Systems Biology Format Converter

User Manual

by the SBFC team

7th March 2016

Contents

List of Figures

1 Introduction

This chapter provides the end user with an overwiew of the software Systems Biology Format Converter (SBFC), with particular focus on the installation process and use cases. You can also consult SBFC wiki for additional details: http://sbfc.sourceforge.net.
The idea behind SBFC is to offer an integrated and easy way to convert Systems Biology model formats. Interoperability between formats is a recurring issue in Systems Biology. Most of the available tools for converting computational models from one format to another have been developed independently. This fragmentation highlights the need for a tool combining the existing multiple converters and providing a solid framework for implementating additional format converters. SBFC represents a generic framework that potentially allows any conversion between two formats. The framework is written in Java and can be used as a standalone executable or as web service. The SBFC framework currently supports conversion from SBML to BioPAX Level 2 and Level3, SBGN-ML, Matlab, Octave, XPP, Dot, and APM. An overview for the software package SBFC is shown in Figure 1. SBFC is free software; you can redistribute it and/or modify it under the terms of the *GNU Lesser General Public License* as published by the Free Software Foundation; either version 2.1 of the License, or any later version. This is a collaborative project and we hope that developers will provide support for more formats by creating new modules.

2 Requirements

To use SBFC as a standalone software, the following software must be installed:

- Java[1] 6+.

The SBFC web service[2], was tested with the following browsers and is currently compatible as follows:

- Google Chrome[3] up to version 44. Both GNU/Linux and Windows are supported;
- Mozilla Firefox[4] up to version 40. Both GNU/Linux and Windows are supported;
- Internet Explorer[5]. Limited support.

3 Download and Installation

You can find and download the package SBFC from the SourceForge project page:
http://sbfc.sourceforge.net/.
In case you discover a bug within SBFC or request new features, please let us know. There are two dedicated trackers:

- https://sourceforge.net/p/sbfc/bugs/ for bug reports; and
- https://sourceforge.net/p/sbfc/feature-requests/ for feature requests.

For general discussions about SBFC, please use the forum: sbfc-forum@googlegroups.com.
If you need further support or you wish to develop new modules in SBFC you can contact the team using the following e-mail address: sbfc-devel@googlegroups.com.

3.1 Using SBFC as standalone package

Download sbfc-x.y.z.zip or the specific converter of interest from the SBFC webpage in SourceForge. After downloading and uncompressing the package, you can put the extracted folder where you prefer. The sbfc folder contain some scripts that help you to launch conversion on the command line as well as a gui.

[1]See https://java.com
[2]See http://www.ebi.ac.uk/biomodels/tools/converters/
[3]See http://www.google.com/chrome/
[4]See https://www.mozilla.org/en-GB/firefox/
[5]See http://windows.microsoft.com/en-GB/internet-explorer/download-ie

3.2 Using SBFC within a program

Download sbfc-x.y.z.zip or the specific converter of interest from the SBFC web page in the SourceForge website. After downloading and uncompressing the package, you should put the file sbfc-x.y.z.jar in the library folder of your program and include this file in the program library path required for compilation. If you have downloaded the standalone version of SBFC (contain '-standalone' in the jar files), you do not need to do anything else. Otherwise, you should also install the dependencies libraries required by SBFC.

4 How to use SBFC

This section explains how to use the software SBFC, providing the user with some use cases.

4.1 Conversion using a console

After unpacking the file sbfc-x.y.z.zip, you will find several shell script files (.sh for Linux/OSX, .bat for windows). These file scripts are used for launching conversion jobs. To test the correct installation of SBFC, a set of **model examples** is available in the folder *examples*:

- examples/SBML.xml : is a very simple model in SBML
- examples/BioPAX.owl : is a very simple model in BioPAX

To retrieve the list of currently supported input models (Java Class names), run the command:

```
./sbfModelList.sh (sbfModelList.bat under windows)

# Example of output
APMModel
BioPAXModel
DotModel
OctaveModel
SBGNMLStringModel
SBGNModel
SBMLModel
XPPModel
```

To retrieve the list of the currently supported converters (Java Class names), run the command:

```
./sbfConverterList.sh (sbfConverterList.bat under windows)

# Example of output
Miriam registry file = Miriam.xml
BioPAXL3Converter
SBML2APM
SBML2BioPAX_l2
SBML2BioPAX_l3
SBML2Dot
SBML2Matlab
SBML2Octave
SBML2SBGNML
SBML2SBML
SBML2XPP
URL2URN
URN2URL
```

For a detailed description of the available converters, please see Section 5.

To convert a model file (or the models in a folder), run the command:

```
./sbfConverter.sh [InputModelClassName] [ConverterClassName] [modelFile | modelsFolder]
```

For instance, to convert an SBML model (e.g. SBML.xml) to an Octave model the command will be:

```
./sbfConverter.sh SBMLModel SBML2Octave examples/SBML.xml
```

A script file is also present for certain converters to simplify the command. To do the previous conversion, a user can also type the following command:

```
./sbml2octave.sh examples/SBML.xml
```

4.2 Conversion using Java

To convert using Java directly in a terminal or in an IDE, you need to add all the jar files present in the folder to your CLASSPATH environment variable or add only the sbfc-x.y.z-standalone.jar file that contains all needed dependencies. Then you can:

- run the Converter class located in the package org.sbfc.converter . In this case the general syntax is:

```
java org.sbfc.converter.Converter [InputModelClassName] [ConverterClassName] [
    ModelFile]
```

  For instance:

```
java org.sbfc.converter.Converter SBMLModel SBML2Octave examples/example/SBML.xml
```

- Alternatively, invoke one of the three following methods in the Converter class from your code directly:

```
/* Input model given as a string */
public static String ConvertFromString([InputModelClassName], [ConverterClassName],
    [ModelString])
/* Input model given as a file, output file path generated automatically */
public static String ConvertFromFile([InputModelClassName], [ConverterClassName], [
    InputFilePath])
/* Input model given as a file, output file path specified */
public static String ConvertFromFile([InputModelClassName], [ConverterClassName], [
    InputFilePath], [OutputFilePath])
```

4.3 Conversion using a GUI

If you don't need to convert a large number of files then instead of using directly the command line or some scripts, you can launch some conversion using a simple GUI:

- run the ConverterGUI class located in the package org.sbfc.converter . In this case the general syntax is:

```
java -jar sbfc-x.y.z-standalone.jar org.sbfc.converter.ConverterGUI
```

  If your system is configured to execute jar files, you can also double click on the sbfc-x.y.z-standalone.jar from a file explorer directly to start the gui.

  Once the gui open, you have to specify your input file and the converter you want to use. Optionaly you can specify an output file. If you don't specify one, an automatic output file path will be generated automatically, it will be located on the same folder as the input file. Then you just need to click on the 'Convert' button to start the conversion. If you let the 'open output file' checkbox selected, at the end of the conversion a new window will open showing the content of the output file. If you unchecked the checkbox, you will get a small dialog window to tell you once the conversion is done.

- Alternatively, invoke the following method in the ConverterGUI class from your code directly:

```
ConverterGUI.getConverterGuiInstance().setVisible(true);
```

5 Overview of the available converters

This section contains the complete list of converters available in SBFC. All the current converters use SBML as input model format. SBML is a machine-readable format for representing models and is the *de facto* standard for representing Systems Biology models. It is oriented towards describing systems where biological entities are involved in, and modified by, processes that occur over time. An example of this is a network of biochemical reactions. SBML framework is suitable for representing models commonly found in research on a number of topics, including cell signalling pathways, metabolic pathways, biochemical reactions, gene regulation, and many others. Below you can find the list of the currently available converters. For a graphical overview of how these converters are organised within the software package SBFC, see Figure 1. If you want to contribute with new converters, please see the Developer Manual. Each converter is described in detail in the following sections.

5.1 Core SBFC converters

This section lists the currently supported converters which are part of the SBFC core package.

- SBML2SBML : converts SBML (any level-version) to SBML (any level-version);
  - URN2URL : converts SBML urn:miriam annotation to SBML Identifiers.org url annotation;
  - URL2URN : converts SBML Identifiers.org url annotation to SBML urn:miriam annotation;
- SBML2BioPAX : converts SBML (any level) to BioPAX level 2 or 3;
- SBML2SBGNML : converts SBML (any level) to SBGN-ML;
- SBML2Matlab : converts SBML (any level) to Matlab;
- SBML2Octave : converts SBML (any level) to Octave;
- SBML2XPP : converts SBML (any level) to XPP;
- SBML2DOT : converts SBML (any level) to DOT;
- SBML2APM : converts SBML (any level) to APM;
- BioPAXL3Converter : converts BioPAX models level 1 or 2 to level 3.

5.2 Additional third-party SBFC converters

Currently, the Systems Biology community has been working on additional converters. Each of these converters developed by third-parties can use specific libraries and library versions which could be different and not fully compatible with the libraries currently required by SBFC. Therefore these converters are separate from the SBF core package.

- MDL2SBML : converts MDL to SBML;
- BioPAX2GPML : converts BioPAX to GPML;

- GPML2BioPAX : converts GPML to BioPAX;
- SBML2Antimony : converts SBML to Antimony (requires libAntimony);
- Antimony2SBML : converts Antimony to SBML (requires libAntimony);
- SBML2CellML : converts SBML to CellML (requires libAntimony + CellML API);
- CellML2SBML : converts CellML to SBML (requires libAntimony + CellML API).

5.3 Conversion from SBML to SBML, including annotation support

This converter translates between SBML models of different level/version, but can also be used for converting model annotation from URN:Miriam to Identifiers.org URL and viceversa.
MIRIAM is an effort to establish the essential, minimal set of information that is sufficient to annotate a model in such a way as to enable its reuse. This standard also describes the syntax by which annotations should be encoded: MIRIAM URIs. If you want to know more, you can read the Miriam FAQs.

Identifiers.org URIs are stable, perennial and globally unique URIs which can be used to annotate a wide range of entities or concepts from a plethora of fields, including proteins, diseases, publications and ontological terms. They can be used unchanged for a wide range of tasks, as they are both unique and directly usable online. The aim is to provide a community-agreed means to create stable, perennial and globally unique identifiers, which can be used to reference data with a primary focus on life sciences. Additionally, the system decouples the identification of records from the physical locations on the web where they can be retrieved. If you want to know more, you can read the Identifiers.org FAQs.

The conversion between different levels-versions of SBML is based on libsbml. Normally, models can be converted upward without difficulty (e.g., from SBML Level 1 to Level 2, or from an earlier Version of Level 2 to the latest Version of Level 2). Sometimes models can be translated downward as well, if they do not use constructs specific to more advanced Levels of SBML. For details about potential information loss on downward conversion, please consult this web page from the SBML website.
The conversion between urn:miriam and identifiers.org url is fully supported without any loss of information.

The converters are:

SBML2SBML : convert from an SBML model of a level and version to another;

URL2URN : convert SBML model annotation from identifiers.org url to miriam:urn;

URN2UR : convert SBML model annotation from miriam:urn to identifiers.org url.

To run SBML2SBML, URL2URN or URN2URL you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert SBML (any *.xml files if you put a folder) to another SBML level-version.
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .xml
# x and y are the SBML level and version of the output model, respectively.
./sbml2sbml.sh [file.xml | folder] LxVy

# Convert SBML model annotation from urn:miriam to identifiers.org url .
./urn2url.sh [file.xml | folder]

# Convert SBML model annotation from identifiers.org url to from urn:miriam .
./url2urn.sh [file.xml | folder]
```

5.4 Conversion from SBML to BioPAX

This converter translates from SBML model format to BioPAX Level 2 or 3 model format. The main objective of the BioPAX initiative is to develop a data exchange format for biological pathways that is flexible, extensible, optionally encapsulated and compatible with other standards and can be widely adopted in a timely manner.

The converter is:

SBML2BioPAX : convert from an SBML model to BioPAX.

To run SBML2BioPAX you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert SBML (any *.xml files if you put a folder) to BioPAX 2 AND 3
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .owl
# To convert specifically to either BioPAX 2 OR BioPAX 3, you can use the scripts:
    sbml2biopax2.sh or sbml2biopax3.sh.
./sbml2biopax.sh [file.xml | folder]
```

5.5 Conversion from SBML to SBGN ML

This converter translates from SBML model format to SBGN ML model format. The mission of the SBGN-ML project is to develop high quality, standard graphical languages for representing biological processes and interactions. To know more, you can read the SBGN-About page.
The converter is:

SBML2SBGNML : convert from an SBML model to SBGN ML.

To run SBML2SBGNM you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert SBML (any *.xml files if you put a folder) to SBGN-ML
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .xml
./sbml2sbgnml.sh [file.xml | folder]
```

5.6 Conversion from SBML to Matlab

This converter translates from SBML model format to Matlab model format. Matlab is a high-level language, primarily intended for numerical computations. It provides a convenient GUI and command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments.
The converter is:

SBML2Matlab : convert from SBML to Matlab.

The converter is based on JSBML, so support all the levels and versions of SBML until SBML level 3 version 1 core. Most of the mathematical parts of the SBML model are converted into a set of declarations or equations that can be executed in Octave or Matlab. The following SBML elements are not supported:

- delay
- constraint
- initialAssignment
- stoichiometryMath

The following information is lost in the conversion:

- compartment
- units
- sboTerm
- notes
- most annotations

To run SBML2Matlab you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert SBML (any *.xml files if you put a folder) to Matlab.
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .m
./sbml2matlab.sh [file.xml | folder]
```

Please note that the name of the file to be converted should not contain any dash (-) symbol, otherwise the converted m file might not be correctly opened by Matlab.

The generated m file using this converter works with Matlab. If you decide to execute it with Octave, you can either use the specific converter sbml2octave, or you can edit this generated file directly by commenting / uncommenting some lines in the source code using the Matlab/Octave comment symbol '%'. In the latter case, you have to comment the Matlab code and uncomment the Octave code for:

- the ODE solver method; and
- for the f function signature

as shown below.

ODE solver call:

```
% Depending on whether you are using Octave or Matlab,
% you should comment / uncomment one of the following blocks.
% This should also be done for the definition of the function f below.
% Start Matlab code
%       tspan=[0:0.01:100];
%       opts = odeset('AbsTol',1e-3);
%       [t,x]=ode23tb(@f,tspan,x0,opts);
% End Matlab code

% Start Octave code
        t=linspace(0,100,100);
        x=lsode('f',x0,t);
% End Octave code
```

f function signature:

```
% Depending on whether you are using Octave or Matlab,
% you should comment / uncomment one of the following blocks.
% This should also be done for the definition of the function f below.
% Start Matlab code
%function xdot=f(t,x)
% End Matlab code

% Start Octave code
function xdot=f(x,t)
% End Octave code
```

5.7 Conversion from SBML to Octave

This converter translates from SBML model format to Octave model format. GNU Octave is a high-level language, primarily intended for numerical computations. It provides a convenient command line interface for solving linear and nonlinear problems numerically, and for performing other numerical experiments using a language that is mostly compatible with Matlab.
The converter is:

SBML2Octave : convert from SBML to Octave.

The converter is based on JSBML, so support all the levels and versions of SBML until SBML level 3 version 1 core. Most of the mathematical parts of the SBML model are converted into a set of declarations or equations that can be executed in Octave or Matlab. The following SBML elements are not supported:

- delay
- constraint
- initialAssignment
- stoichiometryMath

The following information is lost in the conversion:

- compartment
- units
- sboTerm
- notes
- most annotations

To run SBML2Octave you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
    # Convert SBML (any *.xml files if you put a folder) to Octave.
    # The output file(s) will be in the same folder as the SBML file(s) with an
        extension .m
    ./sbml2octave.sh [file.xml | folder]
```

The generated m file using this converter works with Octave. If you decide to execute it with Matlab, you can either use the specific converter sbml2matlab, or you can edit this generated file directly by commenting / uncommenting some lines in the source code using the Matlab/Octave comment symbol '%'. In the latter case, you have to comment the Octave code and uncomment the Matlab code for:

- the ODE solver method; and
- for the f function signature

as shown below.

ODE solver call:

```
% Depending on whether you are using Octave or Matlab,
% you should comment / uncomment one of the following blocks.
% This should also be done for the definition of the function f below.
% Start Matlab code
        tspan=[0:0.01:100];
        opts = odeset('AbsTol',1e-3);
        [t,x]=ode23tb(@f,tspan,x0,opts);
% End Matlab code
% Start Octave code
%       t=linspace(0,100,100);
%       x=lsode('f',x0,t);
% End Octave code
```

f function signature:

```
% Depending on whether you are using Octave or Matlab,
% you should comment / uncomment one of the following blocks.
% This should also be done for the definition of the function f below.
% Start Matlab code
function xdot=f(t,x)
% End Matlab code

% Start Octave code
%function xdot=f(x,t)
% End Octave code
```

5.8 Conversion from SBML to XPP

This converter translates from SBML model format to XPP model format. XPP-Aut is a numerical analysis software. It permits to solve differential equations, difference equations, delay equations, functional equations, boundary value problems, and stochastic equations.
The converter is:

SBML2XPP : convert from SBML to XPP.

The converter is based on JSBML, so support all the levels and versions of SBML until SBML level 3 version 1 core. Most of the mathematical parts of the SBML model are converted into a set of declarations or equations that can be executed in XPP.
Here is what is translated in the ode file:

- all SBML ids are changed to comply to the XPP rules. In XPP id should be 9 character long maximum (may be not true with the latest versions of XPP) and we have to put the local parameter in the same id space as all the others.
- in general we put a comment for each SBML element that we include in the ode file
- we declare several functions used in the MathML expression and that do not exist by default in XPP, like root, power,..
- the compartment elements as para or init
- the parameters elements as para or init (global parameters as well as local paramaters).
- the species elements as para or init
- d species/dt=ODE (if this species actually be affected by reaction/kineticLaw) (ODE should be concentration/time, but kineticLaw usually be substance/time, depending on species unit) (in ODE all the reaction/kineticLaw will be represented as RXNindex)
- rules
- reactions/kineticLaws
- events
- we set several XPP values to be able to simulate the model
- done (the word "done" as to be written at the end of an Xpp ode file)

SBML elements not supported

- delayed event
- constraint
- initialAssignment
- stoichiometryMath
- some math operators allowed in SBML are not supported by XPP

Information lost in the conversion

- compartment
- units
- sboTerm
- notes
- most annotations

To run SBML2XPP you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# To convert SBML (any *.xml files if you put a folder) to XPP
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .xpp
./sbml2xpp.sh [file.xml | folder]
```

5.9 Conversion from SBML to DOT

This converter translates from SBML model format to DOT model format. DOT is the language used to encode the graphics processed by GraphViz. DOT is a plain text graph description language. It is a simple way of describing graphs that both humans and computer programs can use. Various programs can process DOT files. Some, like OmniGraffle, dot, neato, twopi, circo, fdp, and sfdp, will read a DOT file and render it in graphical form. Others, like gvpr, gc, accyclic, ccomps, sccmap, and tred, will read a DOT file and perform calculations on the represented graph. Finally, others, like GVedit, KGraphEditor, lefty, dotty, and grappa, provide an interactive interface. Most programs are part of the GraphViz package or use it internally. (definition from the DOT Article on wikipedia).
The converter is:

SBML2DOT : convert from SBML to DOT.

The converter will first generate a dot file from a SBML file and then will use the dot program from graphviz to generate an svg, png file or any output format supported by dot. All the SBML species will be transformed into a dot node with a shape corresponding to their SBGN closest glyph (Process Description L1). The SBGN glyph is determined using either the sboTerm attribute of the species or using the MIRIAM annotations of the species, both should point to the same SBGN glyph but in case of uncertainty, the sboTerm is being used over the annotations. To know more about these different points, you would have to read the SBML and SBGN specifications. In particular, the section 5 of the SBML level 3 specifications for explanations of how to use and interpret the sboTerm attribute on SBML elements and the section 6 to know more about annotations of SBML elements. The link between an sboTerm and a SBGN glyph is made directly in the SBGN Process Description L1 specifications. The link between the Miriam datatypes and a SBGN glyph is made through a series of constraint define in a XML file that will not be described here. Then all the reactions will be transformed as well into a dot node with a shape corresponding to their SBGN closest glyph. To finish, we use the list of reactants, products and modifers of each reactions to create some edges from the reaction nodes to the species nodes. The shape of the arrow are again determined using the sboTerm attribute of the speciesReferences. The resulting graph is not SBGN compliant, the dot language does not allow us to define easily all the SBGN glyph. The anchor of the edges are not at the proper place as well.
SBML elements not supported

- most of them, only species and reaction are converted

To run SBML2DOT you need java 1.5 or higher and GraphViz (to generate other output than '.dot'). After unpacking the zipped file, move to the folder that have been created and run:

```
# To convert SBML (any *.xml files if you put a folder) to dot directly
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .dot
./sbml2dot.sh [file.xml | folder]
```

5.10 Conversion from SBML to APM

This converter translates from SBML model format to APM model format. APM (APMonitor Modeling Language) is an optimization software for mixed-integer and differential algebraic equations. It is coupled with large-scale solvers for linear, quadratic, nonlinear, and mixed integer programming (LP, QP, NLP, MILP, MINLP). Modes of operation include data reconciliation, real-time optimization, dynamic simulation, and nonlinear predictive control. It is freely available through MATLAB, Python, or from a web browser interface.
The converter is:

SBML2APM : convert from SBML to APM.

The converter is based on JSBML, so support all the levels and versions of SBML until SBML level 3 version 1 core. To run SBML2APM you need java 1.5 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert SBML (any *.xml files if you put a folder) to APM.
# The output file(s) will be in the same folder as the SBML file(s) with an
    extension .apm
./sbml2apm.sh [file.xml | folder]
```

5.11 Conversion from BioPAX L1/2 to BioPAX L3

This converter upgrades a BioPAX model from Level 1 or 2 to Level 3. The main objective of the BioPAX initiative is to develop a data exchange format for biological pathways that is flexible, extensible, optionally encapsulated and compatible with other standards and can be widely adopted in a timely manner.

The converter is:

BioPAXL3Converter : Convert a model from BioPAX L1/L2 to BioPAX L3.

To run BioPAXL3Converter you need java 1.6 or higher. After unpacking the zipped file, move to the folder that have been created and run:

```
# Convert BioPAX L1/L2 (any *owl files if you put a folder) to BioPAX L3
# The output file(s) will be in the same folder with extension .owl
./sbfConverter.sh BioPAXModel BioPAXL3Converter [file.owl | folder]
```

5.12 Conversion from MDL to SBML

This converter translates from MDL model format to SBML model format. MDL (Model Description Language) is a language for modelling large multi-scale models for the program MCell. MCell (Monte Carlo cell) is a program that uses spatially realistic 3-D cellular models and specialized Monte Carlo algorithms to simulate the movements and reactions of molecules within and between cells-cellular microphysiology.

This converter was written by David Tolnay and aims to translate an MDL model into SBML using libsbml-5.8.0 using the spatial package included with this version of libsbml. Due to changes to the spatial package specifications, we cannot guarantee that this works for the latter versions. MCell runs simulations that are specified in model description language (MDL) format. These files typically have the extension .mdl, but are not required to. A MDL file is a text file with commands separated by whitespace. The nature and type of whitespace (space, tab, newline) is unimportant to MCell3. You are thus free to use white space to clarify the contents of the MDL file. For more information about MDL, please visit the page
http://mcell.org/documentation/qrg/mcell3_qrg.html#model-description-language-overview.
The converter is:

MDL2SBML : convert from MDL to SBML.

These converters require Java 1.7+. To run this converter, you have to copy all the jar files from the folder:
https://sourceforge.net/p/sbfc/code/HEAD/tree/converters/mdl2sbml/lib/,
to the folder SBFC_core/lib.

To test the correct installation, you can type the scripts:

```
# Print the list of available converters. MDL2SBML should be printed.
./sbfConverterList.sh
# Print the list of available models. MDLModel and LibSBMLModel should be printed.
./sbfModelList.sh
```

To run the converter, you can use the script sbfConverter.sh with the usual syntax:

```
# Convert a model from MDL to SBML
./sbfConverter.sh MDLModel MDL2SBML [file.mdl | folder]
```

If you want to compile this package, you also need Apache Ant 1.6.5 or higher. After editing the source code you can build the jar file by just typing in the mdl2sbml folder:

```
# Generate the jar file
ant jar
# Generate the java documentation
ant javadoc
```

5.13 Conversion between BioPAX and GPML

These converters aims to translate between BioPAX model and GPML, using the pathvisio-core library. GPML (Graphical Pathway Markup Language) is simply an XML-based format. You can use it to define a pathway consisting of purely graphical elements (such as lines and shapes) or graphical elements with added biological information (such as genes, proteins and datanodes). For mode details about GPML, please refer to the documentation.

The converters are:

BioPAX2GPML : convert from BioPAX to GPML;

GPML2BioPAX : convert from GPML to BioPAX.

These converters require Java 1.7+. To run these converters, you have to copy all the jar files from the folder:
https://sourceforge.net/p/sbfc/code/HEAD/tree/converters/sbfc-pathvisio/lib/,
to the folder SBFC_core/lib. To test the correct installation, you can type the scripts:

```
# Print the list of available converters. GPML2BioPAX and BioPAX2GPML should be
    printed.
./sbfConverterList.sh
# Print the list of available models. GPMLModel and BioPAXOldModel should be printed
    .
./sbfModelList.sh
```

To run the converter, you can use the script sbfConverter.sh with the usual syntax:

```
# Convert a model from BioPAX to GPML
./sbfConverter.sh BioPAXOldModel BioPAX2GPML [file.owl | folder]
# Convert a model from GPML to BioPAX
./sbfConverter.sh GPMLModel GPML2BioPAX [file.gpml | folder]
```

Please note that these converters use an old version of the BioPAX tools. The model for BioPAX is called BioPAXOldModel as it uses an old library, and it should not be confused with BioPAXModel, which is the default BioPAX model in SBFC_core.
To compile the package sbfc-pathvisio from source, you also need Apache Ant 1.6.5 or higher. After editing the source code you can build the jar file by just typing in the sbfc-pathvisio folder:

```
# Generate the jar file
ant jar
# Generate the java documentation
ant javadoc
```

5.14 Conversion between Antimony and SBML

These converters aims to translate between Antimony and SBML, using the Antimony external program sbtranslate. Antimony is a text-based model definition language originally based on Jarnac, and extended to be fully modular. For more information, please visit the http://antimony.sourceforge.net/ website.

The converters are:

Antimony2SBML : convert from Antimony to SBML;

SBML2Antimony : convert from SBML to Antimony.

These converters require Java 1.7+. To run these converters, you have to copy all the jar files from the folder:
https://sourceforge.net/p/sbfc/code/HEAD/tree/converters/sbfc-antimony/lib/,
to the folder SBFC_core/lib.
These converters are based on the external program *sbtranslate* provided in the Antimony website. The user must install the package antimony and configure the path to the sbtranslate command in the sbfc script sbfConverter.sh located in the folder SBFC_core. This can be done in two ways:

```
# FILE: sbfConverter.sh
# Add the path to the program sbtranslate
# by configuring the Java system property SBTRANSLATE_PATH:
#PROPERTIES="$PROPERTIES -DSBTRANSLATE_PATH=/path/to/sbtranslate"
# Configure the environment variable SBTRANSLATE_PATH
#export SBTRANSLATE_PATH=/path/to/sbtranslate
```

To test the correct installation, you can type the scripts:

```
# Print the list of available converters. Antimony2SBML and SBML2Antimony should be
    printed.
./sbfConverterList.sh
# Print the list of available models. AntimonyModel and SBMLModel should be printed.
./sbfModelList.sh
```

To run the converter, you can use the script sbfConverter.sh with the usual syntax:

```
# Convert a model from Antimony to SBML
./sbfConverter.sh AntimonyModel Antimony2SBML [file.txt | folder]
# Convert a model from SBML to Antimony
./sbfConverter.sh SBMLModel SBML2Antimony [file.xml | folder]
```

To compile the package sbfc-antimony from source, you also need Apache Ant 1.6.5 or higher. After editing the source code you can build the jar file by just typing in the sbfc-antimony folder:

```
# Generate the jar file
ant jar
# Generate the java documentation
ant javadoc
```

5.15 Conversion between CellML and SBML

These converters aims to translate between CellML and SBML, using the Antimony external program sbtranslate. The purpose of CellML is to store and exchange computer-based mathematical models. For more information, please visit the http://antimony.sourceforge.net/ website.

The converters are:

CellML2SBML : convert from CellML to SBML;

SBML2CellML : convert from SBML to CellML.

These converters require Java 1.7+. To run these converters, you have to copy all the jar files from the folder:
https://sourceforge.net/p/sbfc/code/HEAD/tree/converters/sbfc-antimony/lib/,
to the folder SBFC_core/lib.
These converters are based on the external program *sbtranslate* provided in the Antimony website. In order to use the program sbtranslate to convert between CellML and SBML, the option **WITH_CELLML** must be turned on (see http://antimony.sourceforge.net/antimony-installation.html). This may involve the additional installation of CellML API and re-compilation of the source code.
Once sbtranslate is ready for supporting the conversion between CellML and SBML, the user must configure the path to the sbtranslate command in the sbfc script sbfConverter.sh located in the folder SBFC_core. This can be done in two ways:

```
# FILE: sbfConverter.sh
# Add the path to the program sbtranslate
# by configuring the Java system property SBTRANSLATE_PATH:
#PROPERTIES="$PROPERTIES -DSBTRANSLATE_PATH=/path/to/sbtranslate"
# Configure the environment variable SBTRANSLATE_PATH
#export SBTRANSLATE_PATH=/path/to/sbtranslate
```

To test the correct installation, you can type the scripts:

```
# Print the list of available converters. CellML2SBML and SBML2CellML should be
    printed.
```

```
./sbfConverterList.sh
# Print the list of available models. CellMLModel and SBMLModel should be printed.
./sbfModelList.sh
```

To run the converter, you can use the script sbfConverter.sh with the usual syntax:

```
# Convert a model from CellML to SBML
./sbfConverter.sh CellMLModel CellML2SBML [file.cellml | folder]
# Convert a model from SBML to CellML
./sbfConverter.sh SBMLModel SBML2CellML [file.xml | folder]
```

To compile the package sbfc-antimony from source, you also need Apache Ant 1.6.5 or higher. After editing the source code you can build the jar file by just typing in the sbfc-antimony folder:

```
# Generate the jar file
ant jar
# Generate the java documentation
ant javadoc
```

6 More about SBFC

6.1 High-demanding conversion jobs

Conversion jobs can be very demanding in terms of CPU and memory, if you have large models or a large number of models to convert. We implemented two modes to convert models in a batch way:

- Use a web browser. In this case the conversion uses the cluster of computer machines at the EBI (European Bioinformatics Institute). Please, visit the web page: http://www.ebi.ac.uk/biomodels/tools/converters/. The user must specify the input and output model formats. Models can be converted via three input methods: multiple file upload, URL pointing, copy/paste. Once the conversion jobs are started, the result files can be downloaded for 72 hours after the conversion finished. As far as privacy concerns neither the original nor the converted models are kept on the servers for a period longer than 72 hours. As the web service uses SBFC in the background, only the converters already integrated in SBFC are available.

- The *ConverterLink* is a Java package that allows the user to run conversion job and get the converted model directly as the return value of a method. Therefore this is convenient if SBFC is integrated in an existing application. A UML class diagram for the class *ConverterLink* and how to use it is shown in Figure 2. Depending on user requirements, two types of methods for submitting conversion jobs can be selected within the class *ConverterLink*:

  blocking calls: the methods *submitAndGetResultFromURL()* and *submitAndGetResultFromString()* start a conversion job and wait until the model is converted and returned from the web server. For large models the conversion process can take several minutes.

  non blocking calls: the methods *submitJobFromURL()* and *submitJobFromString()* start a conversion job and immediately return a *ConversionId* object. The status of the conversion can be check later on with the *getJobStatus()* method.

6.2 Pipelines of conversions

SBFC supports the creation of workflows of multiple converters in a pipeline. This can be achieved in two ways:

A file is passed for each intermediate conversion. The user can invoke the static method: *Converter.convertFromFile(String inputModelType, String converterType, String inputFileName)* in the package *org.sbfc.converter* to convert model formats as a file automatically. SBFC also provides the user with a command to convert a model a input model file by command line:

```
java Converter [Input-Model-Class] [Converter-Class] [Input-Filename]
```

The names of available input model classes and converter classes are in the packages *org.sbfc.converter.models* and *org.sbfc.converter*, respectively.

A string is passed for each intermediate conversion. The user can invoke the static method: *String Convert.convertFromString(String inputModelType, String converterType, String modelString)* in the package *org.sbfc.converter* to convert models as string format automatically.

The method *convertFromFile()* imports and exports models using the methods *void setModelFromFile(String filename)* and *String modelToFile()* as implemented in the model classes which are selected in the string parameter *converterType* and implement the interface *GeneralModel*.
The method *convertFromString()* imports and exports models using the methods *void setModelFromString(String filename)* and *String modelToString()* as implemented in the model classes which are selected in the string parameter *converterType* and implement the interface *GeneralModel*.
For each convertion step the SBFC framework checks that the output format from the previous step is the same as the input format.

6.3 Advanced options

For using SBFC with libsbml, the user must configure the environment variable *LD_LIBRARY_PATH* to include the path to your installed copy of libsbml. For Unix/Linux users, this environment variable is declared in the following files:

- sbfConverter.sh
- sbfConverterList.sh
- sbfConverterOnline.sh
- sbfModelList.sh

For Windows users, please consider the same files but with extension .bat instead of .sh .

6.4 Error messages

SBFC can notify the user with three potential error messages to highlight problems during the conversion process or when reading / writing a model format. These error messages are thrown via the following SBFC exceptions:

ConversionException: The conversion between two model formats was not successful;

ReadModelException: The input model was not imported correctly;

WriteModelException: The output model was not exported correctly.

6.5 Need help or report a bug?

If you have any trouble concerning the use of SBFC, please let us know using the Sourceforge project tracker:
https://sourceforge.net/p/sbfc/bugs/.

6.6 Need another converter?

If you need a converter not currently implemented in SBFC, you can request it using the Sourceforge project tracker in *Feature Request* in the SBFC page:
https://sourceforge.net/p/sbfc/feature-requests/.
Alternatively, you can develop it and submit it to the project. To find out more details about how to develop a new converter, please refer to the Developer Manual.

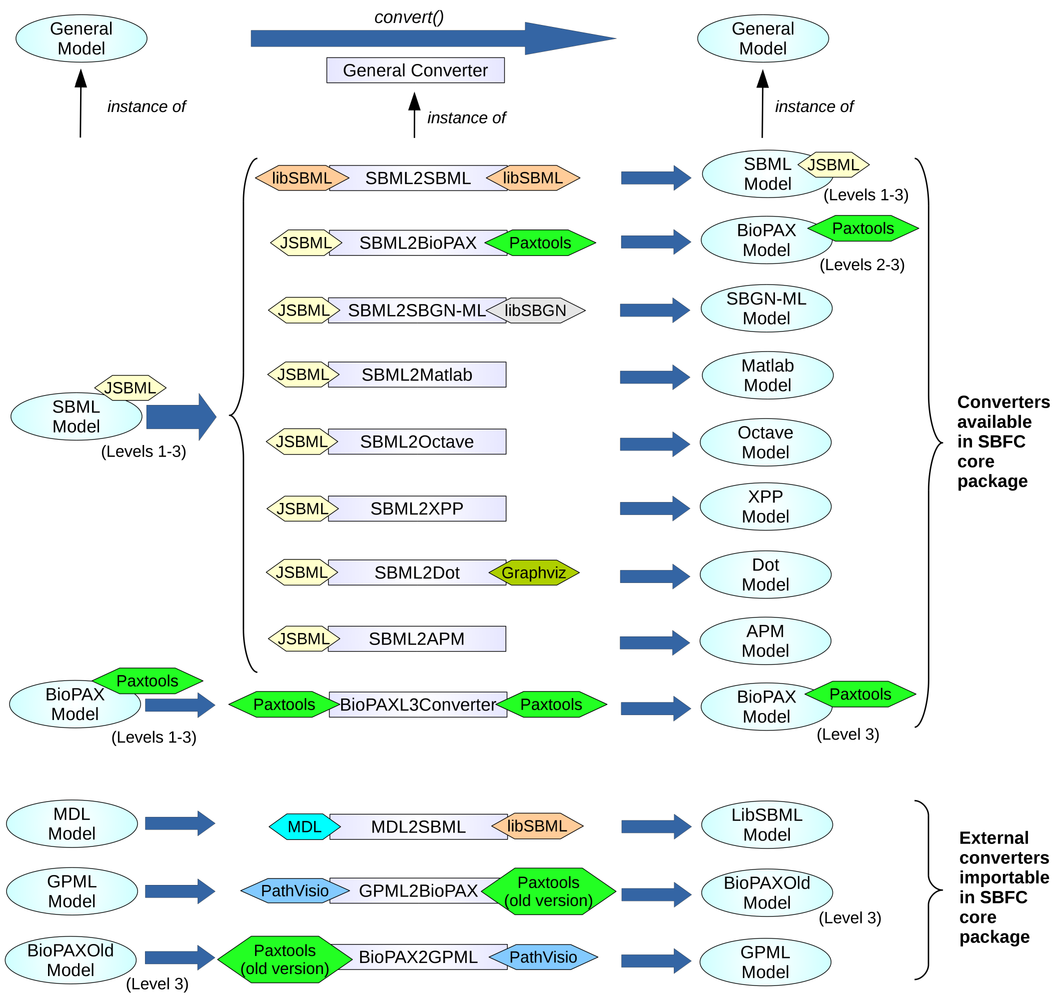

Figure 1: Overview for the software package SBFC. At the SBFC core there is a general converter which translates a general model into another. Instantiations of general model and general converter are easily implemented in SBFC providing the user with a wide range of options for converting between specific model formats. Currently, SBFC supports conversions from SBML format (levels 1-3) to SBML (levels 1-3), BioPAX (levels 2-3), SBGN-ML, Matlab, Octave, XPP, Dot or APM formats. Software libraries for importing or exporting model formats are often available and can be reused by the converters. For instance, the converter SBML2BioPAX currently uses the software libraries JSBML to import an SBML model, and PAXTOOLS to export it.

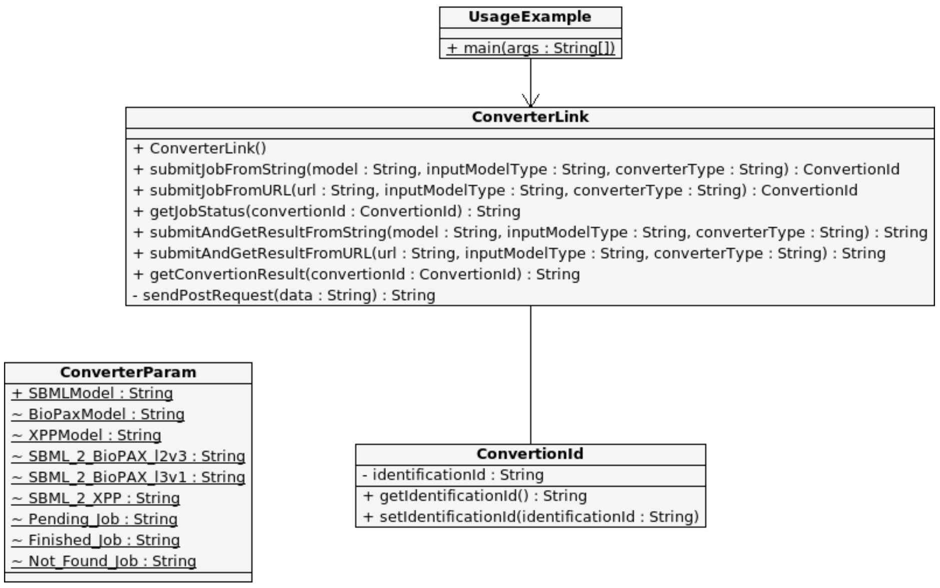
UsageExample
+ main(args : String[])
ConverterLink
+ ConverterLink()
+ submitJobFromString(model : String, inputModelType : String, converterType : String) : ConvertionId
+ submitJobFromURL(url : String, inputModelType : String, converterType : String) : ConvertionId
+ getJobStatus(convertionId : ConvertionId) : String
+ submitAndGetResultFromString(model : String, inputModelType : String, converterType : String) : String
+ submitAndGetResultFromURL(url : String, inputModelType : String, converterType : String) : String
+ getConvertionResult(convertionId : ConvertionId) : String
- sendPostRequest(data : String) : String
ConverterParam
+ SBMLModel : String
~ BioPaxModel : String
~ XPPModel : String
~ SBML_2_BioPAX_l2v3 : String
~ SBML_2_BioPAX_l3v1 : String
~ SBML_2_XPP : String
~ Pending_Job : String
~ Finished_Job : String
~ Not_Found_Job : String
ConvertionId
- identificationId : String
+ getIdentificationId() : String
+ setIdentificationId(identificationId : String)

Figure 2: UML class diagram shows how to use the class *ConverterLink*.