The **Fōrmulæ Desktop** Users Guide

Laurence R. Ugalde

Contents

Chapter 1

Introduction

1.1 Fōrmulæ technology

Welcome to the Desktop program for the Fōrmulæ technology.

Fōrmulæ is an open-source project that defines a framework to do symbolic computation. In other words, it helps to a programmer who wants to do the following:

- To show expressions in a human-like format. it is usually known as *pretty-print*.
- To edit expression in a natural way.
- To define rules for transforming expressions in other expressions. As an example, the expression $2 + 3$ is transformed to the expression 5 because there exists a rule for that.

An important aspect of this framework is that the development of the latter is modular and collaborative. That is, any person can write code to specify how to show, edit and transform expressions related to an specific area -i.e. astronomy-, pack the result in a file -named a *package*- and upload it. Other person will be able to download the package and use it.

1.2 Capabilities

This program was originally created for a demostration of the things that are able to do with the Fōrmulæ framework.

Some of the main activities you can make with the Desktop program are:

- Install packages created by contributors. They are usually downloaded from internet. They can be also configured, updated or deleted.
- Create, edit and visualize expressions from them all. They can be combined even though they come from different packages.
- If a package allows it, it will show information of certain expressions according with your region (language, country, etc). i.e. How to show numbers, or dates.
- Make symbolic calculations with them, according with the rules provided in packages.

- Show a list of expressions created and their calculations.
- Save and load your work onto files.
- Print your work

1.3 Development of packages

If you are a developer and want to write a package, the Desktop program is infinitely useful for testing your work. Once it is ok, it can be packed and released to internet. Share your knowledge!

1.4 Uses

There are many ways the Desktop program can be used -but it is not limited- to:

- As a computer algebra system, such like *Mathematica*, *Maple* or *Mathcad*.
- Since they allow the creation of beautiful notation, graphics, plots, diagrams and images, it can be used as a graphics tool. Such that elements can be further copied/pasted to other programs, such as text documents, or saved to files.
- As a programming language, it is, to create programs intended to automate tasks.
- As a didactic tool. They can be useful for teaching math and programming.
- As a scripting editor. Scripts can be later called for execution from other programs.

Chapter 2

Requisites

2.1 Operating system

You need one of the following operating system:

- Microsoft Windows
- Apple MacOS
- Linux

2.2 Java

You will need to have installed Java, version 1.7 or higher.

Chapter 3

Getting the Desktop program

There are two ways to get the Desktop program:

1. Downloading the program. It will include several predefined packages.
2. Building the program from source code. Packages will be selected, downloaded and loaded into the Desktop in a later time. This option is for hackers who want to create everything from code.

3.1 Downloading the program with or without packages

You can download it form the downloads section of the official web site:

http://www.formulae.org

The file is named `desktop.zip`

The following step is open the zip file and unpack all its content in a new, empty directory. There is no restriction about the name of the directory, but `formulae` or `desktop` are the most commonly used.

Now, skip to the next section.

3.2 Building from source code

You can download the source code form the downloads section of the official web site:

http://www.formulae.org

The file is named `desktop_src.zip`

3.2.1 Compiling from source code

In order to be compiled, you will first need the Fōrmulæ library. If you downloaded it in binary form

```
<JAVA_PATH>/bin/javac -classpath .:formulae.jar -d <DESKTOP_PATH> *.java
```

If you want to build the library from source code, the source code is in the downloads section of the official web site:

http://www.formulae.org

The file is named `formulae.zip`

It is build with:

```
<JAVA_PATH>/bin/javac -classpath . -d <LIBRARY_PATH> org.formulae.*
```

And finally, the Desktop program is built with:

```
<JAVA_PATH>/bin/javac -classpath .:<LIBRARY_PATH> -d <DESKTOP_PATH> *.java
```

Chapter 4

Running the Desktop program

4.1 Running form a Jar file

If you just downloaded the program, you will see that the directory created contains a `desktop.jar` file. Launching the program is as simply as double-clicking on it.

The first time you run the program, it looks like the figure 4.1

The left area is the editors section. If you have not installed any package, it will be empty. Once you start downloading packages and loading them it will be populated of editors. Go to the next chapter Managing packages to found how to install packages.

If you download the program with predefined packages, the edition section of the program will be populated. It is not necessary -by now- to download and install packages, so skip to the chapter Editing expressions, in order to start using the program. In the future, you will need to remove, update, configure packages or install new one. In such that case it is recommended to read the chapter Managing packages.

4.2 Running from command-line

To start the program from commalnd-line:

```
<JAVA_PATH>/bin/java -jar desktop.jar
```

4.3 Running when compiled from source code

If you have the library as a binary Jar file:

```
<JAVA_PATH>/bin/java -classpath .:formulae.jar Desktop
```

If you have built the library from source code:

```
<JAVA_PATH>/bin/java -classpath .:<LIBRARY_PATH> Desktop
```

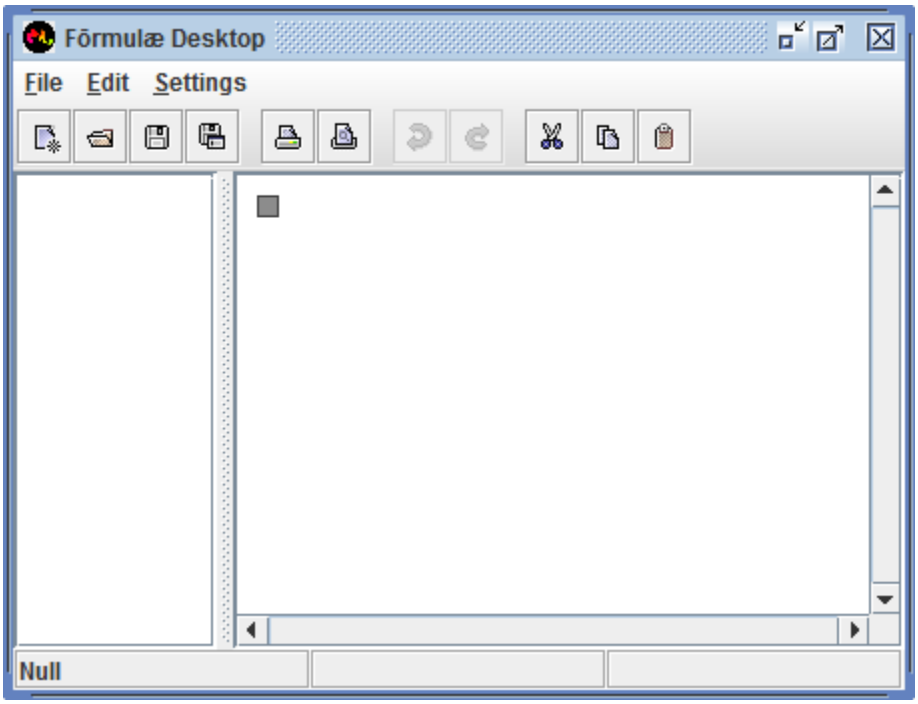


Figure 4.1: The program, the first time it is started

4.4 Running for an specific locale

The Desktop program is intended to work according to the locale detected automatically -it is obtained from the operating system where the Desktop in running on-.

It is possible to specify a different locale, with the following options, specified at command line:

`-Duser.language=LANGUAGE`

`-Duser.country=COUNTRY` (optional)

`-Duser.variant=VARIANT` (optional)

Where LANGUAGE is the 2-character language code, according to the ISO 639 standard, COUNTRY is the 2-character country code, according to the ISO 3166 standard, and the VARIANT is the variant code.

The following are examples:

1. Running the Desktop program for Italian:

```
<JAVA_PATH>/bin/java -Duser.language=it -jar desktop.jar
```

2. Running the Desktop program for French from France:

```
<JAVA_PATH>/bin/java -Duser.language=fr -Duser.country=FR -jar desktop.jar
```

3. Running the Desktop program for French from Canada:

```
<JAVA_PATH>/bin/java -Duser.language=fr -Duser.country=CA -jar desktop.jar
```

4.4.1 Availability of locales

Since the number of different languages and its variants around the world is big, providing a complete set of locales is a huge amount of work, so the availability of an specific one is not guaranteed.

Because the Fōrmulæ project is a collaborative one, an specific locale is available when a volunteer decides to make a translation.

If you want to collaborate creating a local, please refer to chapter Customization, section Creating a new locale at page 33.

If the Desktop program is run with arguments that specifies a locale currently not supported, the English locale will be used.

4.5 Running for an specific timezone

The Desktop program is intended to work according to the timezone detected automatically -it is obtained from the operating system where the Desktop in running on-.

It is possible to specify a different timezone, with the following options, specified at command line:

`-Duser.timezone=TIMEZONE`

The following is an example:

```
<JAVA_PATH>/bin/java -Duser.timezone=Europe/Sofia -jar desktop.jar
```

Chapter 5

Editing expressions

5.1 Basic concepts

When the program has just started, it shows only one expression, the *null expression*, which is the small square shown at the top left part of the working area.

5.1.1 Highlighting expressions

An expression may be formed from multiple *smaller* expressions. When the cursor passes over an expression, the Desktop program calculates the largest subexpression containing the position of the cursor and shows its enclosing rectangle in blue color. This is known as the highlihted expression. The figure 5.1 show examples of higlighting expressions based on the position of the cursor.

5.1.2 Selecting expressions

There are several ways to select a given expression. The simplest one is to click when the desired expression is highlighted, it will make that expression became the selected expression. The selecte dexpression is always shown with a grey background, like in the figure 5.2.

There can be one and only one selected expression at any time. Even you have just started the program, it contains only one expression, the *null* expression and it is already selected.

5.1.3 Tags

Every kind of expression has an universal name intended to bi-univocally identify the expression, called Tag. Every tag is on the form:

```
<Field or knowledge>.<Sub-field of knowledge>.  ...  .<Name>
```

The figure 5.3 show examples of common expression tags.

5.2 Editions

An edition in this context is an operation that modify an expression or part of it. An edition operates on the currently selected expression

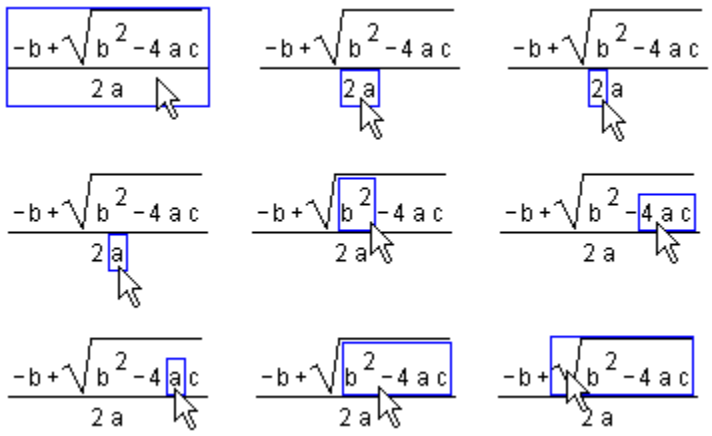


Figure 5.1: Examples of highligting expressions

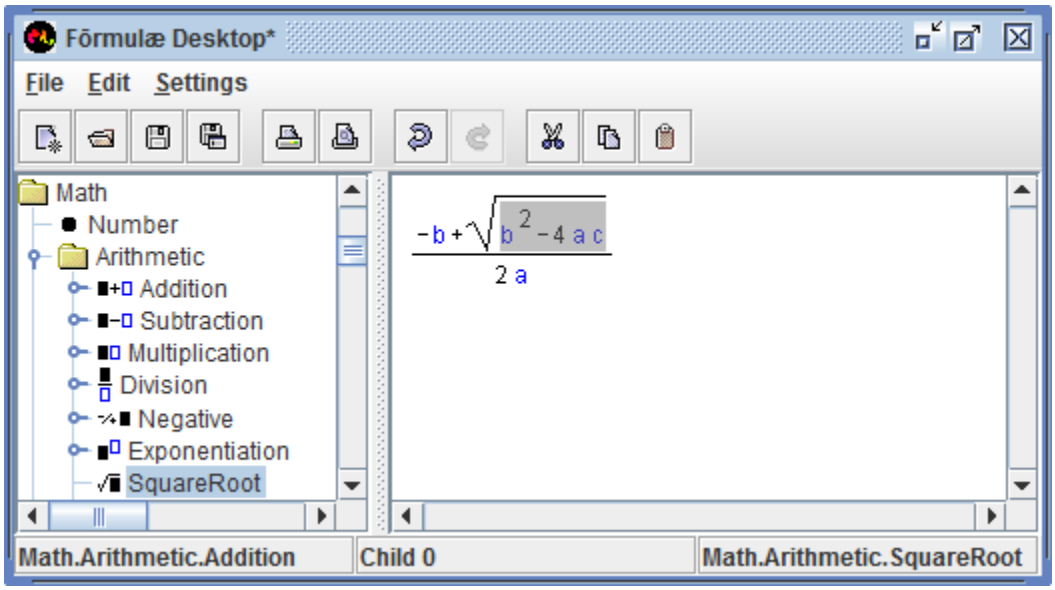


Figure 5.2: Example of the selected expression)

Tag	Expression
`Math.Number`	Real, either integer or decimal number
`Math.Arithmetic.Addition`	The mathematical addition
`Math.Constant.Pi`	The π constant
`Logic.And`	The logical conjunction
`Programming.While`	The while loop in programming

Figure 5.3: Examples of tags

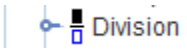


Figure 5.4: Division edition contracted

Let us make several examples of editions. You have to have the standard package already loaded.

5.2.1 Example 1. Creating a number

Select the *number* edition from the left tree, or press the n key.

This reduction means "The currently selected expression will be substituted by a new number expression. This new numbers will became the selected expresion".

5.2.2 Example 2. Creating an addend

Having the number expression just created, select the *Addition* option from the editions tree, or press the + key

This reduction means "The currently selected expression will be substituted by a new addition expression, its first addend will be the currently selected expression, and the second one will be a new null expression. That new null expression will became the selected expression". It sounds fairly complicated but the result seems intuitive and natural.

In fact, the reduction tries to be fairly smart, for example

- If the currently selected expression is actually an addition, no new addition is created, instead, a new *null* expression is created, it becames the last addend and it also becames the selected expression.
- If the currently selected expression is an addend of an existing addition, no new addition is created, instead, a new *null* expression is created, it is placed at the right of the currently selected expressions and it also becames the selected expression

5.2.3 Example 3. Creating a division

In this example, select either the first addend, the second addend, or the entire addition, at your wish. Select the *Division* from the editions tree at the left, or press the combination of keys to produce the character '/', usually / on a numeric keypad or ⇧+7.

it will create a new division expression, the selected expression -no matter how complicated it could be- becames the numerator, and a new *null* expression is the denominator, and also becames the selected expression.

The division edition in the edition tree, has variants (as many other editions). They are represented as a special sign left, indicating that the tree can be further expanded, just as seen in the figure 5.4.

If you click on the expansion/contraction left icon, the edition will be expanded to show all their *flavors*, such like the figure 5.5.

The division edition just used corresponds to the first flavor. The second one is the opposite: A new division expression is created, but selected expression becames the denominator, and a new *null* expression is created to became the numerator and selected expression.

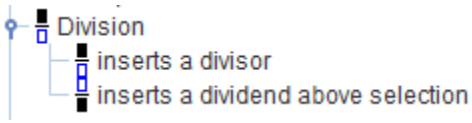


Figure 5.5: Division edition expanded

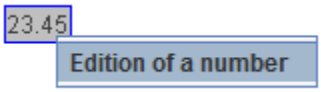


Figure 5.6: the floating menu showing the action for the currently selected expression

5.2.4 Generalities about editions

As the examples showed, functionality of editions are usually different, but all they have the following in common:

- They are intended to produce a natural and intuitive sense to the user.
- It always needs an existing expression -the selected expression- as reference.
- After the execution of the edition, an existing expression should be set as the selected expression. In several cases, it can even be the previously selected expression.
- There is no restriction for an edition to be applied about the nature of the selected expression, any edition can operate over any selected expression.

5.3 Actions

An action is a particular case of an edition. It differs on the following:

- It is always associated to a particular tag.
- it usually neither creates new expressions nor removes expressions, it usually only changes the internal structure of the selected expression. Because of that, after the edition, it usually remains as the selected expression.

They can be seen as *operations* that can be performed on a particular kind of expression.

A tag can have associated zero o more actions. In order to acces the actions for the selected expression, right-click on the mouse and a floating menu will be shown, like the figure 5.6. Once selected the action it will be performed. As an example, if the currently selected expression is a number expression, there is one action: to edit that number, like the figure 5.7.

If the currently selected expression tag has no actions associated with it, right-clicking over it has no effect. If it has one or more actions, the first one is de *default* action. The default action is performed when pressing the space key.

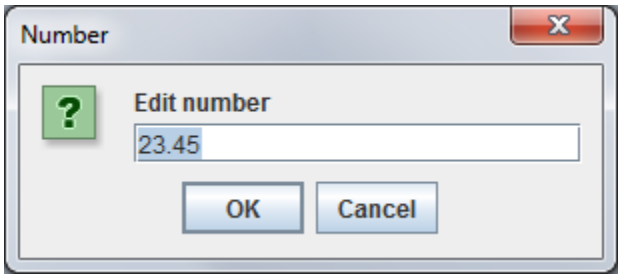


Figure 5.7: The action associated with a number expression

5.4 Knowing the editions and actions from packages

Most of the edition and actions are not part of the desktop program, these are provided in packages, and therefore it is impossible to show all of them, given there are an increasing number of packages being developed, and others are being changed or improved, so this document is necessarily incomplete

http://dictionary.formulae.org

There is common edition functionallity present in the desktop. It is the matter of the next section.

5.5 Common functionality

5.5.1 Insertion

There are expressions that can accept more direct subexpressions that they actually have, for example, it is always possible to add a new addend to an addition, while a division can have exactly two subexpressions, the numerator and the denominator.

If the selected expression is a child of an expression that can hold an additional expression, -or *creating s sibling*- it can be done pressing the Insert key. A new *null* expression will be created and it will be set after the former. This new null expression will then became the selected expression.

If pressing the Alt+Insert keys the new *null* expression will be created before the current selected expression.

For example, given $1+2+3$, when pressing the Insert key the result is $1+2+\square+3$, while pressing Alt+Insert the result is $1+\square+2+3$

If the parent of the selected expression does not support additional expression, pressing Insert or Control+Insert has no effect.

Creating a unique child

There can be expressions having zero children, but can accept one of them. if we want to create that unique child, pressing the Control+Insert.

As an example, a list expression can hold any number of children (its elements). If selected expression is an empty list like $\{\}$, we can create a single element, it becomes $\{\square\}$

5.5.2 Deletion

A deletion is performed when the delete keys is pressed.

Action	Key combination	Alternate key combination	Menu	Button
Cut	Ctrl + x	Ctrl + Insert	Edit > Cut	
Copy	Ctrl + c	Ctrl + Insert	Edit > Copy	
Paste	Ctrl + v	Ctrl + Insert	Edit > Paste	

Figure 5.8: Methods for performing cut/copy/paste operations

. . . trying to be as less *destructive* as possible:

When current selection is not the *null* expression

Deletion of expressions are subjected to the following rules

- If the current selected expression has no subexpressions -i.e. a number- it is replaced by a new *null* expression. ex. $1+2+3$ becomes $1+\square+3$
- If the current selected expression has exactly one subexpression -i.e. a square root- the expression is replaced by its only child. $1+\sqrt{2}+3$ becomes $1+2+3$
- If the current selected expression has more than one subexpression, -i.e. a division- it is replaced by a new *null* expression. This is the most *destructive* kind of deletion, because there is no hint that lets us choose a subexpression. $1+\frac{4}{5}+3$ becomes $1+\square+3$

When current selection is the *null* expression

In this case the deletion tries to remove the selected expression as child of its parent

There are expressions that accept less direct subexpressions that they actually have, for example, it is always possible to remove a addend from an addition, providing it has three or more of them.

If the parent of the selected expression supports having one less child, the selected expression is deleted. The new selected expression is the sibling expression that, after deletion has the position

depends on the following. If the selected expression was the latest child, the penultimate one becomes the selected expression, otherwise the following child becames the selected expression.

There is an special case: if the parent of the selected expression has two children -being the selected expression one of them- and it does not support having one child, the parent expression is replaced by the other child, and this becomes the selected expression. For example, deleting in the following expresion $1+\frac{2}{\square}+3$, where the selected null expression is a denominator of a division. Because a division cannot hold ony one child, the division is replaced by the *other* sibling -the numerator- becoming $1+2+3$

If none of the previous condition is fulfilled, the deletion operation is not performed.

5.5.3 Cut/Copy/Paste operations

A cut/copy/paste operation is performed by any of the methods shown in figure 5.8.

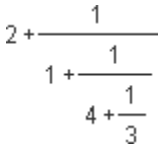


Figure 5.9: Example of a finite continued fraction

Figure 5.10: The expression used as template

A *Cut* operation replaces the currently selected expression by a new *null* expression, and the deleted expression is transferred to the *clipboard*. The *null* expression becomes the new selected expression.

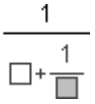
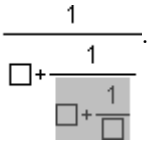
A *Copy* operation makes a copy of the currently selected expression, and such that copy is transferred to the *clipboard*. No changes in the selected expression is performed.

A *Paste* operation replaces the currently selected expression by a copy of the expression in the *clipboard*. That copy becomes the new selected expression.

Example

This example if for creating a continued fraction like the shown in figure 5.9.

The expression shown in figure 5.10 is created. Notice that the number 1 appears in the continued fraction as a pattern. Once the expression is created, it is selected and copied to the clipboard.

The denominator of the division is selected  , and then the expression in the clipboard is pasted to look like  . Again the the lower denominator is selected like  and the expression is pasted again to look like  .

The rest of the work is replacing the *null* expressions by their respective numbers. It will not be shown here.

5.5.4 Undoing/redoing

A undo/redo operation is performed by any of the methods shown in figure 5.11.

Every performed edition by either a package or the desktop program, can be undone or redone.

Action	Key combination	Menu	Button
Undo	Ctrl + z	Edit ⟩ Undo	
Redo	Ctrl + y	Edit ⟩ Redo	

Figure 5.11: Methods for performing cut/copy/paste operations

Chapter 6

Reduction

Reduction is the process of converting an expression to another in base of a set of transformation rules. For example, the expression $2 + 3$ is converted to the expression 5 because there is a expression rule -a *reducer*- that knows how to add numbers.

6.1 Reducers

The desktop program does not contain any reducer, they are provided from packages. Thay are modular like visualizations and edtions.

A reducer is a piece of code that is associated with an specific tag. If an expression has that that, the code of the reducer can be run on it. Usually that such code analyzes the expression to ensure that it can be transformed, in such that case the reducer also performs the change. For example, the *Addition of numbers* reducer is appicable to the expression $2 + 3$, reducing it to the expression 5. This same reducer ai applicable to the expression $5 +$ "*abc*" -since it is also an addition- but it is unable to transforming the expression to another one, leaving the expression unaffected.

There can be zero, one or several reducers associated to a single tag.

6.2 Reduction engine

A complete redution process can involve several reducers. The reduction engine is a program that manages te application of the available reducers to a given expression. Informally, it applies reducers to the expressions and subexpressions repeatedly, until no recucer can be applied, at that moment, the reduction process finishes.

For example, take the expression $2 + (3 \times 5)$. A reducer called *Multiplication of numbers* can be applie to the multiplication, reducing the expression to $2 + 15$ and then the *addition of numbers* can be applied, producing the expression 17. Since there is no reducers associated with a number expresions the reduction process terminates.

6.3 Performing reduction

Once the expression to be reduced has been created, pressing the enter key calls the reduction engine. The resulting expression is created. Note that the resulting expression in shown in a grey background, like the

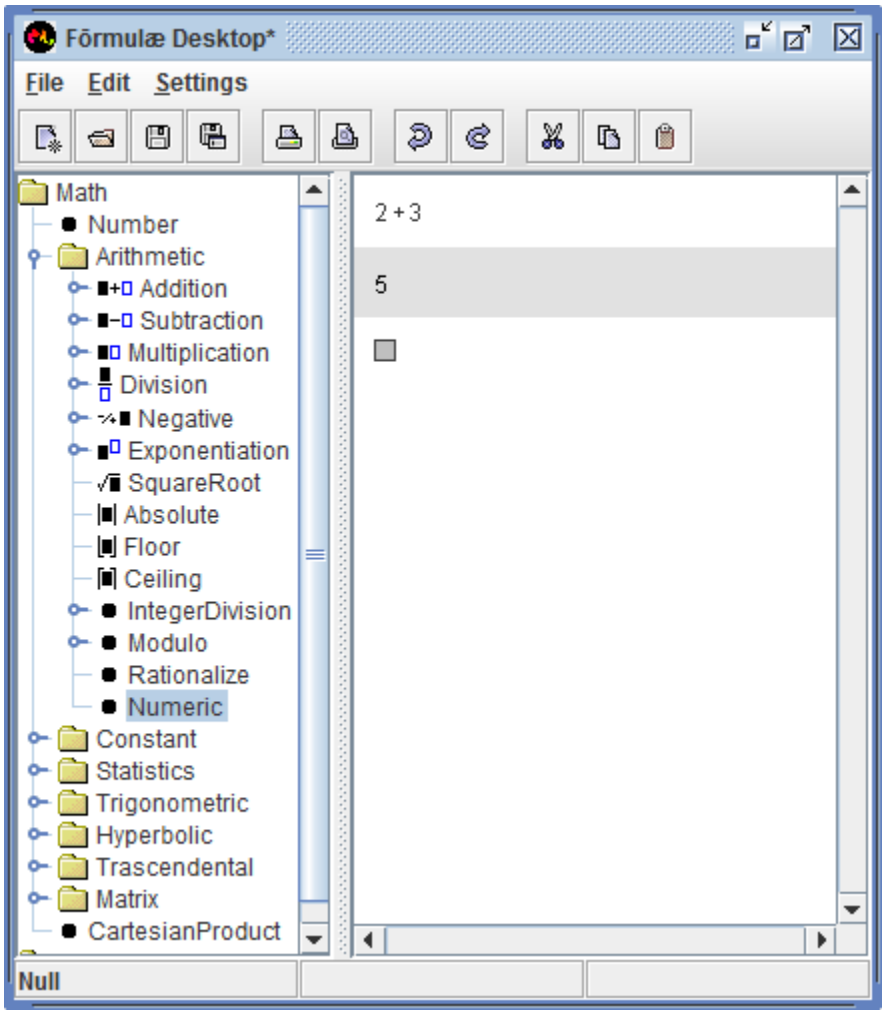


Figure 6.1: A reduction

figure 6.1.

A new *null* expression is also created and selected, in order to be edited and start again.

This create a sequence *question-answer* that can be repeated, much like a *command-line* interface[1], like the figure 6.2.

Unlike command-line interfaces, it is possible to get back to a previous *question* expression to make changes and performs a new reduction. Simply edit the expression and press again the enter key. The old *answer* will be replaced by the new one.

6.3.1 Notes about reduction

- *Answer* expressions are not editable. However they (or any subexpression) can be either highlighted, selected, or copied to the clipboard.

[1]Examples of command-line interfaces are the shell on a *nix operating system, or a and MS-DOS prompt.

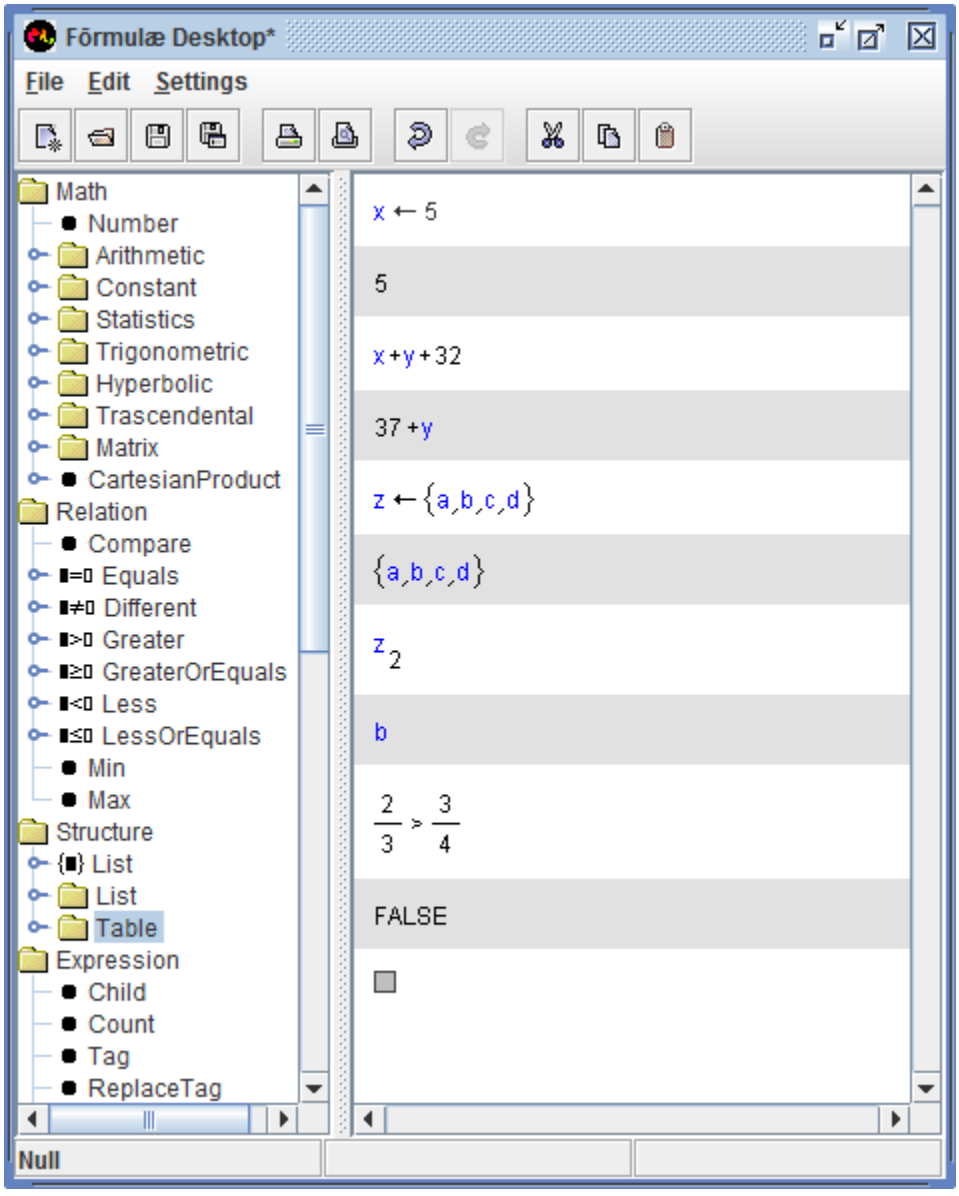


Figure 6.2: A session with a set of "questions" and "answers"

- Reduction is a undoable operation.

6.4 Showing intermediate results

Sometimes it is useful to know the intermediate steps a full reductions takes. Pressing the Alt+Enter keys instead of just the Enter key, the desktop will show all intermediate results, like the figure 6.3.

Intermediate results are also not editable, but they (or any subexpression) can be either highlighted, selected or copied to the clipboard.

Be careful, some reductions can produce a great amount of intermediate results.

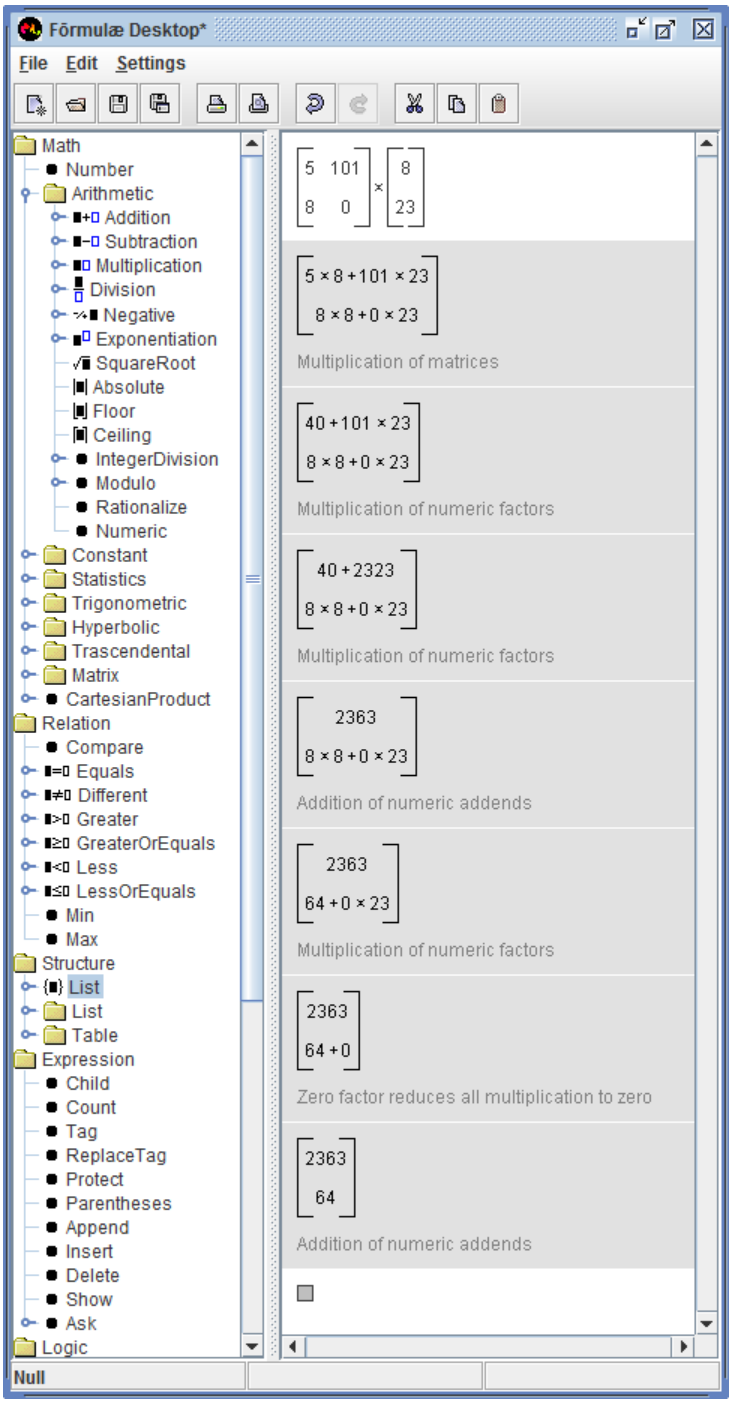


Figure 6.3: A reduction showing intermediate results

Chapter 7

Saving and sharing your work

The work can be saves as files, in order to preserve it or continue working any later.

7.1 Saving to a file

In order to save your work as a file, File » Save Ctrl + S or File » Save as... Ctrl + A

7.2 Loading from a file

File » Open... Ctrl + O

7.3 Creating a new file

File » New Ctrl + N

7.4 Compatibility

Desktop files are not intended solety to save your own work, They are also compatible between different computers, so they can be shared.

However, there can be differences that affect the ways files are shared.

7.4.1 File format

Desktop files are text files. The information is defined in a way that it does not store data about operating system, packages, configurations, localization, etc.

7.4.2 Installed packages

As you know, packages contains specifications of how expressions related to the package are visualized. If a file contains expressions that are about to be restored by a Desktop program that has different or missing

packages respect to the Desktop program where the file were saved, the following differences apply:

- Expressions will be shown according to the packages installed on the target Desktop.
- If the target Desktop does not have an installed package that specify how to show an specific expression, this will be shown as a function where the function name is the tag of the expressions, and its subexpressions are shown as the arguments of the function

  As an example, suppose that a file is saved containing the following expression:

  $$\frac{5}{2} + 3$$

  If the file is loaded on a different Desktop instance containing packages that specify how to show numbers, divisions but no additions, the user will see the following:

  $$\texttt{Math.Arithmetic.Addition(}\tfrac{5}{2}\texttt{, 3)}$$

7.4.3 Localization

if the target Desktop has a package that supports localization for a specific expression, it will show it accordingly.

As an example, suppose a file is created by a user on United Stated. It contains numbers that were shown as:

$$1,234.56$$

The file is sent to, and opened later by an user in Italy, and his/her Desktop program contains a package that is able to show numbers for his/her country. It will show the previous number as:

$$1\ 234{,}56$$

Chapter 8

Managing packages

Functionality in Fōrmulæ is modular. Packages are the heart of the Fōrmulæ framework. You can download packages and load into your Desktop program.

8.1 What does a package contain ?

A package contains the following:

- Expressions, the basic block of functionality in Fōrmulæ.
- Visualization, that define how expressions are shown.
- Editions, that define how expressions can be created or modified
- Reducers, that define how expressions can be transformed in other expressions

A single package can contain zero o several elements of each type. There are packages containing only reducers, for example.

All the files required for a package are usually packed in a single Jar file, but it is possible for a Jar file to contain more than one package.

8.2 Getting packages

8.2.1 Downloading packages

Standard packages

Standard packages are those developed inside the Fōrmulæ project, they all are open source and available at the Fōrmulæ web page:

http://www.formulae.org

They are provided as Jar files, i.e. the Arithmetic package has the name `arithmetic.jar`.

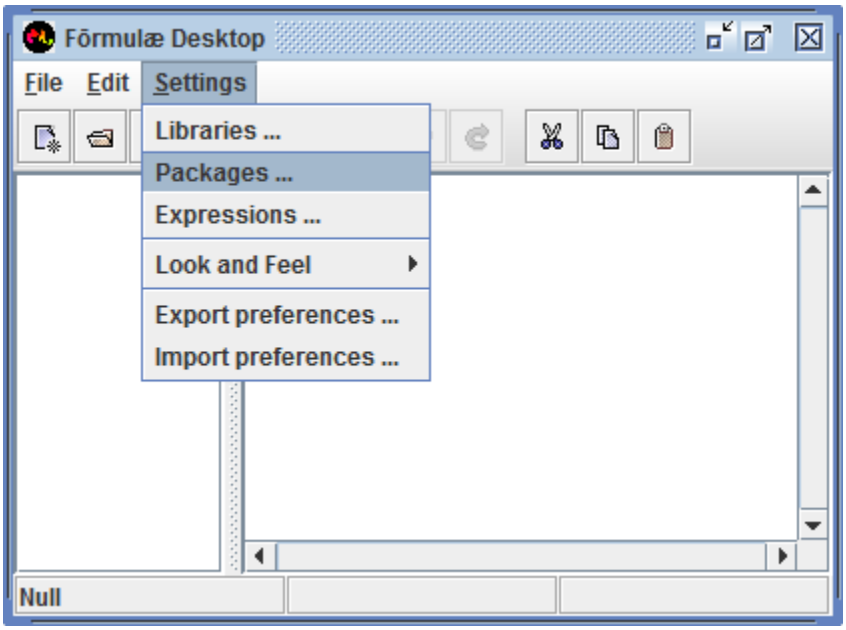


Figure 8.1: Selecting the package menu option

Non standard packages

Non standard packages are those developed outside the Fōrmulæ project, they are not hosted in the Fōrmulæ website

8.2.2 Building packages from source code

I you want to build the standard packages from source code, they are available in the downloads section of the official web site

http://www.formulae.org

Every package has its own souce zip file, for example the file `arithmetic.zip` contains the source code for the standard arithmetic package.

8.2.3 Writing your own package

You can write your own packages. The information about it is out of the scope of this document. If you are interested you should start reading the *Fōrmulæ Developer's Guide*

8.3 Loading a package

From the menu bar, select Settings » Packages... like the figure 8.1.

It will show the packages dialog box, initially empty, like the figure 8.2.

Press the From File ... button. It will show the file dialog, like the figure 8.3. Locate the file into your filesystem. double-click on it, or single-click on it and press the Open button.

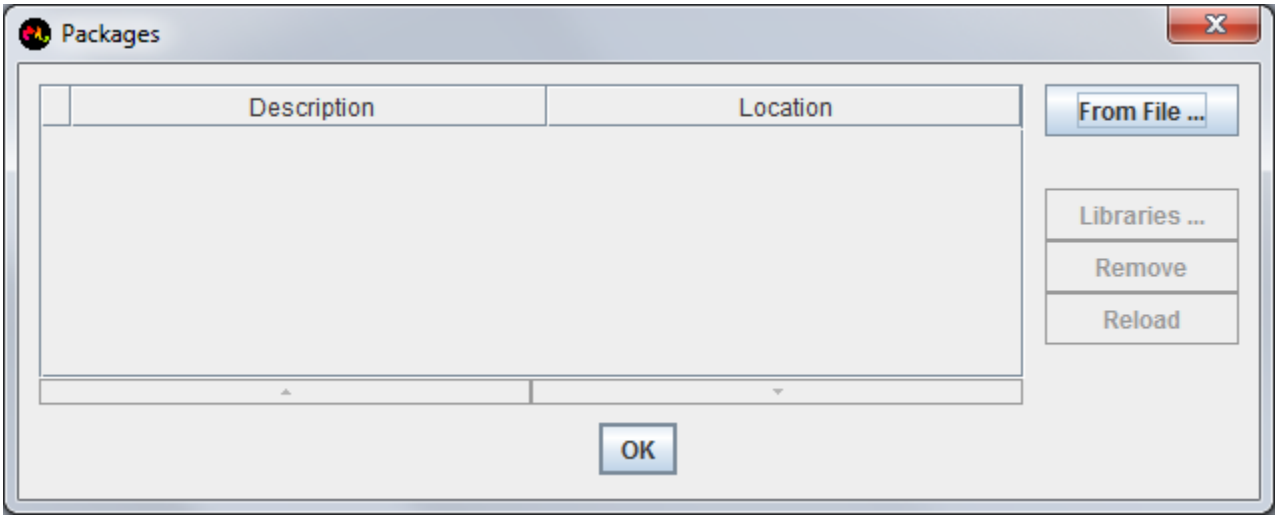


Figure 8.2: The packages dialog box

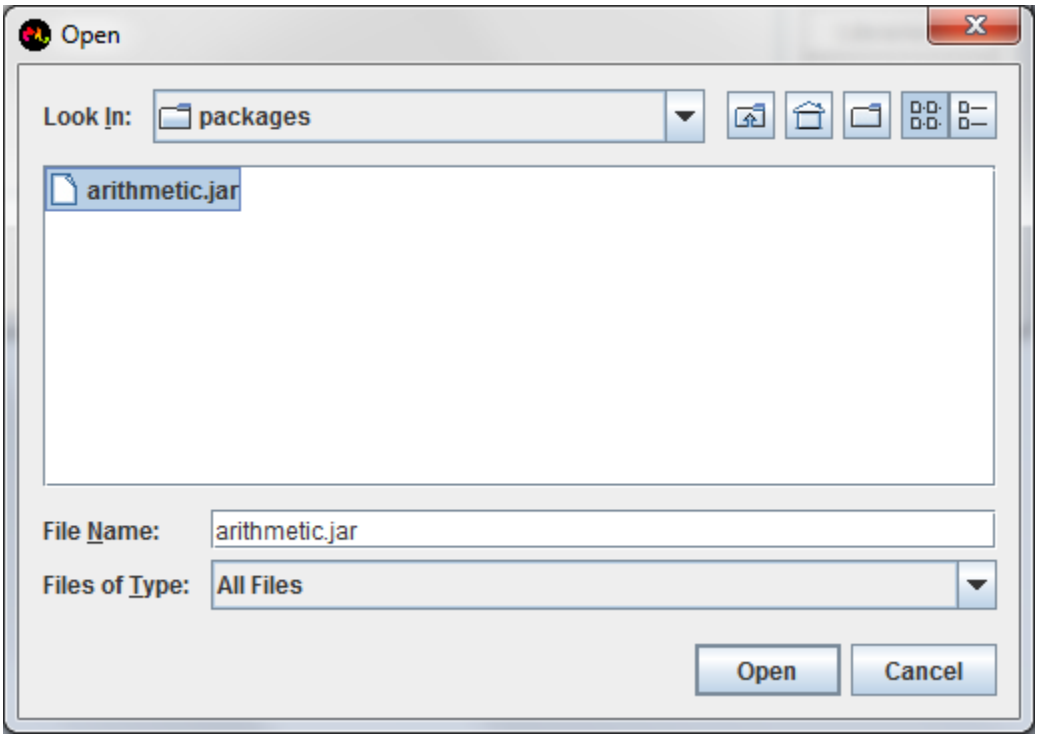


Figure 8.3: The file dialog box

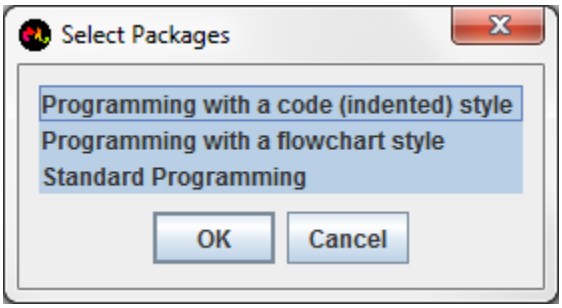


Figure 8.4: A Jar file containing more than one package. The progtam show the individual packages in order to the user can select between them

Repeat the last step for each Jar file you want lo load. Then press the Ok button in the packages dialog box.

8.3.1 Loading from a Jar file or directory containing multiple packages

When you load a Jar file, the Desktop program automatically searches for packages and it is possible to found more than one of them. In such a case, it shows a dialog box in order you can choose the packages you are interested, much like te figure 8.4. In this example a Jar file, named `programming.jar`, intended to provide programming features, contains thee different packages:

- A package that contains visualizations only. All these visualization show programming structures in indented code format, much like programming style.
- A package that contains visualizations only. All these visualization show programming structures in a flowchart style
- A package containing the expressions definitions, editions and reducers related to programming elements, but no visualizations. These are functionality for editing and executing programming structures

In this example, you *might* select the *Standard Programming* option, because this has the common and main capacilities for programming, and you *should* choose between the code and flowcharting style, depending on how you want to see these programming structures.

Moreover. you *can* select both of them and decide later which to enable/disable. It will be discussed in a later chapter.

8.4 Loading a package from a directory

It is specially useful if you are writing your own package and you want to test it, since generally you build your package as a Jar file only after you have tested it for a while and it is ready to be uploaded or distributed.

The directory is that you are using for your classes.

In order to choose a directory instead of a Jar file, When you are in the file dialog box, single-click on the directory (do not double-click, it will enter into the directory, showing the files it contains) and then press the Open button, like the figure 8.5

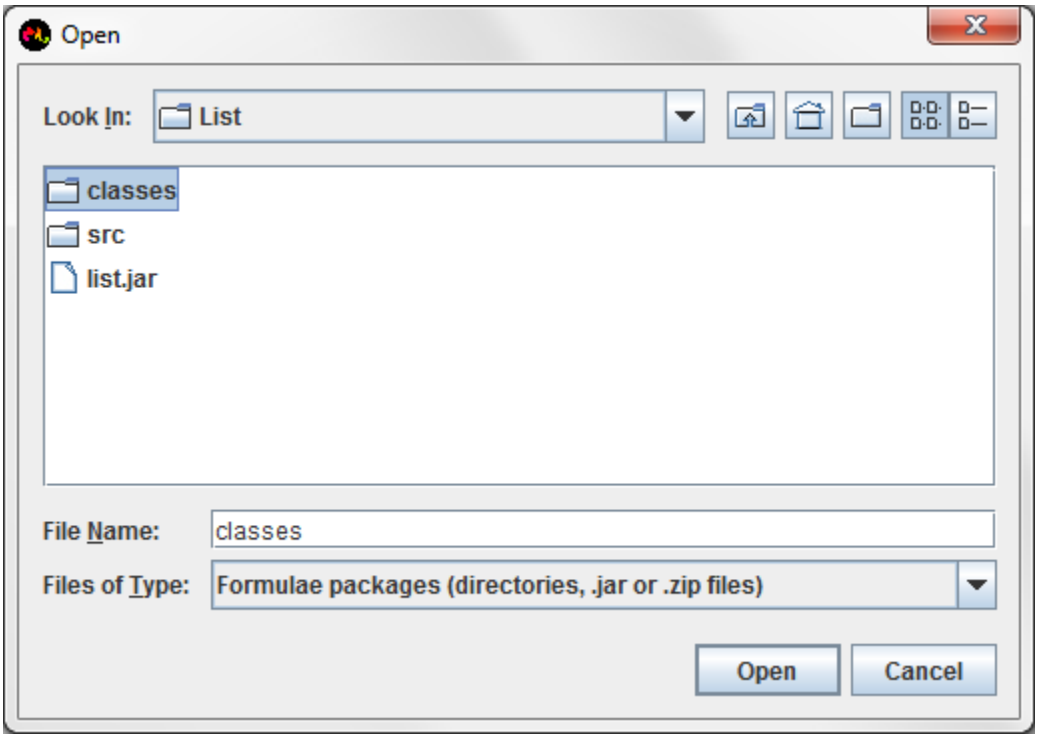


Figure 8.5: Selecting a directory instead of a Jar file

8.5 Reloading a package

Reloading a package is needed for the following situations:

- If you have downloaded a new version of the package
- If you are writing your own package, you have changed something and you want to test it

In order yo reload a package, select Settings > Packages ... highlight the package clicking on it and press the Reload button. A message will be shown, like the figure 8.6.

8.6 Packages that require external Java libraries

Sometimes, a Fōrmulæ package requires additional Java libraries.

As an example, there is a charting standard package, for interactively create common charts. The Fōrmulæ package delegates the work of building charts to a good, open source Java library, knows as *JFreeChart*[1]. This library is intended to use by deveopers, it is not a standaalone application, so the Fōrmulæ package is an intermediate between the user, providing interaction, but using the quality of chart produced by such that library.

To set a set of libraries for a package, we have to a define of libraries first.

If a package required additional Java libraries, the package provider should provide or indicate how to get these libraries.

[1] http://www.jfree.org/jfreechart

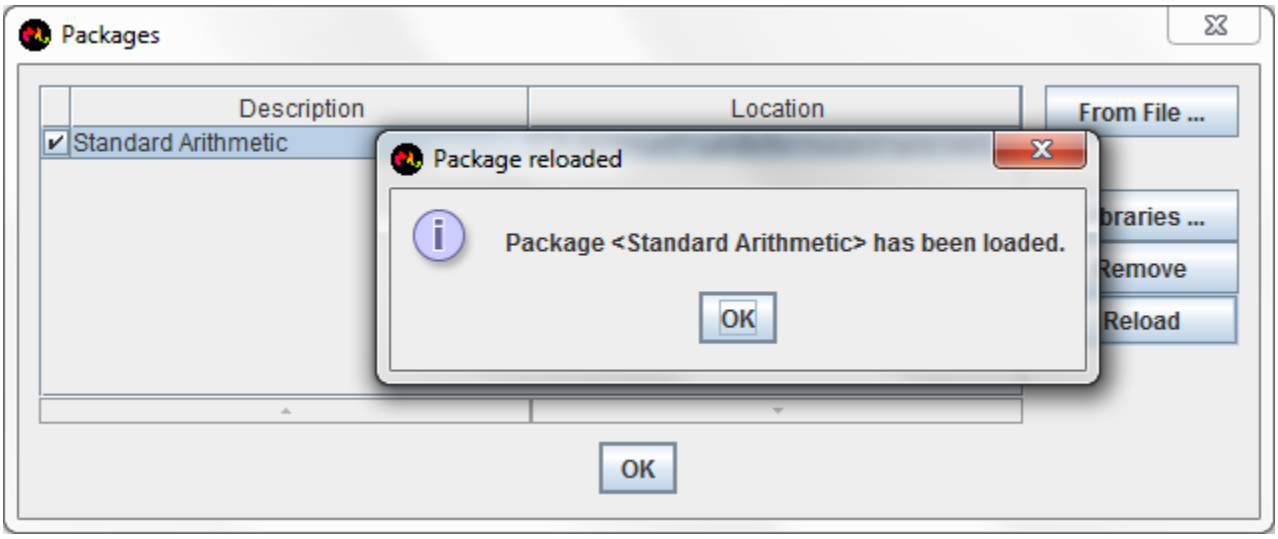


Figure 8.6: Reloading a package

8.6.1 Defining libraries

To define libraries, you have to select Settings » Libraries... from the menu bar, like the figure 8.7.

The libraries dialog box will be shown, see the figure 8.8.

Press the New ... button. A new dialog box is shown, see the figure 8.9

Now press the Browse ... button to open a file dialog box, like the figure 8.10.

Locate the library file and then press the Open button. The new library open dialog now has te library file. Now you have to write a broief description of the library, such its name and version. It i svery important that the version be specified, as you will se later, there can be defined different versions of a same library.

Now the new library looks like the figure 8.11. Press the Ok button.

The libraries dialog box has a library, see the figure 8.12. repeat for each library you have to define. at the end, press the Ok button.

8.6.2 Assigning libraries to packages

Once you have defined libraries, to assign to packages follow the next steps:

Select Settings » Packages... from the menu bar, like the figure 8.13.

Select the desired package -single clicking over it- and then press the Libraries ... A dialog box like the figure like 8.14 will open

For each required library, select it single click on it- and press the < button in order to pass it to the *Required* section, like te figure 8.15.

Once you have finished, press the Ok button to close the dialog box, and press again the Ok button of the packages dialog box to close it.

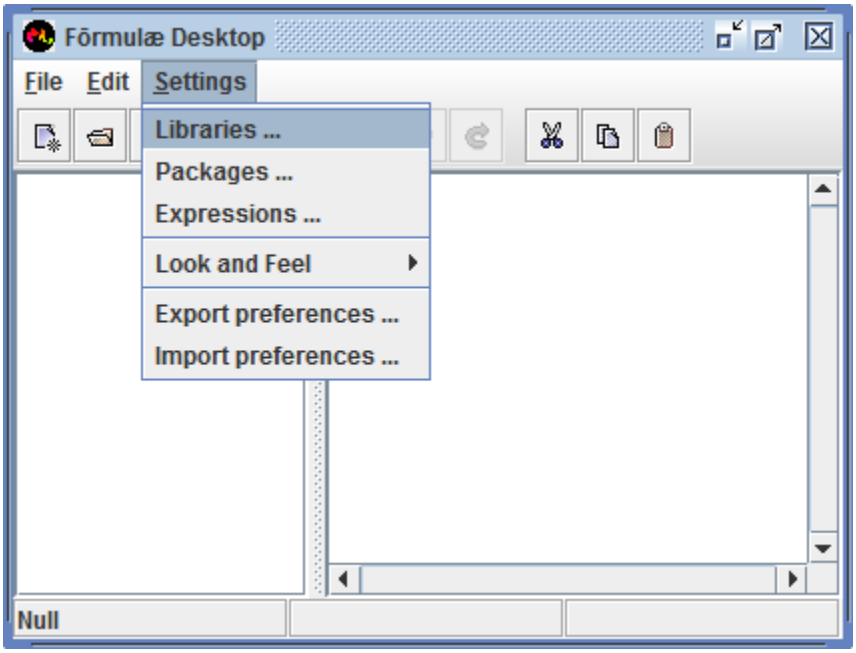


Figure 8.7: Selecting the libraries menu option

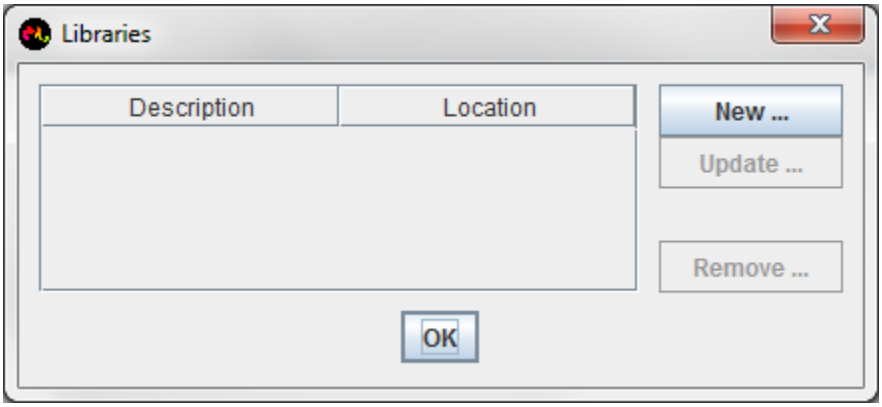


Figure 8.8: The libraries dialog box

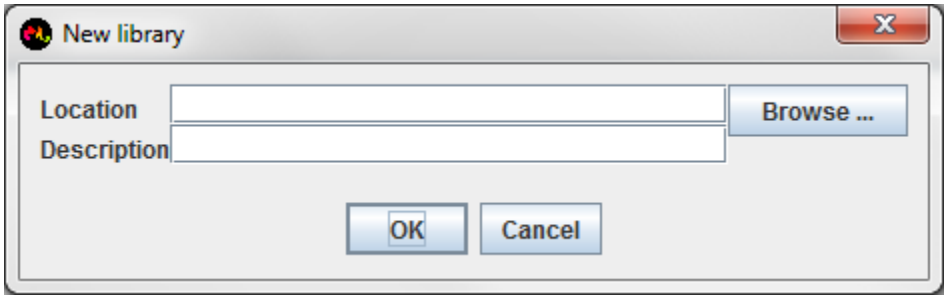


Figure 8.9: Defining a new library

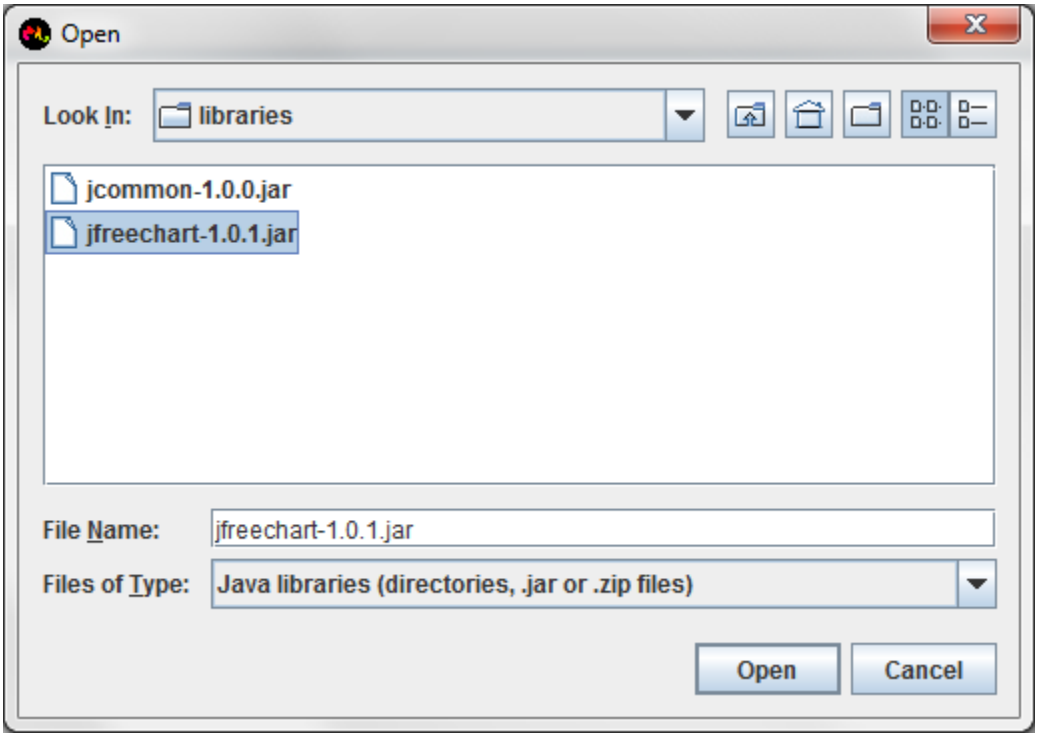


Figure 8.10: Choosing the new library

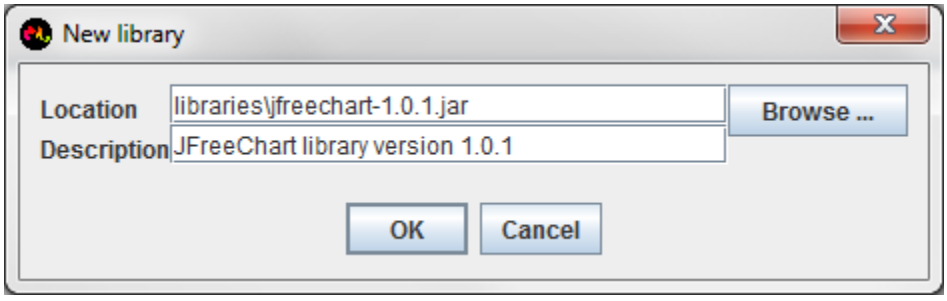


Figure 8.11: The new library dialog box is now populated

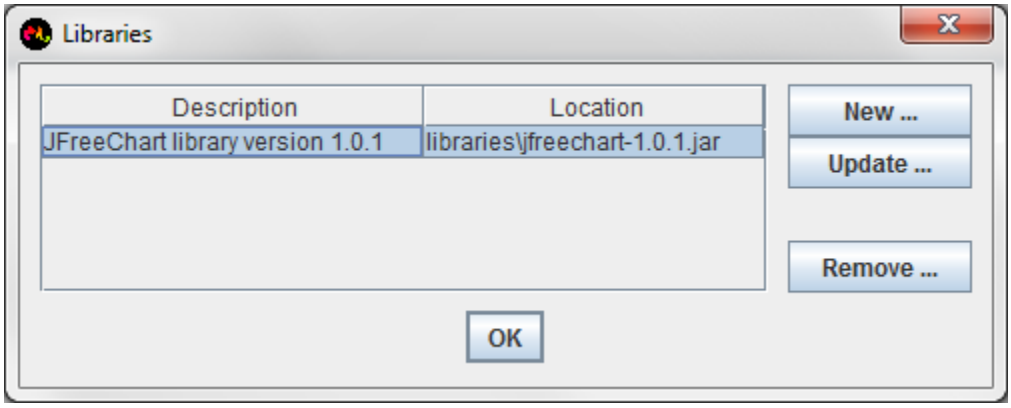


Figure 8.12: The libraries dialog box, with a library defined

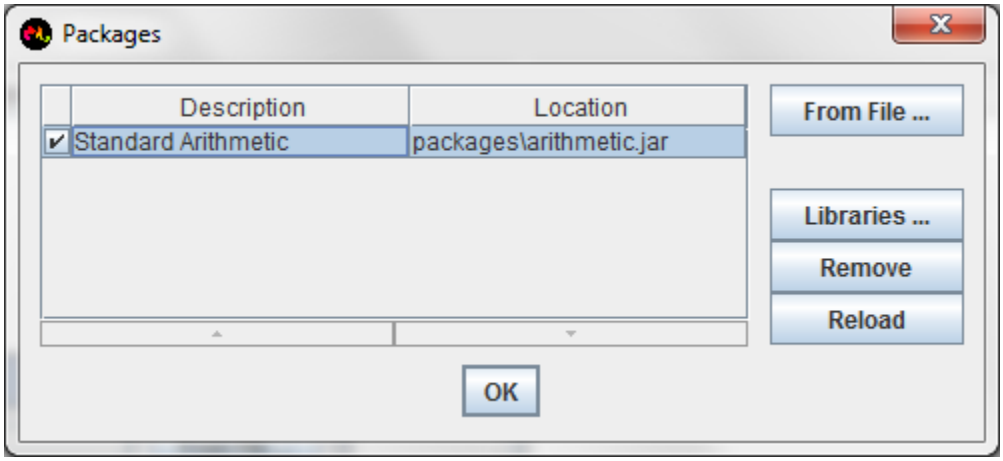


Figure 8.13: The packages dialog box, with the arithmetic package

Figure 8.14: The dialog box with the available libraries

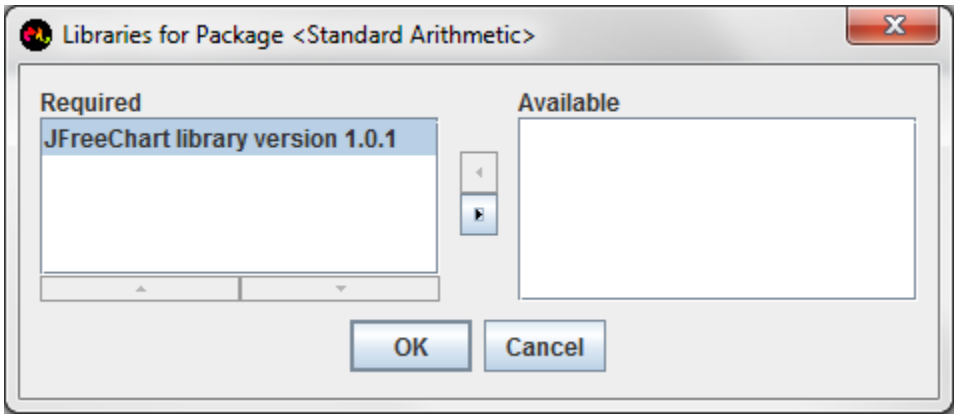


Figure 8.15: Selecting libraries to be associated with a package

8.7 Localization

As the same as Desktop program, packages are also subjected to localization. Customization to a specific language is also not guaranteed, and in case of a specific locale is not available, the English one will be used.

Also notice that packages are developed independently from the Desktop program and other packages, so it will be normal the following situations:

- Several packages support your locale while other do not.
- Desktop program supports your locale while several packages do not.
- Desktop program does not support your locale while several packages do.
- Neither the Desktop program nor any package supports your locale.

Chapter 9

Customization

9.1 Creating a new locale