration files. This directory must have the same hierarchy as the one specified by `testng.directory` and contain `.bioformats` configuration files:

  ```bash
  $ ant -Dtestng.directory=/path/to/data -Dtestng.configDirectory=/path/to/config test-automated
  ```

- **testng.configSuffix**
  Specifies an optional suffix for the configuration files:

  ```bash
  $ ant -Dtestng.directory=/path/to/data -Dtestng.configSuffix=win test-automated
  ```

- **testng.memory**
  Specifies the amount of memory to be allocated to the JVM:

  ```bash
  $ ant -Dtestng.directory=$DATA -Dtestng.memory=4g test-automated
  ```

  Default: 512m.
testng.threadCount

Specifies the number of threads to use for testing:

```
$ ant -Dtestng.directory=${DATA} -Dtestng.threadCount=4 test-automated
```

Default: 1.

You should now see output similar to this:

```
Buildfile: build.xml

init-title:
  [echo] ------------------------ bio-formats-testing-framework ------------------------
  ...

  SmartTest: test-automated:
  → NOT find resource [{logback.configurationFile}]
  → NOT find resource [logback.groovy]
  → NOT find resource [logback-test.xml]
  → resource [logback.xml] at [file:/opt/ome/bioformats/components/test-suite/logback.xml]
  [testng] 17:05:28,835 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → About to instantiate appender of type [ch.qos.logback.core.ConsoleAppender]
  [testng] 17:05:28,837 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → Naming appender as [stdout]
  [testng] 17:05:28,876 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → About to instantiate appender of type [ch.qos.logback.classic.sift.SiftingAppender]
  [testng] 17:05:28,878 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → Naming appender as [SIFT]
  [testng] 17:05:28,891 |-INFO in ch.qos.logback.classic.joran.action.LoggerAction -
  → Setting level of logger [loci.tests.testng] to DEBUG
  [testng] 17:05:28,891 |-INFO in ch.qos.logback.classic.joran.action.RootLoggerAction -
  → Setting level of ROOT logger to INFO
  [testng] 17:05:28,891 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction -
  → Attaching appender named [SIFT] to Logger[ROOT]
  [testng] 17:05:28,892 |-INFO in ch.qos.logback.core.joran.action.AppenderRefAction -
  → Attaching appender named [stdout] to Logger[loci.tests.testng]
  → ConfigurationAction - End of configuration.
  → JoranConfigurer@706a04ae - Registering current configuration as safe fallback point
  [testng] [2015-08-18 17:05:28,904] [main] testng.directory = /ome/data_repo/test_per_
  → commit/
  [testng] 17:05:28,908 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → About to instantiate appender of type [loci.tests.testng.TimestampedLogFileAppender]
  [testng] 17:05:28,909 |-INFO in ch.qos.logback.core.joran.action.AppenderAction -
  → Naming appender as [logfile-main]
  [testng] 17:05:28,955 |-INFO in loci.tests.testng.TimestampedLogFileAppender[logfile-
  [testng] [2015-08-18 17:05:28,963] [main] testng.multiplier = 1.0
  [testng] [2015-08-18 17:05:28,964] [main] testng.in-memory = false
  [testng] [2015-08-18 17:05:28,964] [main] user.language = en
```

(continues on next page)
 Bio-Formats, Release 6.14.0

[2015-08-18 17:05:28,964] [main] user.country = US
[2015-08-18 17:05:28,964] [main] Maximum heap size = 455 MB
[2015-08-18 17:05:32,258] [main] Building list of tests...

and then eventually:

[2015-08-18 17:05:32,258] [main] Total files: 480
[2015-08-18 17:05:32,258] [main] Scan time: 3.293 s (6 ms/file)

BUILD SUCCESSFUL
Total time: 16 minutes 42 seconds

In most cases, test failures should be logged in the main console output as:

[2015-08-18 17:13:13,625] [pool-1-thread-1] SizeZ: FAILED (Series 0...

To identify the file, look for the initialization line preceding the test failures under the same thread:

[2015-08-18 17:13:12,376] [pool-1-thread-1] Initializing /ome/data_repo/test_per_commit/ome-tiff/img_bk_20110701.ome.tif:

The console output is also recorded under components/test-suite/target as bio-formats-software-test-main-$DATE.log where “$DATE” is the date on which the tests started in “yyyy-MM-dd_hh-mm-ss” format. The detailed report of each thread is recorded under bio-formats-software-pool-$POOL-thread-$THREAD-main-$DATE.log.

Configuration files can be generated for files or directories using the ant gen-config target. This generation target supports the same options as ant test-automated:

$ ant -Dtestng.directory=/path/to/data -Dtestng.configDirectory=/path/to/config -Dtestng.memory=4g -Dtestng.threadCount=6 gen-config

MATLAB tests

Tests for the Bio-Formats MATLAB toolbox are written using the xunit framework and are located under https://github.com/ome/bioformats/tree/develop/components/formats-gpl/test/matlab.

To run these tests, you will need to download or clone matlab-xunit, a xUnit framework with JUnit-compatible XML output. Then add this package together with the Bio-Formats MATLAB to your MATLAB path:

% Add the matlab-xunit toolbox to the MATLAB path
cd('/path/to/matlab-xunit');
% Add the Bio-Formats MATLAB source to the MATLAB path
% For developers working against the source code

(continues on next page)
You can run all the MATLAB tests using `runxunit`:

```matlab
cd /path/to/bioformats/components/formats-gpl/test/matlab
runxunit
```

Individual test classes can be run by passing the name of the class:

```matlab
cd /path/to/bioformats/components/formats-gpl/test/matlab
runxunit TestBfsave
```

Individual test methods can be run by passing the name of the class and the name of the method:

```matlab
cd /path/to/bioformats/components/formats-gpl/test/matlab
runxunit TestBfsave:testLZW
```

Finally, to output the test results under XML format, you can use the `-xmlfile` option:

```matlab
cd /path/to/bioformats/components/formats-gpl/test/matlab
runxunit -xmlfile test-output.xml
```

### 3.4.3 Generating test images

Sometimes it is nice to have a file of a specific size or pixel type for testing. Bio-Formats defines an internal format used for generating test images. Test images recognized by Bio-Formats:

- must have an extension of type `.fake`
- must encode the image metadata using key-value pairs separated by `=` either in the filename or in a companion file
- may be accompanied by an INI-style companion file. A companion file must use the same basename as the fake file and be suffixed with `.ini`

**Filename format**

To generate an image file (that contains a gradient image):

```bash
touch "my-special-test-file&pixelType=uint8&sizeX=8192&sizeY=8192.fake"
```

Whatever is before the first `&` is the image name; the remaining key-value pairs, each preceded with `&`, set the pixel type and image dimensions. Just replace the values with whatever you need for testing. See Key-value pairs for the full description of available key/value pairs.
Companion file

You can put metadata values in a separate UTF-8 encoded .fake.ini file with the same basename as the fake file:

```
touch my-special-test-file.fake
echo "pixelType=uint8" >> my-special-test-file.fake.ini
echo "sizeX=8192" >> my-special-test-file.fake.ini
echo "sizeY=8192" >> my-special-test-file.fake.ini
```

In fact, just the .fake.ini file alone suffices:

```
echo "pixelType=uint8" >> my-special-test-file.fake.ini
echo "sizeX=8192" >> my-special-test-file.fake.ini
echo "sizeY=8192" >> my-special-test-file.fake.ini
```

If you include a “[GlobalMetadata]” section to the ini file, then all the included values will be accessible from the global metadata map:

```
echo "[GlobalMetadata]" >> my-special-test-file.fake.ini
echo "my.key=some.value" >> my-special-test-file.fake.ini
```

The .ini file can also contain one section for each series, which allows metadata such as exposure times and positions to be set for each plane:

```
echo "[series_0]" >> my-special-test-file.fake.ini
echo "ExposureTime_0=10" >> my-special-test-file.fake.ini
echo "ExposureTimeUnit_0=ms" >> my-special-test-file.fake.ini
echo "PositionX_0=5" >> my-special-test-file.fake.ini
echo "PositionY_0=5" >> my-special-test-file.fake.ini
echo "PositionZ_0=1" >> my-special-test-file.fake.ini
echo "PositionXUnit_0=mm" >> my-special-test-file.fake.ini
echo "PositionYUnit_0=mm" >> my-special-test-file.fake.ini
echo "PositionZUnit_0=mm" >> my-special-test-file.fake.ini
```

Several keys have support for units and can be expressed as KEY=VALUE UNIT where UNIT is the symbol of the desired unit:

```
touch "physicalSizesUnits&physicalSizeX=1nm&physicalSizeY=1nm&physicalSizeZ=1.5km.fake"
echo "physicalSizeX=1 nm" >> physicalSizes.fake.ini
echo "physicalSizeY=10 pm" >> physicalSizes.fake.ini
echo "physicalSizeZ=.002 mm" >> physicalSizes.fake.ini
```

High-content screening

To generate a simple plate file:

```
touch "simple-plate&plates=1&plateAcqs=1&plateRows=1&plateCols=1&fields=1.fake"
touch "default-plate&plates=1.fake"
touch "default-plate&screens=0&plates=1.fake"
```

These will each create a single plate without a containing screen, by default in the first two cases. In the third case setting screens to zero is used to document the lack of a screen. As above a .fake.ini file can be used.

To generate a simple screen file:
touch "default-screen&screens=1.fake"

This will create a screen containing a single simple plate.

To generate a valid plate at least one of screens, plates, plateAcqs, plateRows, plateCols and fields must be greater than zero. If this condition is met then any other plate-specific values set to zero will be ignored and the defaults used. So, for example, the file:

one-key-set&screens=0&plates=0&plateRows=0&plateCols=0&plateAcqs=0&fields=1.fake

will create a simple plate with no screen.

**Regions**

To generate a fake file containing regions of interest:

```
touch "regions&points=10.fake"
touch "regions&ellipses=20.fake"
touch "regions&rectangles=5&lines=25.fake"
```

Replace `regions` in the above examples with the desired image or plate which will contain the regions, e.g.

```
touch "HCSanalysis&plates=1&plateRows=16&plateCols=24&rectangles=100.fake"
```

For each shape type, the value will specify the number of regions of interest to create where each region of interest contains a single shape of the input type. By convention, all generated regions of interests are not associated to any given Z, C or T plane.

**Sub-resolutions**

New in version 6.0.0.

To generate a fake file containing sub-resolutions:

```
touch "pyramid1&sizeX=20000&sizeY=10000&resolutions=8.fake"
touch "pyramid2&sizeX=20000&sizeY=10000&resolutions=4&resolutionScale=4.fake"
```

The resolutions and resolutionScale specify the number of sub-resolutions for each plane and the downsampling factor between consecutive sub-resolutions.

**Key-value pairs**

There are several other keys that can be added, a complete list of these, with their default values, is shown below.

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sizeX</td>
<td>number of pixels wide</td>
<td>512</td>
</tr>
<tr>
<td>sizeY</td>
<td>number of pixels tall</td>
<td>512</td>
</tr>
<tr>
<td>sizeZ</td>
<td>number of Z sections</td>
<td>1</td>
</tr>
<tr>
<td>sizeC</td>
<td>number of channels</td>
<td>1</td>
</tr>
<tr>
<td>sizeT</td>
<td>number of timepoints</td>
<td>1</td>
</tr>
<tr>
<td>thumbSizeX</td>
<td>number of pixels wide, for the thumbnail</td>
<td>0</td>
</tr>
<tr>
<td>thumbSizeY</td>
<td>number of pixels tall, for the thumbnail</td>
<td>0</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>pixelType</td>
<td>pixel type</td>
<td>uint8</td>
</tr>
<tr>
<td>bitsPerPixel</td>
<td>number of valid bits (&lt;= number of bits implied by pixel type)</td>
<td>0</td>
</tr>
<tr>
<td>rgb</td>
<td>number of channels that are merged together</td>
<td>1</td>
</tr>
<tr>
<td>dimOrder</td>
<td>dimension order (e.g. XYZCT)</td>
<td>XYZCT</td>
</tr>
<tr>
<td>orderCertain</td>
<td>whether or not the dimension order is certain</td>
<td>true</td>
</tr>
<tr>
<td>little</td>
<td>whether or not the pixel data should be little-endian</td>
<td>true</td>
</tr>
<tr>
<td>interleaved</td>
<td>whether or not merged channels are interleaved</td>
<td>false</td>
</tr>
<tr>
<td>indexed</td>
<td>whether or not a color lookup table is present</td>
<td>false</td>
</tr>
<tr>
<td>falseColor</td>
<td>whether or not the color lookup table is just for making the image</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>look pretty</td>
<td></td>
</tr>
<tr>
<td>metadataComplete</td>
<td>whether or not the metadata is complete</td>
<td>true</td>
</tr>
<tr>
<td>thumbnail</td>
<td>whether or not CoreMetadata.thumbnail is set</td>
<td>false</td>
</tr>
<tr>
<td>series</td>
<td>number of series (Images)</td>
<td>1</td>
</tr>
<tr>
<td>lutLength</td>
<td>number of entries in the color lookup table</td>
<td>3</td>
</tr>
<tr>
<td>scaleFactor</td>
<td>the scaling factor for the pixel values on each plane</td>
<td>1</td>
</tr>
<tr>
<td>exposureTime</td>
<td>time of exposure</td>
<td>null</td>
</tr>
<tr>
<td>acquisitionDate</td>
<td>timestamp formatted as “yyyy-MM-dd_HH-mm-ss”</td>
<td>null</td>
</tr>
<tr>
<td>screens</td>
<td>number of screens</td>
<td>0</td>
</tr>
<tr>
<td>plates</td>
<td>number of plates to generate</td>
<td>0</td>
</tr>
<tr>
<td>plateAcqs</td>
<td>number of plate runs</td>
<td>Page 90, 1</td>
</tr>
<tr>
<td>plateRows</td>
<td>number of rows per plate</td>
<td>Page 90, 1</td>
</tr>
<tr>
<td>plateCols</td>
<td>number of columns per plate</td>
<td>Page 90, 1</td>
</tr>
<tr>
<td>fields</td>
<td>number of fields per well</td>
<td>Page 90, 1</td>
</tr>
<tr>
<td>withMicrobeam</td>
<td>whether or not a microbeam should be added to the experiment (HCS</td>
<td>false</td>
</tr>
<tr>
<td></td>
<td>only)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>number of annotations of the given type to generate</td>
<td>0</td>
</tr>
<tr>
<td>physicalSizeX</td>
<td>real width of the pixels, supports units defaulting to microns</td>
<td></td>
</tr>
<tr>
<td>physicalSizeY</td>
<td>real height of the pixels, supports units defaulting to microns</td>
<td></td>
</tr>
<tr>
<td>physicalSizeZ</td>
<td>real depth of the pixels, supports units defaulting to microns</td>
<td></td>
</tr>
<tr>
<td>ChannelName_x</td>
<td>the channel name for channel x</td>
<td></td>
</tr>
<tr>
<td>color</td>
<td>the default color for all channels</td>
<td>null</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>color_x</td>
<td>the color for channel x, overrides the default color for that channel</td>
<td>Page 90, 3</td>
</tr>
<tr>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>ellipses, labels,</td>
<td>the number of ROIs containing one shape of the given type to generate</td>
<td></td>
</tr>
<tr>
<td>lines, points,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>polygons, polylines,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>rectangles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ExposureTime_x</td>
<td>floating point exposure time for plane x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>ExposureTimeUnit_x</td>
<td>string defining the units for the corresponding ExposureTime_x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionX_x</td>
<td>floating point X position for plane x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionXUnit_x</td>
<td>string defining the units for the corresponding PositionX_x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionY_x</td>
<td>floating point Y position for plane x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionYUnit_x</td>
<td>string defining the units for the corresponding PositionY_x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionZ_x</td>
<td>floating point Z position for plane x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>PositionZUnit_x</td>
<td>string defining the units for the corresponding PositionZ_x</td>
<td>Page 90, 2</td>
</tr>
<tr>
<td>resolutions</td>
<td>number of pyramid levels or sub-resolutions for each series</td>
<td>1</td>
</tr>
<tr>
<td>resolutionScale</td>
<td>for images with sub-resolutions, scaling factor between consecutive</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>pyramid levels</td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Key</th>
<th>Value</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>sleepOpenBytes</td>
<td>number of milliseconds to sleep for when openBytes is called</td>
<td>0</td>
</tr>
<tr>
<td>sleepInitFile</td>
<td>number of milliseconds to sleep for when initFile is called</td>
<td>0</td>
</tr>
</tbody>
</table>

For full details of these keys, how unset and default values are handled and further examples see loci.formats.in.FakeReader.

You can often work with the .fake file directly, but in some cases support for those files is disabled and so you will need to convert the file to something else. Make sure that you have Bio-Formats built and the JARs in your CLASSPATH (individual JARs or just bioformats_package.jar):

```
bfconvert test&pixelType=uint8&sizeX=8192&sizeY=8192.fake test.tif
```

If you do not have the command line tools installed, substitute loci.formats.tools.ImageConverter for bfconvert.

### 3.4.4 Writing a new file format reader

This document is a brief guide to writing new Bio-Formats file format readers.

All format readers should extend either loci.formats.FormatReader or an existing reader.

#### Methods to override

- `isSingleFile(java.lang.String)` Whether or not the named file is expected to be the only file in the dataset. This only needs to be overridden for formats whose datasets can contain more than one file.
- `isThisType(loci.common.RandomAccessInputStream)` Check the first few bytes of a file to determine if the file can be read by this reader. You can assume that index 0 in the stream corresponds to the index 0 in the file. Return true if the file can be read; false if not (or if there is no way of checking).
- `fileGroupOption(java.lang.String)` Returns an indication of whether or not the files in a multi-file dataset can be handled individually. The return value should be one of the following:
  - `FormatTools.MUST_GROUP`: the files cannot be handled separately
  - `FormatTools.CAN_GROUP`: the files may be handled separately or as a single unit
  - `FormatTools.CANNOT_GROUP`: the files must be handled separately

  This method only needs to be overridden for formats whose datasets can contain more than one file.

- `getSeriesUsedFiles(boolean)` You only need to override this if your format uses multiple files in a single dataset. This method should return a list of all files associated with the given file name and the current series (i.e. every file needed to display the current series). If the noPixels flag is set, then none of the files returned should contain pixel data. For an example of how this works, see loci.formats.in.PerkinElmerReader. It is recommended that the first line of this method be `FormatTools.assertId(currentId, true, 1)` - this ensures that the file name is non-null.

- `openBytes(int, byte[], int, int, int, int)` Returns a byte array containing the pixel data for a specified subimage from the given file. The dimensions of the subimage (upper left X coordinate, upper left Y coordinate, width, and height) are specified in the final four int parameters. This should throw a `FormatException` if the image number is invalid (less than 0 or >= the number of images). The ordering of the array returned by openBytes should

---

1 Default value set to 1 if any of the screens, plates, plateAcqs, plateRows, plateCols or fields values is set to a value greater than zero.

2 Must be stored in the INI file under a [series_n] section, where n is the 0-based series index.

3 Colors are specified as a single packed integer representing an RGBA value. This can be a decimal value (e.g. 16711935 for green with alpha = 255) or a hexadecimal value (e.g. 0x00FF00FF for green with alpha = 255).
correspond to the values returned by `isLittleEndian` and `isInterleaved`. Also, the length of the byte array should be `[image width * image height * bytes per pixel]`. Extra bytes will generally be truncated. It is recommended that the first line of this method be `FormatTools.checkPlaneParameters(this, no, buf.length, x, y, w, h)` - this ensures that all of the parameters are valid.

- `initFile(java.lang.String)` The majority of the file parsing logic should be placed in this method. The idea is to call this method once (and only once!) when the file is first opened. Generally, you will want to start by calling `super.initFile(String)`. You will also need to set up the stream for reading the file, as well as initializing any dimension information and metadata. Most of this logic is up to you; however, you should populate the core variable (see `loci.formats.CoreMetadata`).

  Note that each variable is initialized to 0 or null when `super.initFile(String)` is called. Also, `super.initFile(String)` constructs a Hashtable called `metadata` where you should store any relevant metadata.

  The most common way to set up the OME-XML metadata for the reader is to initialize the MetadataStore using the `makeFilterMetadata()` method and populate the Pixels elements of the metadata store from the core variable using the `MetadataTools.populatePixels(MetadataStore, FormatReader)` method:

  ```java
  # Initialize the OME-XML metadata from the core variable
  MetadataStore store = makeFilterMetadata();
  MetadataTools.populatePixels(store, this);
  ```

  If the reader includes metadata at the plane level, you can initialize the Plane elements under the Pixels using `MetadataTools.populatePixels(MetadataStore, FormatReader, doPlane)`:  

  ```java
  MetadataTools.populatePixels(store, this, true);
  ```

  Once the metadatastore has been initialized with the core properties, additional metadata can be added to it using the setter methods. Note that for each of the model components, the `setObjectID()` method should be called before any of the `setObjectProperty()` methods, e.g.:

  ```java
  # Add an oil immersion objective with achromat
  String objectiveID = MetadataTools.createLSID("Objective", 0, 0);
  store.setObjectiveID(objectiveID, 0, 0);
  store.setObjectiveImmersion(getImmersion("Oil"), 0, 0);
  ```

- `close(boolean)` Cleans up any resources used by the reader. Global variables should be reset to their initial state, and any open files or delegate readers should be closed.

  Note that if the new format is a variant of a format currently supported by Bio-Formats, it is more efficient to make the new reader a subclass of the existing reader (rather than subclassing `loci.formats.FormatReader`). In this case, it is usually sufficient to override `initFile(java.lang.String)` and `isThisType(byte[])`.

  Every reader also has an instance of `loci.formats.CoreMetadata`. All readers should populate the fields in CoreMetadata, which are essential to reading image planes.

  If you read from a file using something other than `loci.common.RandomAccessInputStream` or `loci.common.Location`, you must use the file name returned by `Location.getMappedId(String)`, not the file name passed to the reader. Thus, a stub for `initFile(String)` might look like this:

  ```java
  protected void initFile(String id) throws FormatException, IOException {
    super.initFile(id);

    RandomAccessInputStream in = new RandomAccessInputStream(id);
    // alternatively,
    // FileInputStream in = new FileInputStream(Location.getMappedId(id));
  }
  ```
// read basic file structure and metadata from stream

For more details, see loci.common.Location.mapId(java.lang.String, java.lang.String) and loci.common.Location.getMappedId(java.lang.String).

Variables to populate

There are a number of global variables defined in loci.formats.FormatReader that should be populated in the constructor of any implemented reader.

These variables are:

- **suffixNecessary** Indicates whether or not a file name suffix is required; true by default
- **suffixSufficient** Indicates whether or not a specific file name suffix guarantees that this reader can open a particular file; true by default
- **hasCompanionFiles** Indicates whether or not there is at least one file in a dataset of this format that contains only metadata (no images); false by default
- **datasetDescription** A brief description of the layout of files in datasets of this format; only necessary for multi-file datasets
- **domains** An array of imaging domains for which this format is used. Domains are defined in loci.formats.FormatTools.

Other useful things

- **loci.common.RandomAccessInputStream** is a hybrid RandomAccessFile/InputStream class that is generally more efficient than either RandomAccessFile or InputStream, and implements the DataInput interface. It is recommended that you use this for reading files.
- **loci.common.Location** provides an API similar to java.io.File, and supports File-like operations on URLs. It is highly recommended that you use this instead of File. See the Javadoc for additional information.
- **loci.common.DataTools** provides a number of methods for converting bytes to shorts, ints, longs, etc. It also supports reading most primitive types directly from a RandomAccessInputStream (or other DataInput implementation).
- **loci.formats.ImageTools** provides several methods for manipulating primitive type arrays that represent images. Consult the source or Javadocs for more information.
- If your reader relies on third-party code which may not be available to all users, it is strongly suggested that you make a corresponding service class that interfaces with the third-party code. Please see Bio-Formats service and dependency infrastructure for a description of the service infrastructure, as well as the loci.formats.services package.
- Several common image compression types are supported through subclasses of loci.formats.codec.BaseCodec. These include JPEG, LZW, LZO, Base64, ZIP and RLE (PackBits).
- If you wish to convert a file’s metadata to OME-XML (strongly encouraged), please see Bio-Formats metadata processing for further information.
- Once you have written your file format reader, add a line to the readers.txt file with the fully qualified name of the reader, followed by a ‘#’ and the file extensions associated with the file format. Note that loci.formats.ImageReader, the master file format reader, tries to identify which format reader to use according to the order given in readers.txt, so be sure to place your reader in an appropriate position within the list.
• The easiest way to test your new reader is by calling “java loci.formats.tools.ImageInfo <file name>”. If all goes well, you should see all of the metadata and dimension information, along with a window showing the images in the file. loci.formats.ImageReader can take additional parameters; a brief listing is provided below for reference, but it is recommended that you take a look at the contents of loci.formats.tools.ImageInfo to see exactly what each one does.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>-version</td>
<td>print the library version and exit</td>
</tr>
<tr>
<td>file</td>
<td>the image file to read</td>
</tr>
<tr>
<td>-nopix</td>
<td>read metadata only, not pixels</td>
</tr>
<tr>
<td>-nocore</td>
<td>do not output core metadata</td>
</tr>
<tr>
<td>-nometa</td>
<td>do not parse format-specific metadata table</td>
</tr>
<tr>
<td>-nofilter</td>
<td>do not filter metadata fields</td>
</tr>
<tr>
<td>-thumbs</td>
<td>read thumbnails instead of normal pixels</td>
</tr>
<tr>
<td>-minmax</td>
<td>compute min/max statistics</td>
</tr>
<tr>
<td>-merge</td>
<td>combine separate channels into RGB image</td>
</tr>
<tr>
<td>-nogroup</td>
<td>force multi-file datasets to be read as individual files</td>
</tr>
<tr>
<td>-stitch</td>
<td>stitch files with similar names</td>
</tr>
<tr>
<td>-separate</td>
<td>split RGB image into separate channels</td>
</tr>
<tr>
<td>-expand</td>
<td>expand indexed color to RGB</td>
</tr>
<tr>
<td>-omexml</td>
<td>populate OME-XML metadata</td>
</tr>
<tr>
<td>-normalize</td>
<td>normalize floating point images*</td>
</tr>
<tr>
<td>-fast</td>
<td>paint RGB images as quickly as possible*</td>
</tr>
<tr>
<td>-debug</td>
<td>turn on debugging output</td>
</tr>
<tr>
<td>-range</td>
<td>specify range of planes to read (inclusive)</td>
</tr>
<tr>
<td>-series</td>
<td>specify which image series to read</td>
</tr>
<tr>
<td>-swap</td>
<td>override the default input dimension order</td>
</tr>
<tr>
<td>-shuffle</td>
<td>override the default output dimension order</td>
</tr>
<tr>
<td>-map</td>
<td>specify file on disk to which name should be mapped</td>
</tr>
<tr>
<td>-preload</td>
<td>pre-read entire file into a buffer; significantly reduces the time required to read the images, but requires more memory</td>
</tr>
<tr>
<td>-crop</td>
<td>crop images before displaying; argument is ‘x,y,w,h’</td>
</tr>
<tr>
<td>-autoscale</td>
<td>used in combination with ‘-fast’ to automatically adjust brightness and contrast</td>
</tr>
<tr>
<td>-novalid</td>
<td>do not perform validation of OME-XML</td>
</tr>
<tr>
<td>-omexml-only</td>
<td>only output the generated OME-XML</td>
</tr>
<tr>
<td>-format</td>
<td>read file with a particular reader (e.g., ZeissZVI)</td>
</tr>
</tbody>
</table>

* = may result in loss of precision

• If you wish to test using TestNG, loci.tests.testng.FormatReaderTest provides several basic tests that work with all Bio-Formats readers. See the FormatReaderTest source code for additional information.

• For more details, please look at the source code and Javadocs. Studying existing readers is probably the best way to get a feel for the API; we would recommend first looking at lociformatsin.ImarisReader (this is the most straightforward one). lociformatsin.LIFReader and lnCellReader are also good references that show off some of the nicer features of Bio-Formats.

If you have questions about Bio-Formats, please contact the forums.
3.4.5 Adding format/reader documentation pages

Documentation pages for the supported formats and readers are auto-generated during the build. This page explains how to amend/extend this part of the Bio-Formats documentation.

Formats

The supported formats pages are generated as part of the documentation build using **maven**. The metadata for each format is contained in `format-pages.txt` and the build generates a reStructuredText file for each format as well as an index page for all supported formats using **Velocity**.

The `format-pages.txt` is an INI file where each section corresponds to a particular format given by the section header. Multiple key/values should be defined for each section:

<table>
<thead>
<tr>
<th>Key</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pagename</td>
<td>The name of the output reStructuredText file. If unspecified, the section header will be used to generate the filename.</td>
</tr>
<tr>
<td>extensions</td>
<td>The list of extensions supported for the format</td>
</tr>
<tr>
<td>owner</td>
<td>The owner of the file format</td>
</tr>
<tr>
<td>developer</td>
<td>The developer of the file format</td>
</tr>
<tr>
<td>bsd</td>
<td>A <code>yes/no</code> flag specifying whether the format readers/writers are under the BSD license</td>
</tr>
<tr>
<td>versions</td>
<td>A comma-separated list of all versions supported for this format</td>
</tr>
<tr>
<td>weHave</td>
<td>A bullet-point list describing the supporting material we have for this format including specification and sample datasets</td>
</tr>
<tr>
<td>weWant</td>
<td>A bullet-point list describing the supporting material we would like to have for this format</td>
</tr>
<tr>
<td>pixelRating</td>
<td></td>
</tr>
<tr>
<td>metadataRating</td>
<td></td>
</tr>
<tr>
<td>opennessRating</td>
<td></td>
</tr>
<tr>
<td>presenceRating</td>
<td></td>
</tr>
<tr>
<td>utilityRating</td>
<td></td>
</tr>
</tbody>
</table>

See :term:`Ratings legend and definitions`. Available choices are:

`Poor`, `Fair`, `Good`, `Very Good`, `Outstanding`.

The `metadataRating` should be set as follows:

* base `metadataRating` is the smaller of `opennessRating` and `pixelsRating`
* increment `metadataRating` by 1 if any combination of `Instrument.ID` plus `Image`.  

(continues on next page)
InstrumentRef', or 'Channel.EmissionWavelength', or 'Channel.ExcitationWavelength' are supported
* increment `metadataRating` by 1 if any SPW metadata is supported or
*:term:`pyramid` is ```yes```

reader
A string or a comma-separated list of all readers for this format

mif
set to ```true``` if the format can store multiple Images (in OME-XML
terminology) or series (in Bio-Formats API terminology)

pyramid
set to ```yes``` if the format can store a single image at multiple
resolutions

notes
Additional relevant information e.g. that we cannot distribute
specification documents to third parties

options
A link to additional reader and writer options documentation where they
are available for the format

**Dataset structure table**

The summary table listing the extensions for each reader is generated by the build process reading their extensions and
descriptions from all Bio-Formats readers (BSD and GPL). A reStructuredText file with a table summary of all file
extensions is created.

**Readers**

The metadata pages for each reader are generated by the build process parsing their metadata support from all Bio-
Formats readers (BSD and GPL). An intermediate meta-support.txt file is created which is then converted into one
reStructuredText page for each reader as well as a metadata summary reStructuredText file using Velocity.

### 3.4.6 Bio-Formats service and dependency infrastructure

**Description**

The Bio-Formats service infrastructure is an interface driven pattern for dealing with external and internal dependen-
cies. The design goal was mainly to avoid the cumbersome usage of ReflectedUniverse where possible and to
clearly define both service dependency and interface between components. This is generally referred to as dependency
injection, dependency inversion or component based design.

It was decided, at this point, to forgo the usage of potentially more powerful but also more complicated solutions such as:

- Spring
- Guice

3.4. Contributing to Bio-Formats
The Wikipedia page for dependency injection contains many other implementations in many languages. An added benefit is the potential code reuse possibilities as a result of decoupling of dependency and usage in Bio-Formats readers. Implementations of the initial Bio-Formats services were completed as part of BioFormatsCleanup and tickets #463 and #464.

Writing a service

- **Interface** – The basic form of a service is an interface which inherits from `loci.common.services.Service`. Here is a very basic example using the (now removed) `OMENotesService`

```java
public interface OMENotesService extends Service {
    /**
     * Creates a new OME Notes instance.
     * @param filename Path to the file to create a Notes instance for.
     */
    public void newNotes(String filename);
}
```

- **Implementation** – This service then has an implementation, which is usually located in the Bio-Formats component or package which imports classes from an external, dynamic or other dependency. Again looking at the `OMENotesService`:

```java
public class OMENotesServiceImpl extends AbstractService implements OMENotesService {
    /**
     * Default constructor.
     */
    public OMENotesServiceImpl() {
        checkClassDependency(Notes.class);
    }

    /* (non-Javadoc)
     * @see loci.formats.dependency.OMENotesService#newNotes()
     */
    public void newNotes(String filename) {
        new Notes(null, filename);
    }
}
```

- **Style**
  - Extension of `AbstractService` to enable uniform runtime dependency checking is recommended. Java does not check class dependencies until classes are first instantiated so if you do not do this, you may end up with `ClassNotFoundException` or the like exceptions being emitted from your service methods. This is to be **strongly** discouraged. If a service has unresolvable classes on its CLASSPATH instantiation should fail, not service method invocation.
  - Service methods should not burden the implementer with numerous checked exceptions. Also external dependency exception instances should not be allowed to directly leak from a service interface. Please
Bio-Formats, Release 6.14.0

wrap these using a ServiceException.
– By convention both the interface and implementation are expected to be in a package named loci.*.
services. This is not a hard requirement but should be followed where possible.
• Registration – A service’s interface and implementation must finally be registered with the
loci.common.services.ServiceFactory via the services.properties file. Following the OMENotesService
again, here is an example registration:
...
# OME notes service (implementation in legacy ome-notes component)
loci.common.services.OMENotesService=loci.ome.notes.services.OMENotesServiceImpl
...

Using a service
OMENotesService service = null;
try {
ServiceFactory factory = new ServiceFactory();
service = factory.getInstance(OMENotesService.class);
}
catch (DependencyException de) {
LOGGER.info("", de);
}
...

3.4.7 Scripts for performing development tasks
The tools directory contains several scripts which are useful for building and performing routine updates to the code
base.
bump_maven_version.py
This updates the Maven POM version numbers for all pom.xml files that set groupId to ome. The script takes a single
argument, which is the new version. For example, to update the POM versions prior to release:
./tools/bump_maven_version.py 5.1.0
and to switch back to snapshot versions immediately after release:
./tools/bump_maven_version.py 5.1.1-SNAPSHOT

3.4. Contributing to Bio-Formats

97


test-build

This is the script used by Travis to test each commit. It compiles and runs tests on each of the components in the Bio-Formats repository according to the arguments specified. Valid arguments are:

- `clean`: cleans the Maven build directories
- `maven`: builds all Java components using Maven and runs unit tests
- `ant`: builds all Java components using Ant and runs unit tests
- `all`: equivalent of `clean maven ant`

update_copyright

This updates the end year in the copyright blocks of all source code files. The command takes no arguments, and sets the end year to be the current year. As `update_copyright` is a Bash script, it is not intended to be run on Windows.

See open Trac tickets for Bio-Formats and the various Trello boards for information on work currently planned or in progress.

For more general guidance about how to contribute to OME projects, see the Contributing developers documentation.
Bio-Formats supports over 140 different file formats. The Dataset Structure Table explains the file extension you should choose to open/import a dataset in any of these formats, while the Supported Formats table lists all of the formats and gives an indication of how well they are supported and whether Bio-Formats can write, as well as read, each format. The Summary of supported metadata fields table shows an overview of the OME data model fields populated for each format.

Further information on each individual format is also provided (click on the entries in the supported formats table).

**We are always looking for examples of files to help us provide better support for different formats.** If you would like to help, you can upload files to Zenodo and let us know. If you have any questions, or would prefer not to use Zenodo, please contact us via the Image.sc forum. If your format is already supported, please refer to the ‘we would like to have’ section on the individual page for that format, to see if your dataset would be useful to us.

All the example files we have permission to share publicly are freely available from our sample image downloads site.

### 4.1 Dataset Structure Table

This table shows the extension of the file that you should choose if you want to open/import a dataset in a particular format.

<table>
<thead>
<tr>
<th>Format name</th>
<th>File to choose</th>
<th>Structure of files</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIM</td>
<td>.aim</td>
<td>Single file</td>
</tr>
<tr>
<td>ARF</td>
<td>.arf</td>
<td>Single file</td>
</tr>
<tr>
<td>Adobe Photoshop</td>
<td>.psd</td>
<td>Single file</td>
</tr>
<tr>
<td>Adobe Photoshop TIFF</td>
<td>.tiff, .tif</td>
<td>Single file</td>
</tr>
<tr>
<td>Alicona AL3D</td>
<td>.al3d</td>
<td>Single file</td>
</tr>
<tr>
<td>Amersham Biosciences GEL</td>
<td>.gel</td>
<td>Single file</td>
</tr>
<tr>
<td>Amira</td>
<td>.am, .amiramesh, .grey, .hx, .labels</td>
<td>Single file</td>
</tr>
<tr>
<td>Analyze 7.5</td>
<td>.img, .hdr</td>
<td>One .img file and one similarly-named file</td>
</tr>
<tr>
<td>Andor SIF</td>
<td>.sif</td>
<td>Single file</td>
</tr>
<tr>
<td>Animated PNG</td>
<td>.png</td>
<td>Single file</td>
</tr>
<tr>
<td>Aperio AFI</td>
<td>.afi</td>
<td>One .afi file and several similarly-named files</td>
</tr>
<tr>
<td>Aperio SVS</td>
<td>.svs</td>
<td>Single file</td>
</tr>
<tr>
<td>Audio Video Interleave</td>
<td>.avi</td>
<td>Single file</td>
</tr>
<tr>
<td>BD Pathway</td>
<td>.exp, .tif</td>
<td>Multiple files (.exp, .dye, .lt, .wm, .pdp)</td>
</tr>
<tr>
<td>BDV</td>
<td>.xml, .h5</td>
<td>Single file</td>
</tr>
<tr>
<td>Bio-Rad GEL</td>
<td>.lsc</td>
<td>Single file</td>
</tr>
<tr>
<td>Bio-Rad PIC</td>
<td>.pic, .xml, .raw</td>
<td>One or more .pic files and an .xml file</td>
</tr>
<tr>
<td>Bio-Rad SCN</td>
<td>.scn</td>
<td>Single file</td>
</tr>
<tr>
<td>Format name</td>
<td>File to choose</td>
<td>Structure of files</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>-----------------------------------------</td>
</tr>
<tr>
<td>Bitplane Imaris</td>
<td>.ims</td>
<td>Single file</td>
</tr>
<tr>
<td>Bitplane Imaris 3 (TIFF)</td>
<td>.ims</td>
<td>Single file</td>
</tr>
<tr>
<td>Bitplane Imaris 5.5 (HDF)</td>
<td>.ims</td>
<td>Single file</td>
</tr>
<tr>
<td>Bruker</td>
<td>(no extension)</td>
<td>One ‘fid’ and one ‘acqp’ plus other metadata files</td>
</tr>
<tr>
<td>Burleigh</td>
<td>.img</td>
<td>Single file</td>
</tr>
<tr>
<td>Canon RAW</td>
<td>.cr2, .crw, .jpg, .thm, .wav</td>
<td>Single file</td>
</tr>
<tr>
<td>CellH5 (HDF)</td>
<td>.ch5</td>
<td>Single file</td>
</tr>
<tr>
<td>CellSens VSI</td>
<td>.vsi, .ets</td>
<td>One .vsi file and an optional .ets file</td>
</tr>
<tr>
<td>CellVoyager</td>
<td>.tif, .xml</td>
<td>Directory with 2 master files</td>
</tr>
<tr>
<td>CellWorx</td>
<td>.pnl, .htd, .log</td>
<td>One .htd file plus one or more .log files</td>
</tr>
<tr>
<td>Cellomics C01</td>
<td>.c01, .dib</td>
<td>One or more .c01 files</td>
</tr>
<tr>
<td>Compix Simple-PCI</td>
<td>.cxd</td>
<td>Single file</td>
</tr>
<tr>
<td>DICOM</td>
<td>.dic, .dcm, .dicom, .jp2, .j2ki, .j2kr, .raw, .ima</td>
<td>One or more .dcm or .dicom files</td>
</tr>
<tr>
<td>DNG</td>
<td>.cr2, .crw, .jpg, .thm, .wav, .tif, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Deltabvision</td>
<td>.dv, .r3d, .r3d_d3d, .dv.log, .r3d.log</td>
<td>One .dv, .r3d, or .r3d_d3d file and an optional .log file</td>
</tr>
<tr>
<td>ECAT7</td>
<td>.v</td>
<td>Single file</td>
</tr>
<tr>
<td>Encapsulated PostScript</td>
<td>.eps, .epsi, .ps</td>
<td>Single file</td>
</tr>
<tr>
<td>Evotec Flex</td>
<td>.flex, .mea, .res</td>
<td>One directory containing one .flex file and one or more .mea or .res files</td>
</tr>
<tr>
<td>Extended leica file</td>
<td>.xlef</td>
<td>Single file</td>
</tr>
<tr>
<td>FEI TIFF</td>
<td>.tif, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>FEI/Philips</td>
<td>.img</td>
<td>Single file</td>
</tr>
<tr>
<td>Flexible Image Transport System</td>
<td>.fits, .fts</td>
<td>Single file</td>
</tr>
<tr>
<td>FlowSight</td>
<td>.cif</td>
<td>Single file</td>
</tr>
<tr>
<td>Fuji LAS 3000</td>
<td>.img, .inf</td>
<td>Single file</td>
</tr>
<tr>
<td>Gatan DM2</td>
<td>.dm2</td>
<td>Single file</td>
</tr>
<tr>
<td>Gatan Digital Micrograph</td>
<td>.dm3, .dm4</td>
<td>Single file</td>
</tr>
<tr>
<td>Graphics Interchange Format</td>
<td>.gif</td>
<td>Single file</td>
</tr>
<tr>
<td>Hamamatsu Aquacosmos</td>
<td>.naf</td>
<td>Single file</td>
</tr>
<tr>
<td>Hamamatsu HIS</td>
<td>.his</td>
<td>Single file</td>
</tr>
<tr>
<td>Hamamatsu NDPI</td>
<td>.ndpi</td>
<td>Single file</td>
</tr>
<tr>
<td>Hamamatsu NDPIS</td>
<td>.ndpis</td>
<td>One .ndpis file and at least one .jpg file</td>
</tr>
<tr>
<td>Hamamatsu VMS</td>
<td>.vms</td>
<td>One .vms file plus several .jpg files</td>
</tr>
<tr>
<td>Hitachi</td>
<td>.txt</td>
<td>One .txt file plus one similar .tif file</td>
</tr>
<tr>
<td>I2I</td>
<td>.i2i</td>
<td>Single file</td>
</tr>
<tr>
<td>IMAGIC</td>
<td>.hed, .img</td>
<td>One .hed file plus one similar .img file</td>
</tr>
<tr>
<td>IMOD</td>
<td>.mod</td>
<td>Single file</td>
</tr>
<tr>
<td>INR</td>
<td>.intr</td>
<td>Single file</td>
</tr>
<tr>
<td>IPLab</td>
<td>.ipl</td>
<td>Single file</td>
</tr>
<tr>
<td>IVision</td>
<td>.ipm</td>
<td>Single file</td>
</tr>
<tr>
<td>Imacon</td>
<td>.fff</td>
<td>Single file</td>
</tr>
<tr>
<td>Image Cytometry Standard</td>
<td>.ics, .ids</td>
<td>One .ics and possibly one .ids file</td>
</tr>
<tr>
<td>Image-Pro Sequence</td>
<td>.seq, .ips</td>
<td>Single file</td>
</tr>
<tr>
<td>Image-Pro Workspace</td>
<td>.ipw</td>
<td>Single file</td>
</tr>
<tr>
<td>Improvision TIFF</td>
<td>.tif, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>InCell 1000/2000</td>
<td>.xdce, .xml, .tiff, .tif, .xlog</td>
<td>One .xdce file with at least one .xml and .xlog file</td>
</tr>
<tr>
<td>InCell 3000</td>
<td>.frm</td>
<td>Single file</td>
</tr>
<tr>
<td>Inveon</td>
<td>.hdr</td>
<td>One .hdr file plus one similar .htd file</td>
</tr>
<tr>
<td>Ionpath MIBI</td>
<td>.tif, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>JEOL</td>
<td>.dat, .img, .par</td>
<td>A single .dat file or an .img file</td>
</tr>
<tr>
<td>Format name</td>
<td>File to choose</td>
<td>Structure of files</td>
</tr>
<tr>
<td>---------------------------------------</td>
<td>---------------------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>JPEG</td>
<td>.jpg, .jpeg, .jpe</td>
<td>Single file</td>
</tr>
<tr>
<td>JPEG-2000</td>
<td>.jp2, .j2k, .jpf</td>
<td>Single file</td>
</tr>
<tr>
<td>JPX Instruments</td>
<td>.jp</td>
<td>Single file</td>
</tr>
<tr>
<td>KPB Instruments</td>
<td>.klb</td>
<td>Single file</td>
</tr>
<tr>
<td>Khoros XV</td>
<td>.xv</td>
<td>Single file</td>
</tr>
<tr>
<td>Kodak Molecular Imaging</td>
<td>.bip</td>
<td>Single file</td>
</tr>
<tr>
<td>LEO</td>
<td>.sxm, .tif, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>LI-FLIM</td>
<td>.fli</td>
<td>Single file</td>
</tr>
<tr>
<td>Laboratory Imaging</td>
<td>.lim</td>
<td>Single file</td>
</tr>
<tr>
<td>Lavision Inspector</td>
<td>.msr</td>
<td>Single file</td>
</tr>
<tr>
<td>Leica</td>
<td>.lei, .tif, .tiff, .raw</td>
<td>One .lei file with at least one .tif or .tiff file</td>
</tr>
<tr>
<td>Leica Image File Format</td>
<td>.tif</td>
<td>Single file</td>
</tr>
<tr>
<td>Leica Object Format</td>
<td>.lof</td>
<td>Single file</td>
</tr>
<tr>
<td>Leica SCN</td>
<td>.scn</td>
<td>Single file</td>
</tr>
<tr>
<td>Leica TCS TIFF</td>
<td>.tiff, .tif, .xml</td>
<td>Single file</td>
</tr>
<tr>
<td>Li-Cor L2D</td>
<td>.l2d, .scn, .tiff</td>
<td>One .l2d file with one or more .tiff files</td>
</tr>
<tr>
<td>MIAS</td>
<td>.tif, .tiff, .txt</td>
<td>One directory per plate containing one or more .tif files</td>
</tr>
<tr>
<td>MINC MRI</td>
<td>.mnc</td>
<td>Single file</td>
</tr>
<tr>
<td>Medical Research Council</td>
<td>.mrc, .st, .ali, .map, .rec, .mrcs</td>
<td>Single file</td>
</tr>
<tr>
<td>MetaXpress TIFF</td>
<td>.htd, .tif</td>
<td>One .htd file plus one or more .tif files</td>
</tr>
<tr>
<td>Metamorph STK</td>
<td>.stk, .nd, .scan, .tiff, .tiff</td>
<td>One or more .stk or .tiff files</td>
</tr>
<tr>
<td>Metamorph TIFF</td>
<td>.tiff, .tif</td>
<td>One or more .tiff files</td>
</tr>
<tr>
<td>Micro-Manager</td>
<td>.tiff, .tiff, .txt, .xml</td>
<td>A file ending in .metadata.txt</td>
</tr>
<tr>
<td>MicroCT</td>
<td>.viff</td>
<td>Directory with XML file and one or more .tif files</td>
</tr>
<tr>
<td>Mikroscan TIFF</td>
<td>.tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Miniolta MRW</td>
<td>.mrv</td>
<td>Single file</td>
</tr>
<tr>
<td>Molecular Imaging</td>
<td>.stp</td>
<td>Single file</td>
</tr>
<tr>
<td>Multiple-image Network Graphics</td>
<td>.mng</td>
<td>Single file</td>
</tr>
<tr>
<td>NIfTI</td>
<td>.nii, .img, .hdr, .nii.gz</td>
<td>A single .nii file or a single .hdr file</td>
</tr>
<tr>
<td>NOAA-HRD Gridded Data Format</td>
<td>(no extension)</td>
<td>Single file</td>
</tr>
<tr>
<td>NRRD</td>
<td>.nrrd, .nhdr</td>
<td>A single .nrrd file or one .nhdr file</td>
</tr>
<tr>
<td>Nikon Elements TIFF</td>
<td>.tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Nikon ND2</td>
<td>.nd2</td>
<td>Single file</td>
</tr>
<tr>
<td>Nikon NEF</td>
<td>.nef, .tiff, .tif</td>
<td>Single file</td>
</tr>
<tr>
<td>Nikon TIFF</td>
<td>.tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>OBF</td>
<td>.obf, .msr</td>
<td>OBF file</td>
</tr>
<tr>
<td>OME-TIFF</td>
<td>.ome.tiff, .ome.tif, .ome.tiff2, .ome.tiff8, .ome.btf, .companion.ome</td>
<td>One or more .ome.tiff files</td>
</tr>
<tr>
<td>OME-XML</td>
<td>.ome, .ome.xml</td>
<td>Single file</td>
</tr>
<tr>
<td>Olympus .omp2info</td>
<td>.omp2info</td>
<td>One .omp2info file and at least one .tiff file</td>
</tr>
<tr>
<td>Olympus APL</td>
<td>.apl, .tnb, .mtb, .tif</td>
<td>One .apl file, one .mtb file, one .tif file</td>
</tr>
<tr>
<td>Olympus FV1000</td>
<td>.oib, .oif, .pt, .lut</td>
<td>Single .oib file or one .oif file</td>
</tr>
<tr>
<td>Olympus Fluoview/ABD TIFF</td>
<td>.tiff, .tif</td>
<td>One or more .tiff, .tif files, and one .oib file</td>
</tr>
<tr>
<td>Olympus OIR</td>
<td>.oir</td>
<td>Single file</td>
</tr>
<tr>
<td>Olympus SIS TIFF</td>
<td>.tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Olympus ScanR</td>
<td>.dat, .xml, .tif</td>
<td>One .xml file, one ‘data’ directory</td>
</tr>
<tr>
<td>Olympus Slidebook</td>
<td>.sld, .spl</td>
<td>Single file</td>
</tr>
<tr>
<td>Openlab LIFF</td>
<td>.liff</td>
<td>Single file</td>
</tr>
<tr>
<td>Openlab RAW</td>
<td>.raw</td>
<td>Single file</td>
</tr>
<tr>
<td>Format name</td>
<td>File to choose</td>
<td>Structure of files</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>----------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Oxford Instruments</td>
<td>.top</td>
<td>Single file</td>
</tr>
<tr>
<td>PCO-RAW</td>
<td>.pcoraw, .rec</td>
<td>A single .pcoraw file with a similarly-named .rec file</td>
</tr>
<tr>
<td>PCX</td>
<td>.pcx</td>
<td>Single file</td>
</tr>
<tr>
<td>PICT</td>
<td>.pict, .pct</td>
<td>Single file</td>
</tr>
<tr>
<td>POV-Ray</td>
<td>.df3</td>
<td>Single file</td>
</tr>
<tr>
<td>Perkin Elmer Densitometer</td>
<td>.hdr, .img</td>
<td>One .hdr file and a similarly-named .img file</td>
</tr>
<tr>
<td>Perkin-Elmer Nuance IM3</td>
<td>.im3</td>
<td>Single file</td>
</tr>
<tr>
<td>PerkinElmer</td>
<td>.ano, .cfg, .csv, .htm, .rec, .tim, .zpo, .tif</td>
<td>One .htm file, several other metadata files, and .tif files or .2, .3, .4, etc. files</td>
</tr>
<tr>
<td>PerkinElmer Columbus</td>
<td>.xml</td>
<td>Directory with XML file and one .tif file per plane</td>
</tr>
<tr>
<td>PerkinElmer Operetta</td>
<td>.tif, .tiff, .xml</td>
<td>Directory with XML file and one .tif file per plane</td>
</tr>
<tr>
<td>PerkinElmer Vectra/QPTIFF</td>
<td>.tif, .tiff, .qptiff</td>
<td>Single file</td>
</tr>
<tr>
<td>PicoQuant Bin</td>
<td>.bin</td>
<td>Single file</td>
</tr>
<tr>
<td>Portable Any Map</td>
<td>.pbm, .pgm, .ppm</td>
<td>Single file</td>
</tr>
<tr>
<td>Prairie TIFF</td>
<td>.tif, .tiff, .cfg, .env, .xml</td>
<td>One .xml file, one .cfg file, and one .tif file per plane</td>
</tr>
<tr>
<td>Princeton Instruments SPE</td>
<td>.spe</td>
<td>Single file</td>
</tr>
<tr>
<td>Pyramid TIFF</td>
<td>.tiff, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Quesant AFM</td>
<td>.afm</td>
<td>Single file</td>
</tr>
<tr>
<td>QuickTime</td>
<td>.mov</td>
<td>Single file</td>
</tr>
<tr>
<td>RCPNL</td>
<td>.rcpnl</td>
<td>One .dv, .r3d, or .d3d file and a similarly-named .dat file</td>
</tr>
<tr>
<td>RHK Technologies</td>
<td>.sm2, .sm3</td>
<td>Single file</td>
</tr>
<tr>
<td>SBIG</td>
<td>(no extension)</td>
<td>Single file</td>
</tr>
<tr>
<td>SM Camera</td>
<td>(no extension)</td>
<td>Single file</td>
</tr>
<tr>
<td>SPC FIFO Data</td>
<td>.spc, .set</td>
<td>One .spc file and similarly-named .set file</td>
</tr>
<tr>
<td>SPC Image Data</td>
<td>.sdt</td>
<td>Single file</td>
</tr>
<tr>
<td>SPIDER</td>
<td>.spi</td>
<td>Single file</td>
</tr>
<tr>
<td>Seiko</td>
<td>.xqd, .xqf</td>
<td>Single file</td>
</tr>
<tr>
<td>SimplePCI TIFF</td>
<td>.tiff, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Simulated data</td>
<td>.fake</td>
<td>Single file</td>
</tr>
<tr>
<td>SlideBook 7 SLD (native)</td>
<td>.sldy</td>
<td>Single file</td>
</tr>
<tr>
<td>Slidebook TIFF</td>
<td>.tiff, .tiff</td>
<td>Single file</td>
</tr>
<tr>
<td>Tagged Image File Format</td>
<td>.tif, .tiff, .tf2, .tf8, .btf</td>
<td>Single file</td>
</tr>
<tr>
<td>Tecan Spark Cyto</td>
<td>.db</td>
<td>SQLite database, TIFF files, etc.</td>
</tr>
<tr>
<td>Text</td>
<td>.txt, .csv</td>
<td>Single file</td>
</tr>
<tr>
<td>TillVision</td>
<td>.vws, .pst, .inf</td>
<td>One .vws file and possibly one .vii file</td>
</tr>
<tr>
<td>TopoMetrix</td>
<td>.tff, .tfr, .zfr, .zfp, .2fl</td>
<td>Single file</td>
</tr>
<tr>
<td>Trestle</td>
<td>.tif</td>
<td>One .tif file plus several other files</td>
</tr>
<tr>
<td>Truevision Targa</td>
<td>.tga</td>
<td>Single file</td>
</tr>
<tr>
<td>UBM</td>
<td>.pr3</td>
<td>Single file</td>
</tr>
<tr>
<td>Unisoku STM</td>
<td>.hdr, .dat</td>
<td>One .HDR file plus one similarly-named .dat file</td>
</tr>
<tr>
<td>VG SAM</td>
<td>.dti</td>
<td>Single file</td>
</tr>
<tr>
<td>Varian FDF</td>
<td>.fdf</td>
<td>Single file</td>
</tr>
<tr>
<td>Veeco</td>
<td>.hdf</td>
<td>Single file</td>
</tr>
<tr>
<td>Ventana .bif</td>
<td>.bif</td>
<td>Single file</td>
</tr>
<tr>
<td>Visitech XYS</td>
<td>.xys, .html</td>
<td>One .html file plus one or more .xml files</td>
</tr>
<tr>
<td>Volocity Library</td>
<td>.mvd2, .aisf, .aiix, .dat, .atsf</td>
<td>One .mvd2 file plus a .dat file</td>
</tr>
<tr>
<td>Volocity Library Clipping</td>
<td>.acff</td>
<td>Single file</td>
</tr>
<tr>
<td>WA Technology TOP</td>
<td>.wat</td>
<td>Single file</td>
</tr>
<tr>
<td>Windows Bitmap</td>
<td>.bmp</td>
<td>Single file</td>
</tr>
<tr>
<td>Woolz</td>
<td>.wlz</td>
<td>Single file</td>
</tr>
</tbody>
</table>
### 4.1.1 Flex Support

OMERO.importer supports importing analyzed Flex files from an Opera system.

Basic configuration is done via the `importer.ini`. Once the user has run the Importer once, this file will be in the following location:

- `C:\Documents and Settings\<username>\omero\importer.ini`

The user will need to modify or add the `[FlexReaderServerMaps]` section of the INI file as follows:

```ini
[FlexReaderServerMaps]
CIA-1 = \\hostname1\mount;\\archivehost1\mount
CIA-2 = \\hostname2\mount;\\archivehost2\mount
```

where the key of the INI file line is the value of the “Host” tag in the .mea measurement XML file (here: `<Host name=“CIA-1”>`) and the value is a semicolon-separated list of escaped UNC path names to the Opera workstations where the Flex files reside.

Once this resolution has been encoded in the configuration file and you have restarted the importer, you will be able to select the .mea measurement XML file from the Importer user interface as the import target.

### 4.2 Supported Formats

**Ratings legend and definitions**

<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>3i SlideBook</td>
<td>.sld</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>3i SlideBook 7</td>
<td>.sldy</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td>Andor Bio-Imaging Division (ABD) TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AIM</td>
<td>.aim</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alicona 3D</td>
<td>.al3d</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amersham Biosciences Gel</td>
<td>.gel</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amira Mesh</td>
<td>.am, .ami, .amesh, .grey, .hx, .labels</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Amnis FlowSight</td>
<td>.cif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analyze 7.5</td>
<td>.img, .hdr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Andor SIF</td>
<td>.sif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Animated PNG</td>
<td>.png</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aperio AFI</td>
<td>.afi, .svs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Aperio SVS TIFF</td>
<td>.svs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Applied Precision Cell-WorX</td>
<td>.htd, .pnl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AVI (Audio Video Interleave)</td>
<td>.avi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Axon Raw Format</td>
<td>.arf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BD Pathway</td>
<td>.exp, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Becker &amp; Hickl SPC FIFO</td>
<td>.spc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Becker &amp; Hickl SPCellImage</td>
<td>.sdt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Big Data Viewer</td>
<td>.xml, .h5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bio-Rad Gel</td>
<td>.1sc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bio-Rad PIC</td>
<td>.pic, .raw, .xml</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bio-Rad SCN</td>
<td>.scn</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bitplane Imaris</td>
<td>.ims</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bruker MRI</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Burleigh</td>
<td>.img</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Canon DNG</td>
<td>.cr2, .crw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CellH5</td>
<td>.ch5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cellomics</td>
<td>.c01, .dib</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cellSens VSI</td>
<td>.vsi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CellVoyager</td>
<td>.xml, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CV7000</td>
<td>.wpi, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DeltaVision</td>
<td>.dv, .rt3d, .rcpnl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DICOM</td>
<td>.dcm, .dicom</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2 – continued from previous page.

continues on next page
Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECAT7</td>
<td>.v</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPS (Encapsulated PostScript)</td>
<td>.eps, .eps, .ps</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evotec/PerkinElmer Opera Flex</td>
<td>.flex, .mea, .res</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEI</td>
<td>.img</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FEI TIFF</td>
<td>.tiff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FITS (Flexible Image Transport System)</td>
<td>.fits</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gatan Digital Micrograph</td>
<td>.dm3, .dm4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gatan Digital Micrograph 2</td>
<td>.dm2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GE MicroCT</td>
<td>.vff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GIF (Graphics Interchange Format)</td>
<td>.gif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamamatsu Aquacosmos NAF</td>
<td>.naf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamamatsu HIS</td>
<td>.his</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamamatsu ndpi</td>
<td>.ndpi, .ndpis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hamamatsu VMS</td>
<td>.vms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hitachi S-4800</td>
<td>.txt, .tif, .bmp, .jpg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>I2I</td>
<td>.i2i</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICS (Image Cytometry Standard)</td>
<td>.ics, .ids</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Imacon</td>
<td>.iff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImagePro Sequence</td>
<td>.seq</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ImagePro Workspace</td>
<td>.ipw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMAGIC</td>
<td>.hed, .img</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IMOD</td>
<td>.mod</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improvision Openlab LIFF</td>
<td>.liff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improvision Openlab Raw</td>
<td>.raw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Improvision TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Inspector OBF</td>
<td>.obf, .msr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Opennes</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>InCell 1000/2000</td>
<td>.xdce, .tif</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>InCell 3000</td>
<td>.frm</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>INR</td>
<td>.inr</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Inveon</td>
<td>.hdr</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Ionpath MIBI</td>
<td>.tif, .tiff</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>IPLab</td>
<td>.ipl</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>IVision</td>
<td>.ipm</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>JEOL</td>
<td>.dat, .img, .par</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>JPEG</td>
<td>.jpg</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>JPEG 2000</td>
<td>.jp2, .j2k, .jpf</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>JPK</td>
<td>.jpk</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>JPX</td>
<td>.jpx</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Keller Lab Block</td>
<td>.klb</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Khoros VIFF (Visualization Image File Format) Bitmap</td>
<td>.xv</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Kodak BIP</td>
<td>.bip</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Lambert Instruments FLIM</td>
<td>.fli</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>LaVision Inspector</td>
<td>.msr</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Leica LCS LEI</td>
<td>.lei, .tif</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Leica LAS AF LIF (Leica Image File Format)</td>
<td>.lif</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Leica LOF</td>
<td>.lof</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Leica SCN</td>
<td>.scn</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Leica XLEF</td>
<td>.xlef</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>LEO</td>
<td>.sxm, .tif, .tiff</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Li-Cor L2D</td>
<td>.l2d, .tif, .scn</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>LIM (Laboratory Imaging/Nikon)</td>
<td>.lim</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>MetaMorph 7.5 TIFF</td>
<td>.tiff</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>MetaMorph Stack (STK)</td>
<td>.stk, .nd</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>MetaXpress</td>
<td>.htd, .tif, .tiff</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>MIAS (Maia Scientific)</td>
<td>.tif</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
<tr>
<td>Micro-Manager</td>
<td>.tif, .txt, .xml</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
<td>☑️</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mikroscan TIFF</td>
<td>.tif, .tiff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MINC MRI</td>
<td>.mnc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Minolta MRW</td>
<td>.mrw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MNG (Multiple-image Network Graphics)</td>
<td>.mng</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Molecular Imaging</td>
<td>.stp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MRC (Medical Research Council)</td>
<td>.mrc, .st, .ali, .map, .rec, .mrsc</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NEF (Nikon Electronic Format)</td>
<td>.nef, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NIfTI</td>
<td>.img, .hdr, .nii, .nii.gz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nikon Elements TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nikon EZ-C1 TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Nikon NIS-Elements ND2</td>
<td>.nd2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NRRD (Nearly Raw Raster Data)</td>
<td>.nrrd, .nhdr, .raw, .txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus CellR/APL</td>
<td>.apl, .mtb, .tnb, .tif, .obsep</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus FluoView FV1000</td>
<td>.oib, .oif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus FluoView TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus OIR</td>
<td>.oir</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus OMP2INFO</td>
<td>.omp2info</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus ScanR</td>
<td>.xml, .dat, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olympus SIS TIFF</td>
<td>.tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OME-TIFF</td>
<td>.ome.tif, .ome.tif, .ome.tf2, .ome.tf8, .ome.btf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OME-XML</td>
<td>.ome, .ome.xml</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OMERo Pyramid</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oxford Instruments</td>
<td>.top</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCORAW</td>
<td>.pcoraw, .rec</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PCX (PC Paintbrush)</td>
<td>.pcx</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2. Supported Formats

continues on next page
### Table 2 – continued from previous page

<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perkin Elmer Densitometer</td>
<td>.pds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerkinElmer Columbus</td>
<td>.xml, .csv, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerkinElmer Nuance</td>
<td>.im3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerkinElmer Operetta</td>
<td>.tif, .xml</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PerkinElmer UltraVIEW</td>
<td>.tif, .2, .3, .4, etc.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Portable Any Map</td>
<td>.pnm, .pgm, .ppm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adobe Photoshop PSD</td>
<td>.psd</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Photoshop TIFF</td>
<td>.tif, .tiff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PicoQuant Bin</td>
<td>.bin</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PICT (Macintosh Picture)</td>
<td>.pict</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PNG (Portable Network Graphics)</td>
<td>.png</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prairie Technologies TIFF</td>
<td>.tif, .xml, .cfg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Princeton Instruments SPE</td>
<td>.spe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quesant</td>
<td>.afm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>QuickTime Movie</td>
<td>.mov</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RHK</td>
<td>.sm2, .sm3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBIG</td>
<td>.spe</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Seiko</td>
<td>.qkd, .qxf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimplePCI &amp; HCImage</td>
<td>.cds</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimplePCI &amp; HCImage TIFF</td>
<td>.tiff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SM Camera</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SPIDER</td>
<td>.spi, .stk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Targa</td>
<td>.tga</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Tecan Spark Cyto Workspace</td>
<td>.db, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td>.txt</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TIFF (Tagged Image File Format)</td>
<td>.tiff, .tif, .tf2, .tf8, .bif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Format</th>
<th>Extensions</th>
<th>Pixels</th>
<th>Metadata</th>
<th>Openness</th>
<th>Presence</th>
<th>Utility</th>
<th>Export</th>
<th>BSD</th>
<th>Multiple Images</th>
<th>Pyramid</th>
</tr>
</thead>
<tbody>
<tr>
<td>TillPhotonics TillVision</td>
<td>.vws</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Topometrix</td>
<td>.tfr, .frr, .zfr, .zfp, .2fl</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Trestle</td>
<td>.tif, .sld, .jpg</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>UBM</td>
<td>.pr3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unisoka</td>
<td>.dat, .hdr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Varian FDF</td>
<td>.fdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vectra OPTIFF</td>
<td>.tif, .qptiff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Veeco AFM</td>
<td>.hdf</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ventana BIF</td>
<td>.bif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VG SAM</td>
<td>.dti</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VisiTech XYS</td>
<td>.xys, .html</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volocity</td>
<td>.mvd2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volocity Library Clipping</td>
<td>.acff</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WA-TOP</td>
<td>.wat</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows Bitmap</td>
<td>.bmp</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Woolz</td>
<td>.wlz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeiss Axio CSM</td>
<td>.lms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeiss AxioVision TIFF</td>
<td>.xml, .tif</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeiss AxioVision ZVI (Zeiss Vision Image)</td>
<td>.zvi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeiss CZI</td>
<td>.czi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zeiss LSM (Laser Scanning Microscope) 510/710</td>
<td>.lsm, .mdb</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Bio-Formats currently supports **162** formats

**Ratings legend and definitions**

<table>
<thead>
<tr>
<th></th>
<th>Outstanding</th>
<th>Very good</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

4.2. Supported Formats
Bio-Formats, Release 6.14.0

Pixels
Our estimation of Bio-Formats’ ability to reliably extract complete and accurate pixel values from files in that format. The better this score, the more confident we are that Bio-Formats will successfully read your file without displaying an error message or displaying an erroneous image.

Metadata
Our certainty in the thoroughness and correctness of Bio-Formats’ metadata extraction and conversion from files of that format into standard OME-XML. The better this score, the more confident we are that all meaningful metadata will be parsed and populated as OME-XML.

Openness
This is not a direct expression of Bio-Formats’ performance, but rather indicates the level of cooperation the format’s controlling interest has demonstrated toward the scientific community with respect to the format. The better this score, the more tools (specification documents, source code, sample files, etc.) have been made available.

Presence
This is also not directly related to Bio-Formats, but instead represents our understanding of the format’s popularity, and is also as a measure of compatibility between applications. The better this score, the more common the format and the more software packages include support for it.

Utility
Our opinion of the format’s suitability for storing metadata-rich microscopy image data. The better this score, the wider the variety of information that can be effectively stored in the format.

Export
This indicates whether Bio-Formats is capable of writing the format (Bio-Formats can read every format on this list).

BSD
This indicates whether format is BSD-licensed. By default, format readers and writers are GPL-licensed.

Multiple Images
This indicates whether the format can store multiple Images (in OME-XML terminology) or series (in Bio-Formats API terminology).

Pyramid
This indicates whether the format can store a single image at multiple resolutions, typically referred to as an image pyramid.

4.2.1 3i SlideBook
Extensions: .sld
Developer: Intelligent Imaging Innovations
Owner: Intelligent Imaging Innovations
Support
BSD-licensed: ❌
Export: ❌
Officially Supported Versions: 4.1, 4.2, 5.0, 5.5, 6.0
Reader: SlidebookReader (Source Code, Supported Metadata Fields)
We currently have:
• Numerous SlideBook datasets
We would like to have:

- A SlideBook specification document
- More SlideBook datasets (preferably acquired with the most recent SlideBook software)

**Ratings**

Pixels: ▲

Metadata: ▼

Openness: ▼

Presence: ▲

Utility: ▼

**Additional Information**

We strongly encourage users to export their .sld files to OME-TIFF using the SlideBook software. Bio-Formats is not likely to support the full range of metadata that is included in .sld files, and so exporting to OME-TIFF from SlideBook is the best way to ensure that all metadata is preserved. Free software from 3i can export the files to OME-TIFF post-acquisition, see https://www.intelligent-imaging.com/slidebook.

3i also develops a native SlideBook reader which works with Bio-Formats. See http://www.openmicroscopy.org/info/slidebook for details.

**See also:**

Slidebook software overview

### 4.2.2 3i SlideBook 7

Extensions: .sldy

Developer: Intelligent Imaging Innovations

Owner: Intelligent Imaging Innovations

**Support**

BSD-licensed: ✔

Export: ✗

Officially Supported Versions:

Reader: SlideBook7Reader (Source Code, Supported Metadata Fields)

Freely Available Software:

- SlideBook

We currently have:

- a small number of datasets

**Ratings**

Pixels: ▲

Metadata: ▲

Openness: ▲
Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

### 4.2.3 Andor Bio-Imaging Division (ABD) TIFF

Extensions: .tif

Developer: Andor Bioimaging Department

Owner: Andor Technology

**Support**

BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: FluoviewReader *(Source Code, Supported Metadata Fields)*

We currently have:

- an ABD-TIFF specification document (from 2005 November, in PDF)
- a few ABD-TIFF datasets

**Ratings**

Pixels: ▲

Metadata: ▲

Openness: ▣

Presence: ▽

Utility: ▣

**Additional Information**

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

With a few minor exceptions, the ABD-TIFF format is identical to the Fluoview TIFF format.

### 4.2.4 AIM

Extensions: .aim

Developer: SCANCO Medical AG

**Support**

BSD-licensed: ✗

Export: ✗
Officially Supported Versions:

**Reader:** AIMReader *(Source Code, Supported Metadata Fields)*

We currently have:
- one .aim file

We would like to have:
- an .aim specification document
- more .aim files

**Ratings**

- Pixels: 
- Metadata: 
- Openness: 
- Presence: 
- Utility: 

### 4.2.5 Alicona 3D

**Extensions:** .al3d

**Owner:** Alicona Imaging

**Support**

- BSD-licensed: 
- Export: 

Officially Supported Versions: 1.0

**Reader:** AliconaReader *(Source Code, Supported Metadata Fields)*

We currently have:
- an AL3D specification document (v1.0, from 2003, in PDF)
- a few AL3D datasets

We would like to have:
- more AL3D datasets (Z series, T series, 16-bit)

**Ratings**

- Pixels: 
- Metadata: 
- Openness: 
- Presence: 
- Utility: 

**Additional Information**

Known deficiencies:
• Support for 16-bit AL3D images is present, but has never been tested.
• Texture data is currently ignored.

### 4.2.6 Amersham Biosciences Gel

Extensions: .gel
Developer: Molecular Dynamics
Owner: Cytiva

**Support**

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: GelReader ([Source Code, Supported Metadata Fields](#))

We currently have:

• a GEL specification document (Revision 2, from 2001 Mar 15, in PDF)
• a few GEL datasets

**Ratings**

Pixels: ↑
Metadata: ▲
Openness: ●
Presence: ▼
Utility: ▼

**Additional Information**

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

See also:

GEL Technical Overview

### 4.2.7 Amira Mesh

Extensions: .am, .amiramesh, .grey, .hx, .labels
Developer: Visage Imaging

**Support**

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: AmiraReader ([Source Code, Supported Metadata Fields](#))
We currently have:
  • a few Amira Mesh datasets
  • public sample images
We would like to have:
  • more Amira Mesh datasets

Ratings
Pixels: ▶
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.8 Amnis FlowSight

Extensions: .cif
Owner: Amnis (now owned by Merck)

Support
BSD-licensed: ✔
Export: ❌

Officially Supported Versions:
Reader: FlowSightReader (Source Code, Supported Metadata Fields)
We currently have:
  • a few sample datasets

Ratings
Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼
4.2.9 Analyze 7.5

Extensions: .img, .hdr
Developer: Mayo Foundation Biomedical Imaging Resource

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: AnalyzeReader (Source Code, Supported Metadata Fields)

We currently have:
• an Analyze 7.5 specification document
• several Analyze 7.5 datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.10 Andor SIF

Extensions: .sif
Developer: Andor Bioimaging Department
Owner: Andor Technology

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: SIFReader (Source Code, Supported Metadata Fields)

We currently have:
• a small number of Andor SIF datasets

We would like to have:
• an Andor SIF specification document

Ratings

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

### 4.2.11 Animated PNG

Extensions: .png
Developer: The Animated PNG Project

**Support**

BSD-licensed: ☑️
Export: ☑️

Officially Supported Versions:

**Reader:** APNGReader ([Source Code, Supported Metadata Fields](#))

**Writer:** APNGWriter ([Source Code](#))

Freely Available Software:

- Firefox 3+
- Opera 9.5+
- KSquirrel

We currently have:

- a specification document
- several APNG files

**Ratings**

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

### 4.2.12 Aperio AFI

Extensions: .afi, .svs
Owner: Leica Biosystems

**Support**

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: AFIReader (Source Code, Supported Metadata Fields)

We currently have:

- several AFI datasets

**Ratings**

**Pixels:**

**Metadata:**

**Openness:**

**Presence:**

**Utility:**

**Additional Information**

**See also:**

Aperio ImageScope

### 4.2.13 Aperio SVS TIFF

**Extensions:** .svs

**Owner:** Leica Biosystems

**Support**

- BSD-licensed: 
- Export: 

**Officially Supported Versions:** 8.0, 8.2, 9.0

Reader: SVSReader (Source Code, Supported Metadata Fields)

**Freely Available Software:**

- OpenSlide

We currently have:

- many SVS datasets
- public sample images
- an SVS specification document
- the ability to generate additional SVS datasets

**Ratings**

**Pixels:**

**Metadata:**

**Openness:**

**Presence:**

**Utility:**

**Additional Information**
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

See also:
Aperio ImageScope

4.2.14 Applied Precision CellWorX

Extensions: .htd, .pnl
Developer: Applied Precision, Inc.

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: CellWorXReader (Source Code, Supported Metadata Fields)

We currently have:
• a few CellWorX datasets
• public sample datasets

We would like to have:
• a CellWorX specification document
• more CellWorX datasets

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.15 AVI (Audio Video Interleave)

Extensions: .avi
Developer: Microsoft

Support
BSD-licensed: ✔
Export: ✔

Officially Supported Versions:
Reader: AVIReader (Source Code, Supported Metadata Fields)
Writer: AVIWriter (Source Code)
Freely Available Software:

- AVI Reader plugin for ImageJ
- AVI Writer plugin for ImageJ

We currently have:

- several AVI datasets

We would like to have:

- more AVI datasets, including:
  - files with audio tracks and/or multiple video tracks
  - files compressed with a common unsupported codec
  - 2+ GB files

**Ratings**

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▲
Utility: ▼

**Additional Information**

- Bio-Formats can save image stacks as AVI (uncompressed).
- The following codecs are supported for reading:
  - Microsoft Run-Length Encoding (MSRLE)
  - Microsoft Video (MSV1)
  - Raw (uncompressed)
  - JPEG

**See also:**

AVI RIFF File Reference AVI on Wikipedia

### 4.2.16 Axon Raw Format

**Extensions:** .arf

**Owner:** INDEC BioSystems

**Support**

- BSD-licensed: x
- Export: x

**Officially Supported Versions:**

**Reader:** ARFReader (Source Code, Supported Metadata Fields)

We currently have:
• one ARF dataset
  • a specification document

We would like to have:
  • more ARF datasets

**Ratings**

- Pixels: ▶
- Metadata: ▶
- Openness: ▶
- Presence: ▼
- Utility: ▼

**4.2.17 BD Pathway**

Extensions: .exp, .tif

Owner: BD Biosciences

**Support**

- BSD-licensed: ❌
- Export: ❌

Officially Supported Versions:

Reader: BDReader (*Source Code*, *Supported Metadata Fields*)

We currently have:
  • a few BD Pathway datasets

We would like to have:
  • more BD Pathway datasets

**Ratings**

- Pixels: ▶
- Metadata: ▶
- Openness: ◀
- Presence: ▼
- Utility: ◀
4.2.18 Becker & Hickl SPC FIFO

Extensions: .spc
Owner: Becker-Hickl

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: SPCReader (Source Code, Supported Metadata Fields)

We currently have:
• an SPC specification document
• public sample images

We would like to have:
• more SPC sample files

Ratings
Pixels: 🔻
Metadata: 🔻
Openness: 🔺
Presence: 🔻
Utility: 🔻

Additional Information
• Only files containing frame, line and pixel clock information are currently supported

4.2.19 Becker & Hickl SPCImage

Extensions: .sdt
Owner: Becker-Hickl

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: SDTReader (Source Code, Supported Metadata Fields)

We currently have:
• an SDT specification document (from 2008 April, in PDF)
• an SDT specification document (from 2006 June, in PDF)
• Becker & Hickl’s SPCImage software
• a large number of SDT datasets
• the ability to produce new datasets

Ratings

Pixels: ▲
Metadata: □
Openness: □
Presence: ▼
Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.20 Big Data Viewer

Extensions: .xml, .h5
Owner: Tobias Pietzsch

Support

BSD-licensed: ✔
Export: ❌

Officially Supported Versions:
Reader: BDVReader (Source Code, Supported Metadata Fields)

We currently have:
• a BDV specification document
• public sample images

Ratings

Pixels: □
Metadata: ▲
Openness: ▲
Presence: □
Utility: □

4.2. Supported Formats
4.2.21 Bio-Rad Gel

Extensions: .1sc
Owner: Bio-Rad

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: BioRadGelReader (Source Code, Supported Metadata Fields)

We currently have:
- software that can read Bio-Rad Gel files
- several Bio-Rad Gel files
- reverse-engineered Bio-Rad Gel (1sc) file format specification
- Bio-Rad’s Image Lab 5.2.1 software

We would like to have:
- more Bio-Rad Gel files

Ratings

Pixels: 🟢
Metadata: 🟢
Openness: 🟢
Presence: 🟢
Utility: 🟢

Additional Information

- Bio-Rad Gel Format (1sc) specification
- Python3 package of a Bio-Rad Gel Format (1sc) reader
  - `pip3 install biorad1sc_reader`
  - `biorad1sc_reader documentation`
  - Includes command-line utilities
    - `bio1sc2tiff` - convert 1sc file image to tiff
    - `bio1scmeta` - report all metadata in a 1sc file
    - `bio1scread` - report details on internal 1sc file structure
  - `biorad1sc_reader on pypi`
  - `biorad1sc_reader on github`
4.2.22 Bio-Rad PIC

Extensions: .pic, .raw, .xml
Developer: Bio-Rad
Owner: ZEISS International

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:

Reader: BioRadReader (Source Code, Supported Metadata Fields)

Freely Available Software:

• Bio-Rad PIC reader plugin for ImageJ

We currently have:

• a PIC specification document (v4.5, in PDF)
• an older PIC specification document (v4.2, from 1996 December 16, in DOC)
• a large number of PIC datasets
• the ability to produce new datasets

Ratings

Pixels: 🔺
Metadata: 🔺
Openness: 🔺
Presence: 🔺
Utility: 🔺

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

• Commercial applications that support this format include:
  – Bitplane Imaris
  – SVI Huygens
4.2.23 Bio-Rad SCN

Extensions: .scn
Developer: Bio-Rad
Owner: Bio-Rad

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: BioRadSCNReader (Source Code, Supported Metadata Fields)

We currently have:
• a few Bio-Rad .scn files

Ratings
Pixels:  
Metadata:  
Openness:  
Presence:  
Utility:  

4.2.24 Bitplane Imaris

Extensions: .ims
Owner: Oxford Instruments (formerly Bitplane)

Support
BSD-licensed: 
Export: 

Officially Supported Versions: 2.7, 3.0, 5.5

Readers:
• ImarisHDFReader (Source Code, Supported Metadata Fields)
• ImarisTiffReader (Source Code, Supported Metadata Fields)
• ImarisReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Imaris writer

We currently have:
• an Imaris (RAW) specification document (from no later than 1997 November 11, in HTML)
• an Imaris 5.5 (HDF) specification document
• Bitplane’s bfFileReaderImaris3N code (from no later than 2005, in C++)
• several older Imaris (RAW) datasets
• one Imaris 3 (TIFF) dataset
• several Imaris 5.5 (HDF) datasets
• public sample images

We would like to have:
• an Imaris 3 (TIFF) specification document
• more Imaris 3 (TIFF) datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information
• There are three distinct Imaris formats:
  1. the old binary format (introduced in Imaris version 2.7)
  2. Imaris 3, a TIFF variant (introduced in Imaris version 3.0)
  3. Imaris 5.5, an HDF variant (introduced in Imaris version 5.5)

4.2.25 Bruker MRI

Developer: Bruker

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: BrukerReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Bruker plugin for ImageJ

We currently have:
• a few Bruker MRI datasets

We would like to have:
• an official specification document

Ratings
Pixels: ▼
Metadata: ▼
4.2.26 Burleigh

Extensions: .img
Owner: Burleigh Instruments

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: BurleighReader (Source Code, Supported Metadata Fields)

We currently have:
- Pascal code that can read Burleigh files (from ImageSXM)
- a few Burleigh files

We would like to have:
- a Burleigh file format specification
- more Burleigh files

Ratings

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.27 Canon DNG

Extensions: .cr2, .crw
Developer: Canon

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: DNGReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• IrfanView

We currently have:
  • a few example datasets

We would like to have:
  • an official specification document

Ratings

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.28 CellH5

Extensions: .ch5
Developer: CellH5

Support

BSD-licensed: ✓
Export: ✓

Officially Supported Versions:
Reader: CellH5Reader (Source Code, Supported Metadata Fields)
Writer: CellH5Writer (Source Code)

Freely Available Software:
  • CellH5

We currently have:
  • public sample images

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▲
4.2.29 Cellomics

Extensions: .c01, .dib
Developer: Thermo Fisher Scientific

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Reader: CellomicsReader (Source Code, Supported Metadata Fields)

We currently have:

- a few Cellomics .c01 datasets
- public sample images

We would like to have:

- a Cellomics .c01 specification document
- more Cellomics .c01 datasets

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.30 cellSens VSI

Extensions: .vsi
Developer: Olympus

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Reader: CellSensReader (Source Code, Supported Metadata Fields)

We currently have:

- a few example datasets
- a VSI specification document (v1.6, 2012 November 27, in PDF)
- a VSI specification document (v1.3, 2010 February 5, in PDF)
4.2.31 CellVoyager

Extensions: .xml, .tif
Owner: Yokogawa

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: CellVoyagerReader (Source Code, Supported Metadata Fields)

We currently have:
• a few example datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▼
Presence: ▼
Utility: ▼

4.2.32 CV7000

Extensions: .wpi, .tif
Owner: Yokogawa

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: CV7000Reader (Source Code, Supported Metadata Fields)
We currently have:
  • many example datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.33 DeltaVision

Extensions: .dv, .r3d, .rcpn1
Owner: Cytiva (formerly GE Healthcare, Applied Precision)

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Readers:
  • DeltavisionReader (Source Code, Supported Metadata Fields)
  • RCPNLReader (Source Code, Supported Metadata Fields)

Freely Available Software:
  • DeltaVision Opener plugin for ImageJ

We currently have:
  • a DV specification document (v2.10 or newer, in HTML)
  • numerous DV datasets
  • public sample images

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▲

Additional Information
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.
• The Deltavision format is based on the Medical Research Council (MRC) file format.
• Commercial applications that support DeltaVision include:
  – Bitplane Imaris
  – SVI Huygens
  – Image-Pro Plus
• RCPNL is a variant of the DeltaVision format and requires a separate reader.

4.2.34 DICOM

Extensions: .dcm, .dicom
Developer: DICOM Standards Committee
Support
BSD-licensed: ✓
Export: ✓
Officially Supported Versions:
Reader: DicomReader (Source Code, Supported Metadata Fields)
Writer: DicomWriter (Source Code)
Freely Available Software:
  • List of converters and viewers
  • List of libraries/toolkits
  • Wikipedia’s list of freeware health software
Sample Datasets:
  • David Clunie’s list of DICOM image samples
  • Whole Slide Images from DICOM Working Group 26
  • NCI Imaging Data Commons Portal, download instructions
  • Converted Whole Slide Images
  • Medical Image Samples from Sebastien Barre’s Medical Imaging page
  • DICOM sample image sets from OsiriX web site
We currently have:
  • DICOM specification documents
  • numerous DICOM datasets (see above)
  • public sample images

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲

4.2. Supported Formats
DICOM stands for “Digital Imaging and Communication in Medicine”.

Bio-Formats supports uncompressed, baseline JPEG, JPEG-2000 (lossy and lossless), and RLE lossless transfer syntaxes.

Support for reading and writing DICOM whole slide images (DICOM WSI format) was implemented through collaboration with NCI Imaging Data Commons and has been funded in whole or in part with Federal funds from the National Cancer Institute, National Institutes of Health, under Task Order No. HHSN26110071 under Contract No. HHSN2612015000031.

If you have a problematic DICOM file which you cannot send us for privacy reasons, please send us the exact error message and be aware that it may take several attempts to fix the problem blind.

See also:
DICOM homepage

4.2.35 ECAT7

Extensions: .v
Developer: Siemens

Support

BSD-licensed: 🚧
Export: 🚧

Officially Supported Versions:
Reader: Ecat7Reader (Source Code, Supported Metadata Fields)

We currently have:
- a few ECAT7 files
- public sample images

We would like to have:
- an ECAT7 specification document
- more ECAT7 files

Ratings

Pixels: ⬇️
Metadata: ⬇️
Openness: ⬇️
Presence: ⬇️
Utility: ⬇️
4.2.36 EPS (Encapsulated PostScript)

Extensions: .eps, .epsi, .ps
Developer: Adobe

Support

BSD-licensed:
Export:

Officially Supported Versions:
Reader: EPSReader (Source Code, Supported Metadata Fields)
Writer: EPSWriter (Source Code)

Freely Available Software:
- EPS Writer plugin for ImageJ

We currently have:
- a few EPS datasets
- the ability to produce new datasets

Ratings

Pixels:
Metadata:
Openness:
Presence:
Utility:

Additional Information
- Bio-Formats can save individual planes as EPS.
- Certain types of compressed EPS files are not supported.

4.2.37 Evotec/PerkinElmer Opera Flex

Extensions: .flex, .mea, .res
Developer: Evotec Technologies, now PerkinElmer

Support

BSD-licensed:
Export:

Officially Supported Versions:
Reader: FlexReader (Source Code, Supported Metadata Fields)

We currently have:
- many Flex datasets
- public sample images
We would like to have:
  • a freely redistributable LuraWave LWF decoder

Ratings
Pixels: 🟢
Metadata: 🟢
Openness: 🟥
Presence: 🟥
Utility: 🟥

Additional Information
The LuraWave LWF decoder library (i.e. lwf_jsdk2.6.jar) with license code is required to decode wavelet-compressed Flex files.

Note that support for the LuraWave code is deprecated and will be removed in Bio-Formats 7.0.0.

See also:
LuraTech (developers of the proprietary LuraWave LWF compression used for Flex image planes)

4.2.38 FEI

Extensions: .img
Developer: FEI

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: FEIReader (Source Code, Supported Metadata Fields)

We currently have:
  • a few FEI files

We would like to have:
  • a specification document
  • more FEI files

Ratings
Pixels: 🟥
Metadata: 🟥
Openness: 🟥
Presence: 🟥
Utility: 🟥
4.2.39 FEI TIFF

Extensions: .tiff
Developer: FEI

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: FEITiffReader (Source Code, Supported Metadata Fields)

We currently have:
  • a few FEI TIFF datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.40 FITS (Flexible Image Transport System)

Extensions: .fits
Developer: National Radio Astronomy Observatory

Support

BSD-licensed: ✔
Export: ✗

Officially Supported Versions:
Reader: FitsReader (Source Code, Supported Metadata Fields)

We currently have:
  • a FITS specification document (NOST 100-2.0, from 1999 March 29, in HTML)
  • several FITS datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▼
Additional Information

See also:

MAST:FITS homepage FITS Support Office

4.2.41 Gatan Digital Micrograph

Extensions: .dm3, .dm4
Owner: Gatan

Support

BSD-licensed: ☒
Export: ☒

Officially Supported Versions: 3, 4
Reader: GatanReader (Source Code, Supported Metadata Fields)

Freely Available Software:

• DM3 Reader plugin for ImageJ
  • EMAN

We currently have:

• Gatan’s ImageReader2003 code (from 2003, in C++)
• numerous DM3 datasets
• public sample images

We would like to have:

• a DM3 specification document

Ratings

Pixels: ▲
Metadata: ◀
Openness: ▼
Presence: ▼
Utility: ▼

Additional Information

Applications that support .dm3 files include Datasqueeze.

Note that the Gatan Reader does not currently support stacks.
4.2.42 Gatan Digital Micrograph 2

Extensions: .dm2
Developer: Gatan

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions: 2
Reader: GatanDM2Reader (Source Code, Supported Metadata Fields)

We currently have:
- Pascal code that can read DM2 files (from ImageSXM)
- a few DM2 files

We would like to have:
- an official DM2 specification document
- more DM2 files

Ratings

Pixels:  
Metadata:  
Openness: ▼
Presence: ▼
Utility: 

4.2.43 GE MicroCT

Extensions: .vff
Developer: GE Healthcare

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: MicroCTReader (Source Code, Supported Metadata Fields)

We currently have:
- several public MicroCT datasets from the CIBC Dataset Archive

Ratings

Pixels: ▲
Metadata: ▼

4.2. Supported Formats
4.2.44 GIF (Graphics Interchange Format)

Extensions: .gif
Developer: CompuServe
Owner: Unisys

Support
BSD-licensed: ✔
Export: ✗

Officially Supported Versions:
Reader: GIFReader (Source Code, Supported Metadata Fields)

Freely Available Software:
  • Animated GIF Reader plugin for ImageJ
  • GIF Stack Writer plugin for ImageJ

We currently have:
  • a GIF specification document (Version 89a, from 1990, in HTML)
  • numerous GIF datasets
  • the ability to produce new datasets

Ratings
Pixels: ⬆
Metadata: ⬇
Openness: ⬇
Presence: ⬆
Utility: ⬇

4.2.45 Hamamatsu Aquacosmos NAF

Extensions: .naf
Developer: Hamamatsu

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: NAFReader (Source Code, Supported Metadata Fields)

We currently have:
- a few NAF files

We would like to have:
- a specification document
- more NAF files

Ratings

Pixels:
Metadata:
Openness:
Presence:
Utility:

4.2.46 Hamamatsu HIS

Extensions: .his
Owner: Hamamatsu

Support

BSD-licensed: x
Export: x

Officially Supported Versions:

Reader: HISReader (Source Code, Supported Metadata Fields)

We currently have:
- Pascal code that can read HIS files (from ImageSX M)
- several HIS files

We would like to have:
- an HIS specification
- more HIS files

Ratings

Pixels:
Metadata:
Openness:
Presence:
Utility:
4.2.47 Hamamatsu ndpi

Extensions: .ndpi, .ndpis
Developer: Hamamatsu

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Readers:
• NDPIReader (Source Code, Supported Metadata Fields)
• NDPISReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• NDP.view2
• OpenSlide

We currently have:
• many example datasets
• public sample images

Ratings
Pixels: ☑
Metadata: ☐
Openness: ☐
Presence: ☑
Utility: ☑

4.2.48 Hamamatsu VMS

Extensions: .vms
Developer: Hamamatsu

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: HamamatsuVMSReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• OpenSlide

We currently have:
• a few example datasets
• public sample images
• developer documentation from the OpenSlide project

We would like to have:
• an official specification document
• more example datasets

Ratings
Pixels:  
Metadata:  
Openness:  
Presence:  
Utility:  

4.2.49 Hitachi S-4800

Extensions: .txt, .tif, .bmp, .jpg
Developer: Hitachi

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: HitachiReader (Source Code, Supported Metadata Fields)

We currently have:
• several Hitachi S-4800 datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence:  
Utility:  

4.2. Supported Formats
4.2.50 I2I

Extensions: .i2i
Developer: Biomedical Imaging Group, UMass Medical School

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: I2IReader (Source Code, Supported Metadata Fields)

We currently have:
• several example datasets
• a specification document
• an ImageJ plugin that can read I2I data

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.51 ICS (Image Cytometry Standard)

Extensions: .ics, .ids
Developer: P. Dean et al.

Support

BSD-licensed: ✔
Export: ✔

Officially Supported Versions: 1.0, 2.0
Reader: ICSReader (Source Code, Supported Metadata Fields)
Writer: ICSWriter (Source Code)

Freely Available Software:
• Libics (ICS reference library)
• ICS Opener plugin for ImageJ
• IrfanView

We currently have:
• numerous ICS datasets
• public sample images
Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▲

Additional Information

- ICS version 1.0 datasets have two files - an .ics file that contains all of the metadata in plain-text format, and an .ids file that contains all of the pixel data.
- ICS version 2.0 datasets are a single .ics file that contains both pixels and metadata.

Commercial applications that can support ICS include:

- Bitplane Imaris
- SVI Huygens

4.2.52 Imacon

Extensions: .fff

Owner: Hasselblad

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:

Reader: ImaconReader (Source Code, Supported Metadata Fields)

We currently have:

- one Imacon file

We would like to have:

- more Imacon files

Ratings

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▲
4.2.53 ImagePro Sequence

Extensions: .seq
Owner: Media Cybernetics
Support
BSD-licensed: x
Export: x

Officially Supported Versions:
Reader: SEQReader (Source Code, Supported Metadata Fields)

We currently have:
• the Image-Pro Plus software
• a few SEQ datasets
• the ability to produce more datasets

We would like to have:
• an official SEQ specification document

Ratings
Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.54 ImagePro Workspace

Extensions: .ipw
Owner: Media Cybernetics
Support
BSD-licensed: x
Export: x

Officially Supported Versions:
Reader: IPWReader (Source Code, Supported Metadata Fields)

We currently have:
• the Image-Pro Plus software
• a few IPW datasets
• the ability to produce more datasets

We would like to have:
• an official IPW specification document
• more IPW datasets:
  – multiple datasets in one file
  – 2+ GB files

Ratings

Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

Additional Information

Bio-Formats uses a modified version of the Apache Jakarta POI library to read IPW files.

4.2.55 IMAGIC

Extensions: .hed, .img
Developer: Image Science

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: ImagicReader (Source Code, Supported Metadata Fields)

Freely Available Software:
  • em2em

We currently have:
  • one example dataset
  • official file format documentation

We would like to have:
  • more example datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information
See also:
IMAGIC specification

### 4.2.56 IMOD

Extensions: .mod

**Developer:** Boulder Laboratory for 3-Dimensional Electron Microscopy of Cells  
**Owner:** Boulder Laboratory for 3-Dimensional Electron Microscopy of Cells

**Support**

BSD-licensed: ✗  
Export: ✗

Officially Supported Versions:

**Reader:** IMODReader ([Source Code, Supported Metadata Fields])

**Freely Available Software:**

- IMOD

We currently have:

- a few sample datasets  
- official documentation

**Ratings**

**Pixels:**  
**Metadata:**  
**Openness:**  
**Presence:**  
**Utility:**

### 4.2.57 Improvision Openlab LIFF

Extensions: .liff

**Developer:** Improvision, now PerkinElmer  
**Owner:** PerkinElmer

**Support**

BSD-licensed: ✗  
Export: ✗

Officially Supported Versions: 2.0, 5.0

**Reader:** OpenlabReader ([Source Code, Supported Metadata Fields])

We currently have:

- an Openlab specification document (from 2000 February 8, in DOC)
• Improvision’s XLIFFFileImporter code for reading Openlab LIFF v5 files (from 2006, in C++)
• several Openlab datasets

We would like to have:
• more Openlab datasets (preferably with 32-bit integer data)

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.58 Improvision Openlab Raw

Extensions: .raw

Developer: Improvision, now PerkinElmer
Owner: PerkinElmer

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: OpenlabRawReader (Source Code, Supported Metadata Fields)

We currently have:
• an Openlab Raw specification document (from 2004 November 09, in HTML)
• a few Openlab Raw datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼
4.2.59 Improvision TIFF

Extensions: .tif
Developer: Improvision, now PerkinElmer
Owner: PerkinElmer

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: ImprovisionTiffReader (Source Code, Supported Metadata Fields)

We currently have:
• an Improvision TIFF specification document
• a few Improvision TIFF datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: 🔴
Utility: ▲

Additional Information
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.60 Imspector OBF

Extensions: .obf, .msr
Developer: Department of NanoBiophotonics, MPI-BPC
Owner: MPI-BPC

Support
BSD-licensed: ✓
Export: ✗

Officially Supported Versions:
Reader: OBFReader (Source Code, Supported Metadata Fields)

We currently have:
• a few OBF datasets
• public OBF sample images
• a specification document
Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.61 InCell 1000/2000

Extensions: .xdce, .tif
Developer: GE Healthcare
Owner: Cytiva

Support

BSD-licensed: X
Export: X

Officially Supported Versions:
Reader: InCellReader (Source Code, Supported Metadata Fields)

We currently have:
- a few InCell 1000 datasets
- public InCell 2000 sample images

We would like to have:
- an InCell 1000 specification document
- more InCell 1000 datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▼
Presence: ▼
Utility: ▼
### 4.2.62 InCell 3000

Extensions: .frm  
Developer: GE Healthcare  
Owner: Cytiva

**Support**

BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:

Reader: InCell3000Reader (Source Code, Supported Metadata Fields)

We currently have:

- a few example datasets  
- public sample images

We would like to have:

- an official specification document

**Ratings**

Pixels: ▼  
Metadata: ▼  
Openness: ▼  
Presence: ▼  
Utility: ▼

### 4.2.63 INR

Extensions: .inr  

**Support**

BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:

Reader: INRReader (Source Code, Supported Metadata Fields)

We currently have:

- several sample .inr datasets

**Ratings**

Pixels: ▲  
Metadata: ▼  
Openness: ▼
4.2.64 Inveon

Extensions: .hdr

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: InveonReader (Source Code, Supported Metadata Fields)

We currently have:
a few Inveon datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.65 Ionpath MIBI

Extensions: .tif, .tiff
Developer: IonPath
Owner: IonPath

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions: 0.1
Reader: IonpathMIBITiffReader (Source Code, Supported Metadata Fields)

We currently have:
• a few sample datasets
• a specification document

We would like to have:
• more sample datasets
Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.66 IPLab

Extensions: .ipl
Developer: Scanalytics
Owner: was BD Biosystems, now BioVision Technologies

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: IPLabReader (Source Code, Supported Metadata Fields)

Freely Available Software:

• IPLab Reader plugin for ImageJ

We currently have:

• an IPLab specification document (v3.6.5, from 2004 December 1, in PDF)
• several IPLab datasets

We would like to have:

• more IPLab datasets (preferably with 32-bit integer or floating point data)

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.
Commercial applications that support IPLab include:

- Bitplane Imaris
- SVI Huygens

See also:

IPLab software review

### 4.2.67 IVision

Extensions: .ipm

Owner: BioVision Technologies

Support

BSD-licensed: ❌

Export: ❌

Officially Supported Versions:

Reader: IvisionReader (Source Code, Supported Metadata Fields)

We currently have:

- a few iVision-Mac datasets
- a specification document

We would like to have:

- more iVision-Mac datasets

Ratings

Pixels: ▲

Metadata: ▲

Openness: ▲

Presence: ▼

Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

iVision-Mac was formerly called IPLab for Macintosh.
4.2.68 JEOL

Extensions: .dat, .img, .par

Owner: JEOL

Support

BSD-licensed: 

Export: 

Officially Supported Versions:

Reader: JEOLReader (Source Code, Supported Metadata Fields)

We currently have:

- Pascal code that reads JEOL files (from ImageSXM)
- a few JEOL files

We would like to have:

- an official specification document
- more JEOL files

Ratings

Pixels: 

Metadata: 

Openness: 

Presence: 

Utility: 

4.2.69 JPEG

Extensions: .jpg

Developer: Independent JPEG Group

Support

BSD-licensed: ✔

Export: ✔

Officially Supported Versions:

Reader: JPEGReader (Source Code, Supported Metadata Fields)

Writer: JPEGWriter (Source Code)

We currently have:

- a JPEG specification document (v1.04, from 1992 September 1, in PDF)
- numerous JPEG datasets
- the ability to produce more datasets
Bio-Formats, Release 6.14.0

Ratings

Pixels: 🔺
Metadata: 🔺
Openness: 🔺
Presence: 🔺
Utility: 🔻

Additional Information

Bio-Formats can save individual planes as JPEG. Bio-Formats uses the Java Image I/O API to read and write JPEG files. JPEG stands for “Joint Photographic Experts Group”.

See also:

JPEG homepage

4.2.70 JPEG 2000

Extensions: .jp2, .j2k, .jpf
Developer: Independent JPEG Group

Support

BSD-licensed: 🟢
Export: 🟢

Officially Supported Versions:

Reader: JPEG2000Reader (Source Code, Supported Metadata Fields)
Writer: JPEG2000Writer (Source Code)

Freely Available Software:

• JJ2000 (JPEG 2000 library for Java)

We currently have:

• a JPEG 2000 specification document (free draft from 2000, no longer available online)
• a few .jp2 files

Ratings

Pixels: 🔺
Metadata: 🔺
Openness: 🔺
Presence: 🟢
Utility: 🔻

Additional Information

Bio-Formats uses the JAI Image I/O Tools library to read JP2 files. Conflicting versions of this no-longer-maintained library may cause errors, see JAI ImageIO component for details.

4.2. Supported Formats
JPEG stands for “Joint Photographic Experts Group”.

### 4.2.71 JPK

Extensions: .jpk  
Developer: JPK Instruments  
Support  
BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:  
Reader: JPKReader ([Source Code, Supported Metadata Fields](#))  
We currently have:  
- Pascal code that can read JPK files (from ImageSXM)  
- a few JPK files  
We would like to have:  
- an official specification document  
- more JPK files  
Ratings  
Pixels: 🅰️  
Metadata: 🅲️  
Openness: 🅲️  
Presence: 🅲️  
Utility: 🅲️

### 4.2.72 JPX

Extensions: .jpx  
Developer: JPEG Committee  
Support  
BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:  
Reader: JPXReader ([Source Code, Supported Metadata Fields](#))  
We currently have:  
- a few .jpx files
Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ★
Utility: ◀

4.2.73 Keller Lab Block

Extensions: .klb
Developer: Keller Lab (Janelia Research Campus)
Support
BSD-licensed: ✔
Export: ✗
Officially Supported Versions:
Reader: KLBReader (Source Code, Supported Metadata Fields)
We currently have:
  • public sample images

Ratings

Pixels: ★
Metadata: ▲
Openness: ▲
Presence: ★
Utility: ◀

4.2.74 Khoros VIFF (Visualization Image File Format) Bitmap

Extensions: .xv
Developer: Khoral
Owner: AccuSoft
Support
BSD-licensed: ✗
Export: ✗
Officially Supported Versions:
Reader: KhorosReader (Source Code, Supported Metadata Fields)
Sample Datasets:
• VIFF Images
We currently have:
  • several VIFF datasets

Ratings
Pixels: 🔻
Metadata: 🔻
Openness: 🔻
Presence: 🔻
Utility: 🔻

4.2.75 Kodak BIP

Extensions: .bip
Developer: Kodak/Carestream

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: KodakReader (Source Code, Supported Metadata Fields)
We currently have:
  • a few .bip datasets
We would like to have:
  • an official specification document

Ratings
Pixels: 🔺
Metadata: 🔻
Openness: 🔻
Presence: 🔻
Utility: 🔻
4.2.76 Lambert Instruments FLIM

Extensions: .fli
Developer: Lambert Instruments

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions: 1.0, 2.0
Reader: LiFlimReader (Source Code, Supported Metadata Fields)

We currently have:
- an LI-FLIM specification document
- several example LI-FLIM datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

Additional Information
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.77 LaVision Imspector

Extensions: .msr
Developer: LaVision BioTec

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions: 4.0, 4.1
Reader: InspectorReader (Source Code, Supported Metadata Fields)

We currently have:
- a few .msr files

Ratings

Pixels: ▼
Metadata: ▼
Openness: ▼
4.2.78 Leica LCS LEI

Extensions: .lei, .tif
Developer: Leica Microsystems CMS GmbH
Owner: Leica Microsystems

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: LeicaReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Leica LCS Lite

We currently have:
• an LEI specification document (beta 2.000, from no later than 2004 February 17, in PDF)
• many LEI datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▲

Additional Information
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

LCS stands for “Leica Confocal Software”. LEI presumably stands for “Leica Experimental Information”.

Commercial applications that support LEI include:
• Bitplane Imaris
• SVI Huygens
• Image-Pro Plus
4.2.79 Leica LAS AF LIF (Leica Image File Format)

Extensions: .lif
Developer: Leica Microsystems CMS GmbH
Owner: Leica Microsystems
Support
BSD-licensed: ✗
Export: ✗
Officially Supported Versions: 1.0, 2.0
Reader: LIFReader (Source Code, Supported Metadata Fields)
Freely Available Software:
• Leica Application Suite X
We currently have:
• a LIF/XLLF/XLEF/LOF specification document (version 3.2, from no later than 2016 December 15, in PDF)
• a LIF specification document (version 2, from no later than 2007 July 26, in PDF)
• a LIF specification document (version 1, from no later than 2006 April 3, in PDF)
• numerous LIF datasets
• public sample images
Ratings
Pixels: ✖
Metadata: ✖
Openness: ✖
Presence: ✖
Utility: ✖
Additional Information
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.
Additional options are available for reading or writing this format type, see Additional reader and writer options for information.
LAS stands for “Leica Application Suite”. AF stands for “Advanced Fluorescence”.
Commercial applications that support LIF include:
• Bitplane Imaris
• SVI Huygens
• Amira
Versions of Bio-Formats prior to 5.3.3 incorrectly calculated the physical pixel width and height. The physical image width and height were divided by the number of pixels, which was inconsistent with the official Leica LIF specification documents. Versions 5.3.3 and later correctly calculate physical pixel sizes by dividing the physical image size by the
number of pixels minus one. To revert to the old method of physical pixel size calculation in 5.3.3 and later, set the leicalif.old_physical_size option to true as described in Additional reader and writer options.

4.2.80 Leica LOF

Extensions: .lof
Developer: Leica Microsystems CMS GmbH
Owner: Leica Microsystems

Support

BSD-licensed: X
Export: X

Officially Supported Versions:
Reader: LOFReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Leica Application Suite X

We currently have:
• numerous LOF datasets
• public sample images

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▲

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.81 Leica SCN

Extensions: .scn
Developer: Leica Microsystems

Support

BSD-licensed: X
Export: X

Officially Supported Versions: 2012-03-10
Reader: LeicaSCNReader (Source Code, Supported Metadata Fields)
Freely Available Software:
  • OpenSlide
We currently have:
  • a few sample datasets
  • public sample images
We would like to have:
  • an official specification document
  • sample datasets that cannot be opened

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.82 Leica XLEF

Extensions: .xlef
Developer: Leica Microsystems CMS GmbH
Owner: Leica Microsystems

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: XLEFReader (Source Code, Supported Metadata Fields)

Freely Available Software:
  • Leica Application Suite X
We currently have:
  • numerous XLEF datasets
  • public sample images

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2. Supported Formats 165
Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

4.2.83 LEO

Extensions: .sxm, .tif, .tiff
Owner: ZEISS International

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

 Reader: LEOReader (Source Code, Supported Metadata Fields)

We currently have:

• Pascal code that can read LEO files (from ImageSXM)
• a few LEO files
  • public sample images

We would like to have:

• an official specification document
• more LEO files

Ratings

Pixels:
Metadata:
Openness:
Presence:
Utility:

4.2.84 Li-Cor L2D

Extensions: .l2d, .tif, .scn
Owner: LiCor Biosciences

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

 Reader: L2DReader (Source Code, Supported Metadata Fields)

We currently have:
• a few L2D datasets

We would like to have:
• an official specification document
• more L2D datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ◼
Presence: ◼
Utility: ◼

Additional Information

L2D datasets cannot be imported into OME using server-side import. They can, however, be imported from ImageJ, or using the omeul utility.

4.2.85 LIM (Laboratory Imaging/Nikon)

Extensions: .lim
Owner: Laboratory Imaging

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: LIMReader (Source Code, Supported Metadata Fields)

We currently have:
• several LIM files
• the ability to produce more LIM files

We would like to have:
• an official specification document

Ratings

Pixels: ◼
Metadata: ◼
Openness: ◼
Presence: ◼
Utility: ◼

Additional Information

Bio-Formats only supports uncompressed LIM files.

4.2. Supported Formats
Commercial applications that support LIM include:

- NIS Elements

### 4.2.86 MetaMorph 7.5 TIFF

Extensions: .tiff

Owner: Molecular Devices

**Support**

BSD-licensed: ❌

Export: ❌

Officially Supported Versions:

Reader: MetamorphTiffReader *(Source Code, Supported Metadata Fields)*

We currently have:

- a few Metamorph 7.5 TIFF datasets

**Ratings**

Pixels: 🔷

Metadata: 🔷

Openness: 🔷

Presence: 🔻

Utility: 🟢

### 4.2.87 MetaMorph Stack (STK)

Extensions: .stk, .nd

Owner: Molecular Devices

**Support**

BSD-licensed: ❌

Export: ❌

Officially Supported Versions:

Reader: MetamorphReader *(Source Code, Supported Metadata Fields)*

We currently have:

- an STK specification document (from 2006 November 21, in DOC)
- an older STK specification document (from 2005 March 25, in DOC)
- an ND specification document (from 2002 January 24, in PDF)
- a large number of datasets
Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▼

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Commercial applications that support STK include:

- Bitplane Imaris
- SVI Huygens
- DIMIN

See also:

Metamorph imaging system overview

4.2.88 MetaXpress

Extensions: .htd, .tif, .tiff

Owner: Molecular Devices

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:

Reader: CellWorxReader (Source Code, Supported Metadata Fields)

We currently have:

- several MetaXpress datasets
- public sample datasets

We would like to have:

- a MetaXpress specification document

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼
4.2.89 MIAS (Maia Scientific)

Extensions: .tif
Developer: Maia Scientific

Support
BSD-licensed:
Export:

Officially Supported Versions:
Reader: MIASReader (Source Code, Supported Metadata Fields)

We currently have:
• several MIAS datasets

Ratings
Pixels:
Metadata:
Openness:
Presence:
Utility:

4.2.90 Micro-Manager

Extensions: .tif, .txt, .xml
Developer: Vale Lab

Support
BSD-licensed:
Export:

Officially Supported Versions: Up to 1.4.22
Reader: MicromanagerReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Micro-Manager

We currently have:
• many Micro-manager datasets
• public sample images

Ratings
Pixels:
Metadata:
Openness:
Bio-Formats will recognize a *metadata.txt file as part of a Micro-Manager fileset if pointed at it and will load the fileset including the companion TIFF files.

- If pointed at a companion .ome.tif file, Bio-Formats will recognize an OME-TIFF format instead. This means it may load the fileset if there are multiple .ome.tif but it will not include *metadata.txt in this fileset and therefore the extended Micro-Manager metadata will be skipped.

- See Micro-Manager for more information.

### 4.2.91 Mikroscan TIFF

**Extensions:** .tif, .tiff  
**Owner:** Mikroscan  
**Support**  
**BSD-licensed:** ✗  
**Export:** ✗  
**Officially Supported Versions:**  
**Reader:** MikroscanTiffReader ([Source Code, Supported Metadata Fields](#))  

We currently have:  
- some Mikroscan datasets

**Ratings**

**Pixels:**  
**Metadata:**  
**Openness:**  
**Presence:** ✗  
**Utility:** ✗  

**Additional Information**

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.
4.2.92 MINC MRI

Extensions: .mnc
Developer: McGill University

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: MINCReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• MINC

We currently have:
• a few MINC files

Ratings

Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.93 Minolta MRW

Extensions: .mrw
Developer: Minolta

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: MRWReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• dcraw

We currently have:
• several .mrw files

Ratings

Pixels: ▲
Metadata: ▼
4.2.94 **MNG (Multiple-image Network Graphics)**

Extensions: .mng

Developer: MNG Development Group

**Support**

BSD-licensed: ✅

Export: ❌

Officially Supported Versions:

Reader: MNGReader ([Source Code, Supported Metadata Fields](#))

Freely Available Software:

- libmng (MNG reference library)

Sample Datasets:

- MNG sample files

We currently have:

- the libmng-testsuites package (from 2003 March 05, in C)
- a large number of MNG datasets

**Ratings**

Pixels: 🟠

Metadata: 🟠

Openness: 🔴

Presence: 🔴

Utility: 🔴

**Additional Information**

**See also:**

MNG homepage MNG specification
4.2.95 Molecular Imaging

Extensions: .stp
Owner: Molecular Imaging Corp, San Diego CA (closed)

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: MolecularImagingReader (Source Code, Supported Metadata Fields)

We currently have:
• Pascal code that reads Molecular Imaging files (from ImageSXM)
• a few Molecular Imaging files

We would like to have:
• an official specification document
• more Molecular Imaging files

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.96 MRC (Medical Research Council)

Extensions: .mrc, .st, .ali, .map, .rec, .mrcs
Developers:
• MRC Laboratory of Molecular Biology
• CCP-EM

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: MRCReader (Source Code, Supported Metadata Fields)

Sample Datasets:
• golgi.mrc

We currently have:
• specification documents for MRC2000 and MRC2014
• an IMOD-specific MRC specification document (in TXT)
• public sample images
• a few MRC datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ■
Utility: ■

Additional Information

Commercial applications that support MRC include:

• Bitplane Imaris

Bio-Formats provides support for the base MRC2000/MRC2014 specifications. Limited support is provided for vendor-specific extended headers, as described in the MRC2014 specification. IMOD extended metadata is perhaps the best-supported variant.

See also:

MRC on Wikipedia

4.2.97 NEF (Nikon Electronic Format)

Extensions: .nef, .tif
Developer: Nikon

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: NikonReader (Source Code, Supported Metadata Fields)

Sample Datasets:

• nefile1.zip
• Sample NEF images

We currently have:

• a NEF specification document (v0.1, from 2003, in PDF)
• several NEF datasets

Ratings

Pixels: ▲
Metadata: ■
Openness: 📈
Presence: 📈
Utility: 📈

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

See also:
NEF Conversion

4.2.98 NIfTI

Extensions: .img, .hdr, .nii, .nii.gz

Developer: National Institutes of Health

Support

BSD-licensed: 🗑️
Export: 🗑️

Officially Supported Versions:

Reader: NiftiReader (Source Code, Supported Metadata Fields)

Sample Datasets:
• Official test data

We currently have:
• NIfTI specification documents
• several NIfTI datasets
• public sample images

Ratings

Pixels: 📈
Metadata: 📈
Openness: 📈
Presence: 📈
Utility: 📈
4.2.99 Nikon Elements TIFF

Extensions: .tiff
Developer: Nikon

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: NikonElementsTiffReader (Source Code, Supported Metadata Fields)

We currently have:
- a few Nikon Elements TIFF files

We would like to have:
- more Nikon Elements TIFF files

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.100 Nikon EZ-C1 TIFF

Extensions: .tiff
Developer: Nikon

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: NikonTiffReader (Source Code, Supported Metadata Fields)

We currently have:
- a few Nikon EZ-C1 TIFF files

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2. Supported Formats
Utility: 

4.2.101 Nikon NIS-Elements ND2

Extensions: .nd2
Developer: Nikon USA

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Readers:
- NativeND2Reader (Source Code, Supported Metadata Fields)
- LegacyND2Reader (Source Code, Supported Metadata Fields)

Freely Available Software:
- NIS-Elements Viewer from Nikon

We currently have:
- many ND2 datasets
- public sample images

We would like to have:
- an official specification document

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

Additional Information

Additional options are available for reading or writing this format type, see Additional reader and writer options for information.

There are two distinct versions of ND2: an old version, which uses JPEG-2000 compression, and a new version which is either uncompressed or Zip-compressed. We are not aware of the version number or release date for either format.

Bio-Formats uses the JAI Image I/O Tools library to read ND2 files compressed with JPEG-2000.

There is also a legacy ND2 reader that uses Nikon’s native libraries. To use it, you must be using Windows 32-bit and have Nikon’s ND2 reader plugin for ImageJ installed. Additionally, you will need to download LegacyND2Reader.dll and place it in your ImageJ plugin folder.

Note that the legacy ND2 reader is deprecated and will be removed in Bio-Formats 7.0.0.
4.2.102 NRRD (Nearly Raw Raster Data)

Extensions: .nrrd, .nhdr, .raw, .txt
Developer: Teem developers

Support

BSD-licensed: ✅
Export: ❌

Officially Supported Versions:

Reader: NRRDReader (Source Code, Supported Metadata Fields)

Freely Available Software:

• nrrd (NRRD reference library)

Sample Datasets:

• Diffusion tensor MRI datasets

We currently have:

• an nrrd specification document (v1.9, from 2005 December 24, in HTML)
• a few nrrd datasets
• public sample images

Ratings

Pixels: 🔹
Metadata: 🔹
Openness: 🔹
Presence: 🔻
Utility: 🔹

4.2.103 Olympus CellR/APL

Extensions: .apl, .mtb, .tnb, .tif, .obsep
Owner: Olympus

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:

Reader: APLReader (Source Code, Supported Metadata Fields)

We currently have:

• a few CellR datasets

We would like to have:

• more CellR datasets

4.2. Supported Formats
* an official specification document

**Ratings**

- Pixels: ▲
- Metadata: ▼
- Openness: ▼
- Presence: ▼
- Utility: ▼

### 4.2.104 Olympus FluoView FV1000

**Extensions:** .oib, .oif  
**Owner:** Olympus

**Support**

- BSD-licensed: ❌
- Export: ❌

- Officially Supported Versions: 1.0, 2.0

**Reader:** FV1000Reader ([Source Code, Supported Metadata Fields](#))

**Freely Available Software:**

- FV-Viewer from Olympus

We currently have:

- an OIF specification document (v2.0.0.0, from 2008, in PDF)
- an FV1000 specification document (v1.0.0.0, from 2004 June 22, in PDF)
- older FV1000 specification documents (draft, in DOC and XLS)
- public sample images
- many FV1000 datasets

We would like to have:

- more OIB datasets (especially 2+ GB files)
- more FV1000 version 2 datasets

**Ratings**

- Pixels: ▲
- Metadata: ▲
- Openness: ☐
- Presence: ☐
- Utility: ▲

**Additional Information**
Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Bio-Formats uses a modified version of the Apache POI library to read OIB files. OIF stands for “Original Imaging Format”. OIB stands for “Olympus Image Binary”. OIF is a multi-file format that includes an .oif file and a directory of .tif, .roi, .pty, .lut, and .bmp files. OIB is a single file format.

Commercial applications that support this format include:

- Bitplane Imaris
- SVI Huygens

See also:

Olympus FluoView Resource Center

### 4.2.105 Olympus FluoView TIFF

**Extensions**: .tif  
**Owner**: Olympus

**Support**

BSD-licensed: ✗  
Export: ✗

**Officially Supported Versions**:

Reader: FluoviewReader ([Source Code, Supported Metadata Fields](#))

**Freely Available Software**:

- [DIMIN](#)

We currently have:

- a FluoView specification document (from 2002 November 14, in DOC)
- Olympus’ FluoView Image File Reference Suite (from 2002 March 1, in DOC)
- several FluoView datasets

**Ratings**

**Pixels**: ▲  
**Metadata**: ▲  
**Openness**: ▲  
**Presence**: ▼  
**Utility**: ▼

**Additional Information**

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Commercial applications that support this format include:

- Bitplane Imaris
• SVI Huygens

4.2.106 Olympus OIR

Extensions: .oir
Owner: Olympus

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: OIRReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Olympus Viewer Plugin for ImageJ

We currently have:
• several OIR datasets
• public sample images

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

Additional Information

Support for this format was added in partnership with OLYMPUS EUROPA SE & Co. KG

4.2.107 Olympus OMP2INFO

Extensions: .omp2info
Owner: Olympusinfo

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: OlympusTileReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• Olympus Viewer Plugin for ImageJ

We currently have:
• several OMP2INFO datasets

Ratings
Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

Additional Information
Support for this format was added in partnership with OLYMPUS EUROPA SE & Co. KG

4.2.108 Olympus ScanR

Extensions: .xml, .dat, .tif
Developer: Olympus
Owner: Olympus

Support
BSD-licensed: ❌
Export: ❌
Officially Supported Versions: Up to 2.5.1
Reader: ScanRReader (Source Code, Supported Metadata Fields)
We currently have:
• several ScanR datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▼
Presence: ▼
Utility: ▼

4.2.109 Olympus SIS TIFF

Extensions: .tiff
Developer: Olympus

Support
BSD-licensed: ❌
Export: ❌

4.2. Supported Formats 183
Officially Supported Versions:
Reader: SISReader (Source Code, Supported Metadata Fields)
We currently have:
- a few example SIS TIFF files

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.110 OME-TIFF

Extensions: .ome.tiff, .ome.tif, .ome.tf2, .ome.tf8, .ome.btf
Developer: Open Microscopy Environment
Support
BSD-licensed: ✅
Export: ✅

Reader: OMETiffReader (Source Code, Supported Metadata Fields)
Writer: OMETiffWriter (Source Code)
We currently have:
- an OME-TIFF specification document
- many OME-TIFF datasets
- public sample images
- the ability to produce additional datasets

Ratings
Pixels: 🔺
Metadata: 🔺
Openness: 🔺
Presence: 
Utility: 🔺

Additional Information
Additional options are available for reading or writing this format type, see Additional reader and writer options for information.
Bio-Formats can save image stacks as OME-TIFF.
Commercial applications that support OME-TIFF are listed on our commercial partners page

See also:
OME-TIFF technical overview

### 4.2.111 OME-XML

Extensions: .ome, .ome.xml
Developer: Open Microscopy Environment

**Support**

BSD-licensed: ✔️

Export: ✔️


Reader: OMEXMLReader (Source Code, Supported Metadata Fields)
Writer: OMEXMLWriter (Source Code)

We currently have:

- OME-XML specification documents
- many OME-XML datasets
- public sample images
- the ability to produce more datasets

**Ratings**

Pixels: 🔹
Metadata: 🔹
Openness: 🔹
Presence: 🔴
Utility: 🔹

**Additional Information**

Bio-Formats uses the OME-XML Java library to read OME-XML files.
Commercial applications that support OME-XML include:

- Bitplane Imaris
- SVI Huygens
4.2.112 OMERO Pyramid

Developer: Open Microscopy Environment

Support

BSD-licensed: ✔

Export: ❌

Officially Supported Versions: 1.0.0

Reader: TiffReader (Source Code, Supported Metadata Fields)

We currently have:

• a specification document
• several OMERO Pyramid datasets
• the ability to produce additional datasets

Ratings

Pixels: 📈

Metadata: 📈

Openness: 📈

Presence: 📉

Utility: 📉

4.2.113 Oxford Instruments

Extensions: .top

Owner: Oxford Instruments

Support

BSD-licensed: ❌

Export: ❌

Officially Supported Versions:

Reader: OxfordInstrumentsReader (Source Code, Supported Metadata Fields)

We currently have:

• Pascal code that can read Oxford Instruments files (from ImageSXM)
• a few Oxford Instruments files

We would like to have:

• an official specification document
• more Oxford Instruments files

Ratings

Pixels: 📉
4.2.114 PCORAW

Extensions: .pcoraw, .rec
Developer: PCO

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: PCORAWReader (Source Code, Supported Metadata Fields)
We currently have:
  • a few example datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.115 PCX (PC Paintbrush)

Extensions: .pcx
Developer: ZSoft Corporation

Support

BSD-licensed: ✔
Export: ✗

Officially Supported Versions:
Reader: PCXReader (Source Code, Supported Metadata Fields)
We currently have:
  • several .pcx files
  • the ability to generate additional .pcx files
4.2.116 Perkin Elmer Densitometer

Extensions: .pds
Developer: Perkin Elmer

Support
BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: PDSReader (Source Code, Supported Metadata Fields)

We currently have:
• a few PDS datasets

We would like to have:
• an official specification document
• more PDS datasets

Ratings
Pixels: ▲
Metadata:▼
Openness:▼
Presence:▼
Utility:▼
4.2.117 PerkinElmer Columbus

Extensions: .xml, .csv, .tif

Owner: PerkinElmer

Support

BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: ColumbusReader (Source Code, Supported Metadata Fields)

We currently have:

• a few example datasets
• public sample images

We would like to have:

• an official specification document

Ratings

Pixels:

Metadata:

Openness:

Presence:

Utility:

4.2.118 PerkinElmer Nuance

Extensions: .im3

Developer: PerkinElmer

Support

BSD-licensed:

Export: ✗

Officially Supported Versions:

Reader: IM3Reader (Source Code, Supported Metadata Fields)

We currently have:

• a few sample datasets

Ratings

Pixels:

Metadata:

Openness:

Presence:

Utility:
4.2.119 PerkinElmer Operetta

Extensions: .tiff, .xml
Developer: PerkinElmer

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: OperettaReader (Source Code, Supported Metadata Fields)

We currently have:
• a few sample datasets
• public sample images

We would like to have:
• an official specification document
• more sample datasets

Ratings
Pixels: 🔴
Metadata: 🔴
Openness: 🟢
Presence: 🔴
Utility: 🟢

4.2.120 PerkinElmer UltraVIEW

Extensions: .tif, .2, .3, .4, etc.
Owner: PerkinElmer

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: PerkinElmerReader (Source Code, Supported Metadata Fields)

We currently have:
• several UltraVIEW datasets
**Ratings**

Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

**Additional Information**

Other associated extensions include: .tim, .zpo, .csv, .htm, .cfg, .ano, .rec

Commercial applications that support this format include:

- Bitplane Imaris
- Image-Pro Plus

See also:
PerkinElmer UltraVIEW system overview (pdf)

### 4.2.121 Portable Any Map

Extensions: .pbm, .pgm, .ppm

Developer: Netpbm developers

**Support**

BSD-licensed: ✔
Export: ❌

Officially Supported Versions:

Reader: PGMReader ([Source Code, Supported Metadata Fields](#))

Freely Available Software:

- Netpbm graphics filter

We currently have:

- a PGM specification document (from 2003 October 3, in HTML)
- a few PBM, PPM and PGM files

**Ratings**

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼
4.2.122 Adobe Photoshop PSD

Extensions: .psd
Developer: Adobe

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions: 1.0
Reader: PSDReader (Source Code, Supported Metadata Fields)

We currently have:
- a PSD specification document (v3.0.4, 16 July 1995)
- a few PSD files

We would like to have:
- more PSD files

Ratings

Pixels: 🙆
Metadata: 🙆
Openness: 🙆
Presence: 🙆
Utility: 🔄

4.2.123 Photoshop TIFF

Extensions: .tif, .tiff
Developer: Adobe

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: PhotoshopTiffReader (Source Code, Supported Metadata Fields)

We currently have:
- a Photoshop TIFF specification document
- a few Photoshop TIFF files

Ratings

Pixels: 🙆
Metadata: 🙆
4.2.124 PicoQuant Bin

Extensions: .bin
Developer: PicoQuant

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Reader: PQBinReader (Source Code, Supported Metadata Fields)

Freely Available Software:

• SymphoTime64

We currently have:

• a few example datasets

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.125 PICT (Macintosh Picture)

Extensions: .pict
Developer: Apple Computer

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Reader: PictReader (Source Code, Supported Metadata Fields)

We currently have:

• many PICT datasets
Bio-Formats, Release 6.14.0

Ratings

Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▲
Utility: ▼

Additional Information

QuickTime for Java is required for reading vector files and some compressed files but note that this is no longer available from Apple.

See also:
PICT technical overview Another PICT technical overview

4.2.126 PNG (Portable Network Graphics)

Extensions: .png
Developer: PNG Development Group

Support

BSD-licensed: ✔
Export: ✔

Officially Supported Versions:
Reader: APNGReader (Source Code, Supported Metadata Fields)
Writer: APNGWriter (Source Code)

Freely Available Software:
  • PNG Writer plugin for ImageJ

We currently have:
  • a PNG specification document (W3C/ISO/IEC version, from 2003 November 10, in HTML.)
  • several PNG datasets
  • public sample images

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▼

Additional Information

Bio-Formats uses the Java Image I/O API to read and write PNG files.
See also:

PNG technical overview

### 4.2.127 Prairie Technologies TIFF

Extensions: .tif, .xml, .cfg
Developer: Prairie Technologies

**Support**

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: PrairieReader ([Source Code, Supported Metadata Fields](#))
We currently have:
- many Prairie datasets

**Ratings**

Pixels: ▲
Metadata: ▲
Openness: ☐
Presence: ◯
Utility: ◯

### 4.2.128 Princeton Instruments SPE

Extensions: .spe
Developer: Princeton Instruments

**Support**

BSD-licensed: ✗
Export: ✗

Officially Supported Versions: 3.0
Reader: SPEReader ([Source Code, Supported Metadata Fields](#))
We currently have:
- An official specification document
- two SPE files

We would like to have:
- more SPE files
Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.129 Quesant

Extensions: .afm
Developer: Quesant Instrument Corporation
Owner: KLA-Tencor Corporation

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: QuesantReader (Source Code, Supported Metadata Fields)

We currently have:
• Pascal code that can read Quesant files (from ImageSXM)
• several Quesant files

We would like to have:
• an official specification document
• more Quesant files

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

196 Chapter 4. Formats
4.2.130 QuickTime Movie

Extensions: .mov
Owner: Apple Computer

Support

BSD-licensed: ✔️
Export: ✔️

Officially Supported Versions:

Readers:
- NativeQTRenderer (Source Code, Supported Metadata Fields)
- LegacyQTRenderer (Source Code, Supported Metadata Fields)

Writer: QTWriter (Source Code)

Freely Available Software:
- QuickTime Player

We currently have:
- a QuickTime specification document
- several QuickTime datasets
- the ability to produce more datasets

We would like to have:
- more QuickTime datasets, including:
  - files compressed with a common, unsupported codec
  - files with audio tracks and/or multiple video tracks

Ratings

Pixels: ★
Metadata: ▼
Openness: ▼
Presence: ★★
Utility: ▼

Additional Information

Bio-Formats has two modes of operation for QuickTime:
- The legacy QTJava mode requires QuickTime for Java which will only run with a 32-bit JVM and is no longer available from Apple.
- Native mode works on systems with no QuickTime (e.g. Linux).

Bio-Formats can save image stacks as QuickTime movies. The following table shows supported codecs:
<table>
<thead>
<tr>
<th>Codec</th>
<th>Description</th>
<th>Native</th>
<th>LegacyQTJava</th>
</tr>
</thead>
<tbody>
<tr>
<td>raw</td>
<td>Full Frames (Uncompressed)</td>
<td>read &amp; write</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>iraw</td>
<td>Intel YUV Uncompressed</td>
<td>read only</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>rle</td>
<td>Animation (run length encoded RGB)</td>
<td>read only</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>jpeg</td>
<td>Still Image JPEG DIB</td>
<td>read only</td>
<td>read only</td>
</tr>
<tr>
<td>rpza</td>
<td>Apple Video 16 bit “road pizza”</td>
<td>read only (partial)</td>
<td>read only</td>
</tr>
<tr>
<td>mjp</td>
<td>Motion JPEG codec</td>
<td>read only</td>
<td>read only</td>
</tr>
<tr>
<td>cvpd</td>
<td>Cinepak</td>
<td>.</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>svq1</td>
<td>Sorenson Video</td>
<td>.</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>svq3</td>
<td>Sorenson Video 3</td>
<td>.</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>mp4v</td>
<td>MPEG-4</td>
<td>.</td>
<td>read &amp; write</td>
</tr>
<tr>
<td>h263</td>
<td>H.263</td>
<td>.</td>
<td>read &amp; write</td>
</tr>
</tbody>
</table>

Note that the legacy QT reader/writer are deprecated and will be removed in Bio-Formats 7.0.0.

See also:
QuickTime software overview

### 4.2.131 RHK

Extensions: .sm2, .sm3

Owner: RHK Technologies

Support

BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: RHKReader ([Source Code, Supported Metadata Fields](#))

We currently have:

- Pascal code that can read RHK files (from ImageSXM)
- a few RHK files

We would like to have:

- an official specification document
- more RHK files

Ratings
4.2.132 SBIG

Owner: Diffraction Limited (formerly Santa Barbara Instrument Group)

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: SBIGReader (Source Code, Supported Metadata Fields)

We currently have:
• an official SBIG specification document
• a few SBIG files

We would like to have:
• more SBIG files

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ✗
Utility: ✗

4.2.133 Seiko

Extensions: .xqd, .xqf
Owner: Seiko

Support

BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: SeikoReader (Source Code, Supported Metadata Fields)

We currently have:
Bio-Formats, Release 6.14.0

- Pascal code that can read Seiko files (from ImageSXM)
- a few Seiko files

We would like to have:
- an official specification document
- more Seiko files

**Ratings**

Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

### 4.2.134 SimplePCI & HCImage

Extensions: .cxd
Developer: Compix

**Support**

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: PCIReader ([Source Code, Supported Metadata Fields](#))

We currently have:
- several SimplePCI files

**Ratings**

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

**Additional Information**

Bio-Formats uses a modified version of the Apache POI library to read CXD files.

**See also:**

SimplePCI software overview
4.2.135 SimplePCI & HClmage TIFF

Extensions: .tiff
Developer: Hamamatsu

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: SimplePCITiffReader (Source Code, Supported Metadata Fields)

We currently have:
• a few SimplePCI TIFF datasets

We would like to have:
• more SimplePCI TIFF datasets

Ratings
Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.136 SM Camera

Support
BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: SMCameraReader (Source Code, Supported Metadata Fields)

We currently have:
• Pascal code that can read SM-Camera files (from ImageSXM)
• a few SM-Camera files

We would like to have:
• an official specification document
• more SM-Camera files

Ratings
Pixels: ▼
Metadata: ▼
4.2.137 SPIDER

Extensions: .spi, .stk
Developer: Wadsworth Center

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: SpiderReader (Source Code, Supported Metadata Fields)

Freely Available Software:
- SPIDER

We currently have:
- a few example datasets
- official file format documentation

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.138 Targa

Extensions: .tga
Developer: Truevision

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: TargaReader (Source Code, Supported Metadata Fields)

We currently have:
- a Targa specification document
• a few Targa files

Ratings
- Pixels: ▲
- Metadata: ▲
- Openness: ▲
- Presence: ▼
- Utility: ▼

4.2.139 Tecan Spark Cyto Workspace

Extensions: .db, .tif
Owner: Tecan Trading

Support
- BSD-licensed: ✗
- Export: ✗

Officially Supported Versions:
- Reader: TecanReader (Source Code, Supported Metadata Fields)

We currently have:
- • a few Tecan Spark Cyto workspaces

Ratings
- Pixels: ▲
- Metadata: ▲
- Openness: ▼
- Presence: ▼
- Utility: ▼

Additional Information
Fiji users will need to enable the “Tissue Analyzer” update site and install “sqlite-jdbc.jar”

4.2.140 Text

Extensions: .txt

Support
- BSD-licensed: ✔
- Export: ✗

Officially Supported Versions:
- Reader: TextReader (Source Code, Supported Metadata Fields)

4.2. Supported Formats
We currently have:

**Ratings**

Pixels: ![Rating Icon]
Metadata: ![Rating Icon]
Openness: ![Rating Icon]
Presence: ![Rating Icon]
Utility: ![Rating Icon]

**Additional Information**

Reads tabular pixel data produced by a variety of software.

### 4.2.141 TIFF (Tagged Image File Format)

**Extensions:** .tiff, .tif, .tf2, .tf8, .btf

**Developer:** Aldus and Microsoft

**Owner:** Adobe

**Support**

BSD-licensed: ![License Icon]
Export: ![License Icon]

**Officially Supported Versions:**

**Reader:** TiffReader *(Source Code, Supported Metadata Fields)*

**Writer:** TiffWriter *(Source Code)*

**Sample Datasets:**

- Big TIFF

We currently have:

- TIFF specification documents from Adobe
- a TIFF specification document
- public sample images
- many TIFF datasets
- a few BigTIFF datasets

**Ratings**

Pixels: ![Rating Icon]
Metadata: ![Rating Icon]
Openness: ![Rating Icon]
Presence: ![Rating Icon]
Utility: ![Rating Icon]

**Additional Information**
Bio-Formats can also read BigTIFF files (TIFF files larger than 4 GB). Bio-Formats can save image stacks as TIFF or BigTIFF.

TIFF files written by ImageJ are also supported, including ImageJ TIFFs larger than 4GB. ImageJ TIFFs are detected based upon the text in the first IFD’s “ImageDescription” tag; this tag’s value is then used to determine Z, C, and T sizes as well as physical sizes and timestamps. For ImageJ TIFFs larger than 4GB, a single IFD is expected (instead of one IFD per image plane). The “ImageDescription” is used to determine the number of images, the pixel data for which are expected to be stored contiguously at the offset indicated in the sole IFD. This differs from standard TIFF and BigTIFF; if the “ImageDescription” tag is missing or invalid, only the first image will be read.

See also:

TIFF technical overview BigTIFF technical overview ImageJ TIFF overview Source code for ImageJ’s native TIFF reader

### 4.2.142 TillPhotonics TillVision

Extensions: .vws

Developer: TILL Photonics

Support

BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: TillVisionReader (Source Code, Supported Metadata Fields)

We currently have:

- several TillVision datasets

We would like to have:

- an official specification document

Ratings

Pixels: 

Metadata: 

Openness: 

Presence: 

Utility: 

---

4.2. Supported Formats
4.2.143 Topometrix

Extensions: .tfr, .ffr, .zfr, .zfp, .2fl
Owner: TopoMetrix (now Veeco)

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: TopometrixReader (Source Code, Supported Metadata Fields)

We currently have:
  • Pascal code that reads Topometrix files (from ImageSXM)
  • a few Topometrix files

We would like to have:
  • an official specification document
  • more Topometrix files

Ratings

Pixels: ☑
Metadata: ☑
Openness: ☑
Presence: ☑
Utility: ☑

4.2.144 Trestle

Extensions: .tif, .sld, .jpg

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: TrestleReader (Source Code, Supported Metadata Fields)

Sample Datasets:
  • OpenSlide

We currently have:
  • a few example datasets
  • public sample images
  • developer documentation from the OpenSlide project
Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.145 UBM

Extensions: .pr3

Support

BSD-licensed: 
Export: 

Officially Supported Versions:

Reader: UBMReader (Source Code, Supported Metadata Fields)

We currently have:

• Pascal code that can read UBM files (from ImageSXM)
• one UBM file

We would like to have:

• an official specification document
• more UBM files

Ratings

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.146 Unisoku

Extensions: .dat, .hdr

Owner: Unisoku

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: UnisokuReader (Source Code, Supported Metadata Fields)

We currently have:

- Pascal code that can read Unisoku files (from ImageSXM)
- a few Unisoku files

We would like to have:

- an official specification document
- more Unisoku files

**Ratings**

Pixels: 

Metadata: 

Openness: 

Presence: 

Utility: 

### 4.2.147 Varian FDF

Extensions: .fdf

Developer: Varian, Inc.

Owner: Agilent Technologies

**Support**

BSD-licensed: 

Export: 

Officially Supported Versions:

Reader: VarianFDFReader (Source Code, Supported Metadata Fields)

We currently have:

- a few Varian FDF datasets

We would like to have:

- an official specification document
- more Varian FDF datasets

**Ratings**

Pixels: 

Metadata: 

Openness: 

Presence: 

Utility: 

4.2.148 Vectra QPTIFF

Extensions: .tif, .qptiff
Owner: Akoya Biosciences, formerly owned by PerkinElmer

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: VectraReader (Source Code, Supported Metadata Fields)

We currently have:
• a specification document
• public sample images
• several datasets

Ratings
 Pixels: 🔺
 Metadata: 🔺
 Openness: 🔺
 Presence: 🔻
 Utility: 🔻

Additional Information
Support for this format was added in partnership with PerkinElmer and updated in partnership with Akoya Biosciences

4.2.149 Veeco AFM

Extensions: .hdf
Developer: Veeco

Support
BSD-licensed: ✗
Export: ✗

Officially Supported Versions:
Reader: VeecoReader (Source Code, Supported Metadata Fields)

We currently have:
• a few sample datasets

Ratings
 Pixels: 🟢
 Metadata: 🟢
4.2.150 Ventana BIF

Extensions: .bif
Owner: Roche Digital Diagnostics

Support
BSD-licensed: 
Export: 
Officially Supported Versions:
Reader: VentanaReader (Source Code, Supported Metadata Fields)

We currently have:
• some Ventana datasets
• public sample images

Ratings
Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

4.2.151 VG SAM

Extensions: .dti

Support
BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: VGSAMReader (Source Code, Supported Metadata Fields)

We currently have:
• a few VG-SAM files

We would like to have:
• an official specification document
• more VG-SAM files
4.2.152 VisiTech XYS

Extensions: .xys, .html
Developer: VisiTech International

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: VisitechReader (Source Code, Supported Metadata Fields)

We currently have:
- several VisiTech datasets

We would like to have:
- an official specification document

4.2.153 Volocity

Extensions: .mvd2
Developer: PerkinElmer

Support

BSD-licensed: 
Export: 

Officially Supported Versions:
Reader: VolocityReader (Source Code, Supported Metadata Fields)
Sample Datasets:

- PerkinElmer Downloads

We currently have:

- many example Volocity datasets

We would like to have:

- an official specification document
- any Volocity datasets that do not open correctly

**Ratings**

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

**Additional Information**

.mvd2 files are Metakit database files.

### 4.2.154 Volocity Library Clipping

Extensions: .acff

Developer: PerkinElmer

**Support**

BSD-licensed: ✗

Export: ✗

Officially Supported Versions:

Reader: VolocityClippingReader (Source Code, Supported Metadata Fields)

We currently have:

- several Volocity library clipping datasets

We would like to have:

- any datasets that do not open correctly
- an official specification document

**Ratings**

Pixels: 
Metadata: 
Openness: 
Presence: 
Utility: 

Additional Information
RGB .acff files are not yet supported. See #6413.

4.2.155 WA-TOP

Extensions: .wat
Developer: WA Technology
Owner: Oxford Instruments

Support
BSD-licensed: x
Export: x

Officially Supported Versions:
Reader: WATOPReader (Source Code, Supported Metadata Fields)

We currently have:
• Pascal code that can read WA-TOP files (from ImageSXM)
• a few WA-TOP files

We would like to have:
• an official specification document
• more WA-TOP files

Ratings
Pixels: ▼
Metadata: ▼
Openness: ▼
Presence: ▼
Utility: ▼

4.2.156 Windows Bitmap

Extensions: .bmp
Developer: Microsoft and IBM

Support
BSD-licensed: ✔
Export: x

Officially Supported Versions:
Reader: BMPReader (Source Code, Supported Metadata Fields)

Freely Available Software:
• BMP Writer plugin for ImageJ
We currently have:
  • many BMP datasets

**Ratings**

Pixels: ▲
Metadata: ▼
Openness: ▼
Presence: ▲
Utility: ▼

**Additional Information**

Compressed BMP files are currently not supported.

See also:

Technical Overview

### 4.2.157 Woolz

Extensions: .wlz

Developer: MRC Human Genetics Unit

**Support**

BSD-licensed: ✗
Export: ✓

Officially Supported Versions:

Reader: WlzReader (Source Code, Supported Metadata Fields)

Writer: WlzWriter (Source Code)

Freely Available Software:

  • Woolz

We currently have:

  • a few Woolz datasets

**Ratings**

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

**Additional Information**

Both the reader and the writer requires the Woolz image processing library to be installed - see https://www.emouseatlas.org/emap/analysis_tools_resources/software/woolz.html for more details
Support for the Woolz format will be removed from Bio-Formats 7.0.0 onwards. For users who wish to retain support for existing Woolz files, we recommend converting these file-sets to an actively maintained open format such as OME-TIFF. This can be achieved using the Bio-Formats `bfconvert` command line tool as outlined in command line tools.

### 4.2.158 Zeiss Axio CSM

Extensions: .lms  
Developer: ZEISS International  
Owner: ZEISS International  
**Support**  
BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:  
Reader: ZeissLMSReader ([Source Code](#), [Supported Metadata Fields](#))  
We currently have:  
- one example dataset  

**Ratings**  
Pixels:  
Metadata:  
Openness:  
Presence:  
Utility:  

**Additional Information**  
This should not be confused with the more common Zeiss LSM format, which has a similar extension. As far as we know, the Axio CSM 700 system is the only one which saves files in the .lms format.

### 4.2.159 Zeiss AxioVision TIFF

Extensions: .xml, .tif  
Developer: ZEISS International  
Owner: ZEISS International  
**Support**  
BSD-licensed: ✗  
Export: ✗  
Officially Supported Versions:  
Reader: ZeissTIFFReader ([Source Code](#), [Supported Metadata Fields](#))  
Freely Available Software:  
- Zeiss ZEN Lite
Bio-Formats, Release 6.14.0

We currently have:

- many example datasets

We would like to have:

- an official specification document

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▼
Utility: ▼

4.2.160 Zeiss AxioVision ZVI (Zeiss Vision Image)

Extensions: .zvi
Developer: Carl Zeiss Microscopy GmbH
Owner: ZEISS International

Support

BSD-licensed: ✗
Export: ✗
Officially Supported Versions: 1.0, 2.0
Reader: ZeissZVIReader (Source Code, Supported Metadata Fields)

Freely Available Software:

- Zeiss Zen

We currently have:

- a ZVI specification document (v2.0.5, from 2010 August, in PDF)
- an older ZVI specification document (v2.0.2, from 2006 August 23, in PDF)
- an older ZVI specification document (v2.0.1, from 2005 April 21, in PDF)
- an older ZVI specification document (v1.0.26.01.01, from 2001 January 29, in DOC)
- Zeiss’ ZvImageReader code (v1.0, from 2001 January 25, in C++)
- many ZVI datasets

Ratings

Pixels: ▲
Metadata: ▲
Openness: ▲
Presence: ▲
Utility: ▲
Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Bio-Formats uses a modified version of the Apache POI library to read ZVI files. ImageJ/FIJI will use the ZVI reader plugin in preference to Bio-Formats if both are installed. If you have a problem which is solved by opening the file using the Bio-Formats Importer plugin, you can just remove the ZVI_Reader.class from the plugins folder.

Commercial applications that support ZVI include Bitplane Imaris.

As of May 2021, the proprietary ZEISS AxioVision software is classed as End of Support. Zeiss ZEN is the successor programme to AxioVision.

### 4.2.161 Zeiss CZI

**Extensions:** .czi

**Developer:** ZEISS International

**Support**

- BSD-licensed:  
- Export:  

**Officially Supported Versions:**

- Reader: ZeissCZIReader ([Source Code, Supported Metadata Fields](#))

**Freely Available Software:**

- Zeiss ZEN
- libCZI

We currently have:

- many example datasets
- public sample images

**Ratings**

- Pixels: ▲
- Metadata: ▲
- Openness: ▲
- Presence: ▼
- Utility: ■

**Additional Information**

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Additional options are available for reading or writing this format type, see [Additional reader and writer options](#) for information.

JPEG-XR compressed CZI files are supported on the following 64-bit platforms:

- Windows 7 and above with Visual Studio 2015 C++ Redistributable
Bio-Formats, Release 6.14.0

- CentOS 6 and above, Ubuntu 12.04 and above
- OS X 10.10 and above

4.2.162 Zeiss LSM (Laser Scanning Microscope) 510/710

Extensions: .lsm, .mdb
Owner: ZEISS International

Support

BSD-licensed: ❌
Export: ❌

Officially Supported Versions:
Reader: ZeissLSMReader (Source Code, Supported Metadata Fields)

Freely Available Software:

- Zeiss LSM Image Browser
- LSM Toolbox plugin for ImageJ
- LSM Reader plugin for ImageJ
- DIMIN

We currently have:

- LSM specification v3.2, from 2003 March 12, in PDF
- LSM specification v5.5, from 2009 November 23, in PDF
- LSM specification v6.0, from 2010 September 28, in PDF
- many LSM datasets

Ratings

Pixels: 🔺
Metadata: 🔺
Openness: 🟡
Presence: 🔺
Utility: 🟡

Additional Information

Please note that while we have specification documents for this format, we are not able to distribute them to third parties.

Bio-Formats uses the MDB Tools Java port

Commercial applications that support this format include:

- SVI Huygens
- Bitplane Imaris
- Amira
4.3 Summary of supported metadata fields

4.3.1 Format readers

<table>
<thead>
<tr>
<th>Reader</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>AFIReader</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>445</td>
</tr>
<tr>
<td>AIMReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>APLReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>APNGReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ARFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>AVIReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>AliconaReader</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>443</td>
</tr>
<tr>
<td>AmiraReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>AnalyzeReader</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>452</td>
</tr>
<tr>
<td>BDRReader</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>417</td>
</tr>
<tr>
<td>BDVReader</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>439</td>
</tr>
<tr>
<td>BIFormatReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>BMPReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>BaseTiffReader</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>448</td>
</tr>
<tr>
<td>BaseZeissReader</td>
<td>84</td>
<td>0</td>
<td>0</td>
<td>392</td>
</tr>
<tr>
<td>BioRadGelReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>BioRadReader</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>436</td>
</tr>
<tr>
<td>BioRadSCNReader</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>447</td>
</tr>
<tr>
<td>BrukerReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>BurleighReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>CV7000Reader</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>420</td>
</tr>
<tr>
<td>CanonRawReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>CellH5Reader</td>
<td>41</td>
<td>0</td>
<td>0</td>
<td>435</td>
</tr>
<tr>
<td>CellSensReader</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>428</td>
</tr>
<tr>
<td>CellVoyagerReader</td>
<td>39</td>
<td>0</td>
<td>0</td>
<td>437</td>
</tr>
<tr>
<td>CellWorxReader</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>426</td>
</tr>
<tr>
<td>CellomicsReader</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>440</td>
</tr>
<tr>
<td>ColumbusReader</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td>436</td>
</tr>
<tr>
<td>DNGReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>DeltavisionReader</td>
<td>51</td>
<td>0</td>
<td>0</td>
<td>425</td>
</tr>
<tr>
<td>DicomReader</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>449</td>
</tr>
<tr>
<td>EPSReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>Ecat7Reader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>FEIRReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>FEITiffReader</td>
<td>39</td>
<td>0</td>
<td>0</td>
<td>437</td>
</tr>
<tr>
<td>FV1000Reader</td>
<td>113</td>
<td>0</td>
<td>0</td>
<td>363</td>
</tr>
<tr>
<td>FakeReader</td>
<td>91</td>
<td>0</td>
<td>0</td>
<td>385</td>
</tr>
<tr>
<td>FilePatternReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>FitsReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>FlexReader</td>
<td>71</td>
<td>0</td>
<td>0</td>
<td>405</td>
</tr>
<tr>
<td>FlowSightReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Reader</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>FluoviewReader</td>
<td>49</td>
<td>0</td>
<td>0</td>
<td>427</td>
</tr>
<tr>
<td>FujiReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>GIFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>GatanDM2Reader</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>446</td>
</tr>
<tr>
<td>GatanReader</td>
<td>69</td>
<td>0</td>
<td>0</td>
<td>407</td>
</tr>
<tr>
<td>GelReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>HISReader</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>449</td>
</tr>
<tr>
<td>HRDGDFReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>HamamatsuVMSReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>HitachiReader</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>445</td>
</tr>
<tr>
<td>T2IRReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ICSReader</td>
<td>72</td>
<td>0</td>
<td>0</td>
<td>404</td>
</tr>
<tr>
<td>IM3Reader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>IMODReader</td>
<td>44</td>
<td>0</td>
<td>0</td>
<td>432</td>
</tr>
<tr>
<td>INRReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>IPLabReader</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>445</td>
</tr>
<tr>
<td>IPWReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>ImaconReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>ImageIOReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ImagicReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>ImarisHDFReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>ImarisReader</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>444</td>
</tr>
<tr>
<td>ImarisTiffReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>ImprovisionTiffReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>InspectorReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>InCell3000Reader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>InCellReader</td>
<td>69</td>
<td>0</td>
<td>0</td>
<td>407</td>
</tr>
<tr>
<td>InveonReader</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>446</td>
</tr>
<tr>
<td>IonpathMIBITiffReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>IvisionReader</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>442</td>
</tr>
<tr>
<td>JEOLReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>JPEG2000Reader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>JPEGReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>JPReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>JPXReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>KLBReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>KhorosReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>KodakReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>L2DReader</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>447</td>
</tr>
<tr>
<td>LEOReader</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>449</td>
</tr>
<tr>
<td>LIFReader</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>391</td>
</tr>
<tr>
<td>LIMReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>LOFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>LegacyND2Reader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>LegacyQTRReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>LeicaReader</td>
<td>56</td>
<td>0</td>
<td>0</td>
<td>420</td>
</tr>
<tr>
<td>LeicaSCCNReader</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>443</td>
</tr>
<tr>
<td>LiFlimReader</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>451</td>
</tr>
</tbody>
</table>

continues on next page
### Table 3 – continued from previous page

<table>
<thead>
<tr>
<th>Reader</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIASReader</td>
<td>67</td>
<td>0</td>
<td>0</td>
<td>409</td>
</tr>
<tr>
<td>MINCReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>MNGReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>MRCReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>MRWReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>MetamorphReader</td>
<td>59</td>
<td>0</td>
<td>0</td>
<td>417</td>
</tr>
<tr>
<td>MetamorphTiffReader</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>433</td>
</tr>
<tr>
<td>MetaxpressTiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>MicroCTReader</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>452</td>
</tr>
<tr>
<td>MicrometerManagerReader</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>434</td>
</tr>
<tr>
<td>MikrosanTiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>MinimalTiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>MolecularImagingReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>NAFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ND2Reader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>NDPIReader</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>444</td>
</tr>
<tr>
<td>NDPISReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>NRRDReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>NativeND2Reader</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>424</td>
</tr>
<tr>
<td>NativeQTReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>NifitiReader</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>452</td>
</tr>
<tr>
<td>NikonElementsTiffReader</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>426</td>
</tr>
<tr>
<td>NikonReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>NikonTiffReader</td>
<td>47</td>
<td>0</td>
<td>0</td>
<td>429</td>
</tr>
<tr>
<td>OBFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>OIRReader</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>428</td>
</tr>
<tr>
<td>OMETiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>OMEXMLReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>OlympusTileReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>OpenlabRawReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>OpenlabReader</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>444</td>
</tr>
<tr>
<td>OperettaReader</td>
<td>60</td>
<td>0</td>
<td>0</td>
<td>416</td>
</tr>
<tr>
<td>OxfordInstrumentsReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>PCIReader</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>447</td>
</tr>
<tr>
<td>PCORAWReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>PCXReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PDSReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>PGMReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PQBinReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>PSDReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PerkinElmerReader</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>446</td>
</tr>
<tr>
<td>PhotoshopTiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PictReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PovrayReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>PrairieReader</td>
<td>46</td>
<td>0</td>
<td>0</td>
<td>430</td>
</tr>
<tr>
<td>PyramidTiffReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>QTReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>QuesantReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Reader</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCPNLReader</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>451</td>
</tr>
<tr>
<td>RHKReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>SBIGReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>SDTReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>SEQReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>SIFReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>SISReader</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>443</td>
</tr>
<tr>
<td>SMCameraReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>SPCReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>SPEReader</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>446</td>
</tr>
<tr>
<td>SVSReader</td>
<td>31</td>
<td>0</td>
<td>0</td>
<td>445</td>
</tr>
<tr>
<td>ScanrReader</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>433</td>
</tr>
<tr>
<td>SeikoReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>SimplePCITiffReader</td>
<td>33</td>
<td>0</td>
<td>0</td>
<td>443</td>
</tr>
<tr>
<td>SlideBook7Reader</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>439</td>
</tr>
<tr>
<td>SlidebookReader</td>
<td>34</td>
<td>0</td>
<td>0</td>
<td>442</td>
</tr>
<tr>
<td>SlidebookTiffReader</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>446</td>
</tr>
<tr>
<td>SpiderReader</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>455</td>
</tr>
<tr>
<td>TCSReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>TargaReader</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>456</td>
</tr>
<tr>
<td>TecanReader</td>
<td>36</td>
<td>0</td>
<td>0</td>
<td>440</td>
</tr>
<tr>
<td>TextReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>TiffDelegateReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>TiffJAIReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>TiffReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>TileJPEGReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>TillVisionReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>TopometrixReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>TrestleReader</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>449</td>
</tr>
<tr>
<td>UBMReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>UnisokuReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>VGSAMReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>VarianFDFReader</td>
<td>25</td>
<td>0</td>
<td>0</td>
<td>451</td>
</tr>
<tr>
<td>VectraReader</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>433</td>
</tr>
<tr>
<td>VeecoReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>VentanaReader</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>448</td>
</tr>
<tr>
<td>VisitechReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>VelocityClippingReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>VelocityReader</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>438</td>
</tr>
<tr>
<td>WATOPReader</td>
<td>22</td>
<td>0</td>
<td>0</td>
<td>454</td>
</tr>
<tr>
<td>WzReader</td>
<td>26</td>
<td>0</td>
<td>0</td>
<td>450</td>
</tr>
<tr>
<td>XLEFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ZeissCZIReader</td>
<td>175</td>
<td>0</td>
<td>0</td>
<td>301</td>
</tr>
<tr>
<td>ZeissLMSReader</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>453</td>
</tr>
<tr>
<td>ZeissLSMReader</td>
<td>101</td>
<td>0</td>
<td>0</td>
<td>375</td>
</tr>
<tr>
<td>ZeissTIFFReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ZeissZVIReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
<tr>
<td>ZipReader</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>457</td>
</tr>
</tbody>
</table>
## 4.3.2 Metadata fields

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Arc - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Arc - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Arc - Model</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Arc - Power</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Arc - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Arc - Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>BooleanAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>BooleanAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>BooleanAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>BooleanAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>BooleanAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Channel - AcquisitionMode</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Channel - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Channel - Color</td>
<td>20</td>
<td>0</td>
<td>0</td>
<td>165</td>
</tr>
<tr>
<td>Channel - ContrastMethod</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Channel - EmissionWavelength</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>161</td>
</tr>
<tr>
<td>Channel - ExcitationWavelength</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>Channel - FilterSetRef</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Channel - Fluor</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Channel - ID</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Channel - IlluminationType</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Channel - LightSourceSettingsAttenuation</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Channel - LightSourceSettingsID</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Channel - LightSourceSettingsWavelength</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Channel - NDFilter</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Channel - Name</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>137</td>
</tr>
<tr>
<td>Channel - PinholeSize</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>174</td>
</tr>
<tr>
<td>Channel - PockelCellSetting</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Channel - SamplesPerPixel</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CommentAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>CommentAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>CommentAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>CommentAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>CommentAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Dataset - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - ExperimenterGroupRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - ExperimenterRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - ImageRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dataset - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Detector - AmplificationGain</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Detector - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Detector - Gain</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Detector - ID</td>
<td>37</td>
<td>0</td>
<td>0</td>
<td>148</td>
</tr>
</tbody>
</table>

continues on next page
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detector - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Detector - Manufacturer</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Detector - Model</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>Detector - Offset</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Detector - SerialNumber</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Detector - Type</td>
<td>28</td>
<td>0</td>
<td>0</td>
<td>157</td>
</tr>
<tr>
<td>Detector - Voltage</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Detector - Zoom</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>DetectorSettings - Binning</td>
<td>19</td>
<td>0</td>
<td>0</td>
<td>166</td>
</tr>
<tr>
<td>DetectorSettings - Gain</td>
<td>21</td>
<td>0</td>
<td>0</td>
<td>164</td>
</tr>
<tr>
<td>DetectorSettings - ID</td>
<td>35</td>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>DetectorSettings - Offset</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>DetectorSettings - ReadOutRate</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>DetectorSettings - Voltage</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Dichroic - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Dichroic - ID</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Dichroic - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Dichroic - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Dichroic - Model</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Dichroic - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>DoubleAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>DoubleAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>DoubleAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>DoubleAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>DoubleAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Ellipse - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - FontSize</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Ellipse - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - ID</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Ellipse - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - RadiusX</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Ellipse - RadiusY</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Ellipse - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Ellipse - StrokeDashArray</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - StrokeWidth</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Ellipse - Text</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Ellipse - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Ellipse - TheT</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Ellipse - TheZ</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Ellipse - Transform</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Ellipse - X</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Ellipse - Y</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Experiment - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Experiment - Description</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Experiment - ExperimenterRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Experiment - ID</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
</tbody>
</table>

continues on next page
### Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment - Type</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Experimenter - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Experimenter - Email</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Experimenter - FirstName</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Experimenter - ID</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Experimenter - Institution</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Experimenter - LastName</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>Experimenter - MiddleName</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Experimenter - UserName</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>ExperimenterGroup - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ExperimenterGroup - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ExperimenterGroup - ExperimenterRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ExperimenterGroup - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ExperimenterGroup - Leader</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ExperimenterGroup - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Filament - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Filament - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filament - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filament - Model</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filament - Power</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filament - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filament - Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>FileAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>FileAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>FileAnnotation - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Filter - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Filter - FilterWheel</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Filter - ID</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Filter - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filter - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filter - Model</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Filter - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Filter - Type</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - DichroicRef</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - EmissionFilterRef</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - ExcitationFilterRef</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - ID</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>FilterSet - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>FilterSet - Model</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>FilterSet - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Folder - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Folder - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Folder - FolderRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Folder - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Folder - ImageRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Folder - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
</tbody>
</table>

continues on next page

### 4.3. Summary of supported metadata fields

225
## Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Folder - ROIRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Image - AcquisitionDate</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Image - AnnotationRef</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Image - Description</td>
<td>48</td>
<td>0</td>
<td>0</td>
<td>137</td>
</tr>
<tr>
<td>Image - ExperimentRef</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Image - ExperimenterGroupRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Image - ExperimenterRef</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Image - ID</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Image - InstrumentRef</td>
<td>52</td>
<td>0</td>
<td>0</td>
<td>133</td>
</tr>
<tr>
<td>Image - MicrobeamManipulationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Image - Name</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Image - ROIRef</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>ImagingEnvironment - AirPressure</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ImagingEnvironment - CO2Percent</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ImagingEnvironment - Humidity</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ImagingEnvironment - Temperature</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>175</td>
</tr>
<tr>
<td>Instrument - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Instrument - ID</td>
<td>58</td>
<td>0</td>
<td>0</td>
<td>127</td>
</tr>
<tr>
<td>Label - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - FontSize</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Label - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - ID</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Label - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Label - StrokeDashArray</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - StrokeWidth</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Label - Text</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Label - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - TheT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - TheZ</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - Transform</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Label - X</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Label - Y</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Laser - FrequencyMultiplication</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Laser - ID</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>174</td>
</tr>
<tr>
<td>Laser - LaserMedium</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Laser - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Laser - Manufacturer</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Laser - Model</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Laser - PockelCell</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Laser - Power</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Laser - Pulse</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Laser - Pump</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Laser - RepetitionRate</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Laser - SerialNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Laser - Tuneable</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
</tbody>
</table>

continues on next page
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Laser - Type</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Laser - Wavelength</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>LightEmittingDiode - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>LightEmittingDiode - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LightEmittingDiode - Manufacturer</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LightEmittingDiode - Model</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LightEmittingDiode - Power</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LightPath - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>LightPath - DichroicRef</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>LightPath - EmissionFilterRef</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>LightPath - ExcitationFilterRef</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Line - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - FontStyle</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Line - TextStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - ID</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Line - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - MarkerEnd</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - MarkerStart</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Line - StrokeDashArray</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - StrokeWidth</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Line - Text</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Line - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Line - TheT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Line - TheZ</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Line - Transform</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Line - X1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Line - X2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Line - Y1</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>Line - Y2</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>178</td>
</tr>
<tr>
<td>ListAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ListAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ListAnnotation - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ListAnnotation - Namespace</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>LongAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>LongAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>LongAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LongAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>LongAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Mask - BinData</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Mask - BinDataBigEndian</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Mask - BinDataBigLength</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - BinDataCompression</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - FillColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Mask - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
</tbody>
</table>

continues on next page
Table 4 – continued from previous page

<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mask - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - FontSize</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - Height</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Mask - ID</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Mask - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Mask - StrokeDashArray</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - StrokeWidth</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - Text</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - TheT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - TheZ</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - Transform</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Mask - Width</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Mask - X</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Mask - Y</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>MicrobeamManipulation - ExperimenterRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>MicrobeamManipulation - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>MicrobeamManipulation - ROIRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>MicrobeamManipulation - Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>MicrobeamManipulationLightSourceSettings - Attenuation</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>MicrobeamManipulationLightSourceSettings - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Microscope - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Microscope - Manufacturer</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Microscope - Model</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>172</td>
</tr>
<tr>
<td>Microscope - SerialNumber</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Microscope - Type</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Objective - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Objective - CalibratedMagnification</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Objective - Correction</td>
<td>27</td>
<td>0</td>
<td>0</td>
<td>158</td>
</tr>
<tr>
<td>Objective - ID</td>
<td>43</td>
<td>0</td>
<td>0</td>
<td>142</td>
</tr>
<tr>
<td>Objective - Immersion</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>156</td>
</tr>
<tr>
<td>Objective - Iris</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Objective - LensNA</td>
<td>24</td>
<td>0</td>
<td>0</td>
<td>161</td>
</tr>
<tr>
<td>Objective - LotNumber</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Objective - Manufacturer</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Objective - Model</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>168</td>
</tr>
<tr>
<td>Objective - NominalMagnification</td>
<td>35</td>
<td>0</td>
<td>0</td>
<td>150</td>
</tr>
<tr>
<td>Objective - SerialNumber</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Objective - WorkingDistance</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>ObjectiveSettings - CorrectionCollar</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ObjectiveSettings - ID</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>147</td>
</tr>
<tr>
<td>ObjectiveSettings - Medium</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ObjectiveSettings - RefractiveIndex</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>175</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pixels - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Pixels - BigEndian</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - DimensionOrder</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - ID</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - Interleaved</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - PhysicalSizeX</td>
<td>97</td>
<td>0</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>Pixels - PhysicalSizeY</td>
<td>97</td>
<td>0</td>
<td>0</td>
<td>88</td>
</tr>
<tr>
<td>Pixels - PhysicalSizeZ</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>135</td>
</tr>
<tr>
<td>Pixels - SignificantBits</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - SizeC</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - SizeT</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - SizeX</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - SizeY</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - SizeZ</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pixels - TimeIncrement</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>168</td>
</tr>
<tr>
<td>Pixels - Type</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plane - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Plane - DeltaT</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>156</td>
</tr>
<tr>
<td>Plane - ExposureTime</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>147</td>
</tr>
<tr>
<td>Plane - HashSHA1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Plane - PositionX</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>147</td>
</tr>
<tr>
<td>Plane - PositionY</td>
<td>38</td>
<td>0</td>
<td>0</td>
<td>147</td>
</tr>
<tr>
<td>Plane - PositionZ</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>153</td>
</tr>
<tr>
<td>Plane - TheC</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plane - TheT</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plane - TheZ</td>
<td>185</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Plate - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Plate - ColumnNamingConvention</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Plate - Columns</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>Plate - Description</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Plate - ExternalIdentifier</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>Plate - ID</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>Plate - Name</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>172</td>
</tr>
<tr>
<td>Plate - RowNamingConvention</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Plate - Rows</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>170</td>
</tr>
<tr>
<td>Plate - Status</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Plate - WellOriginX</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Plate - WellOriginY</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>PlateAcquisition - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>PlateAcquisition - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>PlateAcquisition - EndTime</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>PlateAcquisition - ID</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>174</td>
</tr>
<tr>
<td>PlateAcquisition - MaximumFieldCount</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>175</td>
</tr>
<tr>
<td>PlateAcquisition - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>PlateAcquisition - StartTime</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>PlateAcquisition - WellSampleRef</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>174</td>
</tr>
<tr>
<td>Point - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
</tbody>
</table>

continues on next page

4.3. Summary of supported metadata fields 229
<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Point - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - FontSize</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Point - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - ID</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Point - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Point - StrokeDashArray</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Point - StrokeWidth</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Point - Text</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Point - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - TheT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Point - TheZ</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Point - Transform</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Point - X</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Point - Y</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Polygon - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - FontSize</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Polygon - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - ID</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Polygon - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - Points</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>177</td>
</tr>
<tr>
<td>Polygon - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polygon - StrokeDashArray</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polygon - StrokeWidth</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Polygon - Text</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Polygon - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polygon - TheT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polygon - TheZ</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Polygon - Transform</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polyline - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - FontSize</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Polyline - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - ID</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Polyline - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - MarkerEnd</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - MarkerStart</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - Points</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>179</td>
</tr>
<tr>
<td>Polyline - StrokeColor</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polyline - StrokeDashArray</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polyline - StrokeWidth</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Polyline - Text</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Polyline - TheC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Polyline - TheT</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Polyline - TheZ</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
</tbody>
</table>

Table 4 – continued from previous page

continues on next page
<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsupported</th>
<th>Partial</th>
<th>Unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polylines - Transform</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Project - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - DatasetRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - ExperimenterGroupRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - ExperimenterRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Project - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ROI - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>ROI - Description</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>ROI - ID</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>ROI - Name</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Reagent - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Reagent - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Reagent - ID</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Reagent - Name</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Reagent - ReagentIdentifier</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - FillColor</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - FillRule</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - FontFamily</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - FontSize</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Rectangle - FontStyle</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - Height</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Rectangle - ID</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Rectangle - Locked</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - StrokeColor</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Rectangle - StrokeDashArray</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Rectangle - StrokeWidth</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Rectangle - Text</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>Rectangle - TheC</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>Rectangle - TheT</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Rectangle - TheZ</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>Rectangle - Transform</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Rectangle - Width</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Rectangle - X</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Rectangle - Y</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td>173</td>
</tr>
<tr>
<td>Screen - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Screen - Name</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Screen - PlateRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - ProtocolDescription</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - ProtocolIdentifier</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - ReagentSetDescription</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - ReagentSetIdentifier</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Screen - Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>StageLabel - Name</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>StageLabel - X</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
</tbody>
</table>

continues on next page
<table>
<thead>
<tr>
<th>Field</th>
<th>Supported</th>
<th>Unsup-</th>
<th>Partial</th>
<th>Un-unknown/Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>StageLabel - Y</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>182</td>
</tr>
<tr>
<td>StageLabel - Z</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>181</td>
</tr>
<tr>
<td>TagAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TagAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TagAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TagAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TagAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TermAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TermAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TermAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TermAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TermAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TiffData - FirstC</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TiffData - FirstT</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TiffData - FirstZ</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TiffData - IFD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TiffData - PlaneCount</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TimestampAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TimestampAnnotation - Description</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>TimestampAnnotation - ID</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TimestampAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TimestampAnnotation - Value</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TransmittanceRange - CutIn</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>TransmittanceRange - CutInTolerance</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TransmittanceRange - CutOut</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>180</td>
</tr>
<tr>
<td>TransmittanceRange - CutOutTolerance</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>TransmittanceRange - Transmittance</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>UUID - FileName</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>UUID - Value</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Well - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Well - Color</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Well - Column</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>Well - ExternalDescription</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Well - ExternalIdentifier</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
<tr>
<td>Well - ID</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>Well - ReagentRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>Well - Row</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>Well - Type</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>WellSample - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>WellSample - ID</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>WellSample - ImageRef</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>WellSample - Index</td>
<td>16</td>
<td>0</td>
<td>0</td>
<td>169</td>
</tr>
<tr>
<td>WellSample - PositionX</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>WellSample - PositionY</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>WellSample - Timepoint</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>XMLAnnotation - AnnotationRef</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>185</td>
</tr>
<tr>
<td>XMLAnnotation - ID</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>183</td>
</tr>
<tr>
<td>XMLAnnotation - Namespace</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>184</td>
</tr>
</tbody>
</table>

continues on next page
This page lists supported metadata fields for the Bio-Formats Aperio AFI format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 31 of them (6%).
- Of those, Bio-Formats fully or partially converts 31 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Aperio AFI format reader:
- Channel: Color
- Channel: EmissionWavelength
- Channel: ExcitationWavelength
- Channel: ID
- Channel: Name
- Channel: SamplesPerPixel
- Image: AcquisitionDate
- Image: ID
- Image: InstrumentRef
- Image: Name
- Instrument: ID
- Objective: ID
- Objective: NominalMagnification
- ObjectiveSettings: ID
- Pixels: BigEndian
- Pixels: DimensionOrder
- Pixels: ID
- Pixels: Interleaved
- Pixels: PhysicalSizeX
- Pixels: PhysicalSizeY
- Pixels: SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 31
Total unknown or missing: 445

AIMReader

This page lists supported metadata fields for the Bio-Formats AIM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 22 of them (4%).
• Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats AIM format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 22

Total unknown or missing: 454

APLReader

This page lists supported metadata fields for the Bio-Formats Olympus APL format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 21 of them (4%).
• Of those, Bio-Formats fully or partially converts 21 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Olympus APL format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 21
Total unknown or missing: 455

APNGReader

This page lists supported metadata fields for the Bio-Formats Animated PNG format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 19 of them (3%).
  • Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Animated PNG format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

ARFReader

This page lists supported metadata fields for the Bio-Formats ARF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats ARF format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

AVIReader

This page lists supported metadata fields for the Bio-Formats Audio Video Interleave format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g., physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Audio Video Interleave format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ
Total supported: 19
Total unknown or missing: 457

AliconaReader

This page lists supported metadata fields for the Bio-Formats Alicona AL3D format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g., physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 33 of them (6%).
- Of those, Bio-Formats fully or partially converts 33 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Alicona AL3D format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- DetectorSettings : ID
- DetectorSettings : Voltage
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : CalibratedMagnification
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : WorkingDistance
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX

4.3. Summary of supported metadata fields
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 33

**Total unknown or missing:** 443

**AmiraReader**

This page lists supported metadata fields for the Bio-Formats Amira format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Amira format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
Bio-Formats, Release 6.14.0

- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

**AnalyzeReader**

This page lists supported metadata fields for the Bio-Formats Analyze 7.5 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 24 of them (5%).
- Of those, Bio-Formats fully or partially converts 24 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Analyze 7.5 format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
These fields are fully supported by the Bio-Formats BD Pathway format reader:

- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
4.3. Summary of supported metadata fields

• DetectorSettings : Offset
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• Instrument : ID
• Objective : ID
• Objective : LensNA
• Objective : Manufacturer
• Objective : NominalMagnification
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : ColumnNamingConvention
• Plate : Columns
• Plate : Description
• Plate : ID
• Plate : Name
• Plate : RowNamingConvention
• Plate : Rows
• PlateAcquisition : ID
• PlateAcquisition : MaximumFieldCount
• PlateAcquisition : WellSampleRef
• ROI : ID
• Rectangle : Height
• Rectangle : ID
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index

Total supported: 59
Total unknown or missing: 417

BDVReader

This page lists supported metadata fields for the Bio-Formats BDV format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 37 of them (7%).
• Of those, Bio-Formats fully or partially converts 37 (100%).

Supported fields

These fields are fully supported by the Bio-Formats BDV format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Image : ROIRef
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• ROI : ID
• ROI : Name
• Rectangle : Height
• Rectangle : ID
• Rectangle : StrokeColor
• Rectangle : Text
• Rectangle : TheC
• Rectangle : TheT
• Rectangle : TheZ
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• XMLAnnotation : ID
• XMLAnnotation : Value

Total supported: 37
Total unknown or missing: 439
**BIFormatReader**

This page lists supported metadata fields for the Bio-Formats BIFormatReader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the *metadata summary table*:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats BIFormatReader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457
BMPReader

This page lists supported metadata fields for the Bio-Formats Windows Bitmap format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Windows Bitmap format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455
BaseTiffReader

This page lists supported metadata fields for the Bio-Formats BaseTiffReader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 28 of them (5%).
• Of those, Bio-Formats fully or partially converts 28 (100%).

Supported fields

These fields are fully supported by the Bio-Formats BaseTiffReader:

• Channel : ID
• Channel : SamplesPerPixel
• Experimenter : Email
• Experimenter : FirstName
• Experimenter : ID
• Experimenter : LastName
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
- Plane: TheC
- Plane: TheT
- Plane: TheZ

**Total supported:** 28

**Total unknown or missing:** 448

**BaseZeissReader**

This page lists supported metadata fields for the Bio-Formats BaseZeissReader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**
- The file format itself supports 84 of them (17%).
- Of those, Bio-Formats fully or partially converts 84 (100%).

**Supported fields**

**These fields are fully supported by the Bio-Formats BaseZeissReader:**
- Channel: Color
- Channel: EmissionWavelength
- Channel: ExcitationWavelength
- Channel: ID
- Channel: Name
- Channel: SamplesPerPixel
- Detector: ID
- Detector: Type
- DetectorSettings: Gain
- DetectorSettings: ID
- DetectorSettings: Offset
- Ellipse: ID
- Ellipse: RadiusX
- Ellipse: RadiusY
- Ellipse: Text
- Ellipse: X
- Ellipse: Y
- Experimenter: FirstName
- Experimenter: ID
- Experimenter : Institution
- Experimenter : LastName
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Image : ROIRef
- Instrument : ID
- Label : ID
- Label : Text
- Label : X
- Label : Y
- Line : ID
- Line : Text
- Line : X1
- Line : X2
- Line : Y1
- Line : Y2
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : NominalMagnification
- Objective : WorkingDistance
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Point : ID
• Point : Text
• Point : X
• Point : Y
• Polygon : ID
• Polygon : Points
• Polygon : Text
• Polyline : ID
• Polyline : Points
• Polyline : Text
• ROI : ID
• ROI : Name
• Rectangle : Height
• Rectangle : ID
• Rectangle : Text
• Rectangle : Width
• Rectangle : X
• Rectangle : Y

Total supported: 84
Total unknown or missing: 392
Bio-Formats, Release 6.14.0

BioRadGelReader

This page lists supported metadata fields for the Bio-Formats Bio-Rad GEL format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Bio-Rad GEL format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455
Bio-Formats, Release 6.14.0

BioRadReader

This page lists supported metadata fields for the Bio-Formats Bio-Rad PIC format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 40 of them (8%).
- Of those, Bio-Formats fully or partially converts 40 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Bio-Rad PIC format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : Gain
- Detector : ID
- Detector : Offset
- Detector : Type
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : Offset
- Experiment : ID
- Experiment : Type
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Model
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 40
Total unknown or missing: 436

BioRadSCNReader

This page lists supported metadata fields for the Bio-Formats Bio-Rad SCN format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 29 of them (6%).
• Of those, Bio-Formats fully or partially converts 29 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Bio-Rad SCN format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Detector : ID
• DetectorSettings : Binning
• DetectorSettings : Gain
• DetectorSettings : ID
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Instrument : ID
• Microscope : Model
• Microscope : SerialNumber
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 29
Total unknown or missing: 447

BrukerReader

This page lists supported metadata fields for the Bio-Formats Bruker format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 23 of them (4%).
• Of those, Bio-Formats fully or partially converts 23 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Bruker format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Experimenter : ID
- Experimenter : Institution
- Experimenter : LastName
- Image : AcquisitionDate
- Image : ExperimenterRef
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 23
Total unknown or missing: 453

BurleighReader

This page lists supported metadata fields for the Bio-Formats Burleigh format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Burleigh format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

CV7000Reader

This page lists supported metadata fields for the Bio-Formats Yokogawa CV7000 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 56 of them (11%).
- Of those, Bio-Formats fully or partially converts 56 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Yokogawa CV7000 format reader:

- Channel : Color
- Channel : ExcitationWavelength
- Channel : ID
- Channel : LightSourceSettingsID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : Power
- Laser : Wavelength
- Objective : ID
- Objective : Model
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : Columns
- Plate : Description
- Plate : ExternalIdentifier
- Plate : ID
- Plate : Name
- Plate : Rows
- PlateAcquisition : EndTime
- PlateAcquisition : ID
- PlateAcquisition : MaximumFieldCount
- PlateAcquisition : StartTime
- PlateAcquisition : WellSampleRef
- Well : Column
- Well : ID
- Well : Row
- WellSample : ID
- WellSample : ImageRef
- WellSample : Index
- WellSample : PositionX
- WellSample : PositionY

Total supported: 56
Total unknown or missing: 420

**CanonRawReader**

This page lists supported metadata fields for the Bio-Formats Canon RAW format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**
- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Canon RAW format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

CellH5Reader

This page lists supported metadata fields for the Bio-Formats CellH5 (HDF) format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 41 of them (8%).
- Of those, Bio-Formats fully or partially converts 41 (100%).
Supported fields

These fields are fully supported by the Bio-Formats CellH5 (HDF) format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Image : ROIRef
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : ID
- Plate : Name
- ROI : ID
- ROI : Name
- Rectangle : Height
- Rectangle : ID
- Rectangle : StrokeColor
- Rectangle : Text
- Rectangle : TheC
- Rectangle : TheT
- Rectangle : TheZ
- Rectangle : Width
- Rectangle : X
- Rectangle : Y
Bio-Formats, Release 6.14.0

- Well : Column
- Well : ExternalIdentifier
- Well : ID
- Well : Row
- WellSample : ID
- WellSample : ImageRef
- WellSample : Index

Total supported: 41
Total unknown or missing: 435

CellSensReader

This page lists supported metadata fields for the Bio-Formats CellSens VSI format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 48 of them (10%).
- Of those, Bio-Formats fully or partially converts 48 (100%).

Supported fields

These fields are fully supported by the Bio-Formats CellSens VSI format reader:

- Channel : EmissionWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : Gain
- Detector : ID
- Detector : Manufacturer
- Detector : Model
- Detector : Offset
- Detector : SerialNumber
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : Offset
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Objective : ID
• Objective : LensNA
• Objective : Model
• Objective : NominalMagnification
• Objective : WorkingDistance
• ObjectiveSettings : ID
• ObjectiveSettings : RefractiveIndex
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 48
Total unknown or missing: 428
**CellVoyagerReader**

This page lists supported metadata fields for the Bio-Formats CellVoyager format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 39 of them (8%).
- Of those, Bio-Formats fully or partially converts 39 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats CellVoyager format reader:

- Channel : Color
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : Columns
- Plate : Name
• Plate : Rows
• PlateAcquisition : EndTime
• PlateAcquisition : ID
• PlateAcquisition : MaximumFieldCount
• PlateAcquisition : StartTime
• PlateAcquisition : WellSampleRef
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index
• WellSample : PositionX
• WellSample : PositionY

Total supported: 39
Total unknown or missing: 437

CellWorxReader

This page lists supported metadata fields for the Bio-Formats CellWorx format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 50 of them (10%).
• Of those, Bio-Formats fully or partially converts 50 (100%).

Supported fields

These fields are fully supported by the Bio-Formats CellWorx format reader:
• Channel : EmissionWavelength
• Channel : ExcitationWavelength
• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Detector : ID
• DetectorSettings : Gain
• DetectorSettings : ID
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Microscope : SerialNumber
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : Columns
• Plate : ID
• Plate : Name
• Plate : Rows
• PlateAcquisition : EndTime
• PlateAcquisition : ID
• PlateAcquisition : MaximumFieldCount
• PlateAcquisition : StartTime
• PlateAcquisition : WellSampleRef
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index
• WellSample : PositionX
• WellSample : PositionY

Total supported: 50
Total unknown or missing: 426

CellomicsReader

This page lists supported metadata fields for the Bio-Formats Cellomics C01 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 36 of them (7%).
• Of those, Bio-Formats fully or partially converts 36 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Cellomics C01 format reader:

• Channel : Color
• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT

4.3. Summary of supported metadata fields 267
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : ColumnNamingConvention
• Plate : Columns
• Plate : ID
• Plate : Name
• Plate : RowNamingConvention
• Plate : Rows
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index

Total supported: 36
Total unknown or missing: 440

ColumbusReader

This page lists supported metadata fields for the Bio-Formats PerkinElmer Columbus format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 40 of them (8%).
• Of those, Bio-Formats fully or partially converts 40 (100%).
Supported fields

These fields are fully supported by the Bio-Formats PerkinElmer Columbus format reader:

- Channel : Color
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : Columns
- Plate : ID
- Plate : Name
- Plate : Rows
- Screen : ID
- Screen : Name
- Well : Column
- Well : ID
Bio-Formats, Release 6.14.0

- Well : Row
- WellSample : ID
- WellSample : ImageRef
- WellSample : Index
- WellSample : PositionX
- WellSample : PositionY

Total supported: 40
Total unknown or missing: 436

DNGReader

This page lists supported metadata fields for the Bio-Formats DNG format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats DNG format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

DeltavisionReader

This page lists supported metadata fields for the Bio-Formats Deltavision format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 51 of them (10%).
• Of those, Bio-Formats fully or partially converts 51 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Deltavision format reader:

• Channel : EmissionWavelength
• Channel : ExcitationWavelength
• Channel : ID
• Channel : NDFilter
• Channel : Name
• Channel : SamplesPerPixel
• Detector : ID
• Detector : Model
• Detector : Type
• DetectorSettings : Binning
• DetectorSettings : Gain
• DetectorSettings : ID
• DetectorSettings : ReadOutRate
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : InstrumentRef
• Image : Name
• ImagingEnvironment : Temperature
• Instrument : ID
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : Manufacturer
• Objective : Model
• Objective : NominalMagnification
• Objective : WorkingDistance
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 51

Total unknown or missing: 425
DicomReader

This page lists supported metadata fields for the Bio-Formats DICOM format reader.
These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the
OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns)
in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 27 of them (5%).
• Of those, Bio-Formats fully or partially converts 27 (100%).

Supported fields

These fields are fully supported by the Bio-Formats DICOM format reader:

• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 27
Total unknown or missing: 449

**EPSReader**

This page lists supported metadata fields for the Bio-Formats Encapsulated PostScript format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Encapsulated PostScript format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ
Total supported: 19
Total unknown or missing: 457

Ecat7Reader

This page lists supported metadata fields for the Bio-Formats ECAT7 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 23 of them (4%).
- Of those, Bio-Formats fully or partially converts 23 (100%).

Supported fields

These fields are fully supported by the Bio-Formats ECAT7 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
FEIReader

This page lists supported metadata fields for the Bio-Formats FEI/Philips format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats FEI/Philips format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
FEITiffReader

This page lists supported metadata fields for the Bio-Formats FEI TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 39 of them (8%).
- Of those, Bio-Formats fully or partially converts 39 (100%).

Supported fields

These fields are fully supported by the Bio-Formats FEI TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Type
- Experimenter : ID
- Experimenter : LastName
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Microscope : Model
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : NominalMagnification
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits

4.3. Summary of supported metadata fields 277
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• StageLabel : Name
• StageLabel : X
• StageLabel : Y
• StageLabel : Z

Total supported: 39
Total unknown or missing: 437

FV1000Reader

This page lists supported metadata fields for the Bio-Formats Olympus FV1000 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 113 of them (23%).
• Of those, Bio-Formats fully or partially converts 113 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Olympus FV1000 format reader:

• Channel : EmissionWavelength
• Channel : ExcitationWavelength
• Channel : ID
• Channel : IlluminationType
• Channel : LightSourceSettingsID
• Channel : LightSourceSettingsWavelength
• Channel : Name
• Channel : SamplesPerPixel
• Detector : Gain
• Detector : ID
• Detector : Type
• Detector : Voltage
• DetectorSettings : ID
• Dichroic : ID
• Dichroic : Model
• Ellipse : FontSize
• Ellipse : ID
• Ellipse : RadiusX
• Ellipse : RadiusY
• Ellipse : StrokeWidth
• Ellipse : TheT
• Ellipse : TheZ
• Ellipse : Transform
• Ellipse : X
• Ellipse : Y
• Filter : ID
• Filter : Model
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• Instrument : ID
• Laser : ID
• Laser : LaserMedium
• Laser : Type
• Laser : Wavelength
• LightPath : DichroicRef
• LightPath : EmissionFilterRef
• Line : FontSize
• Line : ID
• Line : StrokeWidth
• Line : TheT
• Line : TheZ

4.3. Summary of supported metadata fields
• Line : Transform
• Line : X1
• Line : X2
• Line : Y1
• Line : Y2
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : Model
• Objective : NominalMagnification
• Objective : WorkingDistance
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : DeltaT
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Point : FontSize
• Point : ID
• Point : StrokeWidth
• Point : TheT
• Point : TheZ
• Point : X
• Point : Y
• Polygon : FontSize
• Polygon : ID
• Polygon : Points
• Polygon : StrokeWidth
• Polygon : TheT
• Polygon : TheZ
• Polygon : Transform
• Polyline : FontSize
• Polyline : ID
• Polyline : Points
• Polyline : StrokeWidth
• Polyline : TheT
• Polyline : TheZ
• Polyline : Transform
• ROI : ID
• Rectangle : FontSize
• Rectangle : Height
• Rectangle : ID
• Rectangle : StrokeWidth
• Rectangle : TheT
• Rectangle : TheZ
• Rectangle : Transform
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• TransmittanceRange : CutIn
• TransmittanceRange : CutOut

Total supported: 113
Total unknown or missing: 363
FakeReader

This page lists supported metadata fields for the Bio-Formats Simulated data format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 91 of them (19%).
- Of those, Bio-Formats fully or partially converts 91 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Simulated data format reader:

- BooleanAnnotation : ID
- BooleanAnnotation : Namespace
- BooleanAnnotation : Value
- Channel : Color
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- CommentAnnotation : ID
- CommentAnnotation : Namespace
- CommentAnnotation : Value
- DoubleAnnotation : ID
- DoubleAnnotation : Namespace
- DoubleAnnotation : Value
- Ellipse : ID
- Ellipse : RadiusX
- Ellipse : RadiusY
- Ellipse : X
- Ellipse : Y
- Image : AcquisitionDate
- Image : AnnotationRef
- Image : ID
- Image : Name
- Image : ROIRef
- Label : ID
- Label : Text
4.3. Summary of supported metadata fields
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Point : ID
• Point : X
• Point : Y
• Polygon : ID
• Polygon : Points
• Polyline : ID
• Polyline : Points
• ROI : ID
• Rectangle : Height
• Rectangle : ID
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• TagAnnotation : ID
• TagAnnotation : Namespace
• TagAnnotation : Value
• TermAnnotation : ID
• TermAnnotation : Namespace
• TermAnnotation : Value
• TimestampAnnotation : ID
• TimestampAnnotation : Namespace
• TimestampAnnotation : Value
• WellSample : PositionX
• WellSample : PositionY
• XMLAnnotation : ID
• XMLAnnotation : Namespace
• XMLAnnotation : Value

Total supported: 91

Total unknown or missing: 385
FilePatternReader

This page lists supported metadata fields for the Bio-Formats File pattern format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 20 of them (4%).
- Of those, Bio-Formats fully or partially converts 20 (100%).

Supported fields

These fields are fully supported by the Bio-Formats File pattern format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 20
Total unknown or missing: 456
Bio-Formats, Release 6.14.0

**FitsReader**

This page lists supported metadata fields for the Bio-Formats Flexible Image Transport System format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Flexible Image Transport System format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457
FlexReader

This page lists supported metadata fields for the Bio-Formats Evotec Flex format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 71 of them (14%).
- Of those, Bio-Formats fully or partially converts 71 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Evotec Flex format reader:

- Channel : ID
- Channel : LightSourceSettingsID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : ID
- Dichroic : ID
- Dichroic : Model
- Filter : FilterWheel
- Filter : ID
- Filter : Model
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : LaserMedium
- Laser : Type
- Laser : Wavelength
- LightPath : DichroicRef
- LightPath : EmissionFilterRef
- LightPath : ExcitationFilterRef
• Objective : CalibratedMagnification
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : ColumnNamingConvention
• Plate : Columns
• Plate : ExternalIdentifier
• Plate : ID
• Plate : Name
• Plate : RowNamingConvention
• Plate : Rows
• PlateAcquisition : ID
• PlateAcquisition : MaximumFieldCount
• PlateAcquisition : StartTime
• PlateAcquisition : WellSampleRef
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index
• WellSample : PositionX
• WellSample : PositionY

Total supported: 71
Total unknown or missing: 405

FlowSightReader

This page lists supported metadata fields for the Bio-Formats FlowSight format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 20 of them (4%).
• Of those, Bio-Formats fully or partially converts 20 (100%).

Supported fields

These fields are fully supported by the Bio-Formats FlowSight format reader:

• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC

4.3. Summary of supported metadata fields
Bio-Formats, Release 6.14.0

- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 20
Total unknown or missing: 456

FluoviewReader

This page lists supported metadata fields for the Bio-Formats Olympus Fluoview/ABD TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 49 of them (10%).
- Of those, Bio-Formats fully or partially converts 49 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Olympus Fluoview/ABD TIFF format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Manufacturer
- Detector : Model
- Detector : Type
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : Offset
- DetectorSettings : ReadOutRate
- DetectorSettings : Voltage
- Image : AcquisitionDate
- Image : Description
4.3. Summary of supported metadata fields
Total supported: 49
Total unknown or missing: 427

FujiReader

This page lists supported metadata fields for the Bio-Formats Fuji LAS 3000 format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g., physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 23 of them (4%).
- Of those, Bio-Formats fully or partially converts 23 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Fuji LAS 3000 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Instrument : ID
- Microscope : Model
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
• Plane : TheZ

Total supported: 23
Total unknown or missing: 453

**GIFReader**

This page lists supported metadata fields for the Bio-Formats Graphics Interchange Format format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Graphics Interchange Format format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

4.3. Summary of supported metadata fields
GatanDM2Reader

This page lists supported metadata fields for the Bio-Formats Gatan DM2 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 30 of them (6%).
- Of those, Bio-Formats fully or partially converts 30 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Gatan DM2 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- DetectorSettings : Binning
- DetectorSettings : ID
- Experimenter : FirstName
- Experimenter : ID
- Experimenter : LastName
- Image : AcquisitionDate
- Image : ExperimenterRef
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 30
Total unknown or missing: 446

GatanReader

This page lists supported metadata fields for the Bio-Formats Gatan Digital Micrograph format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 69 of them (14%).
• Of those, Bio-Formats fully or partially converts 69 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Gatan Digital Micrograph format reader:

• Channel : AcquisitionMode
• Channel : ID
• Channel : SamplesPerPixel
• Detector : ID
• DetectorSettings : ID
• DetectorSettings : Voltage
• Ellipse : FontSize
• Ellipse : ID
• Ellipse : RadiusX
• Ellipse : RadiusY
• Ellipse : StrokeColor
• Ellipse : Text
• Ellipse : X
• Ellipse : Y
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• Instrument : ID
• Label : FontSize
• Label : ID
• Label : StrokeColor
• Label : Text
• Label : X
• Label : Y
• Line : FontSize
• Line : ID
• Line : StrokeColor
• Line : Text
• Line : X1
• Line : X2
• Line : Y1
• Line : Y2
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : NominalMagnification
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
Bio-Formats, Release 6.14.0

- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- ROI : ID
- Rectangle : FontSize
- Rectangle : Height
- Rectangle : ID
- Rectangle : StrokeColor
- Rectangle : Text
- Rectangle : Width
- Rectangle : X
- Rectangle : Y

Total supported: 69
Total unknown or missing: 407

**GelReader**

This page lists supported metadata fields for the Bio-Formats Amersham Biosciences GEL format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Amersham Biosciences GEL format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
Bio-Formats, Release 6.14.0

- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455

**HISReader**

This page lists supported metadata fields for the Bio-Formats Hamamatsu HIS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the *metadata summary table*:

- The file format itself supports 27 of them (5%).
- Of those, Bio-Formats fully or partially converts 27 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Hamamatsu HIS format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Offset
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : ID
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 27
Total unknown or missing: 449

HRDGDFReader

This page lists supported metadata fields for the Bio-Formats NOAA-HRD Gridded Data Format format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 21 of them (4%).
• Of those, Bio-Formats fully or partially converts 21 (100%).
Supported fields

These fields are fully supported by the Bio-Formats NOAA-HRD Gridded Data Format format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455

HamamatsuVMSReader

This page lists supported metadata fields for the Bio-Formats Hamamatsu VMS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 26 of them (5%).
- Of those, Bio-Formats fully or partially converts 26 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Hamamatsu VMS format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : ID
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 26
Total unknown or missing: 450
**HitachiReader**

This page lists supported metadata fields for the Bio-Formats Hitachi format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 31 of them (6%).
- Of those, Bio-Formats fully or partially converts 31 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Hitachi format reader:

- `Channel : ID`
- `Channel : SamplesPerPixel`
- `Image : AcquisitionDate`
- `Image : ID`
- `Image : InstrumentRef`
- `Image : Name`
- `Instrument : ID`
- `Microscope : Model`
- `Microscope : SerialNumber`
- `Objective : ID`
- `Objective : WorkingDistance`
- `ObjectiveSettings : ID`
- `Pixels : BigEndian`
- `Pixels : DimensionOrder`
- `Pixels : ID`
- `Pixels : Interleaved`
- `Pixels : PhysicalSizeX`
- `Pixels : PhysicalSizeY`
- `Pixels : SignificantBits`
- `Pixels : SizeC`
- `Pixels : SizeT`
- `Pixels : SizeX`
- `Pixels : SizeY`
- `Pixels : SizeZ`
- `Pixels : Type`
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 31
Total unknown or missing: 445

I2IReader

This page lists supported metadata fields for the Bio-Formats I2I format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 19 of them (3%).
  • Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats I2I format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
ICSReader

This page lists supported metadata fields for the Bio-Formats Image Cytometry Standard format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 72 of them (15%).
- Of those, Bio-Formats fully or partially converts 72 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Image Cytometry Standard format reader:

- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Manufacturer
- Detector : Model
- Detector : Type
- DetectorSettings : Gain
- DetectorSettings : ID
- Dichroic : ID
- Dichroic : Model
- Experiment : ID
- Experiment : Type
- Experimenter : ID
- Experimenter : LastName
- Filter : ID
4.3. Summary of supported metadata fields

- Filter : Model
- FilterSet : DichroicRef
- FilterSet : EmissionFilterRef
- FilterSet : ExcitationFilterRef
- FilterSet : ID
- FilterSet : Model
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : LaserMedium
- Laser : Manufacturer
- Laser : Model
- Laser : Power
- Laser : RepetitionRate
- Laser : Type
- Laser : Wavelength
- Microscope : Manufacturer
- Microscope : Model
- Objective : CalibratedMagnification
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Model
- Objective : WorkingDistance
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 72
Total unknown or missing: 404

**IM3Reader**

This page lists supported metadata fields for the Bio-Formats Perkin-Elmer Nuance IM3 format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Perkin-Elmer Nuance IM3 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19

Total unknown or missing: 457

IMODReader

This page lists supported metadata fields for the Bio-Formats IMOD format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 44 of them (9%).
  • Of those, Bio-Formats fully or partially converts 44 (100%).

Supported fields

These fields are fully supported by the Bio-Formats IMOD format reader:
  • Channel : ID
  • Channel : SamplesPerPixel
  • Image : AcquisitionDate
  • Image : ID
  • Image : Name
  • Image : ROIRef
  • Pixels : BigEndian
  • Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Point : ID
- Point : StrokeColor
- Point : StrokeDashArray
- Point : StrokeWidth
- Point : TheZ
- Point : X
- Point : Y
- Polygon : ID
- Polygon : Points
- Polygon : StrokeColor
- Polygon : StrokeDashArray
- Polygon : StrokeWidth
- Polygon : TheZ
- Polyline : ID
- Polyline : Points
- Polyline : StrokeColor
- Polyline : StrokeDashArray
- Polyline : StrokeWidth
- Polyline : TheZ
- ROI : ID
- ROI : Name
Total supported: 44
Total unknown or missing: 432

INRReader

This page lists supported metadata fields for the Bio-Formats INR format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 22 of them (4%).
  • Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats INR format reader:
  • Channel : ID
  • Channel : SamplesPerPixel
  • Image : AcquisitionDate
  • Image : ID
  • Image : Name
  • Pixels : BigEndian
  • Pixels : DimensionOrder
  • Pixels : ID
  • Pixels : Interleaved
  • Pixels : PhysicalSizeX
  • Pixels : PhysicalSizeY
  • Pixels : PhysicalSizeZ
  • Pixels : SignificantBits
  • Pixels : SizeC
  • Pixels : SizeT
  • Pixels : SizeX
  • Pixels : SizeY
  • Pixels : SizeZ
  • Pixels : Type
  • Plane : TheC
  • Plane : TheT
  • Plane : TheZ
Total supported: 22
Total unknown or missing: 454

IPLabReader

This page lists supported metadata fields for the Bio-Formats IPLab format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g., physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 31 of them (6%).
- Of those, Bio-Formats fully or partially converts 31 (100%).

Supported fields

These fields are fully supported by the Bio-Formats IPLab format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Image : ROIRef
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : DeltaT
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• ROI : ID
• Rectangle : Height
• Rectangle : ID
• Rectangle : Width
• Rectangle : X
• Rectangle : Y

Total supported: 31
Total unknown or missing: 445

IPWReader

This page lists supported metadata fields for the Bio-Formats Image-Pro Workspace format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 20 of them (4%).
• Of those, Bio-Formats fully or partially converts 20 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Image-Pro Workspace format reader:
• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 20
Total unknown or missing: 456

ImaconReader

This page lists supported metadata fields for the Bio-Formats Imacon format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 23 of them (4%).
• Of those, Bio-Formats fully or partially converts 23 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Imacon format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Experimenter : FirstName
• Experimenter : ID
• Experimenter : LastName
• Image : AcquisitionDate
• Image : ExperimenterRef
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 23
Total unknown or missing: 453

ImageIOReader

This page lists supported metadata fields for the Bio-Formats ImageIOReader.
These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 19 of them (3%).
  • Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats ImageIOReader:
• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
Bio-Formats, Release 6.14.0

- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

**ImagicReader**

This page lists supported metadata fields for the Bio-Formats IMAGIC format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats IMAGIC format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
Supported fields

These fields are fully supported by the Bio-Formats Bitplane Imaris 5.5 (HDF) format reader:

- Channel : Color
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX

Total supported: 22

Total unknown or missing: 454
Bio-Formats, Release 6.14.0

- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported: 23**

**Total unknown or missing: 453**

**ImarisReader**

This page lists supported metadata fields for the Bio-Formats Bitplane Imaris format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**
- The file format itself supports 32 of them (6%).
- Of those, Bio-Formats fully or partially converts 32 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Bitplane Imaris format reader:
- Channel : ID
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : Offset
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 32
Total unknown or missing: 444

**ImarisTiffReader**

This page lists supported metadata fields for the Bio-Formats Bitplane Imaris 3 (TIFF) format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 23 of them (4%).
- Of those, Bio-Formats fully or partially converts 23 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Bitplane Imaris 3 (TIFF) format reader:
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 23
Total unknown or missing: 453

ImprovionTiffReader

This page lists supported metadata fields for the Bio-Formats Improvision TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 26 of them (5%).
• Of those, Bio-Formats fully or partially converts 26 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Improvision TIFF format reader:

• Channel : Color
• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

**Total supported:** 26

**Total unknown or missing:** 450

**ImspectorReader**

This page lists supported metadata fields for the Bio-Formats Lavision Imspector format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Lavision Impector format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

InCell3000Reader

This page lists supported metadata fields for the Bio-Formats InCell 3000 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats InCell 3000 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

InCellReader

This page lists supported metadata fields for the Bio-Formats InCell 1000/2000 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 69 of them (14%).
- Of those, Bio-Formats fully or partially converts 69 (100%).
Supported fields

These fields are fully supported by the Bio-Formats InCell 1000/2000 format reader:

- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- Experiment : ID
- Experiment : Type
- Image : AcquisitionDate
- Image : Description
- Image : ExperimentRef
- Image : ID
- Image : InstrumentRef
- Image : Name
- ImagingEnvironment : Temperature
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Manufacturer
- Objective : NominalMagnification
- ObjectiveSettings : ID
- ObjectiveSettings : RefractiveIndex
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : ColumnNamingConvention
• Plate : Columns
• Plate : ID
• Plate : Name
• Plate : RowNamingConvention
• Plate : Rows
• Plate : WellOriginX
• Plate : WellOriginY
• PlateAcquisition : ID
• PlateAcquisition : MaximumFieldCount
• PlateAcquisition : WellSampleRef
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index
• WellSample : PositionX
• WellSample : PositionY
Total supported: 69
Total unknown or missing: 407

InveonReader

This page lists supported metadata fields for the Bio-Formats Inveon format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 30 of them (6%).
  • Of those, Bio-Formats fully or partially converts 30 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Inveon format reader:
  • Channel : ID
  • Channel : SamplesPerPixel
  • Experimenter : ID
  • Experimenter : Institution
  • Experimenter : UserName
  • Image : AcquisitionDate
  • Image : Description
  • Image : ExperimenterRef
  • Image : ID
  • Image : InstrumentRef
  • Image : Name
  • Instrument : ID
  • Microscope : Model
  • Pixels : BigEndian
  • Pixels : DimensionOrder
  • Pixels : ID
  • Pixels : Interleaved
  • Pixels : PhysicalSizeX
  • Pixels : PhysicalSizeY
  • Pixels : PhysicalSizeZ
  • Pixels : SignificantBits
  • Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 30
Total unknown or missing: 446

IonpathMIBITiffReader

This page lists supported metadata fields for the Bio-Formats Ionpath MIBI format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 22 of them (4%).
  • Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Ionpath MIBI format reader:

• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : Name
• Instrument : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC

4.3. Summary of supported metadata fields
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 22

Total unknown or missing: 454

IvisionReader

This page lists supported metadata fields for the Bio-Formats IVision format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 34 of them (7%).
• Of those, Bio-Formats fully or partially converts 34 (100%).

Supported fields

These fields are fully supported by the Bio-Formats IVision format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Detector : ID
• Detector : Type
• DetectorSettings : Binning
• DetectorSettings : Gain
• DetectorSettings : ID
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : NominalMagnification
• ObjectiveSettings : ID
• ObjectiveSettings : RefractiveIndex
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 34
Total unknown or missing: 442

JEOLReader

This page lists supported metadata fields for the Bio-Formats JEOL format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

4.3. Summary of supported metadata fields
Supported fields

These fields are fully supported by the Bio-Formats JEOL format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

JPEG2000Reader

This page lists supported metadata fields for the Bio-Formats JPEG-2000 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats JPEG-2000 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

JPEGReader

This page lists supported metadata fields for the Bio-Formats JPEG format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats JPEG format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

JPKReader

This page lists supported metadata fields for the Bio-Formats JPK Instruments format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats JPK Instruments format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

JPXReader

This page lists supported metadata fields for the Bio-Formats JPX format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats JPX format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

KLBReader

This page lists supported metadata fields for the Bio-Formats KLB format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).
Supported fields

These fields are fully supported by the Bio-Formats KLB format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

KhorosReader

This page lists supported metadata fields for the Bio-Formats Khoros XV format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Khoros XV format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

KodakReader

This page lists supported metadata fields for the Bio-Formats Kodak Molecular Imaging format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 26 of them (5%).
- Of those, Bio-Formats fully or partially converts 26 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Kodak Molecular Imaging format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- ImagingEnvironment : Temperature
- Instrument : ID
- Microscope : Model
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 26
Total unknown or missing: 450
L2DReader

This page lists supported metadata fields for the Bio-Formats Li-Cor L2D format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 29 of them (6%).
- Of those, Bio-Formats fully or partially converts 29 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Li-Cor L2D format reader:

- Channel : ID
- Channel : LightSourceSettingsID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : LaserMedium
- Laser : Type
- Laser : Wavelength
- Microscope : Model
- Microscope : Type
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
Bio-Formats, Release 6.14.0

- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 29
Total unknown or missing: 447

LEOReader

This page lists supported metadata fields for the Bio-Formats LEO format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 27 of them (5%).
- Of those, Bio-Formats fully or partially converts 27 (100%).

Supported fields

These fields are fully supported by the Bio-Formats LEO format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : WorkingDistance
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 27
**Total unknown or missing:** 449

**LIFReader**

This page lists supported metadata fields for the Bio-Formats Leica Image File Format format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 85 of them (17%).
- Of those, Bio-Formats fully or partially converts 85 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Leica Image File Format format reader:

- Channel : Color
- Channel : ExcitationWavelength
- Channel : ID
- Channel : LightSourceSettingsAttenuation
- Channel : LightSourceSettingsID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Offset
- Detector : Type
- Detector : Zoom
• DetectorSettings : Gain
• DetectorSettings : ID
• DetectorSettings : Offset
• Filter : ID
• Filter : Model
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• Instrument : ID
• Label : FontSize
• Label : ID
• Label : StrokeWidth
• Label : Text
• Label : X
• Label : Y
• Laser : ID
• Laser : LaserMedium
• Laser : Type
• Laser : Wavelength
• LightPath : EmissionFilterRef
• Line : ID
• Line : X1
• Line : X2
• Line : Y1
• Line : Y2
• Microscope : Model
• Microscope : Type
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : Model
• Objective : NominalMagnification

4.3. Summary of supported metadata fields 339
• Objective : SerialNumber
• ObjectiveSettings : ID
• ObjectiveSettings : RefractiveIndex
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Polygon : ID
• Polygon : Points
• ROI : ID
• Rectangle : Height
• Rectangle : ID
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• TransmittanceRange : CutIn
• TransmittanceRange : CutOut
Total supported: 85
Total unknown or missing: 391

LIMReader

This page lists supported metadata fields for the Bio-Formats Laboratory Imaging format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Laboratory Imaging format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
LOFReader

This page lists supported metadata fields for the Bio-Formats Leica Object Format format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Leica Object Format format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
LegacyND2Reader

This page lists supported metadata fields for the Bio-Formats Nikon ND2 (Legacy) format reader. These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Nikon ND2 (Legacy) format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

4.3. Summary of supported metadata fields
LegacyQTReader

This page lists supported metadata fields for the Bio-Formats QuickTime format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats QuickTime format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
LeicaReader

This page lists supported metadata fields for the Bio-Formats Leica format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 56 of them (11%).
- Of those, Bio-Formats fully or partially converts 56 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Leica format reader:

- Channel : Color
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Offset
- Detector : Type
- Detector : Voltage
- DetectorSettings : ID
- Filter : ID
- Filter : Model
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- LightPath : EmissionFilterRef
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Model
- Objective : NominalMagnification
- Objective : SerialNumber
- ObjectiveSettings : ID
- ObjectiveSettings : RefractiveIndex
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- StageLabel : Name
- StageLabel : Z
- TransmittanceRange : CutIn
- TransmittanceRange : CutOut

**Total supported:** 56

**Total unknown or missing:** 420
LeicaSCNReader

This page lists supported metadata fields for the Bio-Formats Leica SCN format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 33 of them (6%).
- Of those, Bio-Formats fully or partially converts 33 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Leica SCN format reader:

- Channel : ID
- Channel : IlluminationType
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : CalibratedMagnification
- Objective : ID
- Objective : LensNA
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
Bio-Formats, Release 6.14.0

- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : PositionX
- Plane : PositionY
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 33
Total unknown or missing: 443

LiFlimReader

This page lists supported metadata fields for the Bio-Formats LI-FLIM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 25 of them (5%).
- Of those, Bio-Formats fully or partially converts 25 (100%).

Supported fields

These fields are fully supported by the Bio-Formats LI-FLIM format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Image : ROIRef
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Polygon : ID
• Polygon : Points
• ROI : ID

Total supported: 25
Total unknown or missing: 451

MIASReader

This page lists supported metadata fields for the Bio-Formats MIAS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 67 of them (14%).
  • Of those, Bio-Formats fully or partially converts 67 (100%).

Supported fields

These fields are fully supported by the Bio-Formats MIAS format reader:

• Channel : Color
• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Ellipse : ID
• Ellipse : RadiusX
• Ellipse : RadiusY
• Ellipse : Text
• Ellipse : TheT
• Ellipse : TheZ
• Ellipse : X
• Ellipse : Y
• Experiment : Description
• Experiment : ID
• Experiment : Type
• Image : AcquisitionDate
• Image : ExperimentRef
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• Instrument : ID
• Mask : BinData
• Mask : FillColor
• Mask : Height
• Mask : ID
• Mask : StrokeColor
• Mask : Width
• Mask : X
• Mask : Y
• Objective : ID
• Objective : Model
• Objective : NominalMagnification
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
This page lists supported metadata fields for the Bio-Formats MINC MRI format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 26 of them (5%).
- Of those, Bio-Formats fully or partially converts 26 (100%).
Supported fields

These fields are fully supported by the Bio-Formats MINC MRI format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 26
Total unknown or missing: 450
**MNGReader**

This page lists supported metadata fields for the Bio-Formats Multiple-image Network Graphics format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Multiple-image Network Graphics format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457
MRCReader

This page lists supported metadata fields for the Bio-Formats Medical Research Council format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Medical Research Council format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454
MRWReader

This page lists supported metadata fields for the Bio-Formats Minolta MRW format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Minolta MRW format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

4.3. Summary of supported metadata fields
MetamorphReader

This page lists supported metadata fields for the Bio-Formats Metamorph STK format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 59 of them (12%).
- Of those, Bio-Formats fully or partially converts 59 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Metamorph STK format reader:

- Channel : EmissionWavelength
- Channel : ID
- Channel : LightSourceSettingsID
- Channel : LightSourceSettingsWavelength
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : ReadOutRate
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- ImagingEnvironment : Temperature
- Instrument : ID
- Laser : ID
- Laser : LaserMedium
- Laser : Type
- Objective : ID
- Objective : LensNA
- ObjectiveSettings : ID
• **ObjectiveSettings**: RefractiveIndex
• **Pixels**: BigEndian
• **Pixels**: DimensionOrder
• **Pixels**: ID
• **Pixels**: Interleaved
• **Pixels**: PhysicalSizeX
• **Pixels**: PhysicalSizeY
• **Pixels**: PhysicalSizeZ
• **Pixels**: SignificantBits
• **Pixels**: SizeC
• **Pixels**: SizeT
• **Pixels**: SizeX
• **Pixels**: SizeY
• **Pixels**: SizeZ
• **Pixels**: Type
• **Plane**: DeltaT
• **Plane**: ExposureTime
• **Plane**: PositionX
• **Plane**: PositionY
• **Plane**: PositionZ
• **Plane**: TheC
• **Plane**: TheT
• **Plane**: TheZ
• **Plate**: ColumnNamingConvention
• **Plate**: Columns
• **Plate**: ID
• **Plate**: RowNamingConvention
• **Plate**: Rows
• **Well**: Column
• **Well**: ID
• **Well**: Row
• **WellSample**: ID
• **WellSample**: ImageRef
• **WellSample**: Index

**Total supported**: 59

**Total unknown or missing**: 417

4.3. **Summary of supported metadata fields**
**MetamorphTiffReader**

This page lists supported metadata fields for the Bio-Formats Metamorph TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

*Of the 476 fields documented in the metadata summary table:*

- The file format itself supports 43 of them (9%).
- Of those, Bio-Formats fully or partially converts 43 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Metamorph TIFF format reader:

- Channel : EmissionWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- ImagingEnvironment : Temperature
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : TheC
• Plane : TheT
• Plane : TheZ
• Plate : ColumnNamingConvention
• Plate : Columns
• Plate : ID
• Plate : RowNamingConvention
• Plate : Rows
• PlateAcquisition : ID
• PlateAcquisition : WellSampleRef
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index

Total supported: 43
Total unknown or missing: 433

MetaxpressTiffReader

This page lists supported metadata fields for the Bio-Formats MetaXpress TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats MetaXpress TIFF format reader:
• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
MicroCTReader

This page lists supported metadata fields for the Bio-Formats MicroCT format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 24 of them (5%).
- Of those, Bio-Formats fully or partially converts 24 (100%).

Supported fields

These fields are fully supported by the Bio-Formats MicroCT format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
Bio-Formats, Release 6.14.0

- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 24
Total unknown or missing: 452

**MicromanagerReader**

This page lists supported metadata fields for the Bio-Formats Micro-Manager format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 42 of them (8%).
- Of those, Bio-Formats fully or partially converts 42 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Micro-Manager format reader:

- Channel : Color
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID

4.3. Summary of supported metadata fields
• Detector : Manufacturer
• Detector : Model
• Detector : SerialNumber
• Detector : Type
• DetectorSettings : Binning
• DetectorSettings : Gain
• DetectorSettings : ID
• DetectorSettings : Voltage
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : InstrumentRef
• Image : Name
• ImagingEnvironment : Temperature
• Instrument : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

**Total supported: 42**

**Total unknown or missing: 434**

**MikroscanTiffReader**

This page lists supported metadata fields for the Bio-Formats Mikroscan TIFF format reader. These fields are from the [OME data model](#). Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Mikroscan TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported: 19**

**Total unknown or missing: 457**
MinimalTiffReader

This page lists supported metadata fields for the Bio-Formats Minimal TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Minimal TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
MolecularImagingReader

This page lists supported metadata fields for the Bio-Formats Molecular Imaging format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Molecular Imaging format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455
NAFReader

This page lists supported metadata fields for the Bio-Formats Hamamatsu Aquacosmos format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Hamamatsu Aquacosmos format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
**ND2Reader**

This page lists supported metadata fields for the Bio-Formats Nikon ND2 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Nikon ND2 format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457
**NDPIReader**

This page lists supported metadata fields for the Bio-Formats Hamamatsu NDPI format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the *metadata summary table*:

- The file format itself supports 32 of them (6%).
- Of those, Bio-Formats fully or partially converts 32 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Hamamatsu NDPI format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Microscope : Model
- Objective : ID
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
NDPISReader

This page lists supported metadata fields for the Bio-Formats Hamamatsu NDPIS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Hamamatsu NDPIS format reader:

- Channel : Color
- Channel : EmissionWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT

Total supported: 32
Total unknown or missing: 444
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 22
Total unknown or missing: 454

NRRDReader

This page lists supported metadata fields for the Bio-Formats NRRD format reader.
These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 22 of them (4%).
  • Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats NRRD format reader:
  • Channel : ID
  • Channel : SamplesPerPixel
  • Image : AcquisitionDate
  • Image : ID
  • Image : Name
  • Pixels : BigEndian
  • Pixels : DimensionOrder
  • Pixels : ID
  • Pixels : Interleaved
  • Pixels : PhysicalSizeX
  • Pixels : PhysicalSizeY
  • Pixels : PhysicalSizeZ
  • Pixels : SignificantBits
  • Pixels : SizeC
  • Pixels : SizeT
Bio-Formats, Release 6.14.0

- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

NativeND2Reader

This page lists supported metadata fields for the Bio-Formats Nikon ND2 format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  - The file format itself supports 52 of them (10%).
  - Of those, Bio-Formats fully or partially converts 52 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Nikon ND2 format reader:

- Channel : AcquisitionMode
- Channel : Color
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : ReadOutRate
Bio-Formats, Release 6.14.0

- DetectorSettings : Voltage
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- ImagingEnvironment : Temperature
- Instrument : ID
- Objective : CalibratedMagnification
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Model
- ObjectiveSettings : ID
- ObjectiveSettings : RefractiveIndex
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
NativeQTReader

This page lists supported metadata fields for the Bio-Formats QuickTime format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats QuickTime format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
**NiftiReader**

This page lists supported metadata fields for the Bio-Formats NIfTI format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**
- The file format itself supports 24 of them (5%).
- Of those, Bio-Formats fully or partially converts 24 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats NIfTI format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
Total supported: 24
Total unknown or missing: 452

NikonElementsTiffReader

This page lists supported metadata fields for the Bio-Formats Nikon Elements TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 50 of them (10%).
- Of those, Bio-Formats fully or partially converts 50 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Nikon Elements TIFF format reader:
- Channel : AcquisitionMode
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : ReadOutRate
- DetectorSettings : Voltage
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- ImagingEnvironment : Temperature
- Instrument : ID
- Objective : CalibratedMagnification
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : Model
• ObjectiveSettings : ID
• ObjectiveSettings : RefractiveIndex
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 50
Total unknown or missing: 426
NikonReader

This page lists supported metadata fields for the Bio-Formats Nikon NEF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Nikon NEF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
NikonTiffReader

This page lists supported metadata fields for the Bio-Formats Nikon TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 47 of them (9%).
- Of those, Bio-Formats fully or partially converts 47 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Nikon TIFF format reader:

- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : Gain
- Detector : ID
- Detector : Type
- Dichroic : ID
- Dichroic : Model
- Filter : ID
- Filter : Model
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : LaserMedium
- Laser : Model
- Laser : Type
- Laser : Wavelength
- Objective : Correction
- Objective : ID
• Objective : Immersion
• Objective : LensNA
• Objective : NominalMagnification
• Objective : WorkingDistance
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 47
Total unknown or missing: 429

**OBFReader**

This page lists supported metadata fields for the Bio-Formats OBF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).
Bio-Formats, Release 6.14.0

Supported fields

These fields are fully supported by the Bio-Formats OBF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

OIRReader

This page lists supported metadata fields for the Bio-Formats Olympus OIR format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 48 of them (10%).
- Of those, Bio-Formats fully or partially converts 48 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Olympus OIR format reader:

- Channel : Color
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : LightSourceSettingsID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : Gain
- Detector : ID
- Detector : Offset
- Detector : Voltage
- DetectorSettings : ID
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Laser : ID
- Laser : Model
- Laser : Wavelength
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Model
- Objective : NominalMagnification
- Objective : WorkingDistance
- ObjectiveSettings : ID
- ObjectiveSettings : RefractiveIndex
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX

4.3. Summary of supported metadata fields
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : DeltaT
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 48
Total unknown or missing: 428

OMETiffReader

This page lists supported metadata fields for the Bio-Formats OME-TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats OME-TIFF format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19

Total unknown or missing: 457

OMEXMLReader

This page lists supported metadata fields for the Bio-Formats OME-XML format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats OME-XML format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC

4.3. Summary of supported metadata fields
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

OlympusTileReader

This page lists supported metadata fields for the Bio-Formats Olympus .omp2info format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Olympus .omp2info format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19

Total unknown or missing: 457

OpenlabRawReader

This page lists supported metadata fields for the Bio-Formats Openlab RAW format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Openlab RAW format reader:
• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

OpenlabReader

This page lists supported metadata fields for the Bio-Formats Openlab LIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 32 of them (6%).
• Of those, Bio-Formats fully or partially converts 32 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Openlab LIFF format reader:

• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Detector : ID
• Detector : Type
• DetectorSettings : Gain
• DetectorSettings : ID
• DetectorSettings : Offset
• Image : AcquisitionDate
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
OperettaReader

This page lists supported metadata fields for the Bio-Formats PerkinElmer Operetta format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 60 of them (12%).
- Of those, Bio-Formats fully or partially converts 60 (100%).

Supported fields

These fields are fully supported by the Bio-Formats PerkinElmer Operetta format reader:

- Channel : AcquisitionMode
- Channel : Color
- Channel : ContrastMethod
- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Experimenter : ID
- Experimenter : LastName
- Image: AcquisitionDate
- Image: ExperimenterRef
- Image: ID
- Image: InstrumentRef
- Image: Name
- Instrument: ID
- Objective: ID
- Objective: LensNA
- Objective: NominalMagnification
- ObjectiveSettings: ID
- Pixels: BigEndian
- Pixels: DimensionOrder
- Pixels: ID
- Pixels: Interleaved
- Pixels: PhysicalSizeX
- Pixels: PhysicalSizeY
- Pixels: PhysicalSizeZ
- Pixels: SignificantBits
- Pixels: SizeC
- Pixels: SizeT
- Pixels: SizeX
- Pixels: SizeY
- Pixels: SizeZ
- Pixels: Type
- Plane: DeltaT
- Plane: ExposureTime
- Plane: PositionX
- Plane: PositionY
- Plane: PositionZ
- Plane: TheC
- Plane: TheT
- Plane: TheZ
- Plate: Columns
- Plate: Description
- Plate: ExternalIdentifier
- Plate: ID
**Bio-Formats, Release 6.14.0**

- Plate : Name
- Plate : Rows
- PlateAcquisition : ID
- PlateAcquisition : MaximumFieldCount
- PlateAcquisition : StartTime
- PlateAcquisition : WellSampleRef
- Well : Column
- Well : ID
- Well : Row
- WellSample : ID
- WellSample : ImageRef
- WellSample : Index
- WellSample : PositionX
- WellSample : PositionY

**Total supported: 60**

**Total unknown or missing: 416**

**OxfordInstrumentsReader**

This page lists supported metadata fields for the Bio-Formats Oxford Instruments format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Oxford Instruments format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder

4.3. **Summary of supported metadata fields**
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

PCIReader

This page lists supported metadata fields for the Bio-Formats Compix Simple-PCI format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 29 of them (6%).
- Of those, Bio-Formats fully or partially converts 29 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Compix Simple-PCI format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- DetectorSettings : Binning
- DetectorSettings : ID
- Image : AcquisitionDate
- Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : DeltaT
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 29
Total unknown or missing: 447

PCORAWReader

This page lists supported metadata fields for the Bio-Formats PCO-RAW format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 26 of them (5%).
• Of those, Bio-Formats fully or partially converts 26 (100%).
Supported fields

These fields are fully supported by the Bio-Formats PCO-RAW format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Detector : ID
- Detector : SerialNumber
- DetectorSettings : Binning
- DetectorSettings : ID
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Instrument : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 26
Total unknown or missing: 450
PCXReader

This page lists supported metadata fields for the Bio-Formats PCX format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats PCX format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
PDSReader

This page lists supported metadata fields for the Bio-Formats Perkin Elmer Densitometer format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 23 of them (4%).
- Of those, Bio-Formats fully or partially converts 23 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Perkin Elmer Densitometer format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : PositionX
- Plane : PositionY
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 23
Total unknown or missing: 453
Supported fields

These fields are fully supported by the Bio-Formats Portable Any Map format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
PQBInReader

This page lists supported metadata fields for the Bio-Formats PicoQuant Bin format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).

Supported fields

These fields are fully supported by the Bio-Formats PicoQuant Bin format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21

Total unknown or missing: 455
PSDReader

This page lists supported metadata fields for the Bio-Formats Adobe Photoshop format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Adobe Photoshop format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
PerkinElmerReader

This page lists supported metadata fields for the Bio-Formats PerkinElmer format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 30 of them (6%).
  • Of those, Bio-Formats fully or partially converts 30 (100%).

Supported fields

These fields are fully supported by the Bio-Formats PerkinElmer format reader:
  • Channel : EmissionWavelength
  • Channel : ExcitationWavelength
  • Channel : ID
  • Channel : SamplesPerPixel
  • Image : AcquisitionDate
  • Image : ID
  • Image : InstrumentRef
  • Image : Name
  • Instrument : ID
  • Pixels : BigEndian
  • Pixels : DimensionOrder
  • Pixels : ID
  • Pixels : Interleaved
  • Pixels : PhysicalSizeX
  • Pixels : PhysicalSizeY
  • Pixels : SignificantBits
  • Pixels : SizeC
  • Pixels : SizeT
  • Pixels : SizeX
  • Pixels : SizeY
  • Pixels : SizeZ
  • Pixels : Type
  • Plane : DeltaT
  • Plane : ExposureTime
  • Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 30
Total unknown or missing: 446

PhotoshopTiffReader

This page lists supported metadata fields for the Bio-Formats Adobe Photoshop TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Adobe Photoshop TIFF format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
Bio-Formats, Release 6.14.0

- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457

**PictReader**

This page lists supported metadata fields for the Bio-Formats PICT format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

*Of the 476 fields documented in the metadata summary table:*

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

*These fields are fully supported by the Bio-Formats PICT format reader:*

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
Total supported: 19
Total unknown or missing: 457

PovrayReader

This page lists supported metadata fields for the Bio-Formats POV-Ray format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats POV-Ray format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
PrairieReader

This page lists supported metadata fields for the Bio-Formats Prairie TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 46 of them (9%).
- Of those, Bio-Formats fully or partially converts 46 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Prairie TIFF format reader:

- Channel : EmissionWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Type
- Detector : Zoom
- DetectorSettings : Gain
- DetectorSettings : ID
- DetectorSettings : Offset
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Instrument : ID
- Laser : ID
- Laser : Power
- Microscope : Model
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : LensNA
- Objective : Manufacturer
- Objective : NominalMagnification
- ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : TimeIncrement
• Pixels : Type
• Plane : DeltaT
• Plane : PositionX
• Plane : PositionY
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 46
Total unknown or missing: 430

PyramidTiffReader

This page lists supported metadata fields for the Bio-Formats Pyramid TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Pyramid TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

QTRenderer

This page lists supported metadata fields for the Bio-Formats QuickTime format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
**Supported fields**

These fields are fully supported by the Bio-Formats QuickTime format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457

**QuesantReader**

This page lists supported metadata fields for the Bio-Formats Quesant AFM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Quesant AFM format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

RCPNLReader

This page lists supported metadata fields for the Bio-Formats RCPNL format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 25 of them (5%).
- Of those, Bio-Formats fully or partially converts 25 (100%).
Supported fields

These fields are fully supported by the Bio-Formats RCPNL format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Objective : Correction
- Objective : Immersion
- Objective : LensNA
- Objective : Manufacturer
- Objective : NominalMagnification
- Objective : WorkingDistance
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 25

Total unknown or missing: 451
**RHKReader**

This page lists supported metadata fields for the Bio-Formats RHK Technologies format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats RHK Technologies format reader:

- **Channel : ID**
- **Channel : SamplesPerPixel**
- **Image : AcquisitionDate**
- **Image : Description**
- **Image : ID**
- **Image : Name**
- **Pixels : BigEndian**
- **Pixels : DimensionOrder**
- **Pixels : ID**
- **Pixels : Interleaved**
- **Pixels : PhysicalSizeX**
- **Pixels : PhysicalSizeY**
- **Pixels : SignificantBits**
- **Pixels : SizeC**
- **Pixels : SizeT**
- **Pixels : SizeX**
- **Pixels : SizeY**
- **Pixels : SizeZ**
- **Pixels : Type**
- **Plane : TheC**
- **Plane : TheT**
- **Plane : TheZ**

**Total supported:** 22

**Total unknown or missing:** 454
**SBIGReader**

This page lists supported metadata fields for the Bio-Formats SBIG format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats SBIG format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 22

**Total unknown or missing:** 454
SDTRender

This page lists supported metadata fields for the Bio-Formats SPCImage Data format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats SPCImage Data format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
SEQReader

This page lists supported metadata fields for the Bio-Formats Image-Pro Sequence format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 20 of them (4%).
- Of those, Bio-Formats fully or partially converts 20 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Image-Pro Sequence format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 20
Total unknown or missing: 456
Bio-Formats, Release 6.14.0

SIFReader

This page lists supported metadata fields for the Bio-Formats Andor SIF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 20 of them (4%).
- Of those, Bio-Formats fully or partially converts 20 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Andor SIF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 20
Total unknown or missing: 456
**SISReader**

This page lists supported metadata fields for the Bio-Formats Olympus SIS TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g., physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 33 of them (6%).
- Of those, Bio-Formats fully or partially converts 33 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Olympus SIS TIFF format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- Detector : Type
- DetectorSettings : ID
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC

4.3. Summary of supported metadata fields
Bio-Formats, Release 6.14.0

- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 33

**Total unknown or missing:** 443

**SMCameraReader**

This page lists supported metadata fields for the Bio-Formats SM Camera format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats SM Camera format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

SPCReader

This page lists supported metadata fields for the Bio-Formats SPC FIFO Data format reader.
These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 19 of them (3%).
  • Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats SPC FIFO Data format reader:
  • Channel : ID
  • Channel : SamplesPerPixel
  • Image : AcquisitionDate
  • Image : ID
  • Image : Name
  • Pixels : BigEndian
  • Pixels : DimensionOrder
  • Pixels : ID
  • Pixels : Interleaved
  • Pixels : SignificantBits
  • Pixels : SizeC
  • Pixels : SizeT
  • Pixels : SizeX
  • Pixels : SizeY
  • Pixels : SizeZ
  • Pixels : Type
  • Plane : TheC
● Plane : TheT
● Plane : TheZ

Total supported: 19
Total unknown or missing: 457

SPEReader

This page lists supported metadata fields for the Bio-Formats Princeton Instruments SPE format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

● The file format itself supports 30 of them (6%).
● Of those, Bio-Formats fully or partially converts 30 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Princeton Instruments SPE format reader:

● Channel : ID
● Channel : SamplesPerPixel
● Image : AcquisitionDate
● Image : ID
● Image : Name
● Image : ROIRef
● Label : ID
● Label : Text
● Label : X
● Label : Y
● Pixels : BigEndian
● Pixels : DimensionOrder
● Pixels : ID
● Pixels : Interleaved
● Pixels : SignificantBits
● Pixels : SizeC
● Pixels : SizeT
● Pixels : SizeX
● Pixels : SizeY
● Pixels : SizeZ
SVSReader

This page lists supported metadata fields for the Bio-Formats Aperio SVS format reader. These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 31 of them (6%).
- Of those, Bio-Formats fully or partially converts 31 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Aperio SVS format reader:

- Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : ID
• Objective : NominalMagnification
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 31
Total unknown or missing: 445

ScanrReader

This page lists supported metadata fields for the Bio-Formats Olympus ScanR format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 43 of them (9%).
• Of those, Bio-Formats fully or partially converts 43 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Olympus ScanR format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : ColumnNamingConvention
- Plate : Columns
- Plate : ID
- Plate : Name
- Plate : RowNamingConvention
- Plate : Rows
- PlateAcquisition : ID
- PlateAcquisition : MaximumFieldCount
Bio-Formats, Release 6.14.0

- PlateAcquisition : WellSampleRef
- Well : Column
- Well : ID
- Well : Row
- WellSample : ID
- WellSample : ImageRef
- WellSample : Index
- WellSample : PositionX
- WellSample : PositionY

Total supported: 43
Total unknown or missing: 433

SeikoReader

This page lists supported metadata fields for the Bio-Formats Seiko format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Seiko format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 22
Total unknown or missing: 454

**SimplePCITiffReader**

This page lists supported metadata fields for the Bio-Formats SimplePCI TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 33 of them (6%).
• Of those, Bio-Formats fully or partially converts 33 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats SimplePCI TIFF format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Detector : ID
• Detector : Model
• Detector : Type
• DetectorSettings : Binning
• DetectorSettings : ID
• Image : AcquisitionDate
• Image : Description
• Image : ID
• Image : InstrumentRef
• Image : Name
• Instrument : ID
• Objective : ID
• Objective : Immersion
• Objective : NominalMagnification
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 33
Total unknown or missing: 443

SlideBook7Reader

This page lists supported metadata fields for the Bio-Formats SlideBook 7 SLD (native) format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 37 of them (7%).
• Of those, Bio-Formats fully or partially converts 37 (100%).
Supported fields

These fields are fully supported by the Bio-Formats SlideBook 7 SLD (native) format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : Model
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 37
Total unknown or missing: 439

**SlidebookReader**

This page lists supported metadata fields for the Bio-Formats Olympus Slidebook format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 34 of them (7%).
- Of those, Bio-Formats fully or partially converts 34 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Olympus Slidebook format reader:

- Channel : ID
- Channel : NDFilter
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : Model
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 34
Total unknown or missing: 442

**SlidebookTiffReader**

This page lists supported metadata fields for the Bio-Formats Slidebook TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 30 of them (6%).
- Of those, Bio-Formats fully or partially converts 30 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Slidebook TIFF format reader:

• Channel : ID
• Channel : Name
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Instrument : ID
Bio-Formats, Release 6.14.0

- Objective: Correction
- Objective: ID
- Objective: Immersion
- Objective: NominalMagnification
- Pixels: BigEndian
- Pixels: DimensionOrder
- Pixels: ID
- Pixels: Interleaved
- Pixels: PhysicalSizeX
- Pixels: PhysicalSizeY
- Pixels: SignificantBits
- Pixels: SizeC
- Pixels: SizeT
- Pixels: SizeX
- Pixels: SizeY
- Pixels: SizeZ
- Pixels: Type
- Plane: PositionX
- Plane: PositionY
- Plane: PositionZ
- Plane: TheC
- Plane: TheT
- Plane: TheZ

**Total supported:** 30

**Total unknown or missing:** 446

**SpiderReader**

This page lists supported metadata fields for the Bio-Formats SPIDER format reader.

These fields are from the OME data model. Bio-Formats standardizes each format's original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 21 of them (4%).
- Of those, Bio-Formats fully or partially converts 21 (100%).
Supported fields

These fields are fully supported by the Bio-Formats SPIDER format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 21
Total unknown or missing: 455

TCSReader

This page lists supported metadata fields for the Bio-Formats Leica TCS TIFF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Leica TCS TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454

TargaReader

This page lists supported metadata fields for the Bio-Formats Truevision Targa format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 20 of them (4%).
- Of those, Bio-Formats fully or partially converts 20 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Truevision Targa format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 20

Total unknown or missing: 456

TecanReader

This page lists supported metadata fields for the Bio-Formats Tecan Spark Cyto format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 36 of them (7%).
- Of those, Bio-Formats fully or partially converts 36 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Tecan Spark Cyto format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : Columns
- Plate : ID
- Plate : Name
- Plate : Rows
- PlateAcquisition : ID
- PlateAcquisition : MaximumFieldCount
- PlateAcquisition : WellSampleRef
- Well : Column
- Well : ID
- Well : Row
- WellSample : ID
- `WellSample : ImageRef`
- `WellSample : Index`

**Total supported:** 36

**Total unknown or missing:** 440

**TextReader**

This page lists supported metadata fields for the Bio-Formats Text format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Text format reader:

- `Channel : ID`
- `Channel : SamplesPerPixel`
- `Image : AcquisitionDate`
- `Image : ID`
- `Image : Name`
- `Pixels : BigEndian`
- `Pixels : DimensionOrder`
- `Pixels : ID`
- `Pixels : Interleaved`
- `Pixels : SignificantBits`
- `Pixels : SizeC`
- `Pixels : SizeT`
- `Pixels : SizeX`
- `Pixels : SizeY`
- `Pixels : SizeZ`
- `Pixels : Type`
- `Plane : TheC`
- `Plane : TheT`
- `Plane : TheZ`
Total supported: 19
Total unknown or missing: 457

TiffDelegateReader

This page lists supported metadata fields for the Bio-Formats Tagged Image File Format format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Tagged Image File Format format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457
TiffJAIReader

This page lists supported metadata fields for the Bio-Formats Tagged Image File Format format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Tagged Image File Format format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
TiffReader

This page lists supported metadata fields for the Bio-Formats Tagged Image File Format format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Tagged Image File Format format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454
**TileJPEGReader**

This page lists supported metadata fields for the Bio-Formats Tile JPEG format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Tile JPEG format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
TillVisionReader

This page lists supported metadata fields for the Bio-Formats TillVision format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats TillVision format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Experiment : ID
- Experiment : Type
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : ExposureTime
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454
TopometrixReader

This page lists supported metadata fields for the Bio-Formats TopoMetrix format reader.
These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats TopoMetrix format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 22
Total unknown or missing: 454
TrestleReader

This page lists supported metadata fields for the Bio-Formats Trestle format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 27 of them (5%).
- Of those, Bio-Formats fully or partially converts 27 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Trestle format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Image : ROIRef
- Mask : BinData
- Mask : Height
- Mask : ID
- Mask : Width
- Mask : X
- Mask : Y
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
• Plane : TheZ
• ROI : ID

Total supported: 27
Total unknown or missing: 449

**UBMReader**

This page lists supported metadata fields for the Bio-Formats UBM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**
- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats UBM format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
Total supported: 19
Total unknown or missing: 457

UnisokuReader

This page lists supported metadata fields for the Bio-Formats Unisoku STM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Unisoku STM format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ
Total supported: 22
Total unknown or missing: 454

VGSAMReader

This page lists supported metadata fields for the Bio-Formats VG SAM format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 19 of them (3%).
• Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats VG SAM format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457
VarianFDFReader

This page lists supported metadata fields for the Bio-Formats Varian FDF format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 25 of them (5%).
- Of those, Bio-Formats fully or partially converts 25 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Varian FDF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ
Total supported: 25
Total unknown or missing: 451

**VectraReader**

This page lists supported metadata fields for the Bio-Formats PerkinElmer Vectra/QPTIFF format reader.

These fields are from the **OME data model**. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 43 of them (9%).
- Of those, Bio-Formats fully or partially converts 43 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats PerkinElmer Vectra/QPTIFF format reader:

- Channel : Color
- Channel : Fluor
- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- Experimenter : ID
- Experimenter : UserName
- Image : AcquisitionDate
- Image : Description
- Image : ExperimenterRef
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Microscope : Model
- Objective : ID
- Objective : Model
• Objective : NominalMagnification
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : ExposureTime
• Plane : PositionX
• Plane : PositionY
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 43
Total unknown or missing: 433

**VeecoReader**

This page lists supported metadata fields for the Bio-Formats Veeco format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Veeco format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19
Total unknown or missing: 457

VentanaReader

This page lists supported metadata fields for the Bio-Formats Ventana .bif format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 28 of them (5%).
- Of those, Bio-Formats fully or partially converts 28 (100%).
**Supported fields**

These fields are fully supported by the Bio-Formats Ventana .bif format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : ID
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : PositionX
- Plane : PositionY
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 28

Total unknown or missing: 448
VisitechReader

This page lists supported metadata fields for the Bio-Formats Visitech XYS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Visitech XYS format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
**VolocityClippingReader**

This page lists supported metadata fields for the Bio-Formats Volocity Library Clipping format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Volocity Library Clipping format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

**Total supported:** 19

**Total unknown or missing:** 457
**VelocityReader**

This page lists supported metadata fields for the Bio-Formats Volocity Library format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

**Of the 476 fields documented in the metadata summary table:**

- The file format itself supports 38 of them (7%).
- Of those, Bio-Formats fully or partially converts 38 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Volocity Library format reader:

- Channel : ID
- Channel : Name
- Channel : SamplesPerPixel
- Detector : ID
- Detector : Model
- DetectorSettings : ID
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Instrument : ID
- Objective : Correction
- Objective : ID
- Objective : Immersion
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
WATOPReader

This page lists supported metadata fields for the Bio-Formats WA Technology TOP format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 22 of them (4%).
- Of those, Bio-Formats fully or partially converts 22 (100%).

Supported fields

These fields are fully supported by the Bio-Formats WA Technology TOP format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 22
Total unknown or missing: 454

WlzReader

This page lists supported metadata fields for the Bio-Formats Woolz format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 26 of them (5%).
• Of those, Bio-Formats fully or partially converts 26 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Woolz format reader:

• Channel : ID
• Channel : SamplesPerPixel
• Image : AcquisitionDate
• Image : ID
• Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

**Supported fields**

These fields are fully supported by the Bio-Formats Extended leica file format reader:
- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : SignificantBits
• Pixels : SizeC
• Pixels : SizeT
• Pixels : SizeX
• Pixels : SizeY
• Pixels : SizeZ
• Pixels : Type
• Plane : TheC
• Plane : TheT
• Plane : TheZ

Total supported: 19
Total unknown or missing: 457

ZeissCZIReader

This page lists supported metadata fields for the Bio-Formats Zeiss CZI format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:
  • The file format itself supports 175 of them (36%).
  • Of those, Bio-Formats fully or partially converts 175 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Zeiss CZI format reader:
• Arc : LotNumber
• Arc : Manufacturer
• Arc : Model
• Arc : Power
• Arc : SerialNumber
• Channel : AcquisitionMode
• Channel : Color
• Channel : EmissionWavelength
- Channel : ExcitationWavelength
- Channel : FilterSetRef
- Channel : Fluor
- Channel : ID
- Channel : IlluminationType
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : AmplificationGain
- Detector : Gain
- Detector : ID
- Detector : LotNumber
- Detector : Manufacturer
- Detector : Model
- Detector : Offset
- Detector : SerialNumber
- Detector : Type
- Detector : Zoom
- DetectorSettings : Binning
- DetectorSettings : Gain
- DetectorSettings : ID
- Dichroic : ID
- Dichroic : LotNumber
- Dichroic : Manufacturer
- Dichroic : Model
- Dichroic : SerialNumber
- Ellipse : ID
- Ellipse : RadiusX
- Ellipse : RadiusY
- Ellipse : Text
- Ellipse : X
- Ellipse : Y
- Experimenter : Email
- Experimenter : FirstName
- Experimenter : ID
- Experimenter : Institution
• Experimenter : LastName
• Experimenter : MiddleName
• Experimenter : UserName
• Filament : LotNumber
• Filament : Manufacturer
• Filament : Model
• Filament : Power
• Filament : SerialNumber
• Filter : FilterWheel
• Filter : ID
• Filter : LotNumber
• Filter : Manufacturer
• Filter : Model
• Filter : SerialNumber
• Filter : Type
• FilterSet : DichroicRef
• FilterSet : EmissionFilterRef
• FilterSet : ExcitationFilterRef
• FilterSet : ID
• FilterSet : LotNumber
• FilterSet : Manufacturer
• FilterSet : Model
• FilterSet : SerialNumber
• Image : AcquisitionDate
• Image : Description
• Image : ExperimenterRef
• Image : ID
• Image : InstrumentRef
• Image : Name
• Image : ROIRef
• ImagingEnvironment : AirPressure
• ImagingEnvironment : CO2Percent
• ImagingEnvironment : Humidity
• ImagingEnvironment : Temperature
• Instrument : ID
• Laser : LotNumber
• Laser : Manufacturer
• Laser : Model
• Laser : Power
• Laser : SerialNumber
• LightEmittingDiode : LotNumber
• LightEmittingDiode : Manufacturer
• LightEmittingDiode : Model
• LightEmittingDiode : Power
• LightEmittingDiode : SerialNumber
• Line : ID
• Line : Text
• Line : X1
• Line : X2
• Line : Y1
• Line : Y2
• Microscope : LotNumber
• Microscope : Manufacturer
• Microscope : Model
• Microscope : SerialNumber
• Microscope : Type
• Objective : CalibratedMagnification
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : Iris
• Objective : LensNA
• Objective : LotNumber
• Objective : Manufacturer
• Objective : Model
• Objective : NominalMagnification
• Objective : SerialNumber
• Objective : WorkingDistance
• ObjectiveSettings : CorrectionCollar
• ObjectiveSettings : ID
• ObjectiveSettings : Medium
• ObjectiveSettings : RefractiveIndex
4.3. Summary of supported metadata fields

- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : PhysicalSizeX
- Pixels : PhysicalSizeY
- Pixels : PhysicalSizeZ
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : DeltaT
- Plane : ExposureTime
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Plate : Columns
- Plate : ID
- Plate : Rows
- Point : ID
- Point : X
- Point : Y
- Polygon : ID
- Polygon : Points
- Polygon : Text
- Polyline : ID
- Polyline : Points
- Polyline : Text
- ROI : Description
• ROI : ID
• ROI : Name
• Rectangle : Height
• Rectangle : ID
• Rectangle : Text
• Rectangle : Width
• Rectangle : X
• Rectangle : Y
• StageLabel : Name
• StageLabel : X
• StageLabel : Y
• StageLabel : Z
• TransmittanceRange : CutIn
• TransmittanceRange : CutInTolerance
• TransmittanceRange : CutOut
• TransmittanceRange : CutOutTolerance
• TransmittanceRange : Transmittance
• Well : Column
• Well : ID
• Well : Row
• WellSample : ID
• WellSample : ImageRef
• WellSample : Index

Total supported: 175
Total unknown or missing: 301

ZeissLMSReader

This page lists supported metadata fields for the Bio-Formats Zeiss LMS format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

• The file format itself supports 23 of them (4%).
• Of those, Bio-Formats fully or partially converts 23 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Zeiss LMS format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Instrument : ID
- Objective : ID
- Objective : NominalMagnification
- ObjectiveSettings : ID
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 23
Total unknown or missing: 453

ZeissLSMReader

This page lists supported metadata fields for the Bio-Formats Zeiss Laser-Scanning Microscopy format reader.

These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 101 of them (21%).
- Of those, Bio-Formats fully or partially converts 101 (100%).
Supported fields

These fields are fully supported by the Bio-Formats Zeiss Laser-Scanning Microscopy format reader:

- Channel : Color
- Channel : ID
- Channel : Name
- Channel : PinholeSize
- Channel : SamplesPerPixel
- Detector : AmplificationGain
- Detector : Gain
- Detector : ID
- Detector : Type
- Detector : Zoom
- DetectorSettings : Binning
- DetectorSettings : ID
- Dichroic : ID
- Dichroic : Model
- Ellipse : FontSize
- Ellipse : ID
- Ellipse : RadiusX
- Ellipse : RadiusY
- Ellipse : StrokeWidth
- Ellipse : Transform
- Ellipse : X
- Ellipse : Y
- Experimenter : ID
- Experimenter : UserName
- Filter : ID
- Filter : Model
- Filter : Type
- Image : AcquisitionDate
- Image : Description
- Image : ID
- Image : InstrumentRef
- Image : Name
- Image : ROIRef
- Instrument : ID
• Label :FontSize
• Label : ID
• Label : StrokeWidth
• Label : Text
• Label : X
• Label : Y
• Laser : ID
• Laser : LaserMedium
• Laser : Model
• Laser : Type
• Laser : Wavelength
• LightPath : DichroicRef
• LightPath : EmissionFilterRef
• Line : FontSize
• Line : ID
• Line : StrokeWidth
• Line : X1
• Line : X2
• Line : Y1
• Line : Y2
• Objective : Correction
• Objective : ID
• Objective : Immersion
• Objective : Iris
• Objective : LensNA
• Objective : NominalMagnification
• ObjectiveSettings : ID
• Pixels : BigEndian
• Pixels : DimensionOrder
• Pixels : ID
• Pixels : Interleaved
• Pixels : PhysicalSizeX
• Pixels : PhysicalSizeY
• Pixels : PhysicalSizeZ
• Pixels : SignificantBits
• Pixels : SizeC

4.3. Summary of supported metadata fields
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : TimeIncrement
- Pixels : Type
- Plane : DeltaT
- Plane : PositionX
- Plane : PositionY
- Plane : PositionZ
- Plane : TheC
- Plane : TheT
- Plane : TheZ
- Polygon : FontSize
- Polygon : ID
- Polygon : Points
- Polygon : StrokeWidth
- Polyline : FontSize
- Polyline : ID
- Polyline : Points
- Polyline : StrokeWidth
- ROI : ID
- Rectangle : FontSize
- Rectangle : Height
- Rectangle : ID
- Rectangle : StrokeWidth
- Rectangle : Width
- Rectangle : X
- Rectangle : Y
- TransmittanceRange : CutIn
- TransmittanceRange : CutOut

Total supported: 101

Total unknown or missing: 375
ZeissTIFFReader

This page lists supported metadata fields for the Bio-Formats Zeiss AxioVision TIFF format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Zeiss AxioVision TIFF format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
ZeissZVIReader

This page lists supported metadata fields for the Bio-Formats Zeiss Vision Image (ZVI) format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Zeiss Vision Image (ZVI) format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
ZipReader

This page lists supported metadata fields for the Bio-Formats Zip format reader. These fields are from the OME data model. Bio-Formats standardizes each format’s original metadata to and from the OME data model so that you can work with a particular piece of metadata (e.g. physical width of the image in microns) in a format-independent way.

Of the 476 fields documented in the metadata summary table:

- The file format itself supports 19 of them (3%).
- Of those, Bio-Formats fully or partially converts 19 (100%).

Supported fields

These fields are fully supported by the Bio-Formats Zip format reader:

- Channel : ID
- Channel : SamplesPerPixel
- Image : AcquisitionDate
- Image : ID
- Image : Name
- Pixels : BigEndian
- Pixels : DimensionOrder
- Pixels : ID
- Pixels : Interleaved
- Pixels : SignificantBits
- Pixels : SizeC
- Pixels : SizeT
- Pixels : SizeX
- Pixels : SizeY
- Pixels : SizeZ
- Pixels : Type
- Plane : TheC
- Plane : TheT
- Plane : TheZ

Total supported: 19

Total unknown or missing: 457
4.4 Grouping files using a pattern file

Individual files can be grouped together into a single fileset using a pattern file. This works for any single-file format that Bio-Formats supports, as long as all files are in the same format. It is most useful for sets of TIFF, JPEG, PNG, etc. files that do not have any associated metadata.

All files to be grouped together should be in the same folder. The pattern file should be in the same folder as the other files; it can have any name, but must have the .pattern extension. The pattern file is what must be opened or imported, so it may be helpful to give it a descriptive or easily-recognizable name.

The pattern file contains a single line of text that is specially formatted to describe how the files should be grouped. The file can be created in any text editor.

The text in the pattern file can take one of several forms. To illustrate, consider a folder with the following file names:

<table>
<thead>
<tr>
<th>red.tiff</th>
<th>green.tiff</th>
<th>blue.tiff</th>
</tr>
</thead>
<tbody>
<tr>
<td>test_Z0_C0.png</td>
<td>test_Z1_C0.png</td>
<td>test_Z0_C1.png</td>
</tr>
<tr>
<td>test_Z0_C2.png</td>
<td>test_Z1_C1.png</td>
<td>test_Z00.tiff</td>
</tr>
<tr>
<td>test_Z01.tiff</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A pattern file that groups red.tiff, green.tiff, and blue.tiff in that order would look like:

<red,green,blue>.tiff

A pattern that groups test_Z0_C0.png, test_Z1_C0.png, test_Z0_C2.png, and test_Z1_C2.png:

test_Z<0-1>_C<0-2:2>.png

The <> notation in general can accept a single literal value, a comma-separated list of literal values, a range of integer values, or a range of integer values with a step value greater than 1 (the range and step are separated by :). Note that inverting the values in a range (e.g. <2-0>) is not supported and will cause an exception to be thrown.

The characters immediately preceding the < can affect which dimension is assigned to the specified values. The values will be interpreted as:

- channels, if c, ch, w, or wavelength precede <
- timepoints, if t, tl, tp, or timepoint precede <
- Z sections, if z, zs, sec, fp, focal, or focalplane precede <
- series, if s, sp, or series precede <

Note that the listed dimension specifier characters are case insensitive. A separator character (underscore or space) must precede the dimension specifier if it is not at the beginning of the filename. In the above example, 2 Z sections and 2 out of 3 channels would be detected according to the dimension specifiers.

Leading zeros in the integer values must be specified. To group test_Z00.tiff and test_Z01.tiff:

test_Z<00-01>.tiff

or:
Note that this pattern would not group the files correctly:

```
(test_Z<0-1>).tiff
```

A pattern file that groups all PNG files beginning with test_ would look like:

```
test_.png
```

This and most other Java-style regular expressions can be used in place of the <> notation above. See the java.util.regex.Pattern Javadoc for more information on constructing regular expressions.

### 4.5 Additional reader and writer options

Some readers and writers have additional options which can be used to inform how Bio-Formats reads or writes files in that format.

#### 4.5.1 Reader options

<table>
<thead>
<tr>
<th>Format name</th>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>cellSens VSI</strong></td>
<td>cellsens. fail_on_missing_ets</td>
<td>false</td>
<td>Throw an exception if an expected associated .ets file is missing</td>
</tr>
<tr>
<td><strong>Gatan Digital Micrograph</strong></td>
<td>gatan. split_montage</td>
<td>true</td>
<td>Split montage image tiles across multiple planes</td>
</tr>
<tr>
<td><strong>Leica LAS AF LIF (Leica Image File Format)</strong></td>
<td>leicalif. old_physical_size</td>
<td>false</td>
<td>Ensure physical pixel sizes are compatible with versions &lt;= 5.3.2</td>
</tr>
<tr>
<td><strong>Nikon NIS-Elements ND2</strong></td>
<td>nativend2.chunkmap</td>
<td>true</td>
<td>Use chunkmap table to read image offsets</td>
</tr>
<tr>
<td><strong>Olympus ScanR</strong></td>
<td>scanr. skip_missing_wells</td>
<td>true</td>
<td>Ignore missing wells</td>
</tr>
<tr>
<td><strong>OME-TIFF</strong></td>
<td>ometiff. fail_on_missing_tiff</td>
<td>true</td>
<td>Fail if a missing file is detected in a partial dataset</td>
</tr>
<tr>
<td><strong>Ventana BIF</strong></td>
<td>ventana. split_tiles</td>
<td>false</td>
<td>Report each tile as a separate series</td>
</tr>
<tr>
<td><strong>Zeiss CZI</strong></td>
<td>zeissczi. attachments</td>
<td>true</td>
<td>Include attachment images</td>
</tr>
<tr>
<td><strong>Zeiss CZI</strong></td>
<td>zeissczi. autostitch</td>
<td>true</td>
<td>Automatically stitch tiled images</td>
</tr>
<tr>
<td><strong>Zeiss CZI</strong></td>
<td>zeissczi. trim_dimensions</td>
<td>false</td>
<td>Trim XY dimensions to match those in ZEN</td>
</tr>
<tr>
<td><strong>Zeiss CZI</strong></td>
<td>zeissczi. relative_positions</td>
<td>false</td>
<td>Use pixel position instead of the physical stage position</td>
</tr>
</tbody>
</table>
Usage

Reader options can be used via the command line with `showinf -option`, in ImageJ via the configuration window, or via the API using the DynamicMetadataOptions class.

4.5.2 Writer options

<table>
<thead>
<tr>
<th>Format name</th>
<th>Option</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OME-TIFF</td>
<td>ometiff.</td>
<td>None</td>
<td>If set, OME-XML will be written to a companion file with a name determined by the option value</td>
</tr>
<tr>
<td>OME-TIFF</td>
<td>ometiff.</td>
<td>false</td>
<td>If set to false, the OME-XML Creator attribute will be updated to include the current Bio-Formats version. This will overwrite any existing Creator in the input metadata. If set to true, the Creator attribute will only be updated if it wasn’t previously set; any existing value will be preserved.</td>
</tr>
<tr>
<td>OME-XML</td>
<td>omexml.</td>
<td>false</td>
<td>If set to false, the OME-XML Creator attribute will be updated to include the current Bio-Formats version. This will overwrite any existing Creator in the input metadata. If set to true, the Creator attribute will only be updated if it wasn’t previously set; any existing value will be preserved.</td>
</tr>
</tbody>
</table>

Usage

Writer options can be used via the command line using `bfconvert -option`, or via the API using the DynamicMetadataOptions class.
5.1 6.14.0 (2023 July)

Deprecation warnings:

- **Legacy ND2 Reader**
  
  The LegacyND2Reader and underlying components have been marked as deprecated in preparation for removal in the upcoming 7.0.0 major release of Bio-Formats. This reader depends on an outdated DLL which has not been built in years, is untested and fully superseded by the new NativeND2Reader. From Bio-Formats 7.0.0 onwards the existing NativeND2Reader will be renamed as ND2Reader and act as the sole reader for the format. Support for the ND2 format will be unaffected and continue via this newly renamed reader.

- **Legacy QuickTime**
  
  The LegacyQT Reader, Writer and Tools have been marked as deprecated in preparation for removal in the upcoming 7.0.0 major release of Bio-Formats. These classes depended on the external QuickTime for Java library which is long obsolete. From Bio-Formats 7.0.0 onwards the existing NativeQTReader will be renamed as QTReader and support for QuickTime will be limited to non legacy readers and writers.

- **LuraWave Codec**
  
  The LuraWave Codec along with the LuraWaveService and associated components have been marked as deprecated in preparation for removal in the upcoming 7.0.0 major release of Bio-Formats. This codec depends on a third-party proprietary library with a license code. Opera Flex is the format that is most likely to be impacted by this change.

- **Woolz**
  
  The Woolz Reader, Writer and Service have been marked as deprecated in preparation for removal in the upcoming 7.0.0 major release of Bio-Formats. The reader and writer are untested and rely on an underlying library which maintenance status is unclear.

File format fixes and improvements:

- **CV7000/CV8000**
  
  - updated CV7000 isThisType test to improve performance by skipping extended type checking
  
  - reader now handles the use case where wells are recorded, but all files were removed
  
  - fixed channel indexing in scenarios where more channels are defined than acquired

- **ICS (Image Cytometry Standard)**
  
  - fixed an Illegal group reference exception when constructing slice label
Bio-Formats, Release 6.14.0

- **KLB (Keller Lab Block)**
  - fixed a bug with the reading of image planes in Z stacks

- **MicroManager**
  - updated handling of truncated files to prevent possible infinite loop and throw an exception

Bio-Formats improvements:
- updated handling of exceptions in Bio-Formats plugins to ensure readers are closed
- updated list of external readers to include new external ZeissQuickStartCZIReader (thanks to Nicolas Chiaruttini)
- updated top-level Git mailmap to normalize commit author variants
- added a new bf-unconfigured tool to the Bio-Formats toolbox. The new tool lists classes which are not configured for testing as part of format reader tests
- replaced use tt tag in java docs with code tag

Documentation improvements:
- fixed a number of broken links
- migrated OME-Model documentation to Read the Docs
- added deprecation warnings to format pages of the affected components

Component updates:
- **ome-common** was upgraded to 6.0.17

5.2 6.13.0 (2023 April)

File format fixes and improvements:
- **Aperio SVS**
  - removed broken special case path when there is only one (pyramidal) image (thanks to Alexandr Virodov)

- **Bio-Rad Gel**
  - fixed pixel data offsets for cropped images

- **cellSens VSI**
  - improved tile decoding to fail faster on invalid tiles

- **CellWorX/MetaXpress**
  - fixed a bug when retrieving channel metadata when the files for the entire first field are missing

- **CV7000**
  - extra folders mixed in with the plate data will now be ignored
  - corrected channel indexing if a subset of channels was acquired

- **Lambert Instruments FLIM**
  - added support for Fli 2.0 (thanks to Shaquille Louisa - Lambert Instruments)

- **MetaMorph**
  - prevented an exception when attempting to read metadata keys that don’t exist
• **MetaMorph TIFF**
  – fixed a NullPointerException when there are missing stage positions

• **MRC**
  – the IMOD flag is now used to determine int8 or uint8 pixel type

• **NIFTI**
  – improved the performance for gzip-compressed files by reducing the number of backwards seeks

• **Nikon ND2**
  – fixed an exception when reading blocks greater than 2GB

• **OME-TIFF**
  – file paths are now normalized rather than resolving symlinks

• **Perkin Elmer Operetta**
  – fixed series validation for single plane images (thanks to Alex Herbert)

• **Zeiss CZI**
  – improved the tile stitching when a pyramid does not exist

Bio-Formats improvements:

• added a new writer option `preserve_creator` for OME-TIFF and OME-XML writers. When set to true the original OME/Creator attribute will be preserved by the writer. If set to false, the OME Creator attribute will be updated to include the current Bio-Formats version. Attribute is set to false by default.

• enabled the configuration of the fill value for undefined pixel values through new API additions to IFormatReader. Two new methods (along with default implementations) have been added for `setFillColor(Byte color)` and `Byte getFillColor()`, the default value if undefined is 0.

• added a new `-fill` option to the command line tools, allowing a byte value to be configured for use with undefined pixels (0-255)

• enabled the configuration of JPEG quality in the JPEGCodec using the existing quality setting on CodecOptions (thanks to Peter Haub)

• improved JPEG performance by using in-memory cache in JPEGCodec (thanks to Pete Bankhead)

• made minor API updates to Memoizer to allow a null reader to be specified in the constructors, in which case a new ImageReader will be wrapped

• prevented file leaks in FileStitcher by closing readers when setting the class list

• fixed exceptions in FileStitcher when a no-op file pattern is encountered

• updated GitHub Actions release process

Documentation improvements:

• updated the format page for MRC to clarify list of developers and add links to latest specifications

• added new developer documentation pages for Bioconductor RBioFormats

• corrected one of the examples in the OME-TIFF specification page (thanks to Tom Boissonnet)

• updated the list of writer options to add documentation for new `preserve_creator` options

• updated the list of command line tools options to add documentation for new `fill` option

• version history and bug reporting are now linked from the main documentation page
• fixed a number of broken links
• updated the Li-Flim format page to add supported versions

Component updates:
• \textit{ome-common} was upgraded to 6.0.16
• \textit{ome-model} was upgraded to 6.3.3
• \textit{ome-codecs} was upgraded to 0.4.5
• \textit{imagej1} was upgraded to 1.54c
• \textit{json} was upgraded to 20230227
• \textit{kryo} was upgraded to 5.4.0
• \textit{snakeyaml} was upgraded to 2.0
• \textit{joda-time} was upgraded to 2.12.2
• \textit{guava} was upgraded to 31.1-jre
• \textit{jackson-annotations} was upgraded to 12.14.2
• \textit{jackson-databind} was upgraded to 12.14.2

5.3 6.12.0 (2023 February)

File format fixes and improvements:
• \textbf{Lambert Instruments FLIM}
  – added support for MSB 12 to 16 bit conversion (thanks to Shaquille Louisa - Lambert Instruments)
• \textbf{Olympus cellSens VSI}
  – corrected tile handling to respect tile origin metadata
  – pixel data will now only be read from ets files with the prefix \texttt{frame\textunderscore} (funded by a partnership between Glencoe Software and Evident Scientific)
• \textbf{Perkin Elmer Operetta/Harmony}
  – plates containing 32-bit TIFF files will now return blank planes
• \textbf{Zeiss CZI}
  – fixed pyramid dimension calculation

Bio-Formats improvements:
• implemented full Java 17 support for Bio-Formats
• \textit{logback} dependencies are now optional as per the SLF4J best practices (thanks to Curtis Rueden)
  – downstream consumers must now declare the specific SLF4J binding they wish to use
• CI build matrix updated to use Java versions 8, 11 and 17
• ant tasks switched to use Maven Artifact Resolver
• fixed a bug when decompressing using the Base64Codec
• added a new ImageJ macro example to Bio-Formats examples
• FakeReader now ensures correct dimension order for RGB image
• added new unit tests for FakeReader
• fixed min/max calculation for RGB data
• artifacts from the GitHub Actions build are now uploaded and archived for 30 days
• release artifacts are now generated via GitHub Actions
• fixed a number of small issues in OME-Model (thanks to Roger Leigh)

Documentation improvements:
• migrated Bio-Formats documentation to Read the Docs
• fixed and updated a number of links in documentation
• updated OME-Model links to comply with Sphinx 5 deprecation warnings
• updated GitHub Actions set-output to comply with deprecation warnings
• updated OME-TIFF documentation to add links to new sample dataset BBBC017

Component updates:
• ome-metakit was upgraded to 5.3.5
• ome-common was upgraded to 6.0.14
• ome-codecs was upgraded to 0.4.4
• ome-model was upgraded to 6.3.2
• jackson-annotations was upgraded to 12.14.1
• jackson-databind was upgraded to 12.14.1
• aircompressor was upgraded to 0.21

5.4 6.11.1 (2022 December)

File format fixes and improvements:
• 3i SlideBook 7
  – refactored code to remove a false positive threat report in Fortinet
• DICOM
  – fixed handling of some datasets with multiple optical paths
• Leica SCN
  – fixed colour correction for datasets from SCN 400 models
• Olympus cellSens VSI
  – corrected the handling of exposure times
• TIFF
  – handle REFERENCE_BLACK_WHITE tag as an array of floats or ints
• Vectra QPTIFF
  – biomarker data in channel name will now be preserved
• Zeiss CZI
Bio-Formats improvements:

- added support for plates with multiple fields

Bio-Formats improvements:

- updated the output command for GitHub actions

Documentation improvements:

- fixed a number of broken links
- added a link to public sample files for Olympus-FluoView

Component updates:

- `snakeyaml` was upgraded to 1.3.2

5.5 6.11.0 (2022 October)

File format fixes and improvements:

- **Amira Mesh**
  - fixed handling of files with empty key values
- **InCell 1000/2000**
  - improved calculation of plane count
- **Nikon NIS-Elements ND2**
  - fixed handling of chunk map skipping logic
- **Olympus FluoView FV1000**
  - fixed an exception when parsing double values in the metadata
- **OME-TIFF**
  - improved initialization performance for files with a large number of planes
  - enabled individual file reading for datasets with one file per series
- **SimplePCI & HCImage**
  - fixed physical size calculation to make use of magnification value
- **TIFF**
  - implemented Zstandard decompression for Tiff formats (thanks to Willem Pomp)

Bio-Formats improvements:

- updated pattern file handling to support a single file wrapped in a pattern file
- Bio-Formats plugin now closes underlying readers after an exception
- contrast will be retained when using VirtualImagePlus in the Bio-Formats plugin (thanks to Tomas Farago)
- fixed overwrite checking in bfconvert tool when output path is a pattern

Documentation improvements:

- fixed a number of broken links
- updated test image documentation to add note on on setting fake file channel colors
- added new public sample files for Olympus-FluoView
• updated OME-TIFF sample files for BBBC017

Component updates:
• ome-codecs was upgraded to 0.4.1
•snakeyaml was upgraded to 1.3.1

5.6 6.10.1 (2022 August)

File format fixes and improvements:
• Hamamatsu NDPI
  – fixed handling of non number tags for files greater than 4GB in size
• OME-TIFF
  – improved performance of initializing a single file OME-TIFF datasets with a larger number of series
• TIFF (Tagged Image File Format)
  – fixed a ClassCastException when the SubfileType tag has the wrong type
• Zeiss CZI
  – scene names will now be taken into account when generating image names

Documentation improvements:
• fixed a number of broken links

Component updates:
• ome-common was upgraded to 6.0.13
• ome-poi was upgraded to 5.3.7
• slf4j was upgraded to 1.7.30
• metadata-extractor was upgraded to 2.18.0

5.7 6.10.0 (2022 May)

New file formats:
• Slidebook 7
  – added support for reading SlideBook Format 7 from SlideBook 2021 This functionality was implemented and contributed by Intelligent Imaging Innovations

File format fixes and improvements:
• CellWorX / MetaXpress
  – channel metadata will now be preserved for all channels by reading from each file
• Cellomics
  – marked the private Pattern field as transient to fix memo file generation with JDK17
• DeltaVision
  – fixed detection of incorrect XY tile counts
• **Hitachi S-4800**
  - enhanced format recognition and identification of datasets
  - corrected the units for physical sizes to nanometers instead of micrometers
• **Nikon NIS-Elements ND2**
  - improved metadata parsing, especially for objective data, timestamps, and exposure times
• **PerkinElmer Operetta**
  - improved checks for invalid TIFFs and supplemental metadata files
  - updated image names to make well names more readable
• **TIFF (Tagged Image File Format)**
  - fixed incorrect SampleValue metadata values
• **Vectra QPTIFF**
  - reader has been updated to support schema version 4
  - funded by a partnership between Glencoe Software and Akoya Biosciences.

Bio-Formats improvements:
  - updated automated testing to ensure files are initialized before all tests

Documentation improvements:
  - fixed broken link in MIPAV documentation
  - updated logback component version in developer docs
  - updated link to View5D software
  - added a new format page for Slidebook 7

Component updates:
  - *ome-metakit* was upgraded to 5.3.4
  - *ome-common* was upgraded to 6.0.9
  - *ome-model* was upgraded to 6.3.1
  - *ome-poi* was upgraded to 5.3.6
  - *ome-codecs* was upgraded to 0.3.2
  - *logback-core* was upgraded to 1.2.9
  - *logback-classic* was upgraded to 1.2.9
  - *xercesImpl* was upgraded to 2.12.2
  - *xml-apis* was upgraded to 1.4.01
  - *snakeyaml* 1.29 was added as a dependency
5.8 6.9.1 (2022 April)

File format fixes and improvements:

- **DeltaVision**
  - allowed partial planes to be read from truncated files
- **MetaMorph**
  - fixed a NumberFormatException when parsing double values in metadata
- **OME-TIFF**
  - performance improvements to reduce the number of open file handles during initialization
- **PerkinElmer Operetta**
  - performance improvements to speed up the reading of datasets (thanks to Nicolas Chiaruttini)
- **Zeiss CZI**
  - switched to a white background for brightfield data to better match Zeiss Zen software
- **Zeiss LSM**
  - fixed a bug that resulted in an incorrect pixel type for some floating point data

Bio-Formats improvements:

- removed the maven deploy step from workflows in forked repositories
- upgraded cdm dependency from 4.6.13 to cdm-core 5.3.3
- configurable sleep time in FakeReader moved to the end of initialization

Documentation improvements:

- DICOM format page updated with improved links for sample datasets, software, and specification
- updated guidelines for submitting sample datasets

5.9 6.9.0 (2022 February)

New file formats:

- **Leica LOF**
  - added support for reading Leica LOF files. This functionality was implemented and contributed by Leica Microsystems
- **Leica XLEF**
  - added support for reading Leica XLEF files. This functionality was implemented and contributed by Leica Microsystems

File format fixes and improvements:

- **FEI TIFF**
  - fixed parsing of physical pixel sizes for Phenom data
- **Inspector OBF**
  - improved handling of deflate errors when opening older OBF files (thanks to Nils Gladitz)
Bio-Formats, Release 6.14.0

- **JPEG**
  - performance improvements to reduce memory required to read tiles from large JPEGs
- **Leica LIF**
  - improved parsing of channel metadata (thanks to Zach Marin)
- **Nikon NIS-Elements ND2**
  - improved parsing of metadata tables with invalid characters
- **OME-TIFF**
  - performance improvements of tile read speeds for some pyramid OME-TIFFs
- **PerkinElmer Operetta**
  - enabled support for handling sparse planes

**Bio-Formats improvements:**
- added new API methods to FormatTools for creating well names
- added a swap option to bfconvert to override input dimension order (thanks to Roberto Calabrese)

**Documentation improvements:**
- new public sample files for Leica XLEF (thanks to Leica Microsystems)
- added documentation for using the swap option with the command line tools
- updated the process for contributing sample files via Zenodo
- updated the link to NDP.view2 software on the Hamamatsu ndpi format page

### 5.10 6.8.1 (2022 January)

**File format fixes and improvements:**
- **Aperio SVS / Aperio AFI**
  - improved handling of macro and label images when no image description is present
- **cellSens VSI**
  - added support for physicalSizeZ metadata
- **Gatan Digital Micrograph DM4**
  - fixed a FormatException when encountering null values in the metadata

**Bio-Formats improvements:**
- fixed a bug in TiffSaver which could result in an invalid TIFF when saving a file with tiling and compression (thanks to Pete Bankhead)
- made performance improvements to TIFF reading and writing (thanks to Pete Bankhead)

**Security improvements:**
- Updated use of the git:// protocol in POM as it has now been deprecated. See the GitHub blog post for further details
- Removed the loci_tools from Bio-Formats builds due to log4j vulnerability, users should instead use bioformats_package. See the OME security advisory for further details
5.11 6.8.0 (2021 December)

New file formats:

- **DICOM**
  - added support for reading and writing DICOM whole slide images (DICOM WSI format). This functionality was implemented through collaboration with NCI Imaging Data Commons, and has been funded in whole or in part with Federal funds from the National Cancer Institute, National Institutes of Health, under Task Order No. HHSN26110071 under Contract No. HHSN2612015000031

- **Olympus omp2info**
  - Added a new reader for the Olympus tile format provided through a partnership with Glencoe Software and OLYMPUS EUROPA SE & Co. KG

File format fixes and improvements:

- **Amira Mesh**
  - added support for Amira 3.0 keys

- **Aperio SVS / Aperio AFI**
  - improved handling of macro and label images

- **AVI (Audio Video Interleave)**
  - set fps from Pixels Time Increment when writing AVI files

- **Nikon NIS-Elements ND2**
  - updated parsing of newer ND2 files to resolve issues with incorrect dimensions (thanks to Ilya Parmon)

- **Olympus OIR**
  - fixed a Null Pointer Exception for Laser Data ID

- **OME-TIFF**
  - fixed handling of partial datasets
  - added a new reader option ometiff.fail_on_missing_tiff to configure behaviour of partial dataset

- **PerkinElmer Operetta**
  - added support for Phenix v6 data

- **TIFF**
  - fixed non-sequential offset correction for TIFF files between 2 and 4 GB

- **Zeiss CZI**
  - added support for zstd compression. This feature will require FIJI users to additionally download the dependency io.airlift.aircompressor 0.18. (funded by a partnership between Glencoe Software and ZEISS)

Bio-Formats improvements:

- updated the ordering of reader.txt

- **bfconvert** tool updated to use multiple of tile size granularity when writing (thanks to Jeremy Muhlich)

- added a new no-sequential option to enable writing in non sequential order

Component updates:
• *jhdf5* was upgraded to 19.04.0
• *commons-lang* 2.6 was added as a dependency
• *aircompressor* 0.18 was added as a dependency

Documentation improvements:
• updated page for using Bio-Formats in Python to add links for AICSImageIO and PyImageJ (thanks to Curtis Rueden)

### 5.12 6.7.0 (2021 August)

File format fixes and improvements:

• **cellSens VSI**
  • corrected offset checking to prevent seeking beyond EOF

• **Deltavision**
  • fixed a bug which resulted in an IndexOutOfBoundsException

• **Hamamatsu ndpis**
  • added support for channel names from NDP Shading Data

• **Nikon NIS-Elements ND2**
  • improved parsing of timestamp values (thanks to Ilya Parmon)
  • improved parsing of channel names and colors

• **Olympus FluoView FV1000**
  • fixed a bug with the handling of filter indexes

• **OME-TIFF**
  • resolution annotations now removed when resolutions are flattened
  • fixed handling of filesets with BinaryOnly across multiple folders

• **PerkinElmer Operetta**
  • channel colors now populated using emission wavelength

• **TIFF**
  • added support for Deflate compressed tiles/strips with lsb2msb order
  • prioritised units from TIFF tag over those from ImageJ comment

• **Zeiss CZI**
  • plates will now be correctly detected
  • added a new reader option `zeissczi.relative_positions` which when set to true will change the PositionX and PositionY values stored in OME-XML to the pixel position instead of the absolute physical stage position
  • added a new reader option `zeissczi.trim_dimensions` which when set to true will use the pixel block metadata to trim XY dimensions to match those reported in ZEN

Bio-Formats improvements:
• the pattern reader now supports populating channel names from pattern tokens
• the channel filler will correctly reset bits per pixel if a lookup table is applied
• existing reader options are now registered in getAvailableOptions command
• improved performance of OME-XML validation (thanks to Nils Gladitz)
• added unit tests for the upgrade of custom attributes in OME-XML transforms

Documentation improvements:
• updated link to a 2005 publication about the OME data model
• bumped low level components logback-core and logback-classic to 1.2.0
• updated the licensing for BDV, KLB and CellH5 readers to BSD
• added documentation for additional reader options
• fixed a number of broken links

The below have been relicensed under the more permissive BSD-2 clause:
• Big Data Viewer reader
• Keller Lab Block reader
• CellH5 reader
• JHDF service
• Bio-Formats GNU Octave package
• Bio-Formats MATLAB functions

Component updates:
• ome-model was upgraded to 6.2.3
• ome-common was upgraded to 6.0.7
• ome-poi was upgraded to 5.3.4
• ome-codecs was upgraded to 0.3.1
• ome-metakit was upgraded to 5.3.3
• logback was upgraded to 1.2.0

5.13 6.6.1 (2021 March)

File format fixes and improvements:
• CV7000
  – fixed the channel-wise ordering of planes. The implementation changes to channel mapping will require existing memo files to be regenerated for CV7000 datasets
• Hamamatsu NDPIS
  – transmittance values will now be used to pick valid channels if no wavelength present
• Leica SCN
  – updated position units from reference frame to nm
  – original metadata now populated correctly for all series
• MetaMorph Stack
Bio-Formats, Release 6.14.0

- support added for parsing NDInfoFile Version to determine correct file suffix
- improved detection of companion binary files

• MetaXpress
  - improved detection of thumbnail paths

• Nikon NIS-Elements ND2
  - made performance improvements to reduce memory usage when reading large datasets

• Olympus OIR
  - fixed a bug which resulted in blank planes when XML blocks end with CRLF

Documentation improvements:
- updated bfconvert documentation for using pattern string on Windows (thanks to Nathanael Reveal)
- added a Fiji usage note to the Tecan Spark Cyto Workspace format page
- added references to the public Imaris IMS format specification page

Bio-Formats improvements:
- added support for ChannelName in Fake series tables

5.14 6.6.0 (2020 December)

New file formats:

• Tecan Spark Cyto Workspace
  - Added a new reader for Tecan Spark Cyto workspace files, provided through a collaboration between Tecan Trading and Glencoe Software

File format fixes and improvements:

• Applied Precision CellWorX / MetaXpress
  - support for CellWorX and MetaXpress has now been split into separate readers

• BD Pathway
  - plate row and column dimensions now being populated

• Bitmap
  - fixed offset calculation for files larger than 2 GB

• Cellomics
  - channel data now being parsed from companion .mdb file
  - corrected Plate/Well/Image mappings for sparse plates
  - plate size now calculated using the maximum row/column index
  - plate row and column dimensions now being populated

• Gatan Digital Micrograph DM4
  - fixed support for montages with single Z dimensions

• MetaMorph Stack
  - corrected laser indexes for multi series datasets
- **MIAS (Maia Scientific)**
  - plate row and column dimensions now being populated

- **Nikon NIS-Elements ND2**
  - fixed a null pointer exception when parsing metadata key value pairs

- **Olympus ScanR**
  - added functionality to handle missing wells through a new option `scanr.skip_missing_wells`. By default the option is set to true and missing wells are skipped

- **Olympus SIS TIFF**
  - corrected parsing of pixel size values (thanks to Stephan Wagner-Conrad)
  - removed trailing null byte from `imageName` and `channelName`
  - improved formatting of the image reader

- **PerkinElmer Opera Flex**
  - plate row and column dimensions now being populated

- **PerkinElmer Operetta**
  - updated metadata files logic to skip plate folders

- **PNG (Portable Network Graphics)**
  - fixed an issue which resulted in a hanging call to `openBytes`

- **Zeiss CZI**
  - scene number is now correctly padded and indexes begin at 1

- **Zeiss LSM**
  - improved handling of cached plane variables

**Documentation improvements:**
- added new OME-TIFF plate companion sample files
- references to image index in the API documentation have been updated to plane index
- updated various links to follow the LOCI site migration
- fixed broken Javadoc links
- removed outdated references to mailing lists

**Component updates:**
- `ome-model` was upgraded to 6.2.2
- `ome-common` was upgraded to 6.0.6
- Memoizer version has been incremented meaning previous memo files are invalidated and will be regenerated
- `kryo` dependency updated to 4.0.2
- Added new `sqlite-jdbc` version 3.28.0 dependency for the Tecan Spark Cyto Workspace format

**Bio-Formats improvements:**
- removed automatic file stitching from format reader tests
- improved stringency of `ChannelName` and `ImageName` testing
Bio-Formats, Release 6.14.0

- improved reader detection for image conversion testing
- reviewed all instances of whitelist/blacklist
- introduced support for GitHub Actions
- improved handling of DynamicMetadataOptions on Windows
- introduced the ability to set metadata options using a .bfoptions file
- test-suite updated to handle new .bfoptions file

5.15 6.5.1 (2020 July)

File format fixes and improvements:
- **Aperio SVS / Aperio AFI**
  - fixed a Null Pointer Exception when exposure time is not defined
- **Big Data Viewer**
  - corrected series indexes for non flattened multi resolution images
- **Cellomics**
  - physical sizes are now set for all series rather than just the first
- **Inspector OBF**
  - file format version and stack version are now recorded as part of global metadata
- **MetaMorph**
  - improved wavelength parsing using Metamorph XML or original metadata
- **Mikroscan TIFF**
  - stricter format recognition now used to prevent erroneous use of the reader
- **Ventana BIF**
  - added support for LEFT overlap direction (thanks to Joan Gibert)
- **Zeiss CZI**
  - fixed a bug to ensure Channel Illumination Type is not overridden by display settings

Documentation improvements:
- added link from OME-TIFF page to commercial partners page
- updated links for Biplane to now use Oxford Instruments
- fixed a number of broken hyperlinks in documentation

Component updates:
- **jxrlib** was upgraded to 0.2.4

Bio-Formats improvements:
- fixed a bug in bfconvert for multi-series files with varying image sizes
- removed the logging OMERO IDs passed to FormatReader and ImageReader
5.16 6.5.0 (2020 April)

File format fixes and improvements:

- **Big Data Viewer**
  - improved performance of tiled reading

- **DeltaVision**
  - implemented additional sanity check to header to ensure correct panel count

- **DICOM**
  - improved performance of initialization of multi-file datasets. Files spread across multiple directories are now handled by a DICOMDIR file that groups the dataset

- **Inspector OBF**
  - added support for OBF Version 6 stacks (thanks to Nils Gladitz)
  - added support for OBF Version 4 stack flush points (thanks to Nils Gladitz)

- **MetaMorph**
  - ensured dimension metadata read from tags and/or .nd file rather than allowing the underlying TIFF reader to treat each IFD as a series

- **MetaXpress**
  - added support for single site HCS variant

- **Zeiss CZI**
  - fixed the position count when only one position is present and the starting index is greater than 0

- **Various Readers**
  - reviewed and updated readers to prevent potential cases of integer overflow

OME-Model updates:

- version of **OME-Model** has been updated to 6.1.0
- added support for Python 3.8 and make code-generation Python 3 only (thanks to Roger Leigh)
- added getters and setters for **OME@Creator** attribute for ome.xml metadata interfaces and implementations (thanks to Nils Gladitz)
- removed unmaintained C++ OME-XML implementation (thanks to Roger Leigh)
- updated code-generation for building on Python 3.6, 3.7 (thanks to Roger Leigh)
- uncapped the Sphinx version for OME-Model documentation

Documentation improvements:

- fixed a number of broken links within the documentation
- added a new format page for MetaXpress
- added a reference to public OBF sample images in the format page
5.17 6.4.0 (2020 March)

File format fixes and improvements:

• **Applied Precision CellWorX**
  – added support for multiple Z sections

• **DeltaVision**
  – added and updated objective metadata based on values from softWoRx 7.2.0 (thanks to David Pinto)

• **Hamamatsu NDPI**
  – added support for JPEG-XR compression
  – added full support for files larger than 4 GB
  – improved support for a number of additional metadata tags
  – the Hamamatsu NDPI reader improvements are provided via work from Glencoe Software Inc.

• **InCell**
  – inverted Y coordinate in plane/field positions to correct stitching of tiles

• **PerkinElmer Vectra QPTIFF**
  – plane position values will now be populated on OME-XML

• **TIFF**
  – values for XPosition and YPosition in original metadata will now be more accurately stored as doubles
  – implemented a fix to prevent integer overflow when reading from a large tile greater than 2 GB

• **Ventana BIF**
  – improved handling of physical sizes for pre-stitched TIFFs

• **Zeiss CZI**
  – added a fix for uncompressed pixels incorrectly flagged as JPEG-XR
  – fixed a bug so that line-scans are now read correctly (thanks to Stephan Wagner-Conrad)
  – improved parsing of detector metadata

Bio-Formats tools improvements:

• added a new nobigtiff option to bfconvert to disable automatic switching to BigTiff based upon the number of pixel bytes (TIFF files larger than 4GB). This may be useful when converting using a compression codec so that the output file size is less than 4GB

• fixed a bug in xmlvalid tool to properly handle lowercasing of file names

• added new bfGetPlaneAtZCT function to MATLAB toolbox to retrieve a particular plane at a ZCT coordinate (thanks to Mark Kittisopikul)

• added a new bfTestInRange helper function to MATLAB toolbox with improved performance and error handling (thanks to Mark Kittisopikul)

• fixed a bug when using bfconvert on multi-series files with only a single timepoint, channel or Z slice selected

Bio-Formats API updates:

• version of jxrlib has been updated to 0.2.2
• version of ome-codecs has been updated to 0.3.0 which includes performance upgrades for LZW compression (thanks to Alexander Popiel)
• moved JPEG-XR codec and service from formats-gpl to formats-bsd component
• TiffParser and TiffSaver have now been updated to implement Closeable
• added a documentation note to use one IFD instance per plane with saveBytes in TiffWriter
• FormatWriter will now create output file’s parent directory if needed
• FakeReader now allows for DeltaT to be set in INI file
• FakeReader now handles INI files in plates created by mkfake
• fixed a number of deprecation warnings in various readers

5.18  6.3.1 (2019 December)

File format fixes and improvements:
• ICS (Image Cytometry Standard)
  – prevented a potential error when writing ICS files with physical units which could not be converted
• Inspector OBF
  – fixed a bug with incorrect dimensions being parsed for some Inspector OBF files
• Leica LAS AF LIF (Leica Image File Format)
  – fixed a NullPointerException in some variants of the LIF file format
• TIFF
  – improved the performance of tiled writing
• Zeiss CZI
  – fixed issues with tile stitching and position size

Bio-Formats tools improvements:
• fixed a potential NullPointerException in SpringUtilities for Bio-Formats plugins (thanks to July Chen)
• updated URL for fetching ImageJ upgrades in ijview
• fixed the XY coordinates for cropped images in bfconvert
• fixed a bug when using a cropped multi-series file in bfconvert (thanks to Matthieu Moisse)
• fixed issues in bfconvert when writing separate tiles with additional options
• added documentation of tile output patterns to utility help in bfconvert
5.19 6.3.0 (2019 October)

File format fixes and improvements:

- **Big Data Viewer**
  - added support for parsing of physical sizes
- **DeltaVision**
  - added a new RCPNL reader which is a variant and split out of the DeltaVision format
- **Hamamatsu NDPI**
  - fixed population of the nominal magnification from the SourceLens TIFF tag
- **ICS (Image Cytometry Standard)**
  - fixed a bug when using tiles to read files from SVI-Huygens
- **Inspector OBF**
  - fixed an `IndexOutOfBoundsException` exception when using `DummyMetadata` (thanks to Nils Gladitz)
- **JPEG 2000**
  - added support for sub-resolutions
- **Leica LIF**
  - updated to parse attachments to determine if XY positions should be flipped or swapped
- **MetaMorph**
  - improved file name construction and plate detection logic
- **Nikon ND2**
  - updated to use floating point for 32 bit values
- **OME-TIFF**
  - reduced memory usage when reading files and memo file size for cached files
- **PerkinElmer Operetta**
  - improved handling of empty fields to prevent series from having X or Y set to 0

API updates:

- added `overwriteIFDValue` signature that takes an IFD offset to `TiffSaver`
- added a new `getRequiredDirectories` method to `FormatTools`
- new `FakeReader` keys added for `sleepOpenBytes` and `sleepInitFile`

Build updates:

- updated deployment mechanism for SNAPSHOT and Release to use Travis CI
- increased the strictness of `AcquisitionDate` checks in `FormatReaderTest`
- improved test coverage of companion file datasets

Component updates:

- `ome-common` was upgraded to 6.0.4
- updated `DateTools` to attempt to parse invalid dates with `Locale.US`
**Documentation improvements:**

- added documentation for sleep options when generating test images
- fixed broken external links in documentation
- corrected suffixes used for JPEG 2000

### 5.20 6.2.1 (2019 August)

**File format fixes and improvements:**

- **Applied Precision CellWorX**
  - corrected plane positions for series index > 0 rather than reusing positions from the first series
- **DeltaVision**
  - added objective info for new Applied Precision 100X/1.4 lensID
  - updated so that date from dv file will override log file date to avoid locale-dependent dates
- **Leica LAS AF LIF (Leica Image File Format)**
  - fixed units and indexing for tile-based plane positions
- **TIFF**
  - fixed a potential exception in MinimalTiffReader when the TIFF is stored using very large tile/strip dimensions
- **Zeiss CZI**
  - fixed a potential index out of bounds exception when populating positions

**Bug fixes and improvements:**

- **bfconvert** has been updated so that when the dimensions of a sub-resolution are smaller than the requested tile size then they default to the size of the sub-resolution
- fixed a bug in the execution of **bfsave** in the GNU Octave environment

**Codec updates:**

- **ome-codecs** was upgraded to 0.2.5
- JPEG codec updated to reduce decompression time for 8-bit RGB images
- Huffman codec updated to allow the decoding tree to go all the way down to the 16-bit depth required by the standard (thanks to Aaron Avery)
- Lossless JPEG codec updated to provide better compliance with the LJPEG standard (thanks to Aaron Avery)

**Documentation improvements:**

- added instructions for building Bio-Formats with IntelliJ IDEA
- corrected command-line tools documentation for the novalid and noncore options
- updated broken links to Barre’s Medical Imaging Samples
- updated the imagej.net link for Zeiss LSM toolbox plugin
- added links to public sample files for the following formats:
New file formats:

- **Mikroscan TIFF**
  - a new reader for Mikroscan TIFF files has been contributed with thanks to Jim Crowe, Mikroscan Technologies, Inc.

- **Ventana BIF**
  - added a new reader for Ventana BIF files which has been commissioned via Glencoe Software

File format fixes and improvements:

- **Cellomics**
  - fixed indexing for plates with a single well or missing fields

- **DeltaVision**
  - added support for the reading of the new panel count field (provided through a collaboration between GE Healthcare and Glencoe Software Inc.)

- **PerkinElmer Operetta**
  - images with smaller XY dimensions than all other TIFF files in dataset will now be padded

- **TIFF**
  - updated functionality for overwriting IFD values to ensure that previous value is completely overwritten and no orphaned tags are left

- **Zeiss CZI**
  - expanded support for auto-stitching of tiles

Bug fixes and improvements:

- added `-cache`, `-cache-dir` and `-no-sas` options to `bfconvert` tool
- deprecated broken TRUNK and DAILY builds from upgrade checker
- disabled Oracle JDK from Travis CI checks

Documentation improvements:
• fixed broken link for discontinued Dcraw software
• updated links for Zeiss formats

5.22 6.1.1 (2019 June)

File format fixes and improvements:

• **DeltaVision**
  – added new lens definitions associated with *rcpnl* files

• **Gatan Digital Micrograph (DM3/DM4)**
  – now parsing the *Montage* tag to determine if tiles are present

• **Leica LAS AF LIF (Leica Image File Format)**
  – added fix to correctly read scale from polygon regions of interest (thanks to Sean Warren)

• **PerkinElmer Columbus**
  – improved handling of truncated TIFF files to return blank planes

• **PerkinElmer Opera Flex**
  – plate barcodes are now used to improve grouping and handling of truncated files

• **TIFF (Tagged Image File Format)**
  – improved parsing times for images stored as uncompressed contiguous strips

• **Zeiss CZI**
  – improved plane position metadata for many CZI datasets

Automated test changes:
• format reader tests have been updated to handle PerkinElmer Columbus datasets with flex files

Documentation improvements:
• added help for missing options in *bfconvert* command line tool

5.23 6.1.0 (2019 May)

New file formats:
• **BDV**
  – added a new reader for Big Data Viewer files

File format fixes and improvements:

• **Applied Precision CellWorX**
  – improved handling of thumbnail files

• **DeltaVision**
  – updated handling of *rcpnl* files to treat each file as a single timepoint

• **FakeReader**
  – removed *header* key from original metadata
• **Hamamatsu VMS**
  – removed *header key* from original metadata
• **Hitachi S-4800**
  – removed *header key* from original metadata
• **ICS (Image Cytometry Standard)**
  – fixed an issue reading .ics/.ids files written by SVI Huygens (thanks to Jan Eglinger)
• **Imaris IMS**
  – fixed issues with newer files which had been failing due to older *netcdf* version
• **JPEG**
  – improved the reading of EXIF data
• **Lambert Instruments FLIM**
  – added support for packed UINT12 datatype (thanks to Johan Herz)
• **LEO**
  – fixed a bug with the parsing of physical sizes
  – improved support for additional global metadata fields
• **Olympus OIR**
  – fixed a bug which would show empty pixels when more than 1000 timepoints

Automated test changes:
• added additional tests for HCS/SPW datasets to ensure Plate, PlateAcquisition, Well, WellSample, and Well-Sample position values are configured where present
• added a new *file-leak-detector* test to flag potential memory leaks

Bio-Formats API changes:
• **ImageConverter** as used in *bfconvert* command line tool is now public
• made **ImageReader** more defensive against exceptions thrown when determining reader type
• fixed an issue when performing a non-sequential write for multi-resolution TIFF files

Component changes:
• **ome-common** was upgraded to 6.0.3
• **perf4j** was upgraded to 0.9.16
• removed *Guava* dependency which will be pulled transitively from the upstream *ome-common* dependency
• **jhdf5** was upgraded to 14.12.6
• **metadata-extractor** was upgraded to 2.11.0
• **xercesImpl** version 2.8.1 was added as it is no longer a dependency of **metadata-extractor**
• **netcdf** was upgraded to 4.6.13
5.24 6.0.1 (2019 March)

File format fixes and improvements:

• **cellSens VSI**
  - improved tag parsing resulting in fixes for missing or incorrect metadata

• **Hamamatsu ndpi**
  - improved handling of variants where a constituent NDPI has no wavelength

• **LaVision Imspector**
  - fixed a potential NullPointerException when ‘xyz-Table Z Resolution’ is false

• **NRRD (Nearly Raw Raster Data)**
  - added support for raw GZIP-compressed data files

• **Olympus OIR**
  - fix to ensure file path is normalized which fixes detection on Windows

• **TIFF**
  - improved handling of direct tile copying to prevent invalid images
  - improved handling of tiles in scenarios of an invalid offset or byte count of 0

Documentation improvements:

• added documentation for -noflat option to the showinf and bfconvert users pages
• updated recommended minimal MATLAB version to R2017b
• documented support for MATLAB versions prior to R2017b
• links to MicroCT public datasets now point to the public archive rather than directly to the zip file

5.25 6.0.0 (2019 February)

Bio-Formats API changes:

• **Java 8** is now the minimum supported version

• **Sub-resolution reading:**
  - added MetadataList and CoreMetadataList classes
  - added a new SubResolutionFormatReader abstract class for handling pyramidal format readers
  - updated all pyramid format readers to use SubResolutionFormatReader
  - deprecated getCoreMetadataList, seriesToCoreIndex, coreIndexToSeries, getCoreIndex and setCoreIndex in IFormatWriter

• **Added a new IPyramidHandler interface with the resolution getter methods**

• **Sub-resolution writing changes:**
  - IFormatWriter now extends IPyramidHandler (breaking)
  - added setResolutions and getResolutions methods to IFormatWriter (breaking)
  - added examples of using the sub-resolution writing API
**Bio-Formats, Release 6.14.0**

- Tiled writing API changes:
  - updated `IFormatWriter` to use `setTileSizeX(0)` and `setTileSizeY(0)` as a way to disable tiling (breaking)
  - updated `FormatWriter` to set 0 as the default values of `getTileSizeX()` and `getTileSizeY` (breaking)
- `IFormatWriter.getCompressionTypes` now returns the types for the selected writer only
- Metadata handling:
  - added getter methods to `MetadataTools` for retrieving OME enumerations by value
  - deprecated OME enumeration getter methods in `FormatReader`
- Refactor `FilePatternReader` logic in a new `WrappedReader` abstract class

**New file formats:**
- **KLB**
  - added a new reader for Keller Lab Block (KLB) files
- **CV7000**
  - added a new reader for Yokogawa CV7000 datasets
- **GE MicroCT**
  - added a new reader for GE MicroCT datasets

**File format fixes and improvements:**
- **Aperio SVS/AFI**
  - removed pyramidal resolutions of mismatching pixel types
  - fixed exposure times, improved image naming of AFI datasets
  - displayed original metadata keys for each channel of AFI datasets
  - added support for multiple Z sections
- **DICOM**
  - improved file grouping and file-to-series mapping for multi-file datasets
- **Fake**
  - added support for multi-resolution test images
  - now populating WellSample positions when present using Plane data
- **Gatan Digital Micrograph**
  - adjusted endianness and record byte count for long values
  - allowed ROIs to be stored in `DocumentObjectList` groups
  - no longer creating an empty ROI when an unsupported shape type is encountered
- **Image Pro**
  - added support for Image Pro Plus .ips set
- **GE InCell**
  - added support for parsing minimum and maximum pixel values
- **Lambert Instruments FLIM**
– fixed an integer overflow error with large files (thanks to Rolf Harkes)

• Leica LIF
  – unified metadata parsing to use DataTools.parseDouble

• Leica SCN
  – improved support for Versa datasets

• Micro-Manager
  – improved handling of very large *_metadata.txt files
  – prevented NumberFormatException for invalid double values
  – add support for parsing ChannelColor from *_metadata.txt files

• Metamorph
  – added support for multi-dimensional .scan dataset created from Scan Slide (thanks to Jeremy Muhlich)

• MRC (Medical Research Council)
  – fixed endian detection for old-style headers

• Nikon ND2
  – prevented integer overflow when reading chunkmaps from files larger than 2GB
  – fixed handling of duplicate and incomplete exposure time lists
  – fixed chunk map handling when CustomData blocks are between ImageDataSeqs

• OME-TIFF
  – added support for reading OME-TIFF with pyramidal resolutions stored as SubIFDs
  – added support for writing OME-TIFF with pyramidal resolutions
  – added support for companion OME-TIFF filesets where TIFF does not link back to the metadata file
  – improved handling of missing planes in TiffData

• PerkinElmer Operetta
  – improved support to handle datasets generated by the Harmony software

• TIFF
  – split IFDs into separate series if the dimensions or pixel type mismatch
  – restricted use case for legacy TIFF JAI reader
  – fixed a bug with FillOrder which resulted in 0 pixel values

• Zeiss CZI
  – reduced duplicate original metadata when reading a pyramid file

• Zeiss TIFF
  – added support for AVI files acquired with Keyence software

• Zeiss ZVI
  – reuse stream for sequential calls to openBytes on the same plane

• updated all pyramidal format readers to consume SubResolutionReader
• updated all readers to consume MetadataTools getter to retrieve enumerations
• reviewed all readers and plugins to close open instances of RandomAccessInputStream
• fixed some deprecation warnings in a number of readers
• for RGB images using ChannelSeparator all channel metadata is now copied instead of just names

ImageJ plugin improvements:
• updated the updater message in the Fiji plugin (thanks to Jan Eglinger)
• disabled LUT writing for any plane that has a default grayscale lookup table
• added macro option to always skip LUT writing

MATLAB toolbox improvements:
• improved performance of bfGetPlane by removing an unnecessary data copy (thanks to Cris Luengo)

Command-line tools improvements:
• bfconvert utility
  – added -no-flat option to the command-line tools to convert files with sub-resolutions
  – added -pyramid-scale and -pyramid-resolutions options to generate sub-resolutions during conversion
  – removed Plate elements when -series is passed as an option
  – extended usage to describe available formats, extensions and compressions
• xmlvalid utility
  – added new validate methods to loci.formats.tools.XMLValidate returning the validation status
  – added a return code to xmlvalid

Component changes:
• ome-common was upgraded to 6.0.0
• ome-codecs was upgraded to 0.2.3
• ome-model was upgraded to 6.0.0

Automated test changes:
• added testng.allow-missing property allowing to skip unconfigured filesets
• added testUnflattenedSaneOMEXML to compare series count to OME-XML images count when resolution flattening is disabled
• added test-equivalent target to compare pixel data between two files
• added support for storing resolution index and resolution count in the configuration files used for automated testing
• tests now fail when a configured file throws UnknownFormatException

Documentation improvements:
• fixed the xmlvalid documentation page (thanks to Kouichi C. Nakamura)
• improved the memory section of the MATLAB documentation page (thanks to Kouichi C. Nakamura)
• extended IFormatReader Javadocs to reflect the reader guide
• added reference to current Adobe TIFF specification
• switched to image.sc as the reference location for public feedback
5.26 5.9.2 (2018 September 03)

File format fixes and improvements:

- **AVI**
  - added support for AVI files acquired with Keyence software

- **Gatan**
  - fixed a bug when reading a file with an empty tag of type 23

- **Deltavision**
  - extended the objective metadata support (thanks to David Pinto)

- **MRC**
  - fixed the reading of MRC files generated with FEI EPU software

- **Zeiss LSM**
  - improved the channel color detection for SIM data

Component changes:

All OME dependencies were upgraded mostly with build changes and documentation improvements:

- ome-common was upgraded from 5.3.2 to 5.3.6
- ome-poi was upgraded from 5.3.1 to 5.3.3
- ome-mdbtools was upgraded from 5.3.1 to 5.3.3
- ome-jai was upgraded from 0.1.0 to 0.1.3
- ome-codecs was upgraded from 0.2.0 to 0.2.2
- ome-stubs was upgraded from 5.3.0 to 5.3.2
- ome-model was upgraded from 5.5.4 to 5.6.3

Documentation improvements:

- added links to public sample files for Imaris IMS, DICOM, Leica-SCN, LEO, MRC, PNG, TIFF and Trestle formats

5.27 5.9.1 (2018 August 14)

File format fixes and improvements:

- **Olympus OIR**
  - fixed a bug to prevent incorrect files from being read when multiple datasets are in the same location

- **LEO**
  - updated parsing of metadata values for image pixel size, working distance, filament, EHT and date (thanks to David Mankus)

- **DeltaVision**
  - reader can now detect up to 12 channels

- **Micro-Manager**
– now logs a warning when an image is acquired with an unsupported version

Documentation improvements:
• added QuPath to the list of visualization and analysis applications
• updated the link to the i3dcore library
• updated the link to Slidebook
• improved MATLAB documentation with information on Java heap memory preferences (thanks to Kouichi C. Nakamura)
• corrected a number of permanently redirected URLs in the component and format pages

5.28 5.9.0 (2018 July 3)

File format fixes and improvements:
• MetaMorph
  – fixed a NullPointerException when a stage label is not present
  – ensured that reported domain is now consistent with the existence of a Plate in OME-XML
  – fixed Metamorph RGB series channel count (thanks to Jeremy Muhlich)
• Leica LIF
  – improved handling of dimension order for non-RGB channels
• Inspector OBF
  – added support for FLIM datasets
• Inveon
  – updated to attempt to locate renamed data files
• Volocity
  – expanded image names to include the stack parent names
• Olympus OIR
  – added a fix for slow tag reading and a potential infinite loop
• TIFF
  – added support in TiffWriter for the writing of DEFLATE (zlib) compression
  – deprecated getIFDs() in TiffParser and added getMainIFDs() and getSubIFDs()
• Zeiss CZI
  – fixed an issue with big images when tiling is present but a pyramid is not
• Nikon NIS-Elements ND2
  – prevented integer overflow exception when reading a tile from a large image
• Amersham Biosciences Gel
  – prevented overflow issue when reading unsigned integer values
• Cellomics
  – fixed indexing when the field counts are variable
Bio-Formats, Release 6.14.0

- **Trestle**
  - updated to ensure consistent ordering of used files

Bug fixes and improvements:
- enabled building and testing with Java 9 and 10
- added CI testing with Java 10 on AppVeyor and Travis
- removed Java 7 from Appveyor matrix
- updated a number of Maven plugins to current versions
- corrected warnings in Maven configuration in sub-components
- added a warning to clarify the behavior when passing metadata with `dimensionOrder` in `bfsave` as part of the Bio-Formats MATLAB toolbox (thanks to Jonathan Armond)
- improved robustness in the detection of patterns as part of the file stitching
- fixed a bug relating to dimension order in the Bio-Formats plugins Exporter
- fixed download URLs in Bio-Formats command-line tools
- updated use of `static final` to match Oracle’s recommendations and convention
- disabled upgrade checker when running unit tests
- added support to data repo test suite for unconfigured tests

Documentation improvements:
- fixed unstable links flagged by automated link checking
- begun adding testing for breakages to memo files
- clarified ordering expectation in `getUsedFiles` Javadocs
- added documentation for `dimensionOrder` in `bfsave` with the MATLAB toolbox
- fixed broken links in previous release notes
- expanded documentation for command-line tools to cover undocumented options and environment variables
- added a new license/copyright section to the About Bio-Formats page
- updated the public format page for the Vectra QPTIFF format

### 5.29 5.8.2 (2018 April 23)

File format fixes and improvements:
- **JPEG**
  - large images with no restart markers now revert to using `DefaultJPEGReader` for improved decoding
- **Micro-Manager**
  - when available `PositionName` will be parsed and used as the image name
- **Hamamatsu ndpi**
  - updated image names to be more meaningful when resolutions are not flattened
- **InCell 2000/6000**
  - fixed an `IllegalArgumentException` and improved well and field indexing
• **AVI**
  – fixed a bug with padding for RGB images

• **NIfTI**
  – the nDimensions field is now used to read additional dimensions when size is greater than 4

• **PerkinElmer Opera Flex**
  – fixed a bug which resulted in an incorrect field count

• **Zeiss CZI**
  – improved handling of files with no extension

### Bug fixes and improvements:

• an error message is now logged by ImageReader when finding a reader for an empty file
• added a new protected helper method to Memoizer to check if a directory is writable
• improved the rounding of PlanePosition values for data repo configuration testing
• prevented a null pointer exception when retrieving plane exposure time using Bio-Formats ImageJ macro extensions
• updated MinMaxCalculator to account for unflattened multi resolution images

### Documentation improvements:

• decoupled the Bio-Formats documentation to the new ome/bio-formats-documentation GitHub repository
• updated Adding format/reader documentation for the new decoupled workflow
• improved link checking in automated builds

## 5.30 5.8.1 (2018 March 22)

### File format fixes and improvements:

• **TIFF**
  – updated TiffWriter so that planes will no longer be split when using non-standard SamplesPerPixel e.g. images with 2 or 4 samples per pixel. This will ensure the TiffData elements represent the structure specified by the user. If users wish to split planes the ChannelSeparator and bfconvert provide the means to do this explicitly
  – updated TiffWriter to use the correct logic for index checking when writing tiled images
  – fixed a ClassCastException when the NEW_SUBFILE_TYPE tag has a non-standard type or count such that the value is not inlined
  – updated to also check the last IFD for an ImageJ comment in the scenario that the image has been processed by other software

• **NRRD (Nearly Raw Raster Data)**
  – added support for space directions and space units fields added in version 4

• **Evotec/PerkinElmer Opera Flex**
  – updated to read rather than calculate image offsets when a single tile is used

### Bug fixes and improvements:
• limited the number of exceptions in the Bio-Formats plugins exporter when an unsupported pixel type is found
• fake test images now allow for per-plane ExposureTime{X,Y,Z} and Position{X,Y,Z} keys in the INI file (for further details see the documentation for Generating test images)
• file patterns now have expanded support for multi-channel pyramids, allowing for the matching of at least two channels rather than three, and the stitching of files containing a pyramid has also been fixed

Documentation improvements:
• improved testing of external links

5.31 5.8.0 (2018 February 21)

New file formats:
• Ionpath MIBI
  – added a new reader to support the reading of Ionpath Multiplexed Ion Beam Imaging (MIBI) files (thanks to Rachel Finck)
• PerkinElmer Vectra QPTIFF
  – added support for PerkinElmer Vectra QPTIFF files (The QPTIFF Bio-Formats reader is provided through a collaboration between PerkinElmer, Inc and Glencoe Software Inc.)

File format fixes and improvements:
• cellSens VSI
  – added support for lossless JPEG compression
• Imspector OBF
  – improved the parsing of OBF files with embedded OME-XML metadata (thanks to Bjoern Thiel)
• Leica LIF
  – companion metadata files are now attached if present
• Micro-Manager
  – fixed a bug related to the parsing of the metadata closing block
• NRRD (Nearly Raw Raster Data)
  – added support for GZIP pixel stream contained within a .nrrd file
• Olympus OIR
  – added support for multi-file datasets
• OME-TIFF
  – when files are ungrouped the dimensions are corrected by checking the indexes for each associated TiffData
• PerkinElmer Operetta
  – added support for additional metadata fields such as Instrument, Wavelength and Exposure time
• TIFF
  – fixed a bug when printing IFD values of type OnDemandLongArray
  – fixed a bug when writing tile sizes for multi-series images
• **Zeiss CZI**
  – when Z positions are not enumerated then values are calculated from a Z step
  – metadata for DisplaySetting will now be preserved in the original metadata table

Bug fixes and improvements:
  • removed unused ScreenReader in preparation for migrating it to be an external reader
  • fixed a bug with the generation of thumbnails in Bio-Formats plugins
  • updated the Maven POM to unify component version property names
  • tile size is now reported in the core metadata when using the showinf tool
  • added `setFilePatternIds` to `ImporterOptions` for use with Bio-Formats plugins
  • improved the precision of format identification for MRC, I2I, and Zeiss LSM

Documentation improvements:
  • fixed and updated a number of external documentation links
  • added links to public NRRD samples

5.32 5.7.3 (2018 January 11)

File format fixes and improvements:
  • **TIFF**
    – fixed a NullPointerException when reading a TIFF file from the root system directory
    – improved support for large images that are stored as a single uncompressed tile with multiple interleaved channels
  • **MRC (Medical Research Council)**
    – added support in original metadata for the fields ISPG and Is data cube
  • **TillPhotonics TillVision**
    – directory listings for .pst files are now sorted
  • **MetaMorph**
    – directory listings are now sorted during file initialization
  • **Amira Mesh**
    – now supports Avizo in the file header in addition to the existing support for AmiraMesh
  • **Becker & Hickl SPCLImage**
    – added a fix for IllegalArgumentException when reading files with compressed data
  • **Zeiss CZI**
    – fixed an IndexOutOfBoundsException when creating ROI objects

Bug fixes and improvements:
  • removed unused target utils-formats-api from ant build
  • automated Memoizer tests updated to use UUID for generating unique memo file directories
  • detect and fix Findbugs’ SBSC_USE_STRINGBUFFER_CONCATENATION using StringBuilder
• configuration files for the automated test suite now use raw physical size rather than formatted size
• added first version of Dockerfile for running the automated test suite standalone

Documentation improvements:
• added a support page to the Bio-Formats project
• updated reference URLs for the Aperio ImageScope and Micro-Manager
• documented issues with conflicts in the *JAI ImageIO component*
• clarified the default values of HCS keys for fake images in the documentation for *Generating test images*
• corrected external links which failed automatic link checking

5.33 5.7.2 (2017 November 21)

File format fixes and improvements:
• Nikon ND2 - fixed a bug which would use the incorrect channel count for small-sized single channel images
• **MetaMorph TIFF**
  – changed the reader’s behaviour to populate exposure times for all planes when only a single exposure time is defined
• **DeltaVision**
  – improved parsing of the associated log files to add additional key value pairs to global metadata
• **EPS (Encapsulated PostScript)**
  – fixed an exception when reading pixel data in cases with embedded TIFF
• **GIF**
  – fixed a bug to display the correct data when reading planes out of order

Bug fixes and improvements:
• fixed failures with Ant build from a clean Maven repository by updating Maven repositories to use HTTPS rather than HTTP
• now using safe version checking for Bio-Formats plugins to prevent a bug with Java 9
• updated the JPEG-XR codec to allow either interleaved or non-interleaved data to be returned

Documentation improvements:
• added clarification regarding Bio-Formats version requirements for using Java 7 or above
• updated download links to latest Bio-Formats release version
• updated the link to the most active fork of JAI ImageIO
• fixed a number of external broken links
• added a Trello link for contributing external developers
• added a link to the page *Adding format/reader documentation pages* to help those contributing to the documentation or supported formats pages
• the *Bio-Rad Gel* page has been updated to add a link to biorad1sc_reader, an external python implementation (thanks to Matthew Clapp)
5.34 5.7.1 (2017 September 20)

File format fixes and improvements:
- **Nikon NIS-Elements ND2**
  - improved parsing of Z position values
- **LaVision Impector**
  - corrected the value of time per FLIM channel
  - fixed a bug which saw the Z and T dimensions swapped
  - fixed a divide by zero exception
  - added a fix for incorrect time-base and number of channels
- **TIFF**
  - added support for handling files with a FillOrder of 2 in which the bits in each byte are reversed
  - improved support for multi-channel ImageJ TIFF files greater than 4GB in size

Performance improvements:
- improved TIFF performance by using non-regexp String replacement (thanks to Thushara Wijeratna)
- improved TIFF handling of Strings for large metadata (thanks to T. Alexander Popiel)

Documentation improvements:
- updated documentation to reference support for ImageJ TIFFs
- added links to format options page to user and developer index pages

---

5.35 5.7.0 (2017 September 4)

File format fixes and improvements:
- **Imaris HDF**
  - fixed resolution problems in which dimensions and resolution order were incorrectly calculated (thanks to Eliana Andreica)
- **Nikon NIS-Elements ND2**
  - fixed a bug in offset calculation when native chunk map is being used
- **MetaMorph**
  - corrected delta T and position Z values for multi-channel images when channels are split across multiple files
- **Amnis FlowSight**
  - better handling of exceptions in isThisType method (thanks to Claire McQuin)
- **PicoQuant Bin**
  - better handling of exceptions in isThisType method (thanks to Claire McQuin)

Bug fixes and improvements:
- reviewed and corrected URLs throughout the Bio-Formats source code
• updated Bio-Formats Macro Extensions list with a missing function
• added a new option in Bio-Formats plugins to configure the slice label display using patterns

Documentation improvements:
• added new format page for **OMERO Pyramid**
• updated the developer page for **Working with whole slide images**
• added new page for configuring options in **Bio-Formats plugins**
• updated documentation sidebar to enable navigation of previous versions

### 5.36 5.6.0 (2017 August 14)

File format fixes and improvements:

- **Zeiss CZI**
  - added support for images from Elyra PALM system
  - prevented a potential infinite loop when a scene with a pyramid is missing
- **cellSens VSI**
  - a new option has been added to throw an exception rather than logging a warning if .ets file is missing. The option, `cellsens.fail_on_missing_ets`, can be used via the MetadataOptions API, as a parameter in the command line tools or via the Bio-Formats configuration dialog in ImageJ
- **MetaMorph Stack (STK)**
  - fixed an error with HCS style datasets always returning the first plane regardless of the requested index
  - updated to use stage labels starting with `Scan` to detect when a whole plate is saved in a single .stk file
  - fixed a bug for `ArrayIndexOutOfBoundsException` when an image contains a single Z plane
- **Gatan Digital Micrograph**
  - added support for Z stacks and ROIs
  - fixed several bugs in tag parsing
- **PerkinElmer Operetta**
  - ensure TIFF files exist before reading
- **JPEG**
  - support added for images with more than `Integer.MAX_VALUE` pixels

Bug fixes and improvements:

- **JPEGTileDecoder**
  - class now implements AutoCloseable to prevent resource leaks
- **Bio-Formats Plugin**
  - improved performance when using options to concatenate multiple series together
- **TiffSaver**
  - made performance improvements to prevent the writing of a new IFD for each tile, resulting in significant file size reductions for images with a large quantity of tiles
Documentation improvements:

- updated website and URL links for new OME Website website
- added missing Andor SIF to supported formats page
- added a new page Working with whole slide images outlining the API support for pyramids/resolutions
- fixed broken documentation links for external resources which are no longer available
- updated the style of Sphinx documentation

Component architecture changes/decoupling:

- decoupled image encoding and decoding routines to the new ome/ome-codecs GitHub repository and consumed as ‘org.openmicroscopy:ome-codecs’ artifact from Maven Central
- removed components/forks/jai - decoupled to the new ome/ome-jai GitHub repository and consumed as part of ‘org.openmicroscopy:ome-jai’ artifact from Maven Central
- replaced components/formats-api/codecs classes with wrappers around ‘org.openmicroscopy:ome-codecs’
- replaced components/formats-bsd/codecs classes with wrappers around ‘org.openmicroscopy:ome-codecs’

Updated build system:

- ant now removes the build files of the bundles during ‘clean’ to prevent a mix of dependencies

5.37 5.5.3 (2017 July 5)

File format fixes and improvements:

- **Zeiss CZI**
  - fix to store Bézier ROIs as polygons, using the control points for the set of Bézier curves to form an approximation of the ROI
  - improved parsing of stage positions in metadata
  - improved parsing of detector gain values
  - removed OME-XML validation errors by fixing potential for duplicate detector IDs
  - removed invalid XML failures for Modulo label elements
  - time increment metadata now populated on Pixels element
  - fix to deal with consecutive empty planes in a series (thanks to Nicholas Trahearn)

- **DICOM**
  - no longer allow core metadata to be modified when determining if files belong to a DICOM dataset

- **Nikon NIS-Elements ND2**
  - fixed calculation for scanline padding

- **Kodak BIP**
  - stricter file type checking enforced by no longer relying only on the file suffix

- **MINC MRI**
  - improved parsing of metadata by correcting units for physical sizes, pixel type and capturing XYZ plane positions in OME-XML

- **Bio-Rad Gel**
– fixed the width of pixel data offset field

• DeltaVision
  – improved accuracy of format detection checking for input streams

• Andor SIF
  – fixed support for cropped images by parsing bounding box of the stored image

Documentation improvements:
  • Olympus cellSens VSI updated to include list of available specifications

5.38 5.5.2 (2017 June 15)

File format fixes and improvements:

• Olympus FluoView FV1000
  – fix for java.lang.ArrayIndexOutOfBoundsException caused by filter names of “—” (thanks to Stefan Helfrich)
  – refactored channel metadata population and increased usage of DataTools utility functions

• Zeiss CZI
  – fixed detection of Z line scans that caused incorrect dimensions in certain filesets
  – improved exception handling of truncated/invalid files

• Veeco AFM
  – fixed reading of tiled images

• Hamamatsu ndpi
  – prevented potential memory leak by ensuring all TiffParser streams are closed

Bug fixes:

• OMEXMLServiceImpl
  – improved exception handling to deal with potential java.lang.NullPointerException when unable to locate OME-XML version while attempting to transform to the latest version

Documentation improvements:
  • updated documentation to be compatible with the latest version of Sphinx 1.6
  • fixed the usage/references of the option markup in documentation
  • fixed the table in the Micro-Manager user page
  • updated metadata ratings for supported formats

Updated build system:

• OME-Model version bump
  – the ome-model component has been updated to 5.5.4 which includes improvements to performance, documentation and the C++ model implementation
5.39 5.5.1 (2017 May 25)

File format improvements:

- **CellH5**
  - fix for HDF5SymbolTableException when recycling an IFormatReader to reopen another CellH5 file
  - bug fix related to opening of subsets of CellH5 files, namely openBytes(r, no, x, y, w, h) for y>0
- **Zeiss CZI**
  - fix pyramid resolution indexing for pyramids of different depths
  - fix for incorrect channel names and colors
- **Zeiss AxioVision ZVI**
  - correct parsing of epoch for Zeiss TIFF and Zeiss ZVI

Bug fixes:

- **Command line tools**
  - fix for java.lang. NegativeArraySizeException caused by incorrect dimensions when using showinf via command line with options set to autoscale and crop
- **Format tools**
  - fix for java.lang. IndexOutOfBoundsException when using getFilename with an image containing multiple samples per pixel channels and a single effective channel

Updated build system:

- **Autogen jobs**
  - fix for gen-meta-support to locate available org.openmicroscopy:ome-xml sources from the Maven repository following the decoupling of the model components
- **FileHandleTest**
  - exclude JHDF5 native libraries from FileHandleTest to enable CellH5 files to be included in daily tests

Documentation improvements:

- added a new example file for reading and writing of XZ and YZ orthogonal planes

5.40 5.5.0 (2017 May 8)

New file formats:

- **Olympus OIR**
  - added support for Olympus .oir data (funded by a partnership between Glencoe Software and OLYMPUS EUROPA SE & Co. KG)
- **PerkinElmer Columbus**
  - added support for PerkinElmer Columbus data

File format improvements:

- **Andor Bio-Imaging Division (ABD) TIFF**
– fixed acquisition date format from MM/dd/yyyy to dd/MM/yyyy

• Nikon NIS-Elements ND2
  – corrected logic used to determine PixelType by parsing uiBpc tags

• Hamamatsu ndpi
  – improved handling of channels in NDPIS datasets (thanks to Manuel Stritt)

• Inspector OBF
  – fix for SAXParseException when description field in metadata is empty

Documentation improvements:
• added links to public sample files for Cellomics
• added links to public sample files for InCell 3000

5.41 5.4.1 (2017 April 13)

File format improvements:
• MIAS (Maia Scientific)
  – added a fix for a possible exception when image files are not found under channel-specific subdirectories

• BD Pathway
  – added fix to check if Experiment.exp is a directory or an experiment file

• Inspector OBF
  – enabled forward compatibility for future versions, as the OBF format is backwards compatible (thanks to Bjoern Thiel)

Documentation improvements:
• updated external homepage link for FocalPoint
• removed Imago from list of visualization and analysis applications as it is no longer available from the Mayachitra website
• added links to public sample files for Hamamatsu NDPI and Hamamatsu VMS
• listed OpenSlide as available software for supported formats
• added a new developer page detailing in-memory reading and writing
• updated the Bio-Formats API versioning policy, which now follows strict semantic versioning
• a new options page has been added, detailing the usage of configurable format-specific options for readers and writers. Links to the available options are also included under the relevant supported formats
5.42 5.4.0 (2017 March 21)

File format improvements:

- **DICOM**
  - added support for DICOMDIR files, which allow multiple DICOM files in a single directory to be opened as a single dataset
  - plane position values for values X, Y and Z are now being set in OME-XML
  - correctly read the physical size X and Y values based on the available specification

- **Nikon NIS-Elements ND2**
  - performance improvements based on reading chunkmap. Processing of the chunkmap can be disabled via the MetadataOptions API using the boolean option `nativend2.chunkmap`. For ImageJ users this option can be accessed via a checkbox in the Nikon ND2 section of the Bio-Formats configuration dialog Plugins → Bio-Formats → Bio-Formats Plugins Configuration (thanks to Christian Sachs)

- **OME-TIFF**
  - added an option to save an OME-TIFF dataset as a binary TIFF and companion XML. This can be used via the bfconvert command line tool by setting the value of option `ometiff.companion` to the name of the companion file to use. For example `bfconvert -option ometiff.companion outputFile.companion.ome inputFile.tiff outputFile.ome.tiff`

- **CellVoyager**
  - metadata fixes specifically the naming of plates. Additional refactoring of the reader for general maintainability

- **Gatan Digital Micrograph**
  - previously missing Image-Instrument reference has been added to OME-XML

- **TiffSaver**
  - ensure open resources are closed under all possible scenarios

- **Zeiss CZI**
  - improved performance of large uncompressed images. When tiles from a large uncompressed image with no internal tiling are requested, only the specific tile specified in the call to `openBytes` is read from disk, instead of the entire image being read and then copied

- **Zeiss AxioVision ZVI (Zeiss Vision Image)**
  - ensure that the `bitsPerPixel` field is always set to match the final pixel type, and populate any channel colors that were parsed in the metadata. The bits per pixel update should only affect `uint16` or `int16` files where the acquisition bit depth is not a multiple of 8, and the RGB channel count is greater than 1

Updated build system:

- updated dependency for NetCDF to 4.3.22
- updated copyright headers from 2016 to 2017 and reviewed and fixed any incorrect header descriptions
- documentation has been migrated to use `.rst` file format for Sphinx files
- reviewed and cleaned up warnings such as unused variables and imports
- added CellVoyager datasets to automated testing via continuous integration
- unified the semantics for creating temporary directories within unit tests
Bio-Formats, Release 6.14.0

Documentation improvements:

• fixed link for PerkinElmer UltraVIEW system
• fixed links for NIfTI public specification and data sets
• available software for Hamamatsu ndpi has been updated from NDP.view to NDP.view2

5.43  5.3.4 (2017 February 21)

Bug fixes:

• ImageJ
  – fix for a NullPointerException when exporting images that were not opened via the Bio-Formats im-
    porter, and thus do not have a complete OMEXMLMetadata store

• Java 1.9
  – fix compile and runtime errors to enable building with Java 1.9

• ECAT7
  – update to add support for different versions of ECAT7 files (thanks to Torsten Stöter)

Updated build system:

• updated dependency for ome-model in the POM to version 5.4.0. This allows for improved ROI handling by
  enabling support for Shape objects with Transform attributes. OME-XML schema version remains unchanged
  as OME schema 2016-06

Documentation improvements:

• new public sample files added for ECAT7 (thanks to Torsten Stöter)
• new public sample files added for Leica LIF (thanks to Michael Goelzer)
• new specification document (Version 3.2) for Leica LIF
• updated links to OMERO documentation as a result of decoupling

5.44  5.3.3 (2017 February 2)

Bug fixes:

• ImageJ
  – fix for issue when exporting from an ImagePlus that represents signed data. The pixel type will now
    remain unchanged as will the pixel values which had previously been scaled incorrectly

• Command line tools
  – fix for java.lang.IllegalArgumentException when using bfconvert via command line with op-
    tion set to only convert a single time-point, channel or Z section

• Tiff writing
  – using TiffWriter to write tiled images now supports the writing of BigTIFF datasets

File format fixes:

• Applied Precision CellWorX
– fix to now display the correct plate name and dimensions

- **NIFTI**
  – a few fixes for problems with byte alignment when reading non-core metadata from NIFTI headers

- **Leica LIF**
  – added support for timestamps of LIF files created with LAS AF 3.1 or newer. In the case of a halted acquisition only non-null timestamps are stored in the OME metadata (thanks to Michael Goelzer)
  – the physical pixel height and width were incorrectly calculated by dividing by the number of pixels. This has now been corrected to match the official Leica LIF specification documents by dividing by the number of pixels minus one (thanks to Michael Goelzer)
  – for backwards compatibility an option to preserve pre-5.3.3 physical sizes has been added. This can be set either via command line tools, through the API with the `loci.formats.in.DynamicMetadataOptions` class, or in the Bio-Formats plugin configuration in ImageJ

- **Improvision TIFF**
  – channel colors are now being read and if present set correctly in image metadata

- **MetaMorph**
  – fix for `java.lang.OutOfMemoryError` exceptions when reading large Metamorph TIFF plates

Updated build system:
- version history file added to MATLAB bundle as NEWS.rst
- increased `TiffWriter` test coverage
- added test coverage framework for command line tools including new `ImageConverterTest`

Documentation improvements:
- improved documentation and new examples for using tiled writing
- updated developer documentation for use of Bio-Formats as a Maven, Gradle or Ivy dependency
- documentation for Leica LIF bug fixes and use of backward compatibility options
- fixes for a number of broken links

### 5.45 5.3.2 (2017 January 9)

Bug fixes:

- **ImageJ**
  – fixed race condition when opening multiple series from a dataset, as introduced by thumbnail loading changes in 5.3.0
  – updated thumbnail generation to be faster for datasets containing an image pyramid

- **MetaMorph**
  – updated to read the refractive index and set `RefractiveIndex` on `ObjectiveSettings` in the generated OME-XML (thanks to Marc Bruce)

- **MetaMorph TIFF**
  – fixed Z and channel dimension counts when each channel has a unique Z position
Bio-Formats, Release 6.14.0

- updated to read the emission wavelength and set EmissionWavelength on LightSourceSettings in the generated OME-XML

**QuickTime**
- fixed incorrect image data when reading of tiles from single channel files

**file grouping**
- fixed handling of loci.formats.in.MetadataOptions objects by the loci.formats.FileStitcher reader

Documentation improvements:
- fixed extensions listed for Zeiss TIFF
- simplified markdown for creating tables

5.46 5.3.1 (2016 December 19)

File format fixes:
- **TIFF**
  - fixed invalid IFD values when writing TIFF or OME-TIFF files with Bio-Formats 5.3.0. This bug affected the writing of TIFF and OME-TIFF via client code using loci.formats.TiffWriter, converting to a TIFF or OME-TIFF using 'bfconvert' command line tool or exporting to TIFF or OME-TIFF using ImageJ/FIJI Bio-Formats exporter.

5.47 5.3.0 (2016 December 12)

New features/API:
- added support for JPEG-XR compressed CZI data (funded by a partnership between Glencoe Software and ZEISS), adding 'ome:jxrlib' as a new dependency of Bio-Formats

**improved tile-based image writing**
- added new methods to the loci.formats.IFormatWriter interface allowing to set and retrieve the tile along the X and Y dimensions
- added default implementations to the loci.formats.FormatWriter abstract class defaulting to the entire image width/height
- added functionality to loci.formats.TiffWriter adding support for tiled images writing for TIFF and derived formats like OME-TIFF
- added developer documentation and samples for tiled reading/writing

- added a new MetadataOptions implementation supporting arbitrary key/value pairs
- updated the display command line utility to support passing key/value options using showinf -option
- added two options to the CZI reader to disable autostitching and exclude pyramid file attachments. Added new checkboxes to the CZI configuration interface of the ImageJ plugin to activate these options

Bug fixes/deprecations:
- deprecated loci.formats.meta.MetadataConverter in favor of ome.xml.meta.MetadataConverter
- updated method deprecated in Octave 4.2.0 (thanks to Carnê Draug)
• OME-XML
  – fixed handling of Mask BinData elements

Component architecture changes/decoupling:

• removed formats-common component - now decoupled to the new ome/ome-common-java GitHub repository and consumed as ‘org.openmicroscopy:ome-common’ artifact from Maven Central
• removed ome-poi component - now decoupled to the new ome/ome-poi GitHub repository and consumed as ‘org.openmicroscopy:ome-poi’ artifact from Maven Central
• removed specification, xsd-fu and ome-xml components - now decoupled to the new ome/ome-model GitHub repository and consumed as ‘org.openmicroscopy:{specification,ome-xml}’ artifacts from Maven Central
• removed mdbtools component - now decoupled to the new ome/ome-mdbtools GitHub repository and consumed as ‘org.openmicroscopy:ome-mdbtools’ artifact from Maven Central
• removed stubs components - now decoupled to the new ome/ome-stubs GitHub repository and consumed as ‘org.openmicroscopy:{lwf-stubs,mipav-stubs}’ artifacts from Maven Central
• removed metakit component - now decoupled to the new ome/ome-metakit GitHub repository and consumed as ‘org.openmicroscopy:metakit’ artifacts from Maven Central
• updated developer documentation for the decoupled components

Updated build system:

• dropped embedded JARs and now use the Maven Ant Tasks plugin to unify the dependencies using the POM
• improved Ant JAR and bundle target
• dropped deprecated osgi targets, OME Tools bundle and ome-jxr component
• removed PDF generation from the docs-sphinx target
• added version number to Javadoc zip bundle name
• migrated unit tests out of test-suite into formats-bsd
• fixed test-suite targets, paths and symlink handling
• fixed test-metadata and migrated it into test-suite
• fixed mismatch between ND2HandlerTest package and location
• cleaned up test-build to remove obsolete and decoupled components and folders
• simplified Travis build
• POM repositories clean-up to reduce complexity and use Maven Central as the first location to look for dependencies
• now storing all versions in the top-level POM
• updated build versioning from Maven by unified versioning strategy, reviewing meta information stored in the manifests of each JAR and introspecting this information in the FormatTools API to retrieve version and revision numbers
• updated developer documentation on updated build system
5.48 5.2.4 (2016 October 18)

Java bug fixes:

- **OME-TIFF**
  - fixed regression when populating plane metadata
- **CZI**
  - populated series metadata with the scene/position information

5.49 5.2.3 (2016 October 5)

Java bug fixes:

- **CZI**
  - fixed imageCount for RGB images
- **ICS writing**
  - fixed ordering of image dimensions
- **DeltaVision**
  - fixed reading of large time dimensions

Command-line tools improvements:

- **bf tools.zip** now includes the version history as NEWS.rst (thanks to Gerhard Burger)

Code clean-up/improvements:

- switched to String.indexOf(int) in GPL-licensed reader code so that a simpler library method can be used
- strings now extended with characters where possible
- completed deprecation of DataTools.sanitizeDouble()
- deprecated unused OSGi and ome-tools bundle build targets

OME-XML changes/improvements:

- bumped schema version number to 2 (schema namespace left unchanged)
- added acquisition modes BrightField, SweptFieldConfocal and SPIM
- added parsing for Laser Scan Confocal and Swept Field Confocal

Documentation improvements:

- documented versioning policy
- clarified supported versions for Micro-Manager and Olympus ScanR files
5.50 5.2.2 (2016 September 13)

Java bug fixes and improvements:

- fixed a regression in which the DataTools number parsing API would not be thread-safe anymore
- **InCell**
  - improved handling of Analyzer 2000 datasets to find TIFF files
- **FV1000**
  - fixed preview names ordering
- **OME-TIFF**
  - enabled all BigTIFF extensions
- various code cleanup across the Java code
- added test coverage for all example codes in the developer documentations
- added tests covering the semantics of the INI parser

ImageJ bug fixes and improvements:

- fixed a bug in ImageJ when swapping dimensions of an image with multiple series of different dimensions
- added an option to the exporter to pad filename indexes with zeros

Command-line tools improvements:

- allowed the binaries to be symlinked (thanks to Gerhard Burger)
- added an option to bfconvert to pad filename indexes with zeros

5.51 5.2.1 (2016 August 25)

Java bug fixes:

- **Zeiss CZI**
  - fixed NumberFormatException when the position object is not null but the values of child are null
- **SimplePCI**
  - made IniParser less stringent to allow reading of imperfectly formatted TIFF description headers
- fixed stitching of file patterns in ImageJ to remove duplication of directory names in the file path
- added an option to bfconvert to allow creation of OME-TIFF without lookup tables
- addition of MetadataOnly elements containing no BinData or TiffData now handled via MetadataTools API in ImageInfo
- example code in developer docs is now tested via a new Maven module
Java format support improvements are listed below.

†Denotes a major breaking change to the reader (typically modification of core metadata). Code changes or re-import may be necessary in ImageJ/FIJI and OMERO.

- added support (and public sample files) for Becker & Hickl .spc FIFO data
- added support for Princeton Instruments .spe data
- bug fixes for many formats including:
  - CellSens VSI†
    * fixes for correctly reading dimensions
  - FlowSight
    * fixes to infer channel count from channel names (thanks to Lee Kamentsky)
  - Hamamatsu VMS†
    * fixed dimensions of full-resolution images
  - ICS writing
    * fixed dimension population for split files
  - Kodak BIP
    * fixed handling of CCD temperature stored in hexadecimal
  - Leica LIF
    * fixed incorrect plane offsets for large multi-tile files
  - LiFlim
    * fixed ExposureTime check and units usage
  - Micro-Manager
    * fixed handling of large datasets saved as image stacks and split over multiple files
    * added user documentation for file saving options
  - MRC and Spider
    * fixed format type checking
  - Nifti
    * fixed planeSize to prevent crashes when loading large files (thanks to Christian Niedworok)
    * added support for gzipped compressed .nii.gz files (thanks to Eric Barnhill)
    * added public samples and updated documented supported file extensions
  - OME-TIFF
    * fixed Plane population errors
    * fixed NullPointerException when closing reader for partial multi-file filesets
    * reduced buffer size for RandomAccessInputStreams to improve performance
    * deprecated getMetadataStoreForConversion and getMetadataStoreForDisplay methods

5.52 5.2.0 (2016 August 18)
- OME-XML
  * fixed metadata store

- PicoQuant
  * updated reader to always buffer data

- PNG writing

- SDT
  * performance improvements for loading of large files

- Slidebook
  * Slidebook6Reader is now completely external and fully maintained by 3i (see http://www.openmicroscopy.org/info/slidebook) and is specified as such in the readers.txt configuration file

- SVS
  * fixed NumberFormatException

- Tiff
  * fixed integer overflow to read resolutions correctly
  * fixed handling of tiled images with tile width less than 64

- Zeiss CZI
  * fixed timestamp indexing when multiple separate channels are present
  * improved slide support - slides are now detected as a complete full-resolution image (instead of each tile being a separate series) and pyramid sub-resolutions and label/overview images are also detected

- Zeiss LSM
  * fixed Plane population errors

- Zeiss ZVI†
  * reworked image ordering calculation to allow for tiles

Top-level Bio-Formats API changes:
- Java 1.7 is now the minimum supported version
- the native-lib-loader dependency has been bumped to version 2.1.4
- the xalan dependency has been bumped to version 2.7.2
- all the ome.jxr classes have been deprecated to make clear that there is no JPEG-XR support implemented in Bio-Formats as yet

- the DataTools API has been extended to add a number of utility functions to:
  - account for decimal separators in different locales
  - parse a String into Double, Float, Integer etc
  - handle NumberFormatException thrown when parsing Unit tests

- the Logging API has been updated to respect logging frameworks (log4j/logback) initialized via a binding-specific configuration file and to prevent DebugTools.enableLogging(String) from overriding initialized logger levels (see Logging for more information)
• helper methods have been added to FormatTools allowing a stage position to be formatted from an input Double and an input unit
• the Formats API has also been updated to add a new validate property to MetadataOptions and support for MetadataOptions has been moved to FormatHandler level to allow it to be used by both Readers and Writers
• initial work on Reader discoverability extended the ClassList API to allow the readers.txt configuration file to be annotated using key/value pairs to mark optional Readers and specify additional per-Reader options

Other general improvements include:
• improved performance of getUsedFiles
• fixes for FilePatternBlock, AxisGuesser, FilePattern
• fixes for the detection of CSV pattern blocks by FilePatternBlock
• bioformats_package.jar now includes bio-formats-tools as a dependency so ImageConverter, ImageFaker and ImageInfo classes are included in the bundle
• the JACE C++ implementation has been decoupled as it does not function with Java 1.8 (see legacy repo)
• ImageJ fixes
  – to allow reader delegation when a legacy reader is enabled but not working
  – to allow ROIs to be imported to the ImageJ ROI manager or added to a new overlay
• MATLAB fixes
  – improved integration with Octave (thanks to Carnê Draug)
  – added logging initialization
• Command-line tools fixes
  – upgrade check no longer run when passing -version
  – common methods refactoring
  – showinf improvements to preload format
  – tiffcomment now warns that it requires an ImageDescription tag to be present in the TIFF file
• added many automated tests and improved FakeReader testing framework
• documentation improvements include:
  – clarifying status of legacy Quicktime and ND2 readers
  – noting that the Gatan reader does not currently support stacks
  – more Java examples added to the developer documentation
  – new units page for developers

The Data Model version 2016-06 has been released to introduce Folders, and to simplify both the graphical aspects of the model and code generation. Full details are available in the OME Model and Formats Documentation. OME-XML changes include:
• Map is now a complexType rather than an element and MapPairs has been dropped
• extended enum metadata has been introduced to better support units
• Shape and LightSource are now complexTypes rather than elements
• BinData has been added to code generation to handle raw binary data
• various code generation improvements to:
– simplify and standardize the generation process
– remove a number of hard-coded exceptional cases allowing for easier maintenance and growth
– allow for genuine abstract model types and enable C++ model implementation

• updated OME-XML and OME-TIFF public sample files

The Bio-Formats C++ native implementation has been decoupled from the Java codebase and will be released as OME-Files C++ from now on, with the exception of OME-XML which is still within Bio-Formats at present (there is a plan to decouple both the Java and the C++ versions of OME-XML in future).

The following components have had their licensing updated to Simplified (2-clause) BSD:

• XSL transforms
• specification code
• xsd-fu Python code

5.53 5.1.10 (2016 May 9)

Java bug fixes:

• fixed warnings being thrown for ImageJ and other non-FIJI users on Windows (these warnings were triggered by the removal of the 3i Slidebook DLLs from the source code repository in Bio-Formats 5.1.9 and should now only be triggered when opening Slidebook files without the update site enabled - http://www.openmicroscopy.org/info/slidebook)
• a fix in the ImageJ plugin for files grouped using the “Dimensions” option
• a fix for writing TIFF files in tiles

5.54 5.1.9 (2016 April 14)

• Java bug fixes, including:
  – SDT
    • fixed width padding calculation for single-pixel image
  – Deltavision
    • fixed the parsing of the new date format
    • added support for parsing and storing the working distance in native units
  – Micromanager
    • cleaned up JSON metadata parsing
  – Olympus Fluoview
    • fixed null pointer exceptions while parsing metadata
  – Leica LIF
    • fixed large multi-tiled files from having incorrect plane offsets after the 2GB mark
  – EM formats (MRC and Spider)
    • added native length support for EM readers
– Gatan
  * fixed erroneous metadata parsing
  * added support for parsing and storing the physical sizes in native units
– OME-TIFF
  * improved handling of OME-TIFF multi-file fileset’s with partial metadata blocks
– Nikon ND2
  * fixed the parsing of emission wavelength
– Olympus CellR (APL)
  * fixed multiple parsing issues with the mtb file
– SlideBook
  * removed slidebook dlls from Bio-Formats repository
  * http://www.openmicroscopy.org/info/slidebook
– Zeiss CZI
  * fixed parsing of files with multiple mosaics and positions
• Documentation updates, including:
  – improved documentation for the export of BigTIFFs in ImageJ
• C++:
  – no changes.

5.55 5.1.8 (2016 February 15)

• Java bug fixes, including:
  – FEI TIFF
    * fixed stage position parsing and whitespace handling (thanks to Antoine Vandecreme)
  – Pyramid TIFF
    * fixed tile reading when a cache (.bfmemo) file is present
  – MicroManager
    * updated to parse JSON data from tags 50839 and 51123
    * fixed to detect *_metadata.txt files in addition to metadata.txt files
    * fixed to handle datasets with each stack in a single file
  – OME-XML
    * updated to make .ome.xml an official extension
  – OME-TIFF
    * fixed to ignore invalid BinaryOnly elements
  – TIFF
    * fixed caching of BigTIFF files
Bio-Formats, Release 6.14.0

- **Slidebook**
  * fixed handling of montages in Slidebook6Reader (thanks to Richard Myers)
  - Performance improvement for writing files to disk (thanks to Stephane Dallongeville)
- **Build system**
  * fixed Maven POMs to reduce calls to artifacts.openmicroscopy.org
  * fixed bioformats_package.jar to include the loci.formats.tools package
- **Documentation updates, including:**
  - updated format pages to include links to example data
  - clarified description of Qu for MATLAB (thanks to Carnë Draug)
  - added installation instructions for Octave (thanks to Carnë Draug)
- **C++:**
  - Bugfixes to the OME-TIFF writer to correct use of the metadata store with multiple series
  - Ensure file and writer state consistency upon close failure

### 5.56 5.1.7 (2015 December 7)

- **Java bug fixes, including:**
  - Prevent physical pixel sizes from being rounded to 0, for all formats
  - **Metamorph**
    * fixed calculation of Z step size
    * fixed detection of post-processed dual camera acquisitions (thanks to Mark Kittisopikul)
  - **OME-XML**
    * fixed XML validation when an ‘xmlns’ value is not present (thanks to Bjoern Thiel)
  - **MINC**
    * fixed endianness of image data
  - **Andor/Fluoview TIFF**
    * fixed calculation of Z step size
  - **MATLAB**
    * improved performance by reducing static classpath checks (thanks to Mark Kittisopikul)
  - **Gatan**
    * fixed physical size parsing in non-English locales
- **Automated testing**
  * fixed handling of non-default physical size and plane position units
- **Documentation updates, including:**
  - updated MapAnnotation example to show linkage of annotations to images
- **C++:**
– no changes, released to keep version numbers in sync with Bio-Formats Java

5.57 5.1.6 (2015 November 16)

• Java bug fixes, including:
  – Updated to use native units for following formats:
    * IMOD
    * Analyze
    * Unisoku
    * Olympus CellR (APL)
  – Metamorph TIFF
    * fixed handling of multi-line descriptions
    * added support for dual camera acquisitions
  – Zeiss LMS
    * fixed exception in type detection
  – Zeiss CZI
    * fixed detection of line scan Airyscan data
  – Slidebook
    * fixed calculation of physical Z size
  – ImageJ plugins
    * fixed handling of non-default units
    * fixed setting of preferences via macros
  – Automated testing
    * fixed handling of non-default units for physical sizes and timings

• C++ changes, including:
  – allow relocatable installation on Windows
  – reduce time required for debug builds

• Documentation updates, including:
  – addition of “Multiple Images” column to the supported formats table
  – addition of a MapAnnotation example
• Java bug fixes, including:
  – ImageJ plugins
    * fixed use of “Group files...” and “Open files individually” options
    * fixed placement of ROIs
    * fixed size of the “About Plugins > Bio-Formats Plugins” window
  – xsd-fu (code generation)
    * removed OMERO-specific logic
  – Metamorph
    * fixed physical Z size calculation
  – Gatan DM3/DM4
    * fixed physical pixel size parsing
  – BMP
    * added support for RLE compression
  – DICOM
    * updated to respect the WINDOW_CENTER tag
    * fixed image dimensions when multiple sets of width and height values are present
  – Fluoview and Andor TIFF
    * fixed physical Z size calculation
  – Imspector OBF
    * updated to parse OME-XML metadata (thanks to Bjoern Thiel)

• C++ changes:
  – TIFF strip/tile row and column calculations corrected to compute the correct row and column count
  – Several compiler warnings removed (false positive warnings in third-party headers disabled, and additional warnings fixed)
  – It is now possible to build with Boost 1.59 and compile with a C++14 compiler

• The source release is now provided in both tar.xz and zip formats

• Documentation updates, including:
  – substantial updates to the format pages
    * improved linking of reader/writer classes to each format page
    * improved supported metadata pages for each format
    * updated format page formatting for clarity
    * added developer documentation for adding and modifying format pages
5.59 5.1.4 (2015 September 7)

- Bug fixes, including:
  - Command line tools
    - fixed display of usage information
  - Automated testing
    - fixed problems with symlinked data on Windows
    - added unit tests for checking physical pixel size creation
  - Cellomics
    - fixed reading of sparse plates
  - SlideBook
    - fixed a few lingering issues with native library packaging
  - SimplePCI/HCImage TIFF
    - fixed bit depth parsing for files from newer versions of HCImage
  - SimplePCI/HCImage .cxd
    - fixed image dimensions to allow for extra padding bytes
  - Leica LIF
    - improved reading of image descriptions
  - ICS
    - fixed to use correct units for timestamps and physical pixel sizes
  - MicroManager
    - fixed to use correct units for timestamps
  - Gatan .dm3/.dm4
    - fixed problems with reading double-precision metadata values
  - Hamamatsu NDPI
    - fixed reading of mask images
  - Leica .lei
    - fixed reading of bit depth and endianness for datasets that were modified after acquisition
  - FEI TIFF
    - updated to read metadata from files produced by FEI Titan systems
  - QuickTime
    - fixed to handle planes with no stored pixels
  - Leica .scn
    - fixed reading of files that contain fewer images than expected
  - Zeiss .czi
    - fixed channel colors when an alpha value is not recorded
Bio-Formats, Release 6.14.0

* fixed handling of pre-stitched image tiles
  - SDT
    * added support for Zip-compressed images
  - Nikon .nd2
    * fixed to read image dimensions from new non-XML metadata
  - OME-XML
    * fixed writing of integer metadata values

* Native C++ updates:
  - completed support for building on Windows

* Documentation updates, including:
  - updated instructions for running automated data tests
  - clarified JVM versions currently supported

5.60 5.1.3 (2015 July 21)

* Native C++ updates:
  - Added cmake superbuild to build core dependencies (zlib, bzip2, png, icu, xerces, boost)
  - Progress on support for Windows

* Bug fixes, including:
  - Fixed segfault in the showinf tool used with the C++ bindings
  - Allow reading from https URLs
  - ImageJ
    * improved performance of displaying ROIs
  - Command line tools
    * fixed bfconvert to correctly create datasets with multiple files
  - Metamorph
    * improved detection of time series
    * fixed .nd datasets with variable Z and T counts in each channel
    * fixed .nd datasets that contain invalid TIFF/STK files
    * fixed dimensions when the number of planes does not match the recorded Z, C, and T sizes
  - SlideBook
    * improved native library detection (thanks to Richard Myers)
  - JPEG
    * fixed decompression of lossless files with multiple channels (thanks to Aaron Avery)
  - Imspector OBF
    * updated to support version 2 files (thanks to Bjoern Thiel)
– **Inspector MSR**
  * improved detection of Z stacks

– **PerkinElmer Opera Flex**
  * improved handling of multiple acquisitions of the same plate

– **Zeiss CZI**
  * fixed error when opening single-file datasets whose names contained “(” and “)"

– **TIFF**
  * improved speed of reading files with many tiles

– **AVI**
  * updated to read frame index (idx1) tables

– **Nikon ND2**
  * fixed channel counts for files with more than 3 channels

– **PNG**
  * fixed decoding of interlaced images with a width or height that is not a multiple of 8

– **PSD**
  * improved reading of compressed images

• **Documentation improvements, including:**
  – updated instructions for writing a new file format reader
  – updated usage information for command line tools
  – new Javadocs for the `MetadataStore` and `MetadataRetrieve` interfaces

### 5.61 5.1.2 (2015 May 28)

• Added OME-TIFF writing support to the native C++ implementation
• OME-TIFF export: switch to BigTIFF if .ome.tf2, .ome.tf8, or .ome.btf extensions are used
• Improved MATLAB developer documentation
• Added SlideBook reader that uses the SDK from 3I (thanks to Richard Myers and 3I - Intelligent Imaging Innovations)
• Preliminary work to make MATLAB toolbox work with Octave
• **Many bug fixes, including:**
  – **ImageJ**
    * fixed regression in getPlanePosition* macro extension methods
    * fixed display of composite color virtual stacks
  – **Nikon ND2**
    * improved parsing of plane position and timestamp data
  – **TIFF**
    * reduced memory required to read color lookup tables
- Zeiss LSM
  * improved parsing of 16-bit color lookup tables
- Zeiss CZI
  * fixed ordering of original metadata table
  * fixed reading of large pre-stitched tiled images
- AIM
  * fixed handling of truncated files
- Metamorph/MetaXpress TIFF
  * improved UIC1 metadata tag parsing

5.62 5.1.1 (2015 April 28)

- Add TIFF writing support to the native C++ implementation
- Fixed remaining functional differences between Windows and Mac/Linux
- Improved performance of ImageJ plugin when working with ROIs
- TIFF export: switch to BigTIFF if .tf2, .tf8, or .btf extensions are used
- Many bug fixes, including:
  - fixed upgrade checking to more accurately report when a new version is available
  - Zeiss CZI
    * fixed ordering of multiposition data
    * improved support for RGB and fused images
  - Nikon ND2
    * improved ordering of multiposition data
  - Leica LIF
    * improved metadata validity checks
    * improved excitation wavelength detection
  - Metamorph STK/TIFF
    * record lens numerical aperture
    * fixed millisecond values in timestamps
  - Gatan DM3
    * correctly detect signed pixel data
  - Imaris HDF
    * fix channel count detection
  - ICS export
    * fix writing of files larger than 2GB
5.63 5.1.0 (2015 April 2)

- Improvements to performance with network file systems
- Improvements to developer documentation
- Initial version of native C++ implementation
- Improved support for opening and saving ROI data with ImageJ
- Added support for CellH5 data (thanks to Christoph Sommer)
- Added support for Perkin Elmer Nuance data (thanks to Lee Kamentsky)
- Added support for Amnis FlowSight data (thanks to Lee Kamentsky and Sebastien Simard)
- Added support for Veeco AFM data
- Added support for Zeiss .lms data (not to be confused with .lsm)
- Added support for .2i data
- Added support for writing Vaa3D data (thanks to Brian Long)
- Updated to OME schema 2015-01
- Update RandomAccessInputStream and RandomAccessOutputStream to read and write bits
- Many bug fixes, including:
  - Leica SCN
    * fix pixel data decompression
    * fix handling of files with multiple channels
    * parse magnification and physical pixel size data
  - Olympus/CellSens .vsi
    * more thorough parsing of metadata
    * improved reading of thumbnails and multi-resolution images
  - NDPI
    * fix reading of files larger than 4GB
    * parse magnification data
  - Zeiss CZI
    * improve parsing of plane position coordinates
  - Inveon
    * fix reading of files larger than 2 GB
  - Nikon ND2
    * many improvements to dimension detection
    * many improvements to metadata parsing accuracy
    * update original metadata table to include PFS data
  - Gatan DM3
    * fix encoding when parsing metadata
- fix physical pixel size parsing
- **Metamorph**
  - fix off-by-one in metadata parsing
  - fix number parsing to be independent of the system locale
- **JPEG**
  - parse EXIF data, if present (thanks to Paul Van Schayck)
- **OME-XML/OME-TIFF**
  - fix handling of missing image data
- **PrairieView**
  - improved support for version 5.2 data (thanks to Curtis Rueden)
- **DICOM**
  - fix dimensions for multi-file datasets
  - fix pixel data decoding for files with multiple images
- **PNG**
  - reduce memory required to read large images
- **Inspector OBF**
  - fix support for version 5 data (thanks to Bjoern Thiel)
- **PCORAW**
  - fix reading of files larger than 4 GB
- **AIM**
  - fix reading of files larger than 4 GB
- **MRC**
  - add support for signed 8-bit data
- Fix build errors in MIPAV plugin
- **ImageJ**
  - fix export from a script/macro
  - fix windowless export
  - allow exporting from any open image window
  - allow the “Group files with similar names” and “Swap dimensions” options to be used from a script/macro
- **bfconvert**
  - fix writing each channel, Z section, and/or timepoint to a separate file
  - add options for configuring the tile size to be used when saving images
5.64 5.0.8 (2015 February 10)

- No changes - release to keep version numbers in sync with OMERO

5.65 5.0.7 (2015 February 5)

- Several bug fixes, including:
  - ND filter parsing for DeltaVision
  - Timepoint count and original metadata parsing for Metamorph
  - Build issues when Genshi or Git are missing
  - LZW image decoding

5.66 5.0.6 (2014 November 11)

- Several bug fixes, including:
  - Pixel sign for DICOM images
  - Image dimensions for Zeiss CZI and Nikon ND2
  - Support for Leica LIF files produced by LAS AF 4.0 and later

5.67 5.0.5 (2014 September 23)

- Documentation improvements
- Support for non-spectral Prairie 5.2 datasets

5.68 5.0.4 (2014 September 3)

- Fix compile and runtime errors under Java 1.8
- Improvements to Nikon .nd2 metadata parsing
- Added support for PicoQuant .bin files (thanks to Ian Munro)

5.69 5.0.3 (2014 August 7)

- Many bug fixes for Nikon .nd2 files
- Several other bug fixes, including:
  - LZW image decoding
  - Stage position parsing for Zeiss CZI
  - Exposure time units for ScanR
– Physical pixel size units for DICOM
– NDPI and Zeiss LSM files larger than 4GB
– Z and T dimensions for InCell 6000 plates
– Export of RGB images in ImageJ

• Improved metadata saving in MATLAB functions

### 5.70 5.0.2 (2014 May 28)

• Many bug fixes for Zeiss .czi files
• **Several other bug fixes, including:**
  – Gatan .dm3 units and step count parsing
  – Imspector .msr 5D image support
  – DICOM reading of nested tags
• Update native-lib-loader version (to 2.0.1)
• Updates and improvements to user documentation

### 5.71 5.0.1 (2014 Apr 7)

• Added image pyramid support for CellSens .vsi data
• **Several bug fixes, including:**
  – Woolz import into OMERO
  – Cellomics file name parsing (thanks to Lee Kamentsky)
  – Olympus FV1000 timestamp support (thanks to Lewis Kraft and Patrick Riley)
  – (A)PNG large image support
  – Zeiss .czi dimension detection for SPIM datasets
• Performance improvements for Becker & Hickl .sdt file reading (thanks to Ian Munro)
• Performance improvements to directory listing over NFS
• Update slf4j and logback versions (to 1.7.6 and 1.1.1 respectively)
• Update jgoodies-forms version (to 1.7.2)
5.72 5.0.0 (2014 Feb 25)

- New bundled ‘bioformats_package.jar’ for ImageJ
- Now uses logback as the slf4j binding by default
- Updated component names, .jar file names, and Maven artifact names
- Fixed support for Becker & Hickl .sdt files with multiple blocks
- Fixed tiling support for TIFF, Hamamatsu .ndpi, JPEG, and Zeiss .czi files
- Improved continuous integration testing
- Updated command line documentation

5.73 5.0.0-RC1 (2013 Dec 19)

- Updated Maven build system and launched new Artifactory repository (http://artifacts.openmicroscopy.org)
- Added support for:
  - Bio-Rad SCN
  - Yokogawa CellVoyager (thanks to Jean-Yves Tinevez)
  - LaVision Inspector
  - PCORAW
  - Woolz (thanks to Bill Hill)
- Added support for populating and parsing ModuloAlong{Z, C, T} annotations for FLIM/SPIM data
- Updated netCDF and slf4j version requirements - netCDF 4.3.19 and slf4j 1.7.2 are now required
- Updated and improved MATLAB users and developers documentation
- Many bug fixes including for Nikon ND2, Zeiss CZI, and CellWorX formats

5.74 5.0.0-beta1 (2013 June 20)

- Updated to 2013-06 OME-XML schema
- Improved the performance in tiled formats
- Added caching of Reader metadata using https://github.com/EsotericSoftware/kryo
- Added support for:
  - Aperio AFI
  - Inveon
  - MPI-BPC Inspector
- Many bug fixes, including:
  - Add ZEN 2012/Lightsheet support to Zeiss CZI
  - Improved testing of autogenerated code
  - Moved OME-XML specification into Bio-Formats repository
5.75 4.4.10 (2014 Jan 15)

- Bug fixes including CellWorx, Metamorph and Zeiss CZI
- Updates to MATLAB documentation

5.76 4.4.9 (2013 Oct 16)

- Many bug fixes including improvements to support for ND2 format
- Java 1.6 is now the minimum supported version; Java 1.5 is no longer supported

5.77 4.4.8 (2013 May 2)

- No changes - release to keep version numbers in sync with OMERO

5.78 4.4.7 (2013 April 25)

- Many bug fixes to improve support for more than 20 formats
- Improved export to multi-file datasets
- Now uses slf4j for logging rather than using log4j directly, enabling other logging implementations to be used, for example when Bio-Formats is used as a component in other software using a different logging system.

5.79 4.4.6 (2013 February 11)

- Many bug fixes
- Further documentation improvements

5.80 4.4.5 (2012 November 13)

- Restructured and improved documentation
- Many bug fixes, including:
  - File grouping in many multi-file formats
  - Maven build fixes
  - ITK plugin fixes
5.81  4.4.4 (2012 September 24)

- Many bug fixes

5.82  4.4.2 (2012 August 22)

- Security fix for OMERO plugins for ImageJ

5.83  4.4.1 (2012 July 20)

- Fix a bug that prevented BigTIFF files from being read
- Fix a bug that prevented PerkinElmer .flex files from importing into OMERO

5.84  4.4.0 (2012 July 13)

- Many, many bug fixes
- **Added support for:**
  - .nd2 files from Nikon Elements version 4
  - PerkinElmer Operetta data
  - MJPEG-compressed AVIs
  - MicroManager datasets with multiple positions
  - Zeiss CZI data
  - IMOD data

5.85  4.3.3 (2011 October 18)

- Many bug fixes, including:
  - Speed improvements to HCImage/SimplePCI and Zeiss ZVI files
  - Reduce memory required by Leica LIF reader
  - More accurately populate metadata for Prairie TIFF datasets
  - Various fixes to improve the security of the OMERO plugin for ImageJ
  - Better dimension detection for Bruker MRI datasets
  - Better thumbnail generation for histology (SVS, NDPI) datasets
  - Fix stage position parsing for Metamorph TIFF datasets
  - Correctly populate the channel name for PerkinElmer Flex files
5.86 4.3.2 (2011 September 15)

- Many bug fixes, including:
  - Better support for Volocity datasets that contain compressed data
  - More accurate parsing of ICS metadata
  - More accurate parsing of cellSens .vsi files
- Added support for a few new formats
  - .inr
  - Canon DNG
  - Hitachi S-4800
  - Kodak .bip
  - JPX
  - Volocity Library Clipping (.acff)
  - Bruker MRI
- Updated Zeiss LSM reader to parse application tags
- Various performance improvements, particularly for reading/writing TIFFs
- Updated OMERO ImageJ plugin to work with OMERO 4.3.x

5.87 4.3.1 (2011 July 8)

- Several bug fixes, including:
  - Fixes for multi-position DeltaVision files
  - Fixes for MicroManager 1.4 data
  - Fixes for 12 and 14-bit JPEG-2000 data
  - Various fixes for reading Volocity .mvd2 datasets
- Added various options to the ‘showinf’ and ‘bfconvert’ command line tools
- Added better tests for OME-XML backwards compatibility
- Added the ability to roughly stitch tiles in a multi-position dataset

5.88 4.3.0 (2011 June 14)

- Many bug fixes, including:
  - Many fixes for reading and writing sub-images
  - Fixes for stage position parsing in the Zeiss formats
  - File type detection fixes
- Updated JPEG-2000 reading and writing support to be more flexible
- Added support for 9 new formats:
- InCell 3000
- Trestle
- Hamamatsu .ndpi
- Hamamatsu VMS
- SPIDER
- Volocity .mvd2
- Olympus SIS TIFF
- IMAGIC
- cellSens VSI

• Updated to 2011-06 OME-XML schema
• Minor speed improvements in many formats
• Switched version control system from SVN to Git
• Moved all Trac tickets into the OME Trac: https://trac.openmicroscopy.org
• Improvements to testing frameworks
• Added Maven build system as an alternative to the existing Ant build system
• Added pre-compiled C++ bindings to the download page

5.89 4.2.2 (2010 December 6)

• Several bug fixes, notably:
  - Metadata parsing fixes for Zeiss LSM, Metamorph STK, and FV1000
  - Prevented leaked file handles when exporting to TIFF/OME-TIFF
  - Fixed how BufferedImages are converted to byte arrays
• Proper support for OME-XML XML annotations
• Added support for SCANCO Medical .aim files
• Minor improvements to ImageJ plugins
• Added support for reading JPEG-compressed AVI files

5.90 4.2.1 (2010 November 12)

• Many, many bug fixes
• Added support for 7 new formats:
  - CellWorX .pnl
  - ECAT7
  - Varian FDF
  - Perkin Elmer Densitometer
Bio-Formats, Release 6.14.0

- FEI TIFF
- Compix/SimplePCI TIFF
- Nikon Elements TIFF

- Updated Zeiss LSM metadata parsing, with generous assistance from Zeiss, FMI, and MPI-CBG
- Lots of work to ensure that converted OME-XML validates
- Improved file stitching functionality; non-numerical file patterns and limited regular expression-style patterns are now supported

5.91 4.2.0 (2010 July 9)

- Fixed many, many bugs in all aspects of Bio-Formats
- Reworked ImageJ plugins to be more user- and developer-friendly
- Added many new unit tests
- Added support for approximately 25 new file formats, primarily in the SPM domain
- Rewrote underlying I/O infrastructure to be thread-safe and based on Java NIO
- Rewrote OME-XML parsing/generation layer; OME-XML 2010-06 is now supported
- Improved support for exporting large images
- Improved support for exporting to multiple files
- Updated logging infrastructure to use slf4j and log4j

5.92 4.1.1 (2009 December 3)

- Fixed many bugs in popular file format readers

4.1 (2009 October 21):

- Fixed many bugs in most file format readers
- Significantly improved confocal and HCS metadata parsing
- Improved C++ bindings
- Eliminated references to Java AWT classes in core Bio-Formats packages
- Added support for reading Flex datasets from multiple servers
- Improved OME-XML generation; generated OME-XML is now valid
- Added support for Olympus ScanR data
- Added OSGi information to JARs
- Added support for Amira Mesh files
- Added support for LI-FLIM files
- Added more informative exceptions
- Added support for various types of ICS lifetime data
- Added support for Nikon EZ-C1 TIFFs
• Added support for Maia Scientific MIAS data

**5.93  4.0.1 (2009 June 1)**

• Lots of bug fixes in most format readers and writers
• Added support for Analyze 7.1 files
• Added support for Nifti files
• Added support for Cellomics .c01 files
• Refactored ImageJ plugins
• Bio- Formats, the common package, and the ImageJ plugins now require Java 1.5
• Eliminated native library dependency for reading lossless JPEGs
• Changed license from GPL v3 or later to GPL v2 or later
• Updated Olympus FV1000, Zeiss LSM, Zeiss ZVI and Nikon ND2 readers to parse ROI data
• Added option to ImageJ plugin for displaying ROIs parsed from the chosen dataset
• Fixed BufferedImage construction for signed data and unsigned int data

**5.94  4.0.0 (2009 March 3)**

• Improved OME data model population for Olympus FV1000, Nikon ND2, Metamorph STK, Leica LEI, Leica LIF, InCell 1000 and MicroManager
• Added TestNG tests for format writers
• Added option to ImageJ plugin to specify custom colors when customizing channels
• Added ability to upgrade the ImageJ plugin from within ImageJ
• Fixed bugs in Nikon ND2, Leica LIF, BioRad PIC, TIFF, PSD, and OME-TIFF
• Fixed bugs in Data Browser and Exporter plugins
• Added support for Axon Raw Format (ARF), courtesy of Johannes Schindelin
• Added preliminary support for IPLab-Mac file format

**5.95  2008 December 29**

• Improved metadata support for DeltaVision, Zeiss LSM, MicroManager, and Leica LEI
• Restructured code base/build system to be component-driven
• Added support for JPEG and JPEG-2000 codecs within TIFF, OME-TIFF and OME-XML
• Added support for 16-bit compressed Flex files
• Added support for writing JPEG-2000 files
• Added support for Minolta MRW format
• Added support for the 2008-09 release of OME-XML

**5.93.  4.0.1 (2009 June 1)**
Bio-Formats, Release 6.14.0

- Removed dependency on JMagick
- Re-added caching support to data browser plugin
- Updated loci.formats.Codec API to be more user-friendly
- Expanded loci.formats.MetadataStore API to better represent the OME-XML model
- Improved support for Nikon NEF
- Improved support for TillVision files
- Improved ImageJ import options dialog
- Fixed bugs with Zeiss LSM files larger than 4 GB
- Fixed minor bugs in most readers
- Fixed bugs with exporting from an Image5D window
- Fixed several problems with virtual stacks in ImageJ

5.96 2008 August 30

- Fixed bugs in many file format readers
- Fixed several bugs with swapping dimensions
- Added support for Olympus CellR/APL files
- Added support for MINC MRI files
- Added support for Aperio SVS files compressed with JPEG 2000
- Added support for writing OME-XML files
- Added support for writing APNG files
- Added faster LZW codec
- Added drag and drop support to ImageJ shortcut window
- Re-integrated caching into the data browser plugin

5.97 2008 July 1

- Fixed bugs in most file format readers
- Fixed bugs in OME and OMERO download functionality
- Fixed bugs in OME server-side import
- Improved metadata storage/retrieval when uploading to and downloading from the OME Perl server
- Improved Bio-Formats ImageJ macro extensions
- Major updates to MetadataStore API
- Updated OME-XML generation to use 2008-02 schema by default
- Addressed time and memory performance issues in many readers
- Changed license from LGPL to GPL
- Added support for the FEI file format
• Added support for uncompressed Hamamatsu Aquacosmos NAF files
• Added support for Animated PNG files
• Added several new options to Bio-Formats ImageJ plugin
• Added support for writing ICS files

5.98 2008 April 17

• Fixed bugs in Slidebook, ND2, FV1000 OIB/OIF, Perkin Elmer, TIFF, Prairie, Openlab, Zeiss LSM, MNG, Molecular Dynamics GEL, and OME-TIFF
• Fixed bugs in OME and OMERO download functionality
• Fixed bugs in OME server-side import
• Fixed bugs in Data Browser
• Added support for downloading from OMERO 2.3 servers
• Added configuration plugin
• Updates to MetadataStore API
• Updates to OME-XML generation - 2007-06 schema used by default
• Added support for Li-Cor L2D format
• Major updates to TestNG testing framework
• Added support for writing multi-series OME-TIFF files
• Added support for writing BigTIFF files

5.99 2008 Feb 12

• Fixed bugs in QuickTime, SimplePCI and DICOM
• Fixed a bug in channel splitting logic

5.100 2008 Feb 8

• Many critical bugfixes in format readers and ImageJ plugins
• Newly reborn Data Browser for 5D image visualization
  – some combinations of import options do not work yet
5.101 2008 Feb 1

- Fixed bugs in Zeiss LSM, Metamorph STK, FV 1000 OIB/OIF, Leica LEI, TIFF, Zeiss ZVI, ICS, Prairie, Openlab LIFF, Gatan, DICOM, QuickTime
- Fixed bug in OME-TIFF writer
- Major changes to MetadataStore API
- Added support for JPEG-compressed TIFF files
- **Added basic support for Aperio SVS files**
  - JPEG2000 compression is still not supported
- Improved “crop on import” functionality
- Improvements to bfconvert and bfview
- Improved OME-XML population for several formats
- Added support for JPEG2000-compressed DICOM files
- EXIF data is now parsed from TIFF files

5.102 2007 Dec 28

- Fixed bugs in Leica LEI, Leica TCS, SDT, Leica LIF, Visitech, DICOM, Imaris 5.5 (HDF), and Slidebook readers
- Better parsing of comments in TIFF files exported from ImageJ
- Fixed problem with exporting 48-bit RGB data
- Added logic to read multi-series datasets spread across multiple files
- Improved channel merging in ImageJ - requires ImageJ 1.39i
- Support for hyperstacks and virtual stacks in ImageJ - requires ImageJ 1.39i
- Added API for reading directly from a byte array or InputStream
- Metadata key/value pairs are now stored in ImageJ’s “Info” property
- Improved OMERO download plugin - it is now much faster
- Added “open all series” option to ImageJ importer
- ND2 reader based on Nikon’s SDK now uses our own native bindings
- Fixed metadata saving bug in ImageJ
- Added sub-channel labels to ImageJ windows
- Major updates to 4D Data Browser
- Minor updates to automated testing suite
5.103 2007 Dec 1

- Updated OME plugin for ImageJ to support downloading from OMERO
- Fixed bug with floating point TIFFs
- Fixed bugs in Visitech, Zeiss LSM, Imaris 5.5 (HDF)
- Added alternate ND2 reader that uses Nikon’s native libraries
- Fixed calibration and series name settings in importer
- Added basic support for InCell 1000 datasets

5.104 2007 Nov 21

- Fixed bugs in ND2, Leica LIF, DICOM, Zeiss ZVI, Zeiss LSM, FV1000 OIB, FV1000 OIF, BMP, Evotec Flex, BioRad PIC, Slidebook, TIFF
- Added new ImageJ plugins to slice stacks and do “smart” RGB merging
- Added “windowless” importer plugin
  - uses import parameters from IJ_Prefs.txt, without prompting the user
- Improved stack slicing and colorizing logic in importer plugin
- Added support for DICOM files compressed with lossless JPEG
  - requires native libraries
- Fixed bugs with signed pixel data
- Added support for Imaris 5.5 (HDF) files
- Added 4 channel merging to importer plugin
- Added API methods for reading subimages
- Major updates to the 4D Data Browser

5.105 2007 Oct 17

- Critical OME-TIFF bugfixes
- Fixed bugs in Leica LIF, Zeiss ZVI, TIFF, DICOM, and AVI readers
- Added support for JPEG-compressed ZVI images
- Added support for BigTIFF
- Added importer plugin option to open each plane in a new window
- Added MS Video 1 codec for AVI
5.106 2007 Oct 1

- Added support for compressed DICOM images
- Added support for uncompressed LIM files
- Added support for Adobe Photoshop PSD files
- Fixed bugs in DICOM, OME-TIFF, Leica LIF, Zeiss ZVI, Visitech, PerkinElmer and Metamorph
- Improved indexed color support
- Addressed several efficiency issues
- Fixed how multiple series are handled in 4D data browser
- Added option to reorder stacks in importer plugin
- Added option to turn off autoscaling in importer plugin
- Additional metadata convenience methods

5.107 2007 Sept 11

- Major improvements to ND2 support; lossless compression now supported
- Support for indexed color images
- Added support for Simple-PCI .cxd files
- Command-line OME-XML validation
- Bugfixes in most readers, especially Zeiss ZVI, Metamorph, PerkinElmer and Leica LEI
- Initial version of Bio-Formats macro extensions for ImageJ

5.108 2007 Aug 1

- Added support for latest version of Leica LIF
- Fixed several issues with Leica LIF, Zeiss ZVI
- Better metadata mapping for Zeiss ZVI
- Added OME-TIFF writer
- Added MetadataRetrieve API for retrieving data from a MetadataStore
- Miscellaneous bugfixes
5.109 2007 July 16

- Fixed several issues with ImageJ plugins
- Better support for Improvision and Leica TCS TIFF files
- Minor improvements to Leica LIF, ICS, QuickTime and Zeiss ZVI readers
- Added searchable metadata window to ImageJ importer

5.110 2007 July 2

- Fixed issues with ND2, Openlab LIFF and Slidebook
- Added support for Visitech XYS
- Added composite stack support to ImageJ importer

5.111 2007 June 18

- Fixed issues with ICS, ND2, MicroManager, Leica LEI, and FV1000 OIF
- Added support for large (> 2 GB) ND2 files
- Added support for new version of ND2
- Minor enhancements to ImageJ importer
- Implemented more flexible logging
- Updated automated testing framework to use TestNG
- Added package for caching images produced by Bio-Formats

5.112 2007 June 6

- Fixed OME upload/download bugs
- Fixed issues with ND2, EPS, Leica LIF, and OIF
- Added support for Khoros XV
- Minor improvements to the importer

5.113 2007 May 24

- Better Slidebook support
- Added support for Quicktime RPZA
- Better Leica LIF metadata parsing
- Added support for BioRad PIC companion files
- Added support for bzip2-compressed files
• Improved ImageJ plugins
• Native support for FITS and PGM

5.114 2007 May 2

• Added support for NRRD
• Added support for Evotec Flex (requires LuraWave Java SDK with license code)
• Added support for gzip-compressed files
• Added support for compressed QuickTime headers
• Fixed QuickTime Motion JPEG-B support
• Fixed some memory issues (repeated small array allocations)
• Fixed issues reading large (> 2 GB) files
• Removed “ignore color table” logic, and replaced with Leica-specific solution
• Added status event reporting to readers
• Added API to toggle metadata collection
• Support for multiple dimensions rasterized into channels
• Deprecated reader and writer methods that accept the ‘id’ parameter
• Deprecated IFormatWriter.save in favor of saveImage and saveBytes
• Moved dimension swapping and min/max calculation logic to delegates
• Separate GUI logic into isolated loci.formats.gui package
• Miscellaneous bugfixes and tweaks in most readers and writers
• Many other bugfixes and improvements

5.115 2007 Mar 16

• Fixed calibration bugs in importer plugin
• Enhanced metadata support for additional formats
• Fixed LSM bug

5.116 2007 Mar 7

• Added support for Micro-Manager file format
• Fixed several bugs – Leica LIF, Leica LEI, ICS, ND2, and others
• Enhanced metadata support for several formats
• Load series preview thumbnails in the background
• Better implementation of openBytes(String, int, byte[]) for most readers
• Expanded unit testing framework
5.117 2007 Feb 28

- Better series preview thumbnails
- Fixed bugs with multi-channel Leica LEI
- Fixed bugs with “ignore color tables” option in ImageJ plugin

5.118 2007 Feb 26

- Many bugfixes: Leica LEI, ICS, FV1000 OIB, OME-XML and others
- Better metadata parsing for BioRad PIC files
- Enhanced API for calculating channel minimum and maximum values
- Expanded MetadataStore API to include more semantic types
- Added thumbnails to series chooser in ImageJ plugin
- Fixed plugins that upload and download from an OME server

5.119 2007 Feb 7

- Added plugin for downloading images from OME server
- Improved HTTP import functionality
- Added metadata filtering – unreadable metadata is no longer shown
- Better metadata table for multi-series datasets
- Added support for calibration information in Gatan DM3
- Eliminated need to install JAI Image I/O Tools to read ND2 files
- Fixed ZVI bugs: metadata truncation, and other problems
- Fixed bugs in Leica LIF: incorrect calibration, first series labeling
- Fixed memory bug in Zeiss LSM
- Many bugfixes: PerkinElmer, DeltaVision, Leica LEI, LSM, ND2, and others
- IFormatReader.close(boolean) method to close files temporarily
- Replaced Compression utility class with extensible Compressor interface
- Improved testing framework to use .bioformats configuration files
5.120 2007 Jan 5

- Added support for Prairie TIFF
- Fixed bugs in Zeiss LSM, OIB, OIF, and ND2
- Improved API for writing files
- Added feature to read files over HTTP
- Fixed bugs in automated testing framework
- Miscellaneous bugfixes

5.121 2006 Dec 22

- Expanded ImageJ plugin to optionally use Image5D or View5D
- Improved support for ND2 and JPEG-2000 files
- Added automated testing framework
- Fixed bugs in Zeiss ZVI reader
- Miscellaneous bugfixes

5.122 2006 Nov 30

- Added support for ND2/JPEG-2000
- Added support for MRC
- Added support for MNG
- Improved support for floating-point images
- Fixed problem with 2-channel Leica LIF data
- Minor tweaks and bugfixes in many readers
- Improved file stitching logic
- Allow ImageJ plugin to be called from a macro

5.123 2006 Nov 2

- Bugfixes and improvements for Leica LIF, Zeiss LSM, OIF and OIB
- Colorize channels when they are split into separate windows
- Fixed a bug with 4-channel datasets
5.124 2006 Oct 31

- Added support for Imaris 5 files
- Added support for RGB ICS images

5.125 2006 Oct 30

- Added support for tiled TIFFs
- Fixed bugs in ICS reader
- Fixed importer plugin deadlock on some systems

5.126 2006 Oct 27

- Multi-series support for Slidebook
- Added support for Alicona AL3D
- Fixed plane ordering issue with FV1000 OIB
- Enhanced dimension detection in FV1000 OIF
- Added preliminary support for reading NEF images
- Added option to ignore color tables
- Fixed ImageJ GUI problems
- Fixed spatial calibration problem in ImageJ
- Fixed some lingering bugs in Zeiss ZVI support
- Fixed bugs in OME-XML reader
- Tweaked ICS floating-point logic
- Fixed memory leaks in all readers
- Better file stitching logic

5.127 2006 Oct 6

- Support for 3i SlideBook format (single series only for now)
- Support for 16-bit RGB palette TIFF
- Fixed bug preventing import of certain Metamorph STK files
- Fixed some bugs in PerkinElmer UltraView support
- Fixed some bugs in Leica LEI support
- Fixed a bug in Zeiss ZVI support
- Fixed bugs in Zeiss LSM support
- Fixed a bug causing slow identification of Leica datasets
• Fixed bugs in the channel merging logic
• Fixed memory leak for OIB format
• Better scaling of 48-bit RGB data to 24-bit RGB
• Fixed duplicate channels bug in “open each channel in a separate window”
• Fixed a bug preventing PICT import into ImageJ
• Better integration with HandleExtraFileTypes
• Better virtual stack support in Data Browser plugin
• Fixed bug in native QuickTime random access
• Keep aspect ratio for computed thumbnails
• Much faster file stitching logic

5.128  2006 Sep 27

• PerkinElmer: support for PE UltraView
• Openlab LIFF: support for Openlab v5
• Leica LEI: bugfixes, and support for multiple series
• ZVI, OIB, IPW: more robust handling of these formats (eliminated custom OLE parsing logic in favor of Apache POI)
• OIB: better metadata parsing (but maybe still not perfect?)
• LSM: fixed a bug preventing import of certain LSMs
• Metamorph STK: fixed a bug resulting in duplicate image planes
• User interface: use of system look & feel for file chooser dialog when available
• Better notification when JAR libraries are missing

5.129  2006 Sep 6

• Leica LIF: multiple distinct image series within a single file
• Zeiss ZVI: fixes and improvements contributed by Michel Boudinot
• Zeiss LSM: fixed bugs preventing the import of certain LSM files
• TIFF: fixed a bug preventing import of TIFFs created with Bio-Rad software
5.130 2006 Mar 31

- First release
6.1 Before filing a bug report

If you think you have found a bug in Bio-Formats, the first thing to do is update your version of Bio-Formats to the latest version to check if the problem has already been addressed. The Fiji updater will automatically do this for you, while in ImageJ you can select Plugins → Bio-Formats → Update Bio-Formats Plugins.

You can also download the latest version of Bio-Formats from the OME website.

6.2 Common issues to check

- If you get an error message similar to:

  ```
  java.lang.UnsupportedClassVersionError: loci/plugins/LociImporter : 
  Unsupported major.minor version 52.0
  ```

  This plugin requires Java 1.8 or later.

  you need to upgrade your system Java version to Java 8 or above, or download a new version of ImageJ/Fiji bundled with Java 8.

- If your 12, 14 or 16-bit images look all black when you open them, typically the problem is that the pixel values are very, very small relative to the maximum possible pixel value (4095, 16383, and 65535, respectively), so when displayed the pixels are effectively black. In ImageJ/Fiji, this is fixable by checking the “Autoscale” option; with the command line tools, the “-autoscale -fast” options should work.

- If the file is very, very small (4096 bytes) and any exception is generated when reading the file, then make sure it is not a Mac OS X resource fork. The ‘file’ command should tell you:

  ```
  $ file /path/to/suspicious-file
  suspicious-file: AppleDouble encoded Macintosh file
  ```

- If you get an OutOfMemory or NegativeArraySize error message when attempting to open an SVS or JPEG-2000 file then the amount of pixel data in a single image plane exceeds the amount of memory allocated to the JVM or 2 GB, respectively. For the former, you can increase the amount of memory allocated; in the latter case, you will need to open the image in sections. If you are using Bio-Formats as a library, this means using the openBytes(int, int, int, int, int) method in loci.formats.IFormatReader. If you are using Bio-Formats within ImageJ, you can use the Crop on import option.

  Note that JPEG-2000 is a very efficient compression algorithm - thus the size of the file on disk will be substantially smaller than the amount of memory required to store the uncompressed pixel data. It is not uncommon for
Bio-Formats, Release 6.14.0

a JPEG-2000 or SVS file to occupy less than 200 MB on disk, and yet have over 2 GB of uncompressed pixel data.

6.3 Sending a bug report

If you can still reproduce the bug after updating to the latest version of Bio-Formats, and your issue does not relate to anything listed above or noted on the relevant file format page, please send a bug report to the forums. You can upload sample files to Zenodo, or for files over 50 GB, we can provide you with an FTP server address.

To ensure that any inquiries you make are resolved promptly, please include the following information:

• **Exact error message.** Copy and paste any error messages into the text of your email. Alternatively, attach a screenshot of the relevant windows.

• **Version information.** Indicate which release of Bio-Formats, which operating system, and which version of Java you are using.

• **Non-working data.** If possible, please send a non-working file. This helps us ensure that the problem is fixed for next release and will not reappear in later releases. Note that any data provided is used for internal testing only; we do not make images publicly available unless given explicit permission to do so.

• **Metadata and screenshots.** If possible, include any additional information about your data. We are especially interested in the expected dimensions (width, height, number of channels, Z slices, and timepoints). Screenshots of the image being successfully opened in other software are also useful.

• **Format details.** If you are requesting support for a new format, we ask that you send as much data as you have regarding this format (sample files, specifications, vendor/manufacturer information, etc.). This helps us to better support the format and ensures future versions of the format are also supported.

**Please be patient** - it may be a few days until you receive a response, but we reply to *every* email inquiry we receive.
### Symbols

- `.1sc`, 123
- `.2`, 190
- `.2fl`, 205
- `.3`, 190
- `.4`, 190
- `.acff`, 212
- `.afi`, 117
- `.afm`, 196
- `.aim`, 112
- `.al3d`, 113
- `.ali`, 174
- `.am`, 114
- `.amiramesh`, 114
- `.apl`, 179
- `.arf`, 120
- `.avi`, 119
- `.bif`, 210
- `.bin`, 193
- `.bip`, 160
- `.bmp`, 143, 213
- `.btf`, 204
- `.c01`, 129
- `.cfg`, 195
- `.ch5`, 129
- `.cif`, 115
- `.cr2`, 128
- `.crw`, 128
- `.csv`, 188
- `.cxd`, 200
- `.czi`, 217
- `.dat`, 155, 183, 207
- `.db`, 203
- `.dcm`, 133
- `.dib`, 129
- `.dicom`, 133
- `.dm2`, 138
- `.dm3`, 138
- `.dm4`, 138
- `.dli`, 210
- `.dv`, 132
- `.eps`, 134
- `.epsi`, 134
- `.exp`, 121
- `.fdf`, 208
- `.fff`, 145
- `.ffr`, 205
- `.fits`, 137
- `.flex`, 135
- `.fli`, 160
- `.frm`, 151
- `.gel`, 114
- `.gif`, 140
- `.grey`, 114
- `.h5`, 123
- `.hdf`, 209
- `.hdr`, 115, 153, 176, 207
- `.hed`, 147
- `.his`, 141
- `.htd`, 119, 169
- `.html`, 211
- `.hx`, 114
- `.i2i`, 143
- `.ics`, 144
- `.ids`, 144
- `.im3`, 189
- `.img`, 115, 128, 136, 147, 155, 176
- `.ims`, 126
- `.inr`, 152
- `.ipl`, 154
- `.ipm`, 155
- `.ipw`, 146
- `.j2k`, 157
- `.jp2`, 157
- `.jpf`, 157
- `.jpg`, 143, 156, 206
- `.jpk`, 158
- `.jpx`, 158
- `.klb`, 159
- `.l2d`, 166
- `.labels`, 114
- `.lei`, 162
- `.lif`, 162
- `.liff`, 148

---

555
showinf command line option, 22
-bigtiff
  bfconvert command line option, 25
-cache
  bfconvert command line option, 26
  showinf command line option, 22
-cache-dir
  bfconvert command line option, 26
  showinf command line option, 23
-channel
  bfconvert command line option, 24
-columns
  mkfake command line option, 31
-compression
  bfconvert command line option, 25
-crop
  bfconvert command line option, 24
  showinf command line option, 21
-debug
  mkfake command line option, 31
  showinf command line option, 22
-fast
  showinf command line option, 22
-fields
  mkfake command line option, 31
-fill
  bfconvert command line option, 27
  showinf command line option, 23
-format
  showinf command line option, 23
-help
  formatlist command line option, 29
-html
  formatlist command line option, 29
-no-sas
  bfconvert command line option, 26
-no-sequential
  bfconvert command line option, 27
-no-upgrade
  showinf command line option, 22
-nobigtiff
  bfconvert command line option, 26
-nocore
  showinf command line option, 22
-noflat
  bfconvert command line option, 23
  showinf command line option, 21
-nolookup
  bfconvert command line option, 25
-nooverwrite
  bfconvert command line option, 25
-nopix
  showinf command line option, 21
-novalid
  showinf command line option, 22
-
-omexml
  showinf command line option, 21
-omexml-only
  showinf command line option, 22
-option
  bfconvert command line option, 23
  showinf command line option, 21
-overwrite
  bfconvert command line option, 25
-padded
  bfconvert command line option, 26
-plates
  mkfake command line option, 30
-pyramid-resolutions
  bfconvert command line option, 26
-pyramid-scale
  bfconvert command line option, 26
-range
  bfconvert command line option, 24
  showinf command line option, 21
-rows
  mkfake command line option, 31
-runs
  mkfake command line option, 30
-series
  bfconvert command line option, 24
  showinf command line option, 21
-swap
  bfconvert command line option, 27
  showinf command line option, 23
-tilex
  bfconvert command line option, 24
-tiley
  bfconvert command line option, 24
-timepoint
  bfconvert command line option, 24
-txt
  formatlist command line option, 29
-xml
  formatlist command line option, 29
-z
  bfconvert command line option, 24
3i SlideBook, 110
3i SlideBook 7, 111

A
Adobe Photoshop PSD, 191
AIM, 112
Alicona 3D, 113
Amersham Biosciences Gel, 114
Amira Mesh, 114
Amnis FlowSight, 115
Analyze 7.5, 115
Andor Bio-Imaging Division (ABD) TIFF, 112
Bio-Formats, Release 6.14.0

Andor SIF, 116
Animated PNG, 117
Aperio AFI, 117
Aperio SVS TIFF, 118
Applied Precision CellWorX, 119
AVI (Audio Video Interleave), 119
Axon Raw Format, 120

B

BD Pathway, 121
Becker & Hickl SPC FIFO, 121
Becker & Hickl SPCImage, 122
BF_CP, 32, 32
BF_DEVEL, 33
BF_FLAGS, 32, 32
BF_MAX_MEM, 32, 32
BF_PROFILE, 32, 32, 33
BF_PROFILE_DEPTH, 32, 32
bfconvert, 31
bfconvert command line option
  -bigtiff, 25
  -cache, 26
  -cache-dir, 26
  -channel, 24
  -compression, 25
  -crop, 24
  -fill, 27
  -no-sas, 26
  -no-sequential, 27
  -nobigtiff, 26
  -noflat, 23
  -nolookup, 25
  -nooverwrite, 25
  -option, 23
  -overwrite, 25
  -padded, 26
  -pyramid-resolutions, 26
  -pyramid-scale, 26
  -range, 24
  -series, 24
  -swap, 27
  -tilex, 24
  -tiley, 24
  -timepoint, 24
  -z, 24

Big Data Viewer, 123
Bio-Rad Gel, 123
Bio-Rad PIC, 124
Bio-Rad SCN, 125
Bitplane Imaris, 126
Bruker MRI, 127
BSD, 110
Burleigh, 128

C

Canon DNG, 128
CellHS, 129
CellSens VSI, 130
CellVoyager, 131
CLASSPATH, 57, 63, 66–68, 71, 72, 90
CV7000, 131

D

DeltaVision, 132
DICOM, 133
domainlist, 32

E

ECAT7, 134
environment variable
  CLASSPATH, 57, 63, 66–68, 71, 72, 90

EPS (Encapsulated PostScript), 134
Evotec/PerkinElmer Opera Flex, 135
Export, 110

F

FEI, 136
FEI TIFF, 136
FITS (Flexible Image Transport System), 137
formatlist, 32

G

Gatan Digital Micrograph, 138
Gatan Digital Micrograph 2, 138

GIF (Graphics Interchange Format), 140

H

Hamamatsu Aquacosmos NAF, 140
Hamamatsu HIS, 141
Hamamatsu ndpi, 141
Hamamatsu VMS, 142
Hitachi S-4800, 143

I

I2I, 143
ICS (Image Cytometry Standard), 144
ijview, 31
Imacon, 145
ImagePro Sequence, 145
ImagePro Workspace, 146
IMAGIC, 147
IMOD, 148
Improvision Openlab LIFF, 148
Improvision Openlab Raw, 149
Improvision TIFF, 149
ImSpecor OBf, 150
InCell 1000/2000, 151
InCell 3000, 151
INR, 152
Inveon, 153
Ionpath MIBI, 153
IPLab, 154
IVision, 155
JEOL, 155
JPEG, 156
JPEG 2000, 157
JPX, 158
K
Keller Lab Block, 159
Khoros VIFF (Visualization Image File Format) Bitmap, 159
Kodak BIP, 160
L
Lambert Instruments FLIM, 160
LaVision ImSpecor, 161
Leica LAS AF LIF (Leica Image File Format), 162
Leica LCS LEI, 162
Leica LOF, 164
Leica SCN, 164
Leica XLEF, 165
LEO, 166
Li-Cor L2D, 166
LIM (Laboratory Imaging/Nikon), 167
Minolta MRW, 172
mkfake, 32
mkfake command line option
-columns, 31
-debug, 31
-fields, 31
-plates, 30
-rows, 31
-runs, 30
MNG (Multiple-image Network Graphics), 173
Molecular Imaging, 173
MRC (Medical Research Council), 174
Multiple Images, 110
N
NEF (Nikon Electronic Format), 175
NIfTI, 176
Nikon Elements TIFF, 176
Nikon EZ-C1 TIFF, 177
Nikon NIS-Elements ND2, 178
NRRD (Nearly Raw Raster Data), 178
O
Olympus CellR/APL, 179
Olympus Fluoview FV1000, 180
Olympus Fluoview TIFF, 181
Olympus OIR, 182
Olympus OMP2INFO, 182
Olympus ScanR, 183
Olympus SIS TIFF, 183
OME-TIFF, 184
OME-XML, 185
OMERO Pyramid, 185
Openness, 110
Oxford Instruments, 186
P
PCORAW, 187
PCX (PC Paintbrush), 187
Perkin Elmer Densitometer, 188
PerkinElmer Columbus, 188
PerkinElmer Nuance, 189
PerkinElmer Operetta, 190
PerkinElmer UltraVIEW, 190
Photoshop TIFF, 192
PicoQuant Bin, 193
PICT (Macintosh Picture), 193
Pixels, 110
PNG (Portable Network Graphics), 194
Portable Any Map, 191
Prairie Technologies TIFF, 195
Presence, 110
Princeton Instruments SPE, 195
Pyramid, 110
Q
Quesant, 196
QuickTime Movie, 196

R
Ratings legend and definitions, 109
RHK, 198

S
SBIG, 199
Seiko, 199
showinf, 31
showinf command line option
  -autoscale, 22
  -cache, 22
  -cache-dir, 23
  -crop, 21
  -debug, 22
  -fast, 22
  -fill, 23
  -format, 23
  -no-upgrade, 22
  -nocore, 22
  -noflat, 21
  -nopix, 21
  -novalid, 22
  -omexml, 21
  -omexml-only, 22
  -option, 21
  -range, 21
  -series, 21
  -swap, 23
SimplePCI & HCImage, 200
SimplePCI & HCImage TIFF, 200
SM Camera, 201
SPIDER, 202

T
Targa, 202
Tecan Spark Cyto Workspace, 203
Text, 203
TIFF (Tagged Image File Format), 204
tiffcomment, 32
TillPhotonics TillVision, 205
Topometrix, 205
Trestle, 206

U
UEM, 207
Unisoku, 207
Utility, 110

V
Varian FDF, 208
Vectra QPTIFF, 208
Veeco AFM, 209
Ventana BIF, 210
VG SAM, 210
VisiTech XYS, 211
Volocity, 211
Volocity Library Clipping, 212

W
WA-TOP, 213
Windows Bitmap, 213
Woolz, 214

X
xmlindent, 32
xmlvalid, 32

Z
Zeiss Axio CSM, 215
Zeiss AxioVision TIFF, 215
Zeiss AxioVision ZVI (Zeiss Vision Image), 216
Zeiss CZI, 217
Zeiss LSM (Laser Scanning Microscope)
  510/710, 218