GNU polyxmass User Manual

(Version 0.8.8)

This User Manual is distributed
in HTML and PDF forms at http://www.polyxmass.org

Filippo Rusconi, PhD
Chargé de recherches au CNRS

Centre national de
la Recherche scientifique

UMR CNRS 5153 - UR INSERM 565 - USM MNHN 0503
Muséum national d'Histoire naturelle
43, rue Cuvier
F-75231 Paris Cedex 05
France

GNU polyxmass User Manual
Copyright (C) 2001, 2002, 2003, 2004, 2005 by Filippo Rusconi

http://www.polyxmass.org

This documentation and all its accompanying files are part of the **GNU polyxmass** project. They are software and are an integral part of the software they document.

The **GNU polyxmass** project is an official GNU project package (see www.gnu.org) released —in its entirety— under the GNU General Public License and was started at the Centre National de la Recherche Scientifique (FRANCE), that granted me the formal authorization to publish it under this Free Software License.

This software is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

A copy of the license is included in the appendix entitled "GNU General Public License".

This software is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this software; if not, write to the Free Software Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA 02110-1301, USA.

For more details see the file `COPYING` in the **GNU polyxmass** distribution files.

Revision History

- december 2005: minor corrections and addition of the Find/Replace section in the **polyxedit** chapter.
- october 2005: update the address of the Free Software Foundation wherever needed.
- august-september 2005: many changes in the doc package so that the LaTeX source can be almost automagically translated to HTML using the tex4ht package.
- july 2005: updated the **polyxedit** chapter about how invalid characters in clipboard-imported sequences can be "purified-out". Also switched the licence from the GFDL to the GPL.
- june 2005: update of the manual to document the window management features and the exporting of results reports. Also made adjustments to the Debian installation part due to **GNU polyxmass** being now part of Debian GNU/Linux.
- 12th of february, 2005: update the `Fink` installation paragraph according to a document that was communicated by D^{r} Mark Tracy.
- 24th of january, 2005: updated the documentation to reflect the addition of the m/z ratio calculations feature. Two chapters were affected: the one for **polyxcalc** and the one for **polyxedit**.
- 7th of december, 2004: updated the documentation to reflect additions in the software. Namely, the calculations of net electrical charges beared by polymers, and the isoelectric point calculations.
- october the 23rd 2004, updated the documentation with details about the way to produce atom/polymer chemistry definition packages and customization, as the scripts initially set up in the **GNU polyxmass** software package were removed.

* october 2004, updated the documentation with the new contents of the `sounds.dic` file and the renaming of `monomer-modif.dic` in `monicons.dic`. Also added a section about self-speaking polymer sequences in the **polyxedit** chapter.

* september 2004, changed the whole documentation due to a big refactoring of the whole **GNU polyxmass** software suite.

* february 2004, added a section on the debian package use. Mentioned the fact that the configuration schema changed, for the configuration files of the different modules go now in /usr/share/polyxmass.d.

* november 2003, added a number of descriptions and made substantial modifications to the **polyxedit** chapter, in particular pertaining to the annotation system (improved heavily recently) and to the user feedback that is given in the polymer sequence editor when the user points monomers with the mouse cursor.

* october 2003, additions to the **polyxedit** chapter to describe the annotation system. Also added a small section on the creation of brand new polymer sequences, as this was not described before.

* august 2003, corrections here and there; modified the **polyxcalc** chapter to reflect the changes in the graphical user interface and the chemical pad's layout configuration file. Corrected a doc bug about Andreas Fink leading the Fink project (which is false);

* july 2003, big work on the polyxmassdata chapter that describes the **GNU polyxmass** file-system hierarchy. Also added some notes on the **GNU polyxmass** installation on the *Mac OS X* system (thanks Mark Tracy for these notes);

* july 2003, continued working on the **polyxedit** module's chapter. Removed any proprietary font embedding. Chapter and Section titles now are typeset using freely available fonts (Palatino).

* june 2003, started a major overhaul of the document as the **GNU polyxmass** software program was entirely rewritten during the last numerous months. The organization of the document will be modified so that the document reflects better the new modularity of the **GNU polyxmass** software suite.

* july 2002, added the section on the molecular calculator (**polyxcalc**).

* july 2002, changed author's address from "University of Bordeaux" to "Present address: Muséum national d'Histoire naturelle" in Paris as I have now moved to the Laboratoire de biophysique.

* july 2002, back to the Computer Modern set of fonts in the text of the manual. pdflatex still best way to get to a nice pdf file.

* july 2002, added a description of the find/replace procedures.

* july 2002, made all the big pictures again better managing their size. The overall size of the document has fallen to 2.2 Mb, while it was of more than 5 Mb.

* april 2002, added a description of how to set some values to customize the program in the resources.tex file. New screen dumps allow to describe the process easily.

* april 2002, moved general-options.tex to resources.tex.

* april 2002, changes related to the fact that the packages are no more relocatable.
* april 2002, changes related to the fact that the directory into which the **GNU polyxmass** program is installed is from now on `polyxmass` and not `polyxmass`. So some macros were edited in order to produce the correct typographical results. The pxm-macros.tex file is not dynamically regenerated from a pxm-macros.tex.in file upon autotools processing. That allows a very close correlation between the version of the software package and the version elements' rendering throughout all the text.
* april 2002, updated the chapter on the **GNU polyxmass**' file-system standard to reflect the changes in the program workings (the fact that the ressource files are now in a hidden directory in the user's home directory).
* april 2002, added a detailed description, in the options' configuration chapter, of each option available.
* april 2002, added a new chapter on the configuration of the options: file general-options.tex.
* end of march 2002, added a section on the chemical bridge support.
* march 2002, moved file name "config-data.tex" to "file-system-config.tex" and corresponding chapter title to reflect the real configuration issue that is dealt with in this chapter and to differentiate this configuration issue with the polymer chemistry "configuration" or "definition".
* march 2002, added the description of the graphical configuration of the **GNU polyxmass** file-system. Updated relevant parts after the `monomer.dic` file was removed from the configuration files, and its contents are moved to `polymer.dic`.
* february 2002, added a section on mass searching, as I have coded it these last days.
* february 2002, chopped polyxmass.tex into chapter parts. This is to allow easily producing chapters one apart from the other.
* february 2002, added the *PDF* thumbnail support with the wonderful package from Heiko Oberdiek (thumbpdf).
* february 2002, changed the organization of the **GNU polyxmass**-specific chapters into one single chapter, with as many sections as needed to describe all aspects of **GNU polyxmass**' operation;
* january 2002, added the "**GNU polyxmass**' Sequence Editor" chapter;
* january 2002, added the "Installing From The `rpm` Source Package" section for the sake of completeness;
* january 2002, started writing the little (but tough) chapter on the **GNU polyxmass**' configuration data hierarchy scheme;
* january 2002, a wealth of corrections after careful reading of the last version while in Christmas holidays in Viterbo (near Rome);
* december 2001, added section on UNIX history... from document by David A. Wheeler: *Secure Programming for GNU/Linux and UNIX HOWTO*;

* november 2001, switched from DocBook SGML format to LaTeX format
* october 2001, initial writing, DocBook SGML format

To MARIA CECILIA,

*To all the admirable people acting in the "*Free Software Movement*" for a better and cleaner computing world,*

To all the readers who helped me with this manual...

Contents

List of Figures

List of Tables

1

Preface

This manual is about the **GNU polyxmass** mass spectrometric software suite, a computing framework that aims at predicting/analyzing mass spectrometric data on (bio)polymers. As such, this manual is intended for people willing to learn how to install and use this multi-modular software suite.

Mass spectrometry has gained popularity across the past five years or so. Indeed, developments in polymer mass spectrometry have made this technique appropriate to accurately measure masses of polymers as heavy as many hundreds of kDa.

There are a number of utilities –sold by mass spectrometer constructors with their machines, usually as a marketing "plus"– that allow predicting/analyzing mass spectrometric data of polymers. These programs are usually different from a constructor to another. Also, there are as many mass spectrometric data prediction/analysis computer programs as there are different polymer types. You will get a program for oligonucleotides, another one for proteins, maybe there is one program for saccharides, and so on. Thus, the biochemist/massist, for example, who happens to work on different biopolymer types will have to learn the use of a number of different software packages. Also, if the software user does not own a mass spectrometer, chances are he will need to buy all these software packages.

The **GNU polyxmass** mass spectrometric computing framework is designed to provide *free* solutions to all these problems. And it does this by:

* Allowing *ex nihilo* polymer chemistry definitions (in the **polyxdef** module that sits in the **GNU polyxmass** program);
* Allowing simple yet powerful mass computations to be made in a mass desktop calculator that is both polymer chemistry definition-aware and fully programmable (that's the **polyxcalc** module also sitting in the **GNU polyxmass** program);
* Allowing highly sophisticated editing of polymer sequences on a polymer chemistry definition-specific basis, along with chemical reaction simulations, finely configured mass spectrometric computations... (all taking place in the **polyxedit** module that is the main module of the **GNU polyxmass** program);

* Allowing customization of the way each monomer will show up graphically during the program operation (in the **polyxedit** module);
* Allowing polymer sequence editing with immediate visualization of the mass changes elicited by the editing activity (in the **polyxedit** module);
* Unlimited number of polymer sequences opened at any given time and of any given polymer chemistry definition type (in the **polyxedit** module).

This manual will progressively introduce all these functionalities in a timely and clear fashion.

UNIX and *GNU/Linux* Histories

Thanks to the **GNU** Free Documentation License, I borrowed (and cosmetically modified it) the material in this section from a remarkable document by David A. Wheeler: *Secure Programming for GNU/Linux and UNIX HOWTO.* [1] I think that it is important to provide some background to the choice of a development platform when the time comes to document the software that one has taken so much time to code...

UNIX

In 1969-1970, Kenneth Thompson, Dennis Ritchie, and others at **AT&T Bell Labs** began developing a small operating system on a little-used *PDP-7*. The operating system was soon christened *UNIX*, a pun on an earlier operating system project called *MULTICS*. In 1972-1973 the system was rewritten in the programming language C, an unusual step that was visionary: due to this decision, *UNIX* was the first widely-used operating system that could switch from and outlive its original hardware. Other innovations were added to *UNIX* as well, in part due to synergies between **Bell Labs** and the academic community. In 1979, the "seventh edition" (V7) version of *UNIX* was released, the grandfather of all extant *UNIX* systems.

After this point, the history of *UNIX* becomes somewhat convoluted. The academic community, led by Berkeley, developed a variant called the Berkeley Software Distribution (*BSD*), while **AT&T** continued developing *UNIX* under the names "*System III*" and later "*System V*". In the late 1980's through early 1990's the "wars" between these two major strains raged. After many years each variant adopted many of the key features of the other. Commercially, *System V* won the "standards wars" (getting most of its interfaces into the formal standards), and most hardware vendors switched to **AT&T**'s *System V*. However, *System V* ended up incorporating many *BSD* innovations, so the resulting system was more a merger of the two branches. The *BSD* branch did not die, but instead became widely used for research, for PC hardware, and for single-purpose servers (e.g., many web sites use a *BSD* derivative).

The result was many different versions of *UNIX*, all based on the original seventh edition. Most versions of *UNIX* were proprietary and maintained by their respective hardware vendor, for example, **Sun** *Solaris* is a variant of *System V*. Three versions of the *BSD* branch of *UNIX* ended up as open source: *FreeBSD* (concentrating on ease-of-installation for PC-type hardware), *NetBSD* (concentrating on many different CPU architectures), and a variant

[1] Get this paper and others at http://www.dwheeler.com

of *NetBSD*, *OpenBSD* (concentrating on security). More general information about *UNIX* history can be found at http://www.levenez.com/unix/.

Free Software Foundation

In 1984 Richard Stallman's **Free Software Foundation** (**FSF**) began the **GNU** project, a project to create a free version of the *UNIX* operating system. By free, Stallman meant software that could be freely used, read, modified, and redistributed. The **FSF** successfully built a vast number of useful components, including the `GNU compiler collection` (`gcc`), an impressive text editor (`GNU Emacs`), and a host of fundamental tools. However, in the 1990's the **FSF** was having trouble developing the operating system kernel; without a kernel the rest of their software would not work.

GNU/Linux

In 1991 Linus Torvalds began developing an operating system kernel, which he named "Linux". This kernel could be combined with the **FSF** material and other components (in particular some of the *BSD* components and Massachusetts Institute of Technology's (**MIT**) `X Window` software) to produce a freely-modifiable and very useful operating system. This book will term the kernel itself the "Linux" kernel and an entire combination as "*GNU/Linux*".

In the *GNU/Linux* community, different organizations have combined the available components differently. Each combination is called a "distribution", and the organizations that develop distributions are called "distributors". Common distributions include **Red Hat**, **Mandrake**, **SuSE** and **Debian**. There are differences between the various distributions, but all distributions are based on the same foundation: the Linux kernel and the **GNU** `glibc` libraries. Since both are covered by "copyleft" style licenses, changes to these foundations generally must be made available to all, a unifying force between the *GNU/Linux* distributions at their foundation that does not exist between the *BSD* and **AT&T**-derived *UNIX* systems.

Open Source vs Free Software

Increased interest in software that is freely shared has made it increasingly necessary to define and explain it. A widely used term is "open source software". Eric Raymond wrote several seminal articles examining its various development processes. Another widely-used term is "free software", where the "free" is short for "freedom": the usual explanation is "free speech, not free beer". Neither phrase is perfect. The term "free software" is often confused with programs whose executables are given away at no charge, but whose source code cannot be viewed, modified, or redistributed. Conversely, the term "open source" is sometimes (ab)used to mean software whose source code is visible, but for which there are limitations on use, modification, or redistribution. This book uses the term "open source" for its usual meaning, that is, software which has its source code freely available for use, viewing, modification, and redistribution; a more detailed definition is contained in the Open Source Definition. Information on the definition of free software, and the motivations behind it, can be found at http://www.fsf.org.

Those interested in reading advocacy pieces for open source software and free software should see http://www.opensource.org and http://www.fsf.org. There are other doc-

uments in the internet which examine such software, for example, authors have found that the open source software were noticeably more reliable than proprietary software (using their measurement technique, which measured resistance to crashing due to random input).

Typographical conventions

Throughout the book the following typographical conventions are used:

* *emphasized text* is used each time a new term or concept is introduced
* `shell-prompt $` shows the prompt at which a command should be entered as non-root
* `shell-prompt #` shows the prompt at which a command should be entered as root
* `this typography` applies to commands that the user enters at the shell prompt along with eventual options
* ↩ symbolizes pressing the *Enter* key.
* `this typography` applies to an output resulting from entering a command at the shell prompt
* `emacs` or `libglib` names of a program or of a library
* `GNOME`, `The Gimp` is the name of a generic software (not a specific executable file)
* `/usr/local/share`, `/usr/bin/polyxmass` are names of a directory or of a file
* http://www.gnu.org is a URL (Uniform Resource Locator)

Program Availability, Technicalities

GNU polyxmass has been initially developed on a *GNU/Linux* system (**RedHat** distribution versions successively 6.0, 7.0, 7.2, 7.3, 8.0, 9.0) using software from the **Free Software Foundation** (**FSF**[2]). Since mid-2002, the development is performed on a *Debian GNU/Linux* system (http://www.debian.org) which I find the ultimate highly-configurable easy-to-use distribution on earth.

Developing for *GNU/Linux* has been utterly exciting and extremely efficient. My warm thanks do go to all the persons who have engaged themselves (energy and time) in *Free Software/true Open Source* by coding, documenting, reviewing... software. The development was mainly centered around the following programs and utilities:

* **GNU** software is central to my developing system:
  - `GNU Emacs`, a text editor that is an environment *per se*
  - `Autotools`, an integrated set of programs to make software development easy and portable. Includes `Autoconf`, `Automake` and others...
    (http://www.gnu.org, home of the *Free Software Movement*);
  - `GDK/GTK+`, two libraries for windowing in the X Window graphic environment (http://www.gtk.org);

[2]For an in-depth coverage of the philosophy behind the **FSF**, specifically creating a *free operating system*, you might desire to visit http://www.gnu.org

  - ✦ `The Gimp`, a wonderful program for doing graphical illustrations in pixel mode (raster images). Think of it as an excellent free replacement for the `Photoshop` program. The "icons" representing each single monomer in the sequence editor were made using `The Gimp`. It saves in *xpm*, *png*, *jpg* and many other graphic formats
    (http://www.gimp.org);
  - ✦ `GNOME`, a graphical environment for the *GNU/Linux* desktop. I used the `GNOME` canvas widget to tailor the sequence editor
    (http://www.gnome.org);

* Thomas Esser has made a TEX/LATEX environment of exceptional quality. I used it everyday, and typeset this manual using it. Of course, Prof. Donald Knuth is the grand-daddy of all this, having invented TEX and Leslie Lamport is the father of LATEX! (http://www.tug.org; search for `teTeX`);

* `Glade` is a wonderful graphical interface builder (by Damon Chaplin) that I used to design the graphical interface of the program. I used it in conjunction with the `libglade` library (by James Henstridge)
  (http://glade.gnome.org and
  http://www.daa.com.au/~james/software/libglade);

* **RedHat** is undoubtedly committed to the success of the *Free Software Movement* and happens to be the maker of a popular (my) *GNU/Linux* distribution
  (http://www.redhat.com);

* Bernhard Herzog has written a vector drawing package that I used for some illustrations in the **GNU polyxmass** package. It is called `Sketch`
  (http://sketch.sourceforge.net);

* Lauris Kaplinski and co-workers have crafted a very powerful program to create and handle scalar vector graphics. This program is called `Sodipodi`
  (http://sodipodi.sourceforge.net);

* Owen Taylor has written a memory profiling tool that I used —during the RedHat*GNU/Linux*-based development— to detect memory leaks. It is called `memprof`
  (otaylor{@}redhat.com, remove the curly brackets);

* Of course I do forget many software packages that I used for this work. Thanks to their authors and to their maintainers: without their hard work my *GNU/Linux* box would not exist!

Organization Of This Manual

After having rapidly explained the general pattern about installing each of the modules that make the **GNU polyxmass** software suite, this manual aims at providing the required concept toolset for understanding what to expect from a computer program project like **GNU polyxmass**. Thus, the general organization of this book is:

* Installation of the **GNU polyxmass** modules;

* The basics of polymer chemistry;

* The basics of mass spectrometry;
* Generalities about the **GNU polyxmass** software;
* The **polyxdef** chapter (definition of atoms and of new polymer chemistries);
* The **polyxcalc** chapter (polymer chemistry-aware programmable calculator);
* The **polyxedit** chapter (sequence editor, biochemical/mass spectrometric simulations);
* The **GNU polyxmass-common** chapter describing the fundamental configuration/data files that are required to run the **GNU polyxmass** software;
* The **GNU polyxmass-data** chapter describing the **GNU polyxmass**' complex chemical configuration hierarchy;
* Appendices.

GNU polyxmass' Licensing Philosophy

The front matter of this manual contains a Copyright statement. I wish to retain the copyright to **GNU polyxmass** and all related writings (source and configuration files, programmer's documentation, user manual...) However, I do not deny others the right to make copies of the work, to redistribute it freely, to modify it according to the **GNU** General Public License for the **GNU polyxmass** computer program, and according to the **GNU** Free Documentation License.

The aim of this licensing is to favor spread of knowledge to the widest public possible. Also, it encourages interested hackers[3] to change the code, to improve it and to send patches to the author so that their improvements get in the program to the benefit of the widest public possible. For an in-depth study of the *free software* philosphy I kindly urge the reader to visit http://www.gnu.org/philosophy.

Contacting The Author

GNU polyxmass program is the fruit of months of work on my part. While I've put a lot of energy into making this program as stable and reliable a piece of software as possible, **GNU polyxmass** comes with no warranty of any kind. I hope that **GNU polyxmass** will help numerous researchers with their mass spectrometric data prediction/analysis work, which will hopefully ease the creation of *scientific knowledge*.

The general policy for directing questions, comments, feature requests, **GNU polyxmass** program and/or **GNU polyxmass** documentation bug reports should be self-explanatory by looking at the addresses below:

polyxmass-webmaster@polyxmass.org
polyxmass-maintainer@polyxmass.org
polyxmass-bugs@polyxmass.org
polyxmass-request@polyxmass.org

[3] *Hacker* is a specialized term to design the programmer who codes programs; this term should *not* be mistaken with *cracker* who is a person who uses computer science knowledge to break information systems' security barriers.

To direct any comment(s) to the author through snail mail, use the following address:

D[r] Filippo Rusconi

Chargé de recherches au CNRS
Centre national de
la Recherche scientifique

UMR CNRS 5153 - UR INSERM 565 - USM MNHN 0503
Muséum national d'Histoire naturelle
43, rue Cuvier
F-75231 Paris Cedex 05
France

2

Installation Overview

The **GNU polyxmass** software suite is a multi-modular software framework. It is made of a number of modular packages that depend on each other. The installation of the **GNU polyxmass** software suite can be achieved with no pain by following the instructions in this chapter.

The dependencies between the modules of the **GNU polyxmass** software framework are "ordered", which means that they require that the modules of the framework be installed on the same system in an ordered manner.

Modules from the **GNU polyxmass** software suite might be modified independently, which means that it is not required that they have the same package version number. For example, a modification in the `polyxmass` program of the **GNU polyxmass** software suite might not necessarily involve changes in the `libpolyxmass` library —that is also part of the **GNU polyxmass** software suite. Thus, the **GNU polyxmass** program will have an incremented version number, while the library itself remains with a constant version number. The dependencies are dealt with at install time, so the best way to make a fresh install of the **GNU polyxmass** software suite is to take all the most recent packages from `http://www.polyxmass.org`. If there are no errors (*errare humanum est*) in the dependency mechanics, all the packages should be installed easily.

Prior to analyzing the installation procedure as a whole, it is necessary to describe the packaging systems that are available for the user to install, manage and remove software packages from the system.

Making packages for distributions comes at the price of big efforts from the packager. The program is always first distributed as a *tar.gz* "tarball" where all the sources required to compile the software are located. This "tarball" packaging is the most natural one for the developer, as it is very near to what she has used to develop the software. Packagers use these "tarball" packages to prepare binary packages so that non-geek users can easily install precompiled software onto their machine. Preparing binary packages from source is a big work, and packagers do not always have the time required to do it in a short delay after new releases of the program. Please, be patient and wait for the packages to be prepared. If you find that the packages come too slowly after software release, learn how to package software and prepare the packages yourself. I'll be happy to offer packages that you prepared on my server http://www.polyxmass.org.

Delivering software to the users can be performed using a number of file formats:

* Uncompiled source package file formats:
  - *tar.gz* files which need to be compiled using the `GNU make` program (these are the "tarball" archives mentioned earlier);
  - *dsc* plus *tar.gz* files which need to be compiled into a binary installable package using the *Debian GNU/Linux* packaging system;
  - *src.rpm* files which need to be compiled into a binary installable package using the `rpm` tool;
* Binary ready-to-install package file formats:
  - *_i386.deb* files that are dependent on the computing platform architecture (must be installed using the `apt-get` or the `dpkg` tools in the *Debian GNU/Linux* distribution);
  - *i386.rpm* files that are dependent on the computing platform architecture (must be installed using the `rpm` tool);
  - *_all.deb* files that are non-dependent on the computing platform architecture (must be installed using the `apt-get` or the `dpkg` tools in the *Debian GNU/Linux* distribution);
  - *noarch.rpm* files that are non-dependent on the computing platform architecture (must be installed using the `rpm` tool).

Installing From The *tar.gz* Sources

Installing a package from the source is as easy as issuing the following commands in the correct order:

```
shell-prompt $ cp polyxmass-0.9.0.tar.gz /tmp ↩ copy the package file into a safe place
shell-prompt $ cd /tmp ↩
shell-prompt $ tar xvzf polyxmass-0.9.0.tar.gz ↩ this unpacks the sources into a source tree in the polyxmass-0.9.0 directory
shell-prompt $ cd polyxmass-0.9.0 ↩
shell-prompt $ ./configure --prefix=/usr --sysconfdir=/etc ↩ these are the two conventional options used with the ./configure command
```

```
shell-prompt $ make ↩P perform the actual compilation of the software
shell-prompt $ su ↩P become root if it is possible
shell-prompt # make install ↩P finally ask that the successfully compiled software be installed to
the destination system directories, as required using the options to the ./configure command
```

As seen in the commands shown above, there are two options that the user might try when running the `configure` script before building the software:

- `--prefix=/usr`: this option governs the file-system tree where the software is to be installed. Binary packages always install new software in the file-system rooted at location `/usr`. Source packages, like the ones being described right now, usually install software in the file-system rooted at location `/usr/local`, as it is assumed that such software is only for use by the local machine, and does not enter in the composition of the default machine setup. It is possible to force source packages to install into the `/usr` file-system tree by using the option `--prefix=/usr`;
- `--sysconfdir=/etc`: this options governs the installation of the most critical configuration files of the whole **GNU polyxmass** software suite. Binary packages set this option to `/etc`, which means that all the critical configuration files are located in the `/etc/polyxmass` directory. The default location for source packages, like the ones described here, is also `/etc`.

The only reason why the user (who is installing software and not merely using it) will want to use/modify the options above, is to install software without having system administrator priviledges (*i.e.* without being root). In this case, it is *essential* that both options be passed identically to the `configure` script of *each package* of the **GNU polyxmass** software suite. This is because each package of the suite is relying on the knowledge that all the other packages have been installed in a determinate file-system tree (`/usr`, `/tmp`, `/opt` or `/home/rusconi`, for example). This is crucial, otherwise all the software suite's modules cannot speak one to each other.

As one example, this is what I tried to make sure the software is —in fact— *a priori* installation directory-agnostic:

- I first installed the **GNU libpolyxmass** source package by issuing the following command lines:
  - `./configure --prefix=/tmp --sysconfdir=/tmp/etc`
  - `make install`[1]

    At this time the library is actually made available to the whole system and the polyxmass program will be able to use it.
- Next I installed the **GNU polyxmass-common** package by issuing the same configuration/installation commands:
  - `./configure --prefix=/tmp --sysconfdir=/tmp/etc`
  - `make install`

    At this time the fundamental configuration/data files are located in a number of locations all rooted in `/tmp`: `/tmp/etc/polyxmass`, `/tmp/share`, `/tmp/sbin`.

[1] Now the `/etc/ld.so.conf` file should be edited (as root) in order to add to it the following line: `/tmp/lib`. Once this line is added and the file saved, the user (as root) must execute the command `/sbin/ldconfig`.

When the operations above were performed successfully I could launch the `polyxmass` program from `/tmp/bin`. Since that directory is not in the PATH, it is necessary to first change the working directory to this directory, using `cd /tmp/bin`. Next it is necessary to execute `polyxmass` by telling the shell that the executable is to be found in the current working directory: `./polyxmass` . The program should launch successfully.

Installing From The `deb` Binary Package

On the *Debian GNU/Linux* distribution, the packaging system makes the installation of software incredibly easy. The `apt-get` package manager is particularly clever in determining the dependencies between packages in an inter-related array of two or more packages. The `apt-get` package manager needs to connect to some place over the network and download a file from that place (which is called a software repository) that will tell what packages are available at that specific place and how they inter-relate. Since the adoption of **GNU polyxmass** as an official *Debian GNU/Linux* distribution package, installing the software is a easy as issuing the following command lines as root:

```
shell-prompt # apt-get update ↩
shell-prompt # apt-get install polyxmass ↩
```

and something like the following will be output to your terminal:

```
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  polyxmass-bin libpolyxmass1 polyxmass-common polyxmass-data
The following NEW packages will be installed:
  polyxmass-bin libpolyxmass1 polyxmass-common polyxmass polyxmass-data
0 upgraded, 3 newly installed, 0 to remove and 9 not upgraded.
Need to get 5020kB of archives.
After unpacking 8577kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

All this output to install one package? —"What is it going on?," you may ask. Well, that's the power of the system: `apt-get` has detected that in order to install the polyxmass package, it is necessary to first install packages onto which polyxmass actually depends. These packages are listed:

```
  The following extra packages will be installed:
  polyxmass-bin libpolyxmass1 polyxmass-common polyxmass-data
```

Which means that, in total, with the initially requested package,

```
The following NEW packages will be installed:
  polyxmass-bin libpolyxmass1 polyxmass-common polyxmass-data polyxmass
0 upgraded, 5 newly installed, 0 to remove and 9 not upgraded.
```

If you accept to continue (key-in [Y] ↩), this is what you get: all the packages get automatically downloaded, installed and configured. That's the *Debian GNU/Linux* packaging system magic: there is no higher standard for such package managing tasks nowadays.

To see all the files that are provided by a given *Debian GNU/Linux* package file, issue the following command:

```
shell-prompt $ dpkg -c polyxmass-bin0.9.0-1.i386.deb ↩
```

The output of this command is a list of all the files —along with their destination directories— that would be installed if the package were installed as above.

Installing From The `rpm` Binary Package

Installing any *`rpm`* package using the `rpm` program[2] is as easy as entering the following commands, as root:

```
shell-prompt # rpm -ivh polyxmass-common-0.8.6-1.i386.rpm ↩
shell-prompt # optionally rpm -ivh polyxmass-data-0.8.5-1.i386.rpm ↩
shell-prompt # rpm -ivh libpolyxmass-0.9.0-1.i386.rpm ↩
shell-prompt # rpm -ivh polyxmass-0.9.0-1.i386.rpm ↩
```

What these commands do is to read the contents in each of the packages (these package files do contain a number of files packed in them) and copy them to their destination directories. The *`rpm`* file format lets the packager define where each of the files contained in a package has to be installed. To see all the files that are provided by a given `rpm`-based package file, issue the following command:

```
shell-prompt $ rpm -qpl polyxmass-0.9.0-1.i386.rpm ↩
```

The output of this command is a list of all the files —along with their destination directories— that would be installed if the package were installed as above.

Installation On A *Mac OS X* System With `Fink`

The *Mac OS-X* operating system can run **GNU** software when the `Fink` porting system is installed (please, visit `http://fink.sourceforge.net` for details on this project). The notes below were kindly provided to me by D^{r} Mark Tracy. If you find errors, they are mine, and I am the only one to be blamed for badly transcribing these notes.

GNU polyxmass was successfully installed on the Mac OS-X/Fink platform. For example, version 0.8.6 of the modules of the **GNU polyxmass** software suite could be installed using the info files maintained by D^{r} Mark Tracy. These `Fink` *`info`* files are scripts much like the `rpm` *`spec`* files. Using the scripts, `Fink` will download the source tarballs, apply any patches, install dependent packages, compile the code and install the finished files. The `Fink` packaging system relies on the usual *`tar.gz`* source files, which may be used without modification. However, the case may arise that the *Mac OS-X/Fink* platform requires that the package maintainer changes the code of the source tree for one or more packages in the **GNU polyxmass** suite. In this case, the patch files would be distributed along with the source

[2] For an in-depth manual on the `rpm` packet manager, you might want to read *Maximum RPM*, a book by Ed Bailey, available from `http://www.rpm.org`.

tarball files and the *info* files. The *info* script will automatically apply the patches, and complain should you forget them. Providing patch files for the software to build correctly on any given platform is the task of the packager. Once you have downloaded all the required files (*info*, *patch*), the installation process is as easy as doing the following: If you obtained the scripts from the Fink server, Fink will automatically install them in the right place and you skip the copying step. The Fink service tends to lag behind the latest version due to their diligent review process. If you obtained the scripts from http://www.polyxmass.org, you need to copy them to the right place:

```
shell-prompt $ sudo cp *.info /sw/fink/10.3/local/main/finkinfo↩
shell-prompt $ sudo cp *.patch /sw/fink/10.3/local/main/finkinfo↩
```

Sudo will want your password. Now comes the time to install the packages by issuing the following commands:

```
shell-prompt $ fink install polyxmass↩
shell-prompt $ fink install polyxmass-common↩ (optional)
shell-prompt $ fink install polyxmass-doc↩ (optional)
shell-prompt $ fink install polyxmass-examples↩ (optional)
```

At the behest of the Fink team, the packages are named somewhat differently under Fink. If the software packager did everything right, Fink will calculate the dependencies, and ask you if you want to install the dependent packages (say yes).[3] Fink will begin downloading the source tarballs. Note that **GNU polyxmass** is dependent on several libraries found in the unstable branch of the 10.3 package tree, therefore Fink must be so configured (sorry, 10.2 is no longer supported). When all is finished, open a new X-terminal window to run the software (yes, it has to be new and it has to be X).

Summary

We have reviewed a number of ways the **GNU polyxmass** software suite might be installed on a variety of platforms. The next chapters will deal with each module separately.

As a final reminder, before the user picks the best installation method and does that installation, I would tell that the software packages in the **GNU polyxmass** software suite should be installed in the following order:

1. `polyxmass-common`
2. `libpolyxmass`
3. `polyxmass-bin` (or `polyxmass`, depending on the distribution)
4. *optionally* `polyxmass-data`
5. *optionally* `polyxmass-doc`

The following chapters will describe the software suite with all the details that are required so that the user gets an intimate knowledge of the way the whole integrated **GNU polyxmass** mass spectrometric software suite works.

[3] If this is your first time installing, there will be a startling number of dependent files.

3

Basics in Polymer Chemistry

This chapter will introduce the basics of polymer chemistry. The way this topic is going to be covered is admittedly biased towards mass spectrometry and biological polymers. Moreover, the aim of this chapter is to provide the reader with the specialized words that will later be used to describe and explain the (inner) workings of the **GNU polyxmass** program. This manual is not a "crash course" in biochemistry!

Polymers? Where? Everywhere!

Indeed, polymers are everywhere. If you ask somebody to show you something polymeric, he/she will point you at the first plastic object in the vicinity. Right, plastic materials are made of hydrocarbon polymers. But we have many different polymers in our body. Proteins are polymers, complex sugars are polymers, DNA (the so-called "molecule of heredity" is a *huge* polymer. There are polymers in wine, in wood... Where? Everywhere!

The *Oxford Advanced Learner's Dictionary of Current English* gives for *polymer* the following definition: ***natural or artificial compound made up of large molecules which are themselves made from combinations of small simple molecules***.

A polymer is indeed made by covalently linking small simple molecules together. These small simple molecules are called *monomer*s, and it is immediate that a *polymer* is made of a number of monomers. A general term to describe the process that leads to the formation of a polymer is *polymerization*. It should be noted that there are many ways to polymerize monomers together. For example, a polymer might be either linear or branched. A polymer is linear if the monomers that are polymerized can be joined at most two times. The first junction links the monomer to an elongating polymer (thus making it the new end of the elongating polymer which, by the way, is longer than before by one unit) and the second junction links the new elongating polymer's end to another monomer. This process goes on until the reaction is stopped, the point at which the polymer reaches its *finished state*. A branched polymer is a polymer in which at least one monomer is able to contract more than two bonds. It is thus clear that a single monomer linked three times to other monomers will yield a "T-structure", which is nothing but a branched structure.

In the following sections we'll describe a number of different kinds of polymers. Each time, they will be described by initially detailing the structure of their constitutive monomers; next the formation of the polymer is described. At each step we shall try to set forth each polymer characteristics in such a manner as to introduce the way **GNU polyxmass**' "thinks polymers" and to introduce specialized terminologies.

Once the basic chemistries (of the different polymers) have all been described, we will enter a more complex subject that is of enormous importance to the mass spectrometry specialist: polymer chain disrupting chemistry. We shall see that this terminology actually involves two kinds of chemistries: cleavage on the one hand and fragmentation on the other hand.

While **GNU polyxmass** is basically oriented to linear single stranded polymer chemistries, it also can be used to simulate highly complex polymer chemistries. Biological polymers are the main focus of this manual, however all the concepts described here may be applied with no modification (or so slight) to synthetic polymer chemistries.

Well, time has come to make a "biochemical polymers" tour. The reader who feels at home with biopolymers may skip joyfully the next sections. However, the section pertaining to polymer lysis and fragmentation should be of interest even to the expert because they are the opportunity to introduce a "funny" terminology that is not encountered anywhere else (have you ever heard of *"leftrighrules"* or of *"fragrules"*?!).

Various Biopolymer Structures

Biopolymers are amongst the most sophisticated and complex polymers on earth and it certainly is not a mistake to take them as examples of how monomers (be these complex or not) can assemble covalently into life-enabling polymers. In this section we will visit three different polymers encountered in the living world: proteins, nucleic acids and polysaccharides. We shall be concerned with 1) the monomers' structure, 2) the polymerization reaction and 3) the final capping reaction responsible for putting the polymer in its *finished state*.

Proteins

These biopolymers are made of amino acids. There are twenty major amino acids in nature, and each protein is made of a number of these amino acids. The combinations are infinite, providing enormous diversity of proteins to the living world.

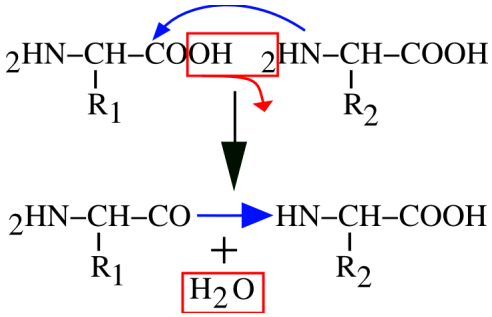

Figure 3.1: **Peptidic bond formation by condensation.** The left end monomer R_1 is condensed to the right end monomer R_2 to yield a peptidic bond. A water molecule is lost during the process.

A protein is a polar polymer: it has a left end and a right end. This means that the polymerization process is something ordered, from left to right.

The Figure 3.1 shows that the chemical reaction at the basis of protein synthesis is a *condensation*. A protein is the result of the condensation of amino acids with each other in an orderly polar fashion. A protein has a left end (called *N terminus; amino terminal end*) and a right end (called *C terminus; carboxyl terminal end*). The left end is an amino group ($_2$HN–) corresponding to the amino group of the non-reacted amino acid. Upon condensation of a new amino acid onto the first one, the carboxyl group of the first amino acid reacts with the amino group of the second amino acid. A water molecule is released, and the formation of a bond between the two amino acids yields a dipeptide. The right end of the dipeptide (and of a polypeptide –*i.e.* of a protein– also, of course) is a carboxyl group (–COOH) corresponding to the un-reacted carboxyl group of the last amino acid to have "polymerized in".

The bond formed by condensation of two amino acids is an amide bond, also called –in protein chemistry– a *peptidic bond*. The elongation of the protein is a simple repetition of the condensation reaction shown in Figure 3.1, granted that the elongation *always* proceeds in the described direction (a new monomer arrives to the right end of the elongating polymer, and elongation is done from left to right).

Now we should point at a protein chemistry-specific terminology issue: we have seen that a protein is a polymer made of a number of monomers, called amino acids. In protein chemistry, there is a subtlety: once a monomer is polymerized into a protein it is no more called a monomer, it is called a *residue*. We could say that a residue is an amino acid less a water molecule.

From what we have seen until now, we could define a protein this way: —"*A protein is a chain of residues linked together in an orderly polar fashion, with the residues being numbered starting from 1 and ending at n, from the first residue on the left end to the last one on the right end*". This definition is still partly inexact, however. Indeed, from what is shown in Figure 3.2, there is still a problem with the extremities of the polymer chain: what about the amino group on the left end of a protein (the amino group sits right onto the first amino acid of the protein), and what about the carboxyl group of the right end of a protein (the carboxyl group sits right onto the last amino acid of the protein)? These two groups are un-reacted, in fact. If we followed the new "residue-based" definition of a protein polymer, we would say that there is a proton in *excess* on the left end and a hydroxyl in *excess* on

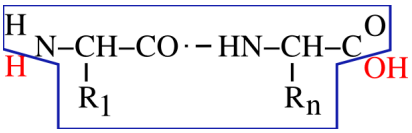

Figure 3.2: **End capping chemistry of the protein polymer.** A protein is made of a chain of residues and of two caps. The left cap is the N-terminal proton and the right cap is the C-terminal hydroxyl. Altogether, the residual chain (enclosed here in the blue polygon) and both red-colored caps (H and OH) do form a complete protein polymer.

the right end. However, these two chemical groups are not actually in *excess*, they are called (in **GNU polyxmass**) the *cappings* or *caps* of the polymer (this terminology is also used in polymer science). They ensure that the polymer is in a *finished state*, which means that it cannot be elongated anymore, on whichever end. The proton is the *left cap* of the protein polymer and the hydroxyl is the *right cap* of the protein polymer.

Now comes the question of unambiguously defining the structure of a protein. It is commonly accepted that the simple ordered sequence of each residue code in the protein, from left to right, constitutes an unambiguous description of the protein's *primary structure*. Of course, proteins have three-dimensional structures, but this is of no interest to a program like **GNU polyxmass**, which is aimed at calculating masses of polymers. To enunciate unambiguously the *sequence* of a protein, you would use a symbology like this:

using the 3-letter code of the amino acids:

Ala Gly Trp Tyr Glu Gly Lys

or, using the 1-letter code of the amino acids:

A G W Y E G K

Alanine is thus the residue 1 and Lysine is the last residue (n = 7).

This primer in protein chemistry should be sufficient for the moment. Let us now go to see how nucleic acids differ from the proteins (and they do no little).

Nucleic Acids

These biopolymers are more complex than the proteins are. This is mainly due to the fact that nucleic acids are composed of monomers that have three different parts, and because those parts differ in DNA and RNA. Nucleic acids are made of *nucleotides*. A nucleotide is the nucleic acid's brick: *a nucleotide consists of a nitrogenous base combined with a ribose/deoxyribose sugar and with a phosphate group*. There are two different kinds of nucleic acids: deoxyribonucleic acid, also known as DNA (the sugar is a deoxyribose) and ribonucleic acid, also known as RNA (the sugar is a ribose). DNA is most often found in its double stranded form, while RNA is most often found in single strand form. There are four nitrogenous bases for each: Adenine, Thymine, Guanine, Cytosine for DNA; in RNA only one of these bases changes: Thymine is replaced by Uracile.

A nucleic acid is a polar polymer: it has a left end and a right end (same as for proteins, remember?). This means that the polymerization process is something ordered, from left to right (sometimes left is up and right is down in certain vertical representations found mainly in textbooks).

This manual is not to teach biochemistry, which is why I am not going to describe the structure of the monomers in atomic detail. However, since it is important to understand how

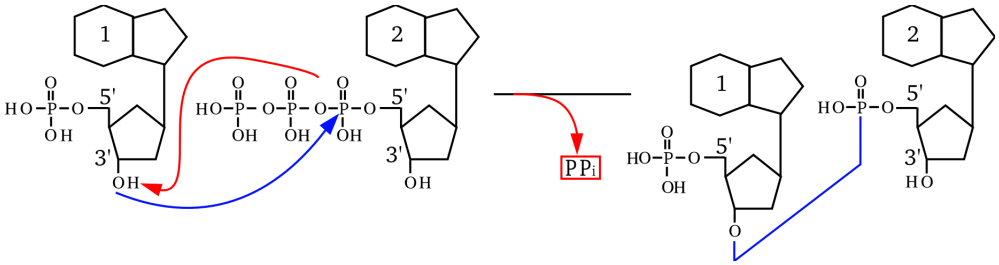

Figure 3.3: **Phosphodiester bond formation by esterification.** The arriving monomer (on the right) has its triphosphate on the 5' carbon of the sugar esterified by nucleophilic attack of the first phosphorus by the alcohol function beared by the 3' carbon of the (deoxy)ribose sugar ring of the left monomer. The bond that is formed is a phosphodiester bond, with release of a pyrophosphate group (P_i). Note that the sugar and nitrogenous bases are schematically represented in this figure.

the polymerization occurs, I drew the Figure 3.3 which shows the polymerization reaction mechanism between a nucleotide and another one, to yield a dinucleotide.

The Figure 3.3 shows that the chemical reaction that is at the basis of nucleic acid synthesis is an *esterification*. A nucleic acid has a left end (called *5' end; often this end is phosphorylated*) and a right end (called *3' end; hydroxyl end*). The reaction is the attack of the phosphorus of the new (deoxy)nucleotide triphosphate by the 3'OH of the right end of the elongating nucleotidic chain. Upon esterification, an *inorganic pyrophosphate* (PP_i) is released, and the formation of a phosphodiester bond between the two nucleotides yields a dinucleotide. The elongation of the nucleic acid polymer is a simple repetition of this esterification reaction so that the chain growth is always in the 5'$\Longrightarrow$3' direction. This is achieved in the living cells by what is called the *5'$\Longrightarrow$3' polymerase enzymatic activity*.

The conventional representation of a nucleic acid involves showing the 5' end on the left, and the 3' end on the right, horizontally. Sometimes, to clearly indicate that the left end is phosphorylated, while the right end is not, the ends are indicated as "5'P" and "3'OH".

Figure 3.4 shows a simple way to formalize what a nucleic acid polymer is. The molecule represented on the left is the representation of the "monomer" in the sense that the polymer is made of a number of these monomers (if you put in the presented formula the proper nitrogenous base and the proper sugar –ribose or deoxyribose–, you will get the nucleotide of your choice). We have seen previously that, in the specific case of the protein polymer chemistry, the monomer is called residue once it is polymerized into the polymer chain. In the case of the nucleic acids, there is no such specific term, we just call the monomeric unit a nucleotide. The formula represented on the left of the Figure 3.4 shows the repetitive element in a nucleic acid polymer, exactly the same way as we had shown the residue formula in the protein polymer chemistry section. Indeed, as we had explained earlier with proteins, the formula shown on the right of the Figure 3.4 illustrates that the nucleic acid polymer needs to be set to a *finished state*. The atoms shown in red (outside the boxed repetitive elements) are the nucleic acid *caps*. Thus, we see clearly that in the case of the nucleic acid polymers, the left cap is a hydroxyl and the right cap is a proton. This anecdotically happens to be the exact converse of what we described earlier for proteins.

Now comes the question of unambiguously defining the structure of a nucleic acid. It is commonly accepted that the simple ordered sequence of the named nitrogenous bases in the nucleic acid, from left (5' end) to right (3' end), constitutes an unambiguous description of

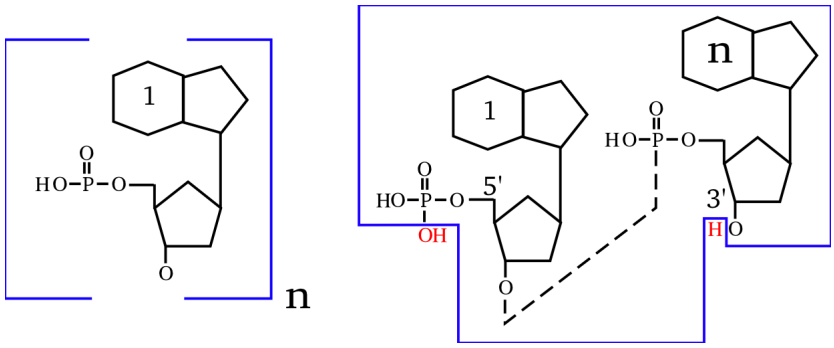

Figure 3.4: **End capping chemistry of the nucleic acid polymer.** A nucleic acid is made of a chain of nucleotides (left formula) and of two caps. The left cap is the hydroxyl group that belongs to the terminal phosphate of the 5' carbon of the sugar. The right cap is the proton that belongs to the hydroxyl group of the 3' carbon of the sugar ring (right formula). Altogether, a finished nucleic acid polymer is made of the nucleotidic chain (enclosed here in the blue polygon), made of the repetitive elements (one of which is shown on the left), and of the two caps (red-colored OH and H, out of the box on the right).

the nucleic acid sequence. To enunciate the sequence of a gene, you would use a symbology like this:
for a DNA, using the 1-letter code of the nitrogenous bases: A T G C A G T C
for an RNA, using the 1-letter code of the nitrogenous bases: A U G C A G U C
Adenine is thus the base 1 and Cytosine is the last base (n = 8).

Polysaccharides

These biopolymers are almost certainly amongst the more complex in the living world. This is mainly due to the fact that saccharides are usually heavily modified in living cells. There are a huge variety of chemical modifications occurring on these biopolymers. Furthermore, the ramifications in the polymer structure are more often the normal situation than not. Interestingly these molecules are first thought of as the "fuel" for the cell, which is certainly far from being total non-sense, but it is clear that their structural role is extremely important. Their ability to form complex structures has been exploited in living systems for identification processes. There are a number of complex sugars on the cell walls...

Nonetheless, the general picture is not that complex, if we only think of the way monomers are polymerized together. As far as we are concerned, in fact, the polymerization mechanism is a simple condensation. In this respect, everything looks much like with proteins; some people do use the same terminology: a monomer sugar becomes a residue once polymerized in the saccharidic chain.

There are two main different kinds of sugars: *pentoses* (in C_5) and *hexoses* (in C_6); it should be noted, however, that there is a variety of other common molecules, like *sialic acids*, *heptose*...

A saccharidic polymer is polar: it has a left end and a right end (same as for proteins and nucleic acid, should you remember!). This means that the polymerization process is something ordered, from left to right. The terminology regarding the ends of a saccharidic polymer is rather unexpected at first sight: the left end is said to be the *non-reducing end* while the

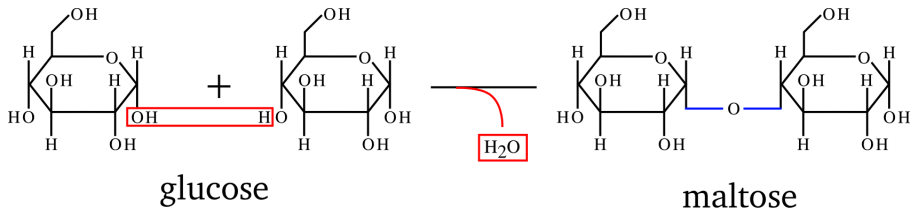

Figure 3.5: **Osidic bond formation by condensation.** The two monomers are subject to condensation with loss of one molecule of water.

right end is said to be the *reducing end*. Historically this was observed with monosaccharides (also called *monoses*), which reduced cupric (Cu^{2+}) ions, thus getting oxydized themselves on the carbonyl (when in the open ring aldehydic form).

Figure 3.5 shows the polymerization reaction between a sugar and another one (2 glucose monomers, actually), to yield a maltose disaccharide. The polymerization mechanism is a simple condensation. The elongation of the polysaccharidic polymer is a simple repetition of this condensation reaction so that the chain growth is always in the same orientation, from non-reducing end to reducing end.

The conventional representation of a polysaccharide involves showing the non-reducing end on the left, and the reducing end on the right, horizontally.

Figure 3.6 shows a simple way to formalize what a saccharidic polymer is. The top formula is the representation of the "monomer" in the sense that the polymer is made of a number of these monomers. The bottom formula represents a polysaccharide, with the repetitive elements boxed (there are n monomers polymerized). The atoms shown in red (outside the boxed repetitive elements) are the saccharidic polymer *caps*. Thus, we see clearly that in the case of polysaccharides, the left cap is a proton and the right cap is a hydroxyl. This anecdotically happens to be identical to the protein case and the exact converse of what we described previously for nucleic acids.

Now comes the question of unambiguously defining the structure of a saccharidic polymer. It is commonly accepted that the simple ordered sequence of the named monoses in the saccharidic polymer, from left (non-reducing end) to right (reducing end), constitutes an unambiguous description of the glycan sequence. To enunciate the sequence of a glycan, you would use a symbology like this:

using a 3-letter code:
Ara Gal Xyl Glc Hep Man Fru
Arabinose is thus the monose 1 and Fructose is the last monose (n = 7).

Incidentally, this is where the ability of **GNU polyxmass** to handle monomer codes of non-limited length comes in handy!

To Sum Up

rapidly made an overview of the three major polymers in the living world. A great many other polymers exist around us.

Table 3.1 on page 22 tries to sum up all the informations gathered so far. Note that the formulae given for the monomers are the "residual" ones. For example, the formula of the glycyl residue corresponds to the formula of the Glycine monomer less one molecule of water.

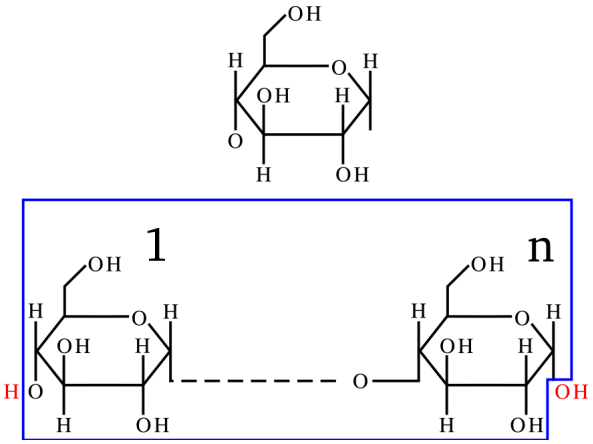

Figure 3.6: **End capping chemistry of the polysaccharidic polymer.** A polysaccharide is made of a chain of osidic residues (blue-boxed formula) and of two caps (red-colored atoms). The left cap is the proton group that belongs to the non-reducing end of the polymer. The right cap is the hydroxyl group that belongs to the reducing end of the polymer.

polymer	name	code	formula	left cap	right cap
protein				H	OH
	Glycine	G	$C_2H_3O_1N_1$		
	Alanine	A	$C_3H_5O_1N_1$		
	Tyrosine	T	$C_9H_9O_2N_1$		
nucleic acid				OH	H
	Adenine	A	$C_{10}H_{12}O_5N_5P_1$		
	Cytosine	C	$C_9H_{12}O_6N_3P_1$		
saccharide				H	OH
	Arabinose	Ara	$C_5H_8O_4$		
	Heptose	Hep	$C_7H_{12}O_8$		
Note: LC=left cap; RC= right cap					

Table 3.1: **Quick comparison of three biopolymers with examples of monomers**

Many synthetic polymers are much simpler than the ones we have rapidly reviewed, and it should be clear that, if **GNU polyxmass** can deal with the complex biopolymers described so far, it certainly will be very proficient with less complex synthetic polymers. Describing the formation of polymers is one thing, but we also have to describe how to disrupt polymers. This is what we shall do in the next section.

Polymer Chain Disrupting Chemistry

As we initially spoke of "polymer chain disrupting chemistry" earlier, we said that this was a complex subject, and that it was of *enormous* importance to the mass spectrometrist. This is why we will treat this subject in a pretty thorough manner.

First of all we should insist on the fact that chemically modifying a polymer does not necessarily mean that the chain structure of the polymer is perturbed. Here, however, we are concerned specifically with the chemical modifications that yield a polymer chain perturbation; *cleavage* and *fragmentation*:

* A CLEAVAGE IS A CHEMICAL PROCESS by which a molecule will act directly on the polymer making it fall into at least two separated pieces (the *oligomers*). As a result of the cleavage reaction, groups originating in the cleaving molecule remain attached to the polymer at the precise cleavage location;

* A FRAGMENTATION IS A CHEMICAL PROCESS by which the polymer structure is disrupted into separated pieces (the *fragments*) mainly because of energy-dependent electron doublet rearrangements leading to bond breakage.

Here are the details pertaining to each one of these two very different processes:

Polymer Cleavage

We said above that, upon cleavage of a polymer, the cleaving molecule reacts with it, and by doing so directly or indirectly "*dissolves*" an inter-monomer bond. A polymer cleavage always occurs in such a way as to generate a set of *true* polymers (smaller in size than the parent polymer, evidently, which is why they are called *oligomers*). Indeed, let us take the example shown in Figure 3.7, where a tripeptide (a very little protein, containing a methionyl residue at position 2) is submitted either to a water-mediated cleavage (hydrolysis, upper panel) or to a cyanogen bromide-mediated cleavage (lower panel). The two cases presented in this figure are similar in some respects but different in other respects:

* in both cases the bond that is cleaved is the inter-monomer bond (in protein chemistry this is a peptidic bond);

* in both cases the Oligomer 2 has the same structure;

* in the first case the molecule that is responsible for the cleavage is water, while in the second case it is cyanogen bromide;

* the structures of the Oligomer 1 species differ when produced using water or cyanogen bromide as the cleaving molecule.

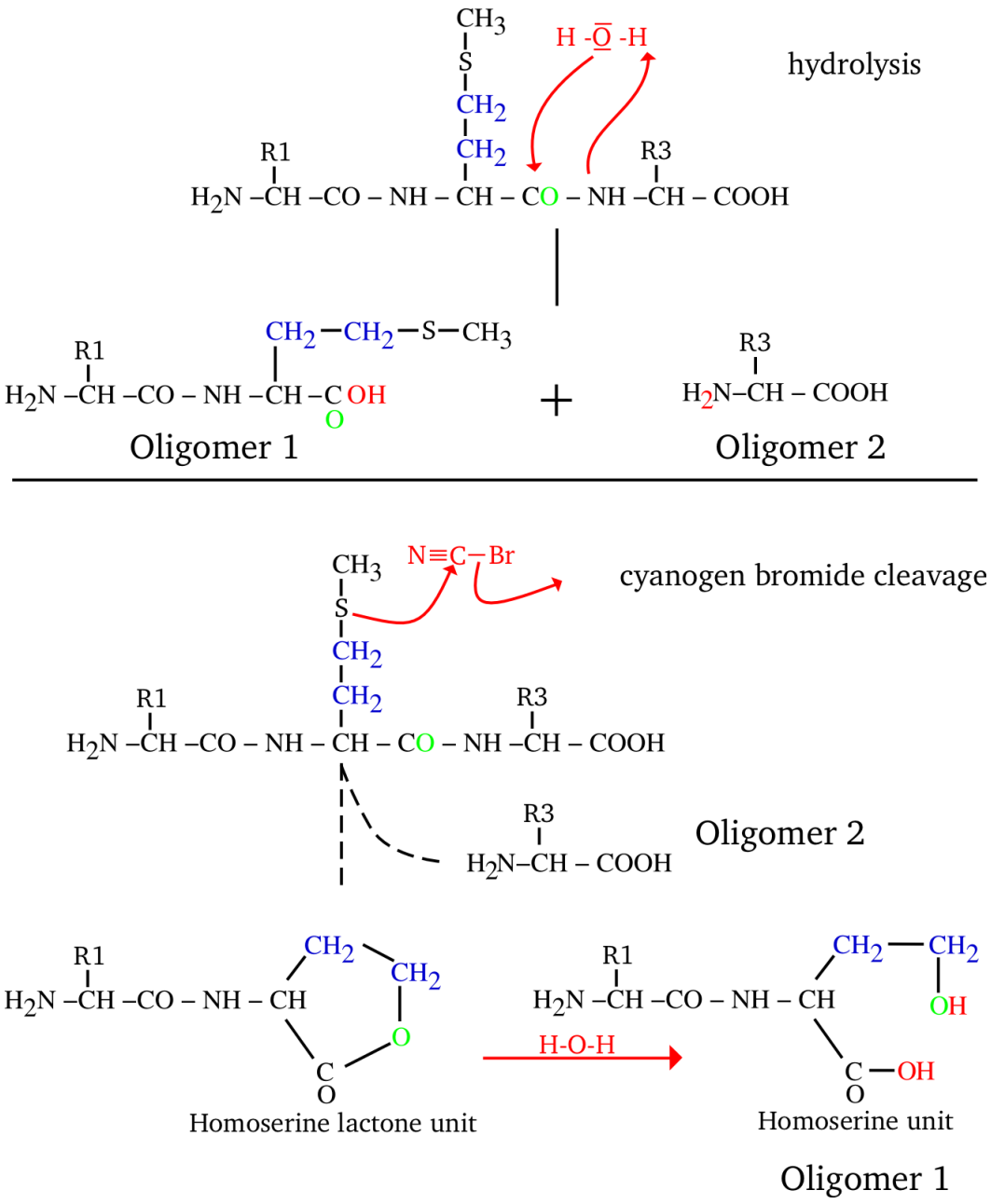

Figure 3.7: **Protein cleavage by water and cyanogen bromide.** A tripeptide (pretty small protein) is cleaved at position 1 either by hydrolysis (top) or by cyanogen bromide (bottom). Cyanogen bromide cleaves specifically on the right of a methionine monomer.

The difference between hydrolysis and cyanogen bromide cleavage is the Oligomer 1 species: the cyanogen bromide cleavage has a side effect of generating a homoserine as the right end monomer of Oligomer 1, while hydrolysis generates a genuine methionine monomer. This is because water reverses in a very symmetrical manner what polymerization did (hydrolysis is the converse of condensation), while cyanogen bromide did some chemical modification onto the generated Oligomer 1 species.

Nonetheless, the reader might have noted that –interestingly– all the four oligomers do effectively have their left cap (a proton) and their right cap (the hydroxyl). This means that in both water and cyanogen bromide-mediated cleavage, all the generated oligomers are indeed true polymers in the sense that: 1) they are a chain of monomers (modified or not) and 2) they are correctly capped (*i.e.* they are polymers in their finished state). This is important because it is the basis on which we shall make the difference between a cleavage process and a fragmentation process.

Thus, the **GNU polyxmass** definition of an oligomer might be: *an oligomer is a polymer (of at least one monomer) in its finished state that was generated upon cleavage of a longer polymer.*

When the polymer cleavage reaction precisely reverses the reaction that was performed for the same polymer's synthesis, there is no special difficulty. But when the cleavage reaction modifies the substrate, then this should be carefully modelled. How? To answer this question we might start by comparing the two different Oligomer 1 species that were yielded upon the water-mediated and the cyanogen bromide-mediated cleavage reactions: "the hydrolysis-generated Oligomer 1 is equal to the cyanogen bromide-generated Oligomer 1 +S1 +C1 +H2 -O1"; this is a big difference! The observations we did so far might be worded this way:

Whenever a protein undergoes a cyanogen bromide-mediated cleavage, the

"-C1H2S1+O1"

chemical reaction should be applied to the resulting oligomers if and only if they have a methionine monomer at their right end. This logical condition is called, in **GNU polyxmass**' jargon, a *leftrightrule*, and will be described later (see page 60).

Well, this sounds reasonable. But what about the "normal" case, when the cleavage is done using water? Nothing special: the mass of the oligomer is calculated by summing the mass of each monomer in the oligomer (since the monomers are not modified this is easily done) and the masses corresponding to both the left and right caps (these are defined in the polymer chemistry definition; in our present case it would be a proton on the left end, and a hydroxyl on the right end). In this way, the oligomer complies with its definition, which states that it is a faithful polymer made of monomers and that it is in its finished state.

Yes, but then how will **GNU polyxmass** manage to calculate the mass of the modified oligomer, like our Oligomer 1 in the case of the cyanogen bromide-mediated cleavage? Simple enough, in a first step it does exactly the same way as for the unmodified oligomer. Next, each oligomer is checked for presence or absence of a methionine residue on its right end. If a methionine is found, the mass corresponding to the "-C1H2S1+O1" chemical reaction is applied. And that's it!

In the previous cyanogen bromide example, the logical condition was involving the identity of the oligomers' right end monomer, but other examples can involve not the right end monomer, but the left end monomer, if some chemical modification was to occur to the monomer sitting right of the cleavage location. In this case the user would have to analyse the situation and provide **GNU polyxmass** with the proper chemical reaction by stating something analog to: *if and only if they have a Xyz monomer at their left end* (note the partial analogy with the case described above).

For the moment this is enough polymer cleavage abstraction, as the rest of the description pertaining to the cleavage specification definition is thoroughly detailed at page 60.

Polymer Fragmentation

In a fragmentation process, the bond that is broken is not necessarily the inter-monomer bond. Indeed, fragmentations are oft-times high energy chemical processes that can affect bonds that belong to the monomers' internal structure. This is one of the reasons why fragmentations do differ from cleavages: they are specific of the polymer type in which they occur. Hydrolyzing a protein and an oligosaccharide is just the same process, from a chemical point of view. But fragmenting a protein or an oligosaccharide are truly different processes because the way that the fragmentation happens in the polymer sequence is so much dependent on the nature of each monomer that makes it.

Another peculiarity of the fragmentations, compared with the cleavages that were described above, is the fact that there is no cleaving molecule starting the process. Instead, a fragmentation process is often initiated by an intra molecular electron doublet rearragement that propagates more or less in the polymer structure to eventually break it. Fragmentations are mainly a gas phase process, not some reaction that happens in solution as a result of putting in contact the polymer and some reagent. It is precisely because no cleaving molecule is involved in the fragmentation process that the fragments are not necessarily capped like a normal polymer should be; and this is another really important difference between cleavage and fragmentation.

Let us illustrate these concepts through two examples: proteins and nucleic acids.

Protein Fragmentation

There is a pretty important number of different kinds of fragments that can be generated upon fragmentation of peptides. We are going to detail the most common ones; the user is invited to use the **GNU polyxmass**' fragmentation-specification grammar to add less frequent (or newly discovered) fragmentation types.

As can be seen from Figure 3.8, the fragmentations do generate fragments of three categories: the ones that include the left end of the precursor polymer (a, b, c), the ones that include the right end of the precursor polymer (x, y, z), and finally the special case in which the fragment is an *internal fragment*, like the immonium ions. When looking at the fragmentations described in the figure it becomes immediately clear why a fragmentation cannot be mistaken for a cleavage: the ionization of the fragment is not necessarily due to the captation of a proton by the fragment. Furthermore, we can also see that a fragmentation is not a cleavage because the fragment that is generated is *absolutely* not necessarily what we call a polymer, in the sense that the fragment might not be capped the same way as the precursor polymer is (in its finished state).

The two observations above should make clear to the reader that calculating masses for fragments is a more difficult process than what was described above for the oligomers. Indeed, while it was simple to calculate the mass of an oligomer (by simply adding the masses of its constitutive monomer units, plus the left and right caps, plus ionization), here there is no chemical formalism generally applicable to all the fragment types. This is why the specification of the fragmentation is left to the user's responsibility.

By looking at Figure 3.8, the reader should have noticed that the fragment naming scheme takes into consideration the fact that the fragment bears the left or the right end of the precursor polymer (or none, also). Indeed, the numbering of fragments holding the

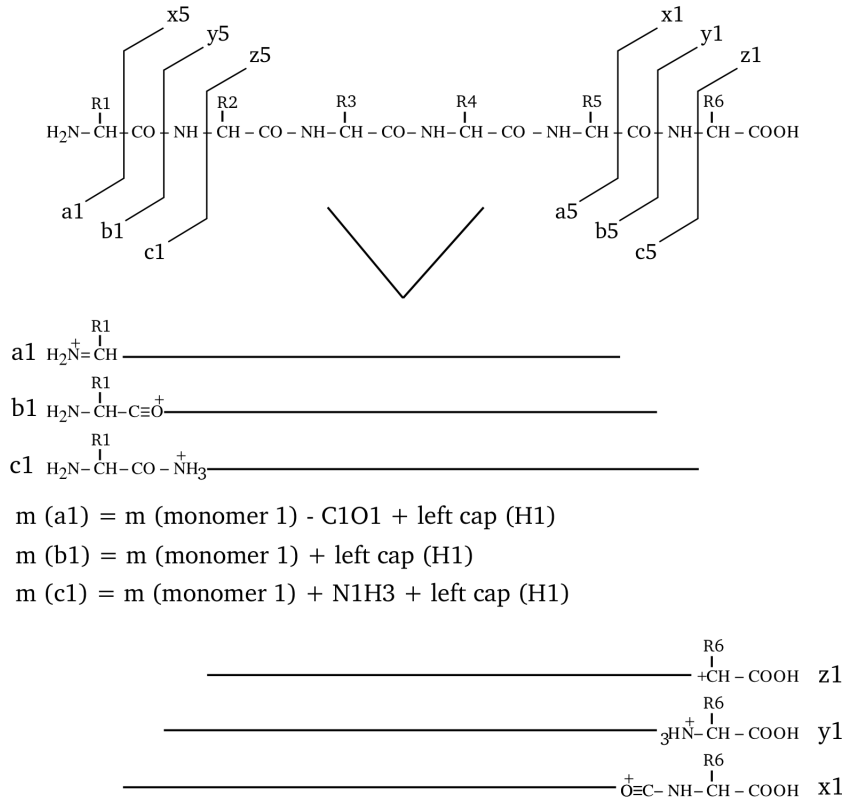

Figure 3.8: **Protein fragmentation patterns most widely encountered.** An hexapeptide is fragmented in the seven most widely encountered manners, such as to generate a, b, c, x, y, z and immonium fragment ions. The figure illustrates the position of the cleavage for each kind of fragment (exemplified using the case of the smallest fragment possible) and the mass calculation method is described for each fragment kind; consider that each fragment bears only *one positive* charge.

left end of the precursor polymer sequence begins at the left end, and for fragments that hold the right end at the right end. Thus the third fragment of series *a* *–a3–* would involve monomers [1→3]; and the third fragment of series *y* *–y3–* would involve monomers [6→4] (in the figure these left-to-right and right-to-left directions are symbolized using arrows). Therefore, it should appear to the reader how important –when specifying a fragmentation– it is to clearly indicate from which end of the precursor polymer the fragment is generated (in **GNU polyxmass** jargon this is "LE" for left end, "RE" for right end and "NE" for no end). **GNU polyxmass** knows what action it should take when it encounters one of these three specifications; for example, if a "LE" specification is found for a given fragmentation specification, **GNU polyxmass** adds to the fragment's mass the mass corresponding to the left cap of the precursor polymer.

Now that the stage is set we can start rationalizing fragment specifications, and thus mass calculations.

***a* fragment series** If we take the *a* fragment series, the Figure 3.8 indicates that the fragments include the left end and that their last monomer lacks its carbonyl group (see, on top of Figure 3.8, that the *a1* arrow goes between the CαH and the CO of monomer 1?). So we would say that each fragment of the *a* series should be challenged with the following chemical treatments: 1) addition of the mass corresponding to the left cap (proton), 2) removal of the mass corresponding to the lacking CO group. This way we have the mass of fragment *a1*. If we were interested in the fragment *a4* we would have summed the masses of monomers 1 to 4, added the mass of the left cap, and finally removed the mass of a CO; that's it. The mass calculation is thus mathematically expressed

$$a_i = LC + \sum_{1}^{i} M_i - CO$$

***b* fragment series** Similarly, the mass calculation is mathematically expressed

$$b_i = LC + \sum_{1}^{i} M_i$$

***c* fragment series** The mass calculation is mathematically expressed

$$c_i = LC + \sum_{1}^{i} M_i + NH_3$$

***x* fragment series** For this series of fragments we do not add the left cap anymore, but replace it with the right cap, since the fragments hold the right end of the precursor polymer. Note also that the numbering of the monomers using the variable i in the following mathematical expressions goes from right to left (contrary to what happened for the *a, b, c* fragment series. All the fragments that hold the precursor polymer right end are numbered this way, so this applies to fragments *x, y, z*. The mass calculation is mathematically expressed

$$x_i = RC + \sum_{1}^{i} M_i + CO$$

***y* fragment series** The calculation is mathematically expressed

$$y_i = RC + \sum_{1}^{i} M_i + H_2$$

***z* fragment series** In low energy CID, the z fragments are expressed this way:

$$z_i = RC + \sum_{1}^{i} M_i - NH$$

which is equivalent to $y\text{-}NH_3$; in high energy CID an additional proton is often measured:

$$z_i = RC + \sum_{1}^{i} M_i - NH + H$$

***immonium* fragment series** These fragments are internal fragments in the sense that they do not hold neither of the two precursor polymer's ends. **GNU polyxmass** understands that the user is speaking of this kind of fragment when the "from which end" piece of data –in the fragmentation specification– states "NE" instead of "LE" or "RE" (see page 62). The mass calculation for these fragments does not take into account the monomers surrounding the one for which the calculation is done. The mass for an immonium ion –at position i in the precursor polymer– will be the mass of the monomer at position i, less the mass of a CO, plus the mass of a proton. The mass calculation for these special internal fragments is expressed

$$imm_i = M_i + H - CO$$

Nucleic Acid Fragmentation

The fragmentations that can be obtained with nucleic acid are numerous and it is more complicated than with proteins to describe them fully. The main reason for this is that there are a big number of fragmentation combinations because of the loss of nitrogenous bases from the skeleton. The mechanisms by which this loss happens are fairly complex, and I am not going to detail any of them. Figure 3.9 shows the most common fragmentations (without taking into consideration the potential loss of bases). An example of fragment is given for each fragment series (pretty the same way as we did before for proteins). Note that the fragment representations are aimed at helping the reader to figure out what the product ion is, not taking into account where the negative charge lies on the fragment, since this charge can float around at every de-protonatable group. All the fragments shown bear one and one only negative charge.

The reader might have noticed –at the bottom of the figure– that a provision is made in the case the fragmented molecular species are not 5' end-phosphorylated but 5' end-hydroxylated. Indeed, the canonical monomer is such that, upon polymerization and left capping, the 5' end is phosphorylated. However, oft-times the oligonucleotides are synthesized chemically without the 5' end phosphate group, thus ending in hydroxyl. This special case should be accounted for by applying to all the fragments that bear the left end of the precursor polymer the following chemical reaction: $-HPO_3$. This chemical reaction should be applied *in addition* to the chemical reaction that yields the fragment *per se*.

Exactly as we did for the protein fragments, we are giving below the mathematical expressions used to calculate the mass of different series of nucleic acid fragments; in these

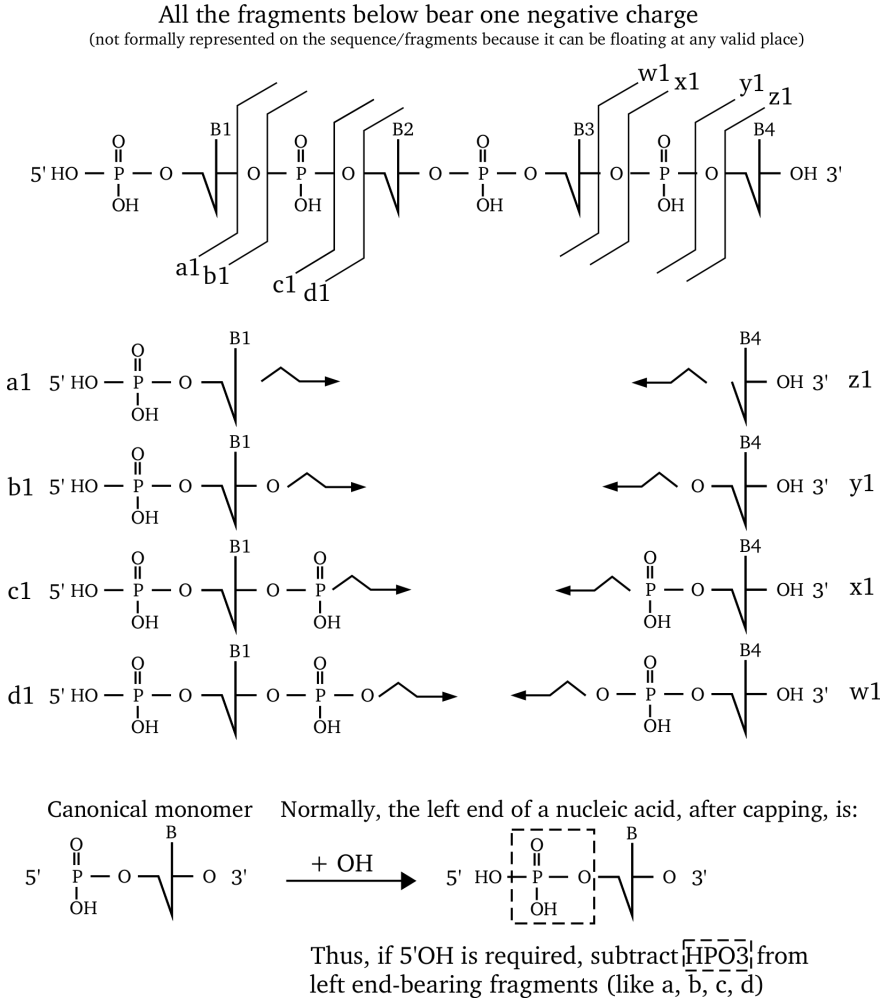

Figure 3.9: **DNA fragmentation patterns most widely encountered.** A short DNA sequence is fragmented in the eight most widely encountered manners, such as to generate a, b, c, d, w, x, y, z fragment ions. The figure illustrates the position of the cleavage for each kind of fragment (exemplified using the case of the smallest fragment possible). and the mass calculation method is described for each fragment kind; considering that each fragment is protonated only once (+1).

calculations we assume that the left end of the precursor polymer is phosphorylated (5' P) and the reader should bear in mind that this precise phosphate might itself be expelled by the fragmentation. The fragment naming scheme consideration that we emitted for protein fragments above (left-to-right or, conversely, right-to-left) applies here also in an identical manner.

***a* fragment series** These fragments most often appear with base loss.

$$a_i = LC + \sum_{1}^{i} M_i - O$$

***b* fragment series**

$$b_i = LC + \sum_{1}^{i} M_i$$

***c* fragment series**

$$c_i = LC + \sum_{1}^{i} M_i - HPO_2$$

***d* fragment series**

$$d_i = LC + \sum_{1}^{i} M_i - HPO_3$$

***w* fragment series**

$$w_i = RC + \sum_{1}^{i} M_i + O$$

***x* fragment series**

$$x_i = RC + \sum_{1}^{i} M_i$$

***y* fragment series**

$$y_i = RC + \sum_{1}^{i} M_i - HPO_2$$

***z* fragment series**

$$z_i = RC + \sum_{1}^{i} M_i - HPO_3$$

There are also a variety of fragments for which a base is lost. But we cannot describe them all!

More Complex Patterns Of Fragmentation

Before finishing with fragmentations, it is necessary to describe a powerful feature of the fragmentation specification grammar available in **GNU polyxmass**. This feature was required for the fragmentation of oligosaccharides and also sometimes for proteins. When the fragmentation (the bond breakage reaction itself) occurs at the level of certain monomers, it might be necessary to be able to specify some particular chemistry that would arise on the monomer in question.

We have seen in the cleavage documentation that, upon cleavage of a protein sequence with cyanogen bromide, for example, a particular chemical reaction had to be applied to the oligomers that were generated with a methionine monomer as their right end monomer. Well, in a fragmentation specification it is possible to apply comparable chemical reactions but in a more thorough manner. Indeed, while in the cleavage it was possible to say something like "*apply a given chemical reaction to the oligomer if the right end monomer is Xyz*", in the fragmentation the logical condition can be bound not only to the identity of the currently fragmented monomer, but also (optionally) to the identity of the previous and/or next monomer in the precursor polymer sequence. For example: —"*Apply a given chemical reaction if fragmentation occurs at the level of "Xyz" monomer only if it is preceded by a "Yxz" monomer and followed by a "Zyx" monomer*".

These logical conditions are called *fragrules*. A *fragspecif* can hold as many *fragrules* as necessary. Thus we see that a fragmentation specification is a multi-part specification, with a *fragspecif* optionally integrating *fragrule* objects... All of this is described in great detail at page 62.

To Sum Up

To sum up all what we have seen so far with polymer chain disrupting chemistries:

* A polymer sequence gets cleaved into oligomers when a chemical reaction occurs in it at the level of one or more inter-monomer bond(s); monomer-specific chemical reactions can be modelled into the cleavage specification using at most one leftrighrule;
* A polymer sequence gets fragmented into fragments when a bond breakage occurs, without the help of any exterior molecule, at any level of the polymer structure, with no limitation to the inter-monomer bond; monomer-specific chemical reactions can be modelled into the fragmentation specification using any number of fragrules;
* Oligomers are automatically capped –*on both ends*– using the rules described in the precursor polymer's definition;
* Fragments are capped automatically only –*on the end they hold, if any*– using the rules described in the precursor polymer's definition;
* Oligomers are automatically ionized (if required by the user) using the rules described in the precursor polymer's definition;
* Fragments are never ionized automatically; ionization (gain/loss of a charged group) is necessarily integrated in the fragmentation specification.

4

Basics in Mass Spectrometry

Mass spectrometry has become a "buzz word" in the field of structural biology. While it has been used for long to measure the molecular mass of little molecules, its recent developments have brought it to the center of the analytical arsenal in the field of structural biology (also of "general" polymer science). It is now current procedure to use mass spectrometry to measure the mass of polypeptides, oligonucleotides (even complete transfer RNAs!) and saccharides, amongst other complex biomolecules.

A mass spectrometer is usually described by giving to its three main different "regions" a name suggestive of their function:

- the source, where production of ionized analytes takes place,
- the analyzer, where the ions are electrically/magnetically "tortured",
- the detector, where the ions arrive, are detected and counted.

Before letting Mass Spectrometry in, I would like to state once for all: *mass spectrometry is aware of ionized molecular species only...*

Now, *enter* Mass Spectrometry

Ion Production: The Source

Indeed, mass spectrometry cannot do anything as long as the molecule to analyze (*analyte*) is not in a charged state. The process of creating an ion from an un-charged analyte is called *ionization*. Well, most of the times the ionization is favored by adapting the sample's pH to a value higher/lower than the isoelectric pH of the analyte, which will elicit the appearance of (a) charge(s) onto it. In cases where the analyte cannot be charged by simple pH variations (small molecule that does not bear any ionizable chemical group), the ionization step might require –on the massist's part– use of starker ionization techniques, like electronic impact ionization or chemical ionization. In biopolymer mass spectrometry, the pH strategy is usually considered the right way to proceed. The ionization process might involve complex charge transfer mechanisms (not fully understood yet, at least for certain ionization/desorption methods) which tend to ionize the analyte in a way not predictable by looking at the analyte's chemical structure.

Ion production should not be uncoupled from one important feature of mass spectrometry: solvent evaporation –in case of liquid sample delivery to the mass spectrometer– and sample *desorption* –in case of solid state sample introduction. The general idea is that mass spectrometry works on gas phase ions. This is because it is of crucial importance, for a correct mass measurement to take place, that the analyte be *totally* freed of its chemical immediate environment. That is, it should be "naked" in the gas phase. Equally important is the fact that ions must be capable of travelling long distances without ever encountering any other molecule in their way. This is achieved by pumping very hard in the two regions called "analyzer" and "detector". In this respect, the source is a special region because, depending on the design of the mass spectrometer, it might be partially at the atmospheric pressure during mass spectrometer operation. It is not the aim of this manual to provide insights into mass spectrometer design topics (I just would not be able to enter into the physics details!), but the general principle is that mass spectrometry involves working on gas phase ions. This is why a mass spectrometer is usually built on extremely reliable pumping technology aimed at maintaining for long periods of time (with no sudden interruption, otherwise the detector might suffer seriously) a good vacuum in the conduit in which ions must flow during operation.

The Analyzer

Once an ion has been generated in the gas phase, its mass should be measured. This is a complex physical process. Depending on the mass spectrometer design, the mass measurement is based on more or less complex physical events. Magnetic mass spectrometers are usually thought of as pretty complex devices; this is also the case for the Fourier transform ion cyclotronic resonance devices. An analyzer like the *time of flight* analyzer is much more simple. I will refrain from trying to explain the physics of the mass measurement, just limit myself saying that –at some stage of the mass measurement process– forces are exerted on the ions by electric/magnetic fields (incidentally, this explains why it is so important that an analyte be ionized, otherwise it would not be subject to these fields). The ionized analytes submitted to these forces have their trajectory modified in such a way that the detector should be able to quantify this modification. Roughly, this is the measurement process.

What Is Really Measured?

Prior to entering into some detail, it seems necessary to make a few definitions[1]:

- unified mass scale (u): IUPAC & IUPAP (1959-1960) agreed upon scale with 1 u equal to 1/12 the mass of the most abundant form of carbon; the dalton is taken as identical to u (but not accepted as standard nomenclature by IUPAC or IUPAP), it is abbreviaed in Da.
- a former unit was "a.m.u." (*i.e.* "atomic mass unit"). It should be considered obsolete, since based on an old 1/16 of ^{16}O standard;
- the mass of a molecule (also "molecular mass") is expressed in daltons. The symbol commonly used is "M" (not "m"), as in "M+H" or "M+Na"... Symbol "m" is already employed for ion mass (as in "m/z");
- the mass-to-charge ratio ("m/z") of an ion is the ion's mass (in daltons) divided by the number (z) of elementary charges. Hence "m/z" is "mass per charge" and units of "m/z" are "daltons per charge";
- nominal mass: the integral sum of the nucleons in an atom (it is also the atomic mass number);
- exact (also known as accurate) mass: the sum of the masses of the protons and neutrons plus the nuclear binding energy;

In the previous sections I used to say that a mass spectrometer's task is to measure masses. Well, this is not 100 % exact. A mass spectrometer actually allows to measure something else: it measures the *m to z ratio* of the analyte, which is denoted m/z. What is this "*m to z ratio*" all about? Well, we said above that a mass spectrometer has to exert forces on the ions in order to determine their m/z. Now, let us say that we have an electric field of constant value, E. We also have two ions of identical masses, one bearing one charge (q) and the other one bearing two charges ($2q$) –positive or negative, no matter in this discussion. These two ions, when put in the same electric field E, will "feel" two different forces exerted on them: F_1 and F_2. It is possible to calculate these forces ($F_1 = qE$ and $F_2 = 2qE$). Evidently, the ion that bears two charges is submitted to a force that is twice as intense as the one exerted on the singly charged ion.

What does this mean? It means simply that the numeric result provided by the mass spectrometer is not going to be the same for both ions, since the physics of the mass spectrometer takes into account the charge level on each different analyte. Our two ions weigh exactly the same, but the mass spectrometer simply can not know that; all it knows is how a given ion reacts to the electric field it is put in. And our two ions, evidently, will react differently.

When we say that a mass spectrometer measures a m/z ratio, the z of this ratio represents the sum of all the charges (this is a net charge!) that sit onto the analyte. But what does the m stand for? The molecular mass? No! The m stands for the mass of the whole analyte ion, which is –in a word– the *measured mass*. This is not the molecular mass (which would be M), it is the molecular mass *plus/less* the mass of the chemical entity that brings the charge to the analyte. When ionizing a molecule, what happens is that something brings (or removes) a charge. In biopolymer chemistry, for example, often the ionization is a simple

[1] Interesting posting signed by Ken I. Mitchelhill in the **ABRF** mailing list at http://www.abrf.org/archives, and a document published by the California Institute of Technology.

protonation/deprotonation. If it is a protonation, that means that an electronic doublet (on some basic group of the analyte) captures a proton. This brings the mass of a proton to the biopolymer ($\simeq$ 1 Da). Conversely, if it is a deprotonation (loss of a proton by some acidic group, say a carboxylic that becomes a carboxylate) the polymer looses the mass of a proton. Of course, if the ionization involves a single electron transfer the mass difference is going to be so feeble as to be un-measurable on a variety of mass spectrometers.

Let us try to formalize this in a less verbose manner by using a sweet amino acid as an example:

- the un-ionized analyte (Glycine) has the following formula: $C_2H_5O_2N_1$;
  the molecular mass is thus $M = 75.033$ Da;
- the analyte gets protonated in the mass spectrometer:

  $$C_2H_5O_2N_1 + H \rightharpoonup C_2H_6O_2N_1$$

  the measured mass of the ion is thus $m = 75.033 + 1.00782$ Da and the charge beared by the ion is thus $z = +1$.
- the peak value read on the mass spectrum for this analyte will thus be:

  $$\text{value} = \frac{m}{z} = \frac{M + 1.00782}{z} = 76.04$$

  with $z = +1$

We see here that the label on the mass spectrum does not correspond to the nominal molecular mass of the analyte: the ionizing proton is "weighed" with the Glycine molecule.

Imagine now that, by some magic, this same Glycine molecule just gets protonated a second time. Let's do exactly the same type of calculation as above, and try to predict what value will be printed onto the mass spectrum:

- the un-ionized analyte (Glycine) has the following formula: $C_2H_5O_2N_1$;
  the molecular mass is thus $M = 75.033$ Da;
- the analyte gets protonated in the mass spectrometer *two times*:

  $$C_2H_5O_2N_1 + 2H \rightharpoonup C_2H_7O_2N_1$$

  the molecular mass of the ion is thus $M = 75.033 + 2.01564$ Da and the charge beared by the ion is thus $z = +2$.
- the peak value read on the mass spectrum for this analyte will thus be:

  $$\text{value} = \frac{m}{z} = \frac{M + 2.01564}{z} = 38.52$$

  with $z = +2$

Oh! yes!, this time it is absolutely clear that a m/z is not a molecular mass! By the way, if the Glycine happened to be ionized *negatively* the calculation would have been analogous to the one above, but instead of *adding* the mass of the proton(s) we would have *removed* it. It is that simple.

Summing up all this in a few words: an ionization involves one or more charge transfer(s) and in most cases (at least in biopolymer mass spectrometry) also involves matter transfer(s). It is crucial *not* to forget the matter transfer(s) when ionizing an analyte. This means that when an ionization process is described, its description ought to be complete, clearly stating three different pieces of information:

* the charge transfer (net charge that is beared by the analyte after the ionization has completed);
* the matter transfer (optional; usually something like "+H1");
* the ionization level (0 means "no ionization"; usually this would be 1 for a single ionization, but might be as large as 30 if, for example, you were ionizing myoglobin with electrospray ionization (protonation). In this case the m/z value would be computed this way:

$$\text{value} = \frac{m}{z} = \frac{M + 30 \cdot \ 1.00782}{30} = \frac{16959 + 30.2346}{30} = 566.30$$

with $z = +30$

By now, the reader should have grasped the importance of understanding well the ionization formalisms for accurately predicting/analyzing mass spectrometric data!

In the next chapters of this manual we will describe how **GNU polyxmass** works and how the user might take advantage of its powerful capabilities. In a first chapter I will introduce some general concepts around the way the program behaves. Next, in the remaining part of this manual, a chapter will be dedicated to each important **GNU polyxmass** function or characteristic.

5

GNU polyxmass Generalities

In this chapter, I wish to introduce some general concepts around the **GNU polyxmass** program.

General **GNU polyxmass** Concepts

The **GNU polyxmass** mass spectrometry software suite has been designed to be able to "work" with every polymer on earth. Well, in a certain way this is true... A more faithful account of the **GNU polyxmass**' capabilities would be: "*The **GNU polyxmass** software suite works with whatever polymer chemistry the user cares to define; the more accurate the polymer chemistry definition, the more **GNU polyxmass** will be accurate*". Sounds like much of the responsibility for the proper functioning of the **GNU polyxmass** framework is in the hands of the user? That is true! However, with **GNU polyxmass** the user has a framework at hand to define polymer chemistries so as to suit his needs.

The main concept that drove the design of the entire **GNU polyxmass** framework is *abstraction*. Indeed, for the program to be able to understand a variety of possibly very different polymers, it had to be written using some *abstraction layer* between the way masses are computed and the way the polymer is described "in memory". This abstraction layer

is implemented by using a "polymer chemistry definition-driven" set of functionalities. The polymer chemistry definition drives all the mass computations, all the polymer sequence editing, all the polymer chemistry reactions... This is how the **GNU polyxmass** software suite makes it possible to handle any polymer type. To implement this abstraction paradigm, the **GNU polyxmass** mass spectrometry framework was designed to be modular, as described below.

The **GNU polyxmass** mass spectrometry software suite comprises the following packages (not all of them installing actual binary/executable programs):

1. **GNU polyxmass-bin** this is the binary package enshrining the `polyxmass` binary program. This is where the user will spend most of his time: doing either polymer chemistry definitions (**polyxdef** menu), mass calculations (**polyxcalc** menu) or real polymer sequence chemical simulations along with mass spectrometry simulations (**polyxedit** menu);

2. **GNU libpolyxmass** this is the library where all the chemical intelligence of the **GNU polyxmass** software framework lies. This library is not graphical and may be interesting to the chemist willing to understand what a *monomer* or an *oligomer* is, from a programmatic standpoint. A number of helper functions having nothing to do with polymer chemistry are implemented in this library also;

3. **GNU polyxmass-common** this is a non-binary package where the essential configuration/data files are stored, like the scripts that are used to update the catalogues of available polymer chemistry or atom definitions. This package comes with the basic atom definition file and an example polymer chemistry definition ("protein");

4. **GNU polyxmass-data** this is an *optional* non-binary package where other example polymer chemistry definitions are delivered, so that the user might learn how to prepare other packages to submit to the **GNU polyxmass** development team for incorporation in the **GNU polyxmass** software suite as official packages;

In the rest of this manual we shall call "module" a set of functionalities that are aimed at a specific task: for example, all the functionalities that are accessible in the `polyxmass` binary program with the aim of defining polymer chemistries will be called the "**polyxdef** module" and will be triggered by using the menu tree rooted at the "**polyxdef**" menu item.

The fact that the **GNU polyxmass** software suite is able to handle any polymer chemistry is, as we said above, due to its ability to interface a polymer sequence with a polymer chemistry definition. To explain this clearly, imagine a protein sequence that would be this tetrapeptide: "ATGC", which reads as "AlanineThreonineGlycineCysteine". Now imagine the same "ATGC" sequence but as a DNA sequence, which reads as "AdenineThymineGuanineCytosine". The two sequences would be entered in a sequence editor by keying in the following key sequence: A T G C. Of course, while the sequence is identical in both cases, you'd expect that the masses for the DNA sequence be much higher than the masses for the protein sequence.

This is where "abstraction" comes in, and modularity also: in order to let the user perform the required computations as flexibly as possible, she first defines two different polymer chemistries: the first named "protein" and the second named "dna". In each of the two distinct polymer chemistry definitions, the user will enter a formula corresponding to each monomer (A,T,G,C). Of course, the monomer formula for a Threonine is very different than the one for a Thymine.

The definition of the polymer chemistry is performed in the **polyxdef** module that is accessible in the polyxmass program under the "**polyxdef**" menu item. Once a polymer

chemistry definition is saved, it may be made available to the system (we'll see how this is done). And when a polymer chemistry definition is made available to the system, any new polymer sequence may be created that abides by this polymer chemistry definition.

By defining precisely the chemical behaviour of a polymer type, and making an association between a given polymer chemistry definition and a polymer sequence, the user makes use of the *abstraction layer* that we mentioned above. Once this is well understood, the originality of the **GNU polyxmass** software framework is understood. This is precisely what sets **GNU polyxmass** apart from the other mass spectrometry-related software offerings.

Since the different functionalities offered by the **GNU polyxmass** framework are well confined in three distinct modules, all accessible from the `polyxmass` binary program, but sitting in clearly distinct menu trees, we'll review each of such "modules" in later chapters.

Before going on with the description of the different modules, I would like to introduce some other more chemistry-oriented concepts that are going to be used throughout the **GNU polyxmass** framework.

On Formulae And Chemical Reactions

It is all the more frequent for any user who runs any of the **GNU polyxmass**' modules to make use of formulae or of chemical reactions. These two chemical entities are not identical in **GNU polyxmass**. While a formula represents a chemical status (a monomer has a given formula, and does not change it), a chemical reaction is something much more dynamic, I should say "active".

This difference is very important in **GNU polyxmass**. Let's take an example: the Lysyl monomer (we call a protein "residue" a "monomer") has the following formula: $C_6H_{12}N_2O$. If I wish to acetylate this Lysyl monomer, the reaction will read this way: "An acetic acid molecule will condense onto the amine of the Lysyl side chain". This can also read: —*"An acetyl group enters the Lysyl side chain while a hydrogen atom leaves the Lysyl side chain; water is lost in the process"*. If we wanted to put this into a more chemistry-oriented representation, we could write this:

$$R - NH_2 + CH_3COOH \rightleftharpoons R - NH - CO - CH_3 + H_2O$$

That is more briefly stated this other way: "$-H_2O + CH_3COOH$". This is exactly what **GNU polyxmass** calls an *"actionformula"* —or, for brevity— an *"actform"*: just because actions are associated with formulae; here the H_2O formula is associated with the $-$, which indicates that the water molecule leaves the molecules being reacted, while the CH_3COOH formula is associated with the $+$, which means that the acetic acid molecule enters in to the target molecule. The net formula is thus, as stated earlier: —*"An acetyl group enters the Lysyl side chain while a hydrogen atom leaves the Lysyl side chain; water is lost in the process"*.

In the **GNU polyxmass** framework, the *formula* and *actform* chemical entities are *not* interchangeable.

The **GNU polyxmass** Framework Data Format

All the data in the **GNU polyxmass** framework are stored on disk as *XML*-formatted files. *XML* is the *eXtensible Markup Language*. This "language" allows to describe the structure of a document. Have you ever opened an *HTML* file with a text editor? If so, you have certainly seen some markup like `<H1>This is the title</H1>`. The browser that loads this file will understand (because it has been programmed to do so) that the title "This is the title" is to be displayed onto the screen using a bold sans-serif font, for example. Well, let us just say that the *XML* file format is an immensely more powerful equivalent of *HTML*.

There would be a lot... a lot to say about *XML* and *Document Type Definition*s: I'll refrain from entering into the details.

The big advantage of using such *XML* format in **GNU polyxmass** is that it is a text format, and not a binary one. This means that any data in the **GNU polyxmass** package is human-readable (even if the *XML* syntax makes it a bit difficult to read data, it is actually possible). Try to read one polymer chemistry definition `.xml` file from the **GNU polyxmass-data** package (say, the `dna-sample.xml` file, for example), and you'll see that this is pure text (the same applies for the `.pxm` polymer sequence files in the same package. The advantages of using text file formats, with respect to binary file formats are:

- if somebody sends you a file and you do not have the program that made it, you still can extract information from the file, because it is readable with any text editor;
- if a text file (such as your most important polymer sequence *XML* file) gets corrupted for some reason (*i.e.* during backup on a bad support, or whatever) you will still be able to extract from the corrupted file all the bits of information that surround the portion that is corrupted, thus minimizing the data loss. This would be impossible with binary files, as they are just totally useless if a single part of them is corrupted;
- imagine you would like to write down a simple script that would allow you to find —in a given directory— all the sequence files that contain the "myo" character string in the polymer's name field (in *XML* a field is called *element*). You can do it easily *without* asking anybody for the file format specification —because your sequence files are just text files.

As an example of how simple it is I'll just write a `bash` shell script below that I'll save into the `polname-find.sh` file in order to execute it afterwards. That is how the shell script looks like in the `polname-find.sh` file:

```
shell-prompt $ cat polname-find.sh ↩


 for i in *.pxm
   do grep "<name>.*myo.*</name>" $i ;
   if [ $? == 0 ]
     then
       echo "in file $i"
   fi
   done
```

Now we should make this brand new file executable so we can run it:
`shell-prompt $ chmod u+x polname-find.sh` ↩
Upon execution of this script, the output looks like this:
`shell-prompt $ ./polname-find.sh` ↩

```
<name>myoglobin-horse</name>
in file myoglob-h.pxm
<name>myosin-chicken</name>
in file myos-chck.pxm
<name>myo-fragment1</name>
in file myofrag1.pxm
<name>apomyoglobin-rabbit</name>
in file apomyo-rbt.pxm
```

The script has gone through all the `*.pxm` files and for each file has searched a start tag `<name>` followed by some string containing "myo" followed by the end tag `</name>`. If "myo" is found, the corresponding line is printed to the screen, and the name of the file containing this pattern is printed also.

With a binary file format this would have been impossible. This little script lets you screen a big database like a snap. That's the power of *UNIX* and *UNIX*-like operating systems.

Editing the Data in **GNU polyxmass** Files

The aim of **GNU polyxmass** is to let people use the software the way they like, with no preconception on the way they interact with it. The *XML* files (polymer sequence or polymer chemistry definition files) can be edited using the graphical interface but also using a simple text editor. Figure 5.1 shows two rather different means to the same end: editing a polymer chemistry definition file. The Document Type Definition (DTD) is not shown on the right pane of the figure, since it is at the top of the file being displayed. This DTD will help the user to determine how to edit the file in a safe way, by telling where each element is authorized to be, and so on... You'll need to learn *XML* if you wish to understand the DTD (a sunday afternoon will suffice). Usually, the safer way to do any editing is by using the graphical interface, not because the **GNU polyxmass** framework understands the edited data better this way, but because the graphical interface layout (acting like a data correctness censor) just prevents the user from writing badly-formed data directly in the *XML* file.

The example shown in Figure 5.1 can be transposed to the polymer sequence *XML* files in a very same way. Of course all the process that leads to "creating" a new polymer chemistry definition is going to be explained in detail in a later chapter (see chapter 6, page 49).

General Polymer Element Naming Policy

Unless otherwise specified, it is *strongly* suggested *not* to insert any non-alphanumeric-non-ASCII character (space, %, #, $...) in the strings that the user enters to identify polymer chemistry definition items. This means that, for example, the user must refrain from using non-alphanumeric-non-ASCII characters for the atom name and symbol, the name, the code or the formula of the monomers or of the modifications, or of the cleavage specifications, or

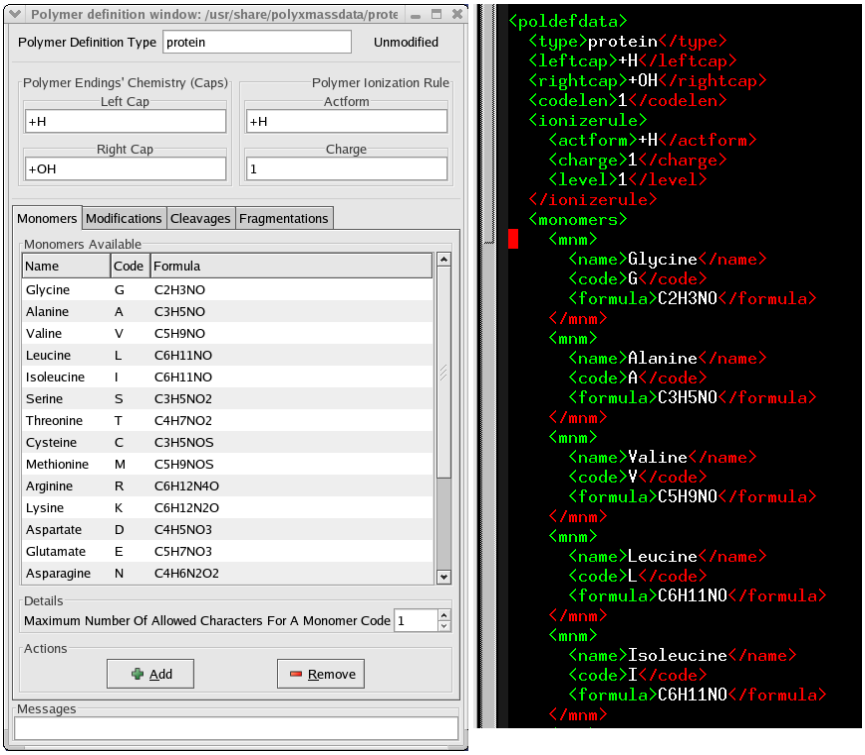

Figure 5.1: **Comparison of a graphical and a text way of editing a polymer chemistry definition file.** The left pane shows the graphical interface that is exposed to the user when defining a polymer. The right pane shows the same *XML* file opened in the `Emacs` editor with the *XML* editing mode switched on.

of the fragmentation specifications... Usually, the accepted delimiting characters are '' and '_'. It is important not to cripple these polymer data for two main reasons:

* so that the program performs smoothly (some parsing processes rely on specific characters (like '#' or '%', for example) to isolate sub-strings starting from larger strings);
* so that the results can be easily and clearly displayed when time comes to print all the data.

Graphical Interface Design

For those coming to *UNIX* after having used *MS Windows* (like me), I would like to state some general graphical interface design specificities of the *UNIX* world. The *MS Windows* graphical environment was designed in such a way that the user is very strictly restricted to a narrow path each time she initiates an action. That policy has often led to arbitrary limitations in the design of software running on the *MS Windows* systems.

This is not going to be exactly the same with a *UNIX* graphical environment: you almost certainly are going to quickly have a great number of windows opened on your desktop; you are the one who knows when to close a results window, not the program designer. When a window is opened, it is not going to be systematically required that it be closed before opening another one. This has a simple reason: imagine that you wanted to compare the oligomers generated by using two different enzymes on the same polymer sequence; you'll need both results windows to be opened at the same time, otherwise how comparison of oligomers could happen? That reasoning is true for a number of situations, and —yes— you'll be responsible for closing the windows you do not need anymore!

This general behaviour is highly desirable, since it indeed allows the user to make comparisons between the data from two different experiments right after having generated the data. But this behaviour introduces a risk: how will it be possible to ascertain that any given set of peptides does come from the cleavage of the first protein using cleaving-agent-1 and not from the cleavage of the first protein using cleaving-agent-2? In other words: how are you going to recognize which results window contains the peptides of the first cleavage, and which results window contains the peptides obtained from the second cleavage? There is an answer: each time a window is displayed —if there is a risk of ambiguity— it will show the identity number (ID number) of the polymer to which it is related. This ID number is nothing else but the *unique* memory address of the polymer sequence editing context to which the window is related.

In any situation where an ambiguity exists about the identity of the data generated on any given polymer sequence, a traceability system is used, as shown in Figure 5.2.

Feedback From **GNU polyxmass** To The User: The Console Window

Something very specific to the *UNIX* and *UNIX*-like systems (and that I really like) is the fact that the programs are usually designed to be "verbose" (if the user asks this). The usual means to giving feedback in other systems is to pop up a "dialog" window in which a message is displayed, and the user has to acknowledge (typically by clicking onto a button widget) in order to close the dialog window. **GNU polyxmass** has been implemented with

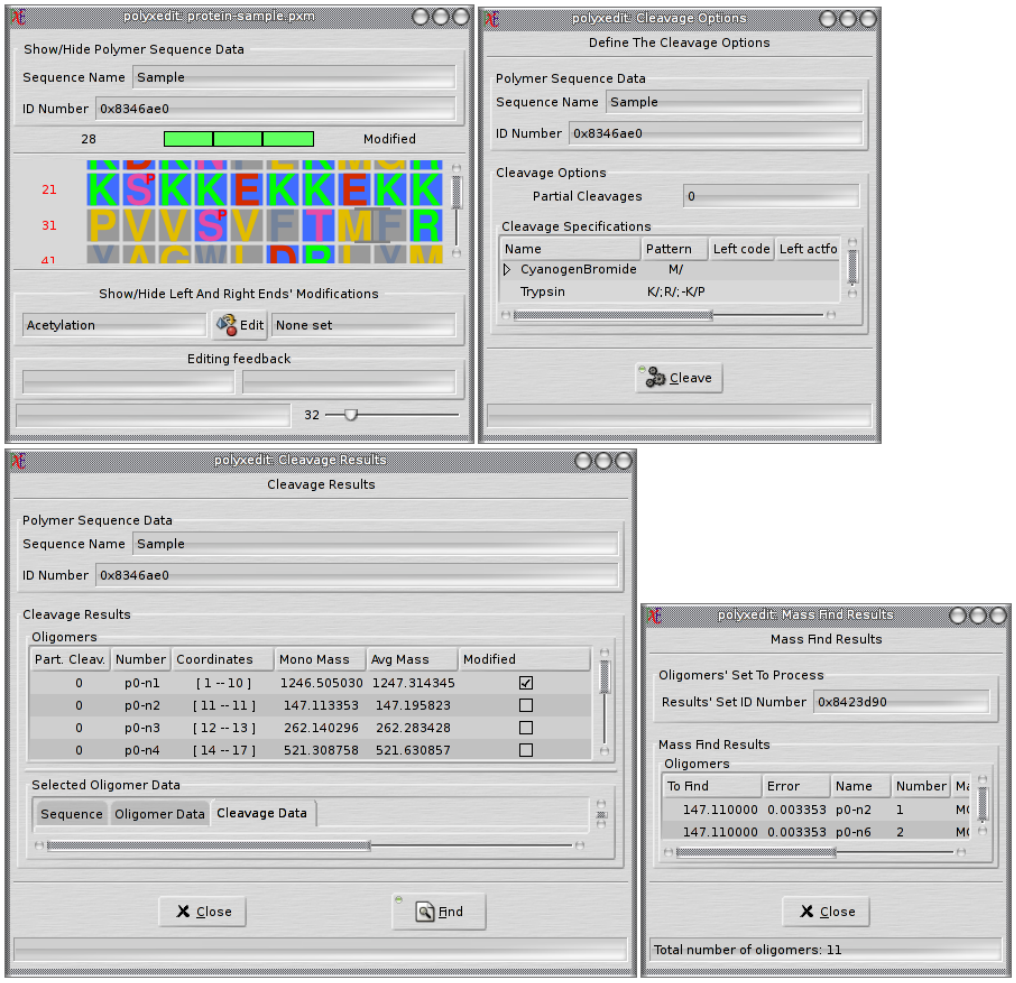

Figure 5.2: **Unambiguous identification of polymer sequences and related data.** When a polymer sequence is loaded/created, it is assigned a numeric value that unambiguously identifies it (for the programmer, this is the pointer to the polymer structure). Each time a window is displayed that contains data pertaining to any given polymer sequence (oligomers generated by cleavage of a given polymer sequence, for example), it is given a reference to the polymer whence the data came, and this reference is the polymer's identity number. This is clearly visible here: the polymer sequence has a given ID Number and all the related windows display that same number. Note that the cleavage results data have another ID Number that is later used to trace the mass find results data (last bottom window).

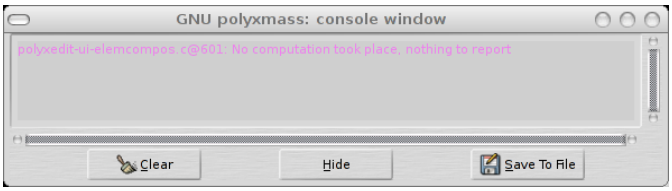

Figure 5.3: **The console window where any messages to the user are displayed.** Depending on the importance level of the message being issued to the user, the color will be more or less "reddish".

the "console" philosophy in mind: every message that it wishes to "hand out" to the user is sent to the terminal window from which the program was started.

There are two levels of very important messages: the *CRITICAL* and the *ERROR* level messages. The CRITICAL-level messages indicate that time has come to make a quick save of all the data, because something bad might happen. ERROR-level messages cannot even be read in the console window, because they elicit an abortion of the program. These abortions are voluntary on the **GNU polyxmass**' part, because the error is so bad that it would crash anyway soon or later.

Each time a message (of any importance level) is issued to the user, the console window is presented to the user (if it was hidden or minimized, this console window is show afresh). Figure 5.3 shows the console window with a warning.

Window Management

GNU polyxmass is powerful and flexible: any number of polymer sequences can be opened at any given time, and any number of simulations might be performed on any of these polymer sequences. This might lead to a huge number of windows opened on the desktop at any given time. There are two main types of windows:

* windows that do not display results. These window are typically windows where the user is provided with options to perform some action. For example, one such window might be the window that allows the user to select an enzyme when an enzymatic cleavage is required on a polymer sequence;

* windows that are responsible for displaying a polymer sequence, the results of some simulation or of any computation. For example, a window displaying results (*results window*, for short) might be the window that displays all the oligomers obtained upon cleavage of a polymer sequence using one enzymatic agent; or the window where all the fragmentation oligomers are displayed after the gas-phase fragmentation of a polymer sequence. Other examples are the windows where the monomeric composition of the polymer sequence is displayed and where the pH/pKa/pI computation results are displayed...

In order to ease the management of all the results windows opened at any given time, a window management facility was devised. Its incarnation is shown in Figure 5.4 on the following page. This window is called by using the main program's window menu

View $\longrightarrow$ *Window List*

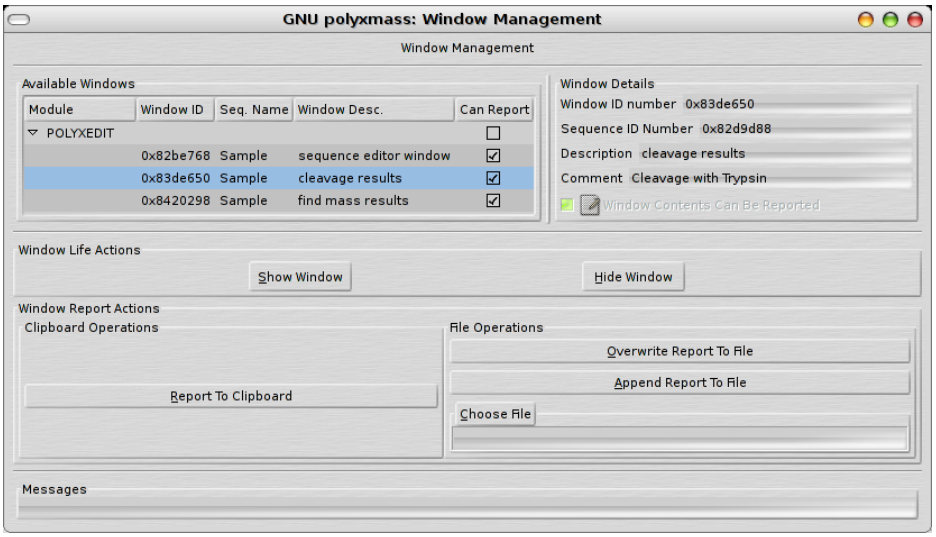

Figure 5.4: **The window management facility.** Each time a polymer sequence window —or a window where results are displayed— is opened, it is registered and appears in the treeview on the left of the depicted window. The user can then select any window of interest and perform actions about this window.

The window management operations include the following actions, that apply to the window item currently selected in the Available Windows treeview on the left of the window:

* Show Window: force a hidden/minimized window to show itself;
* Hide Window: force a window to hide itself.

As soon as a window that is listed in the Available Windows treeview is closed, its corresponding item in the treeview is removed.

6

polyxdef: Definition Of Polymer Chemistries

After having completed this chapter you will be able to accomplish the very first steps needed to use the **GNU polyxmass** framework's features at best. In order to use the program, indeed, it is required that the polymer chemistry on which you would like to experiment be defined according to a number of rules that will be detailed in the remaining sections of this chapter.

The **polyxdef** module is easily called by pulling down the "**polyxdef**" menu item from the **GNU polyxmass** program's menu. The user may accomplish two different tasks in the **polyxdef** module:

* Edit an atom definition;

* Edit a polymer chemistry definition.

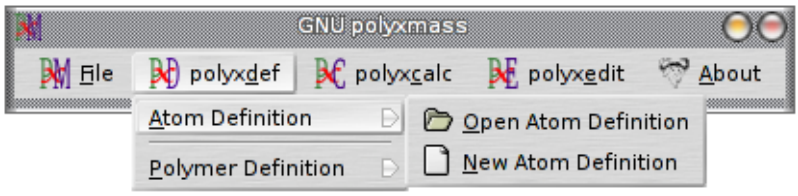

Figure 6.1: **polyxdef atom definition menu** The user might ask that an atom definition file be opened for editing or that a new atom definition be started empty for *ex nihilo* editing.

Editing an atom definition

The editing of an atom definition is performed through the user interface that shows up when the user selects one of the two submenu items shown in Figure 6.1.

When the user asks that an existing atom definition file be found, a "chooser window" shows up like the one shown on Figure 6.2 on the facing page.

When the atom definition editor shows up, the user sees an interface that allows the addition/removal of isotopes or atoms. This interface (Figure 6.3 on page 52) makes it trivial to edit to the highest level of refinement the definitions of the atoms to be used in the **GNU polyxmass** software suite.

Using the atom definition window is absolutely easy. The main idea is that an atom does not exist as something valuable for doing chemistry until it does not have at least one isotope defined as part of it. This means that to define a new atom, the Add Atom button should be clicked, which triggers the creation of a new empty item in the treeview shown in Figure 6.3 on page 52. At this point, the user must first name the new atom and give it a symbol (to edit a cell, just click onto it, make the required editing and validate by pressing ↩). Next, the user adds an isotope to that atom item. Clicking onto the Add Isotope button will trigger the creation of an empty isotope. The user fills the Mono Mass monoisotopic mass field of the newly created empty item. The same has to be done for the Abundance isotopic abundance field.

Each time a new monoisotopic mass/isotopic abundance pair is either edited, added or removed from an atom item, the average mass of that atom is recomputed and shown in the Avg Mass atom average mass cell.

The user may —at any moment— ask that the syntactic validity of the atoms in the definition be checked. For that, clicking onto the Check Syntax button is enough. If something goes wrong, a window shows up to describe the error(s) that were encountered. In our example of Figure 6.4 on page 53, we see that the syntax-checking function has detected that atom "Carbon" has no isotopic data whatsoever; and that is a real error, as we were mentioning earlier.

Once the atom definition is completed, the user has to register it to the **GNU polyxmass** software suite. This task is described in a later chapter about the configuration/data files hierarchy of the **GNU polyxmass** software.

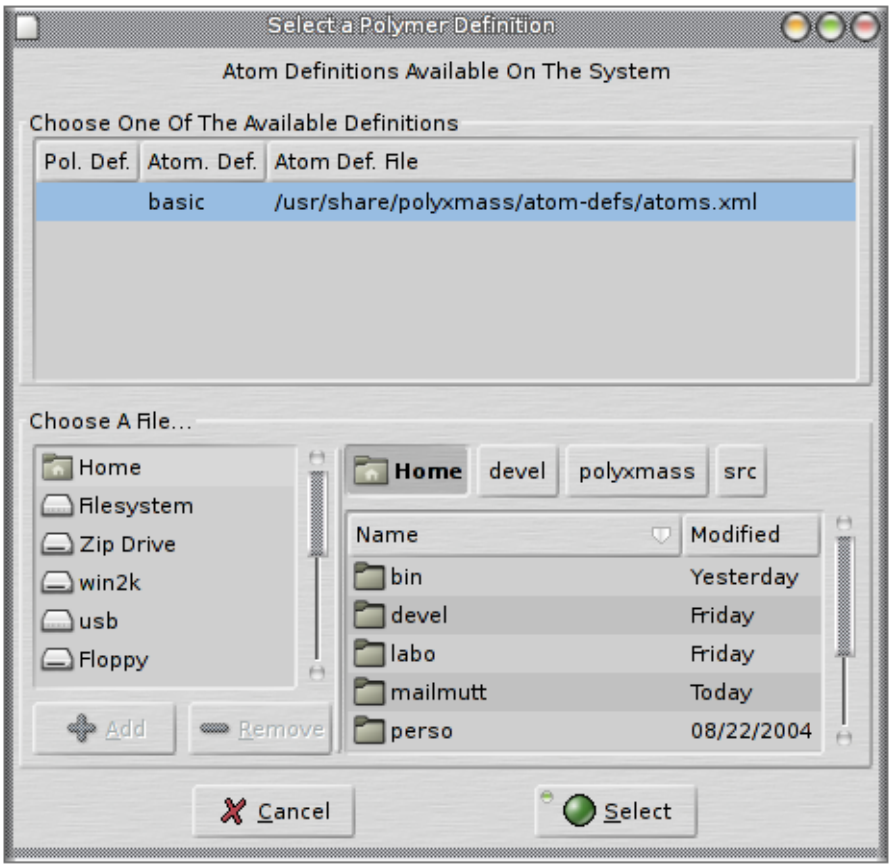

Figure 6.2: **polyxdef atom definition choosing window** The user might either select an atom definition already registered to the **GNU polyxmass** software suite (upper frame) or select an atom definition that is not registered (lower frame).

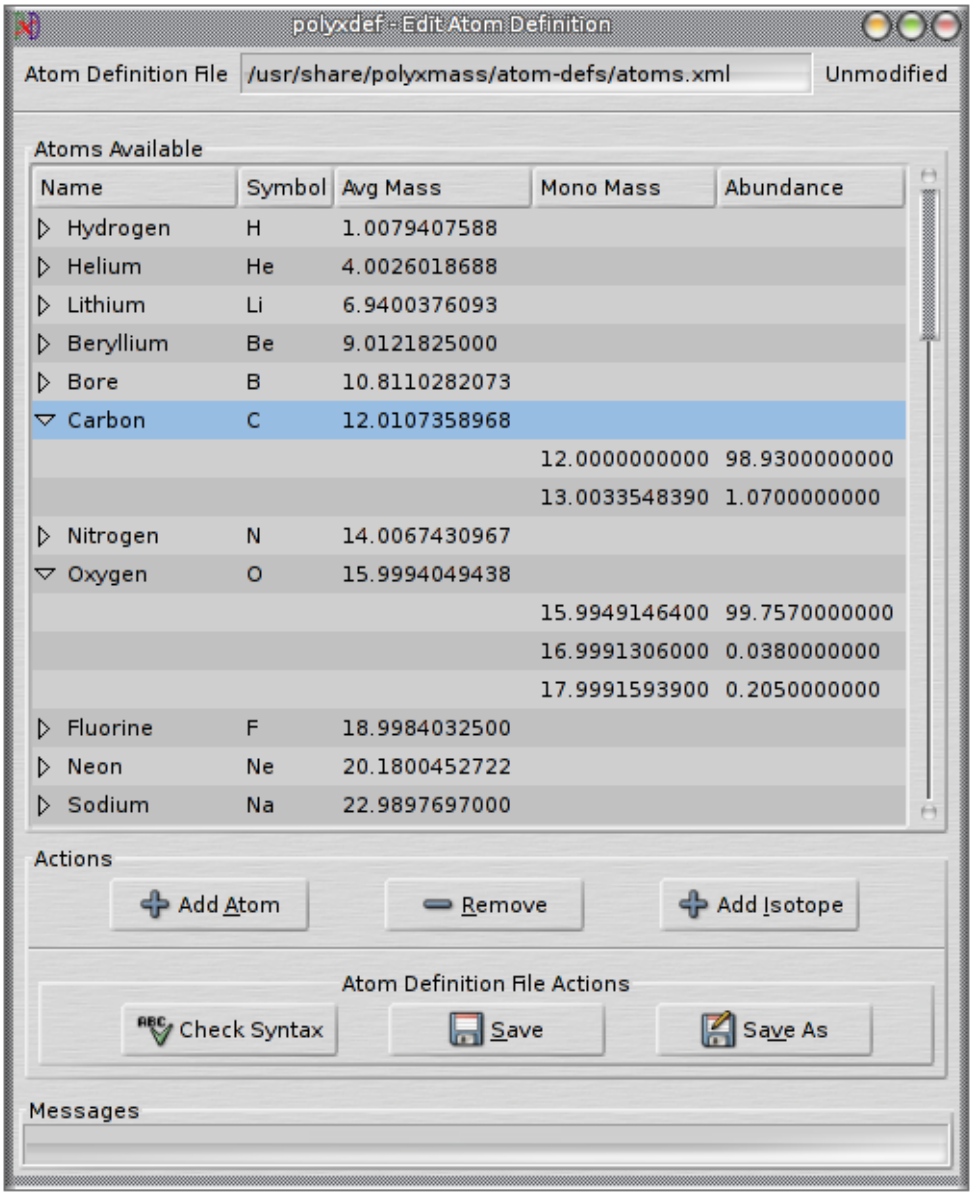

Figure 6.3: **polyxdef atom definition window** The atom items must contain isotope items, otherwise the atom does not have any "raison d'être".

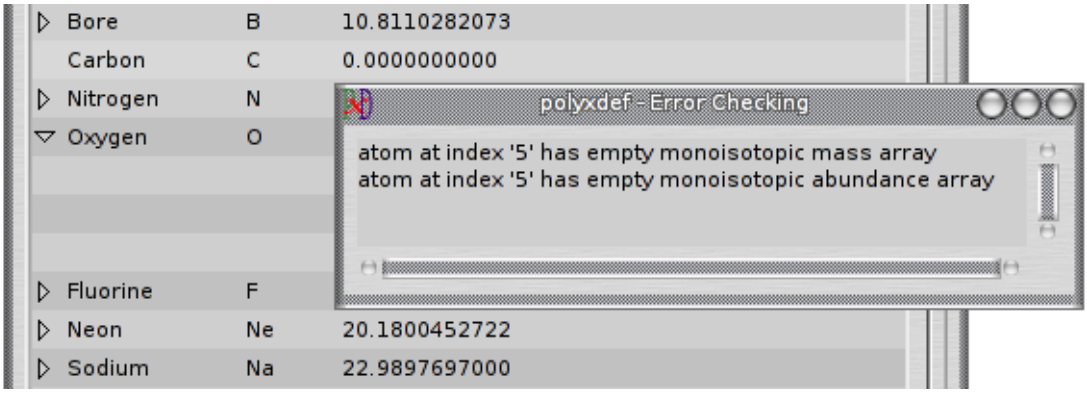

Figure 6.4: **polyxdef atom syntax-checking window** The atom items must contain isotope items, otherwise the atom does not have any "raison d'être". Here, the syntax-checking function has found an error, and the message is displayed in the window overlaid onto the definition window

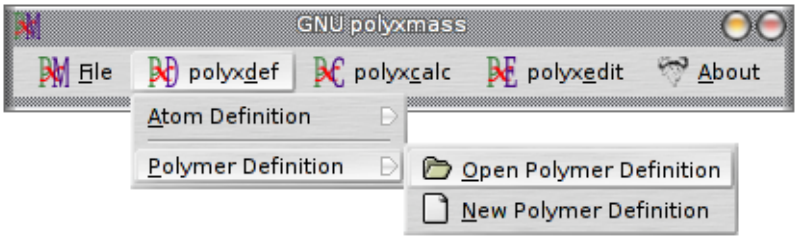

Figure 6.5: **polyxdef polymer chemistry definition menu** The user might ask that a polymer chemistry definition file be opened for editing or that a new polymer chemistry definition be started empty for *ex nihilo* editing.

Editing a polymer chemistry definition

Editing a polymer chemistry definition is performed using the carefully crafted user interface that shows up when the user selects one of the two submenu items shown in Figure 6.5.

When the user asks that an existing polymer chemistry definition file be found, a "chooser window" shows up like the one shown on Figure 6.6 on the facing page.

When the polymer chemistry definition editor shows up, the user sees an interface that allows the addition/removal of a number of chemical items that define the polymer chemistry (Figure 6.7 on page 56). For example, the user might define any number of monomers to be later used in order to create polymer sequences. Equally important is the ability to define any kind of chemical modification (Figure 6.8 on page 57). Doing chemical or enzymatic cleavages on polymer sequences is something rather common in experimental laboratories, and the user can model any kind of chemical/enzymatic cleavage (Figure 6.9 on page 57). Also, it is of crucial importance that the user be able to define any kind of gas phase fragmentations for his newly-defined polymer chemistry (Figure 6.10 on page 57). Also, d

Now that we have made a quick overview of what a polymer chemistry definition looks

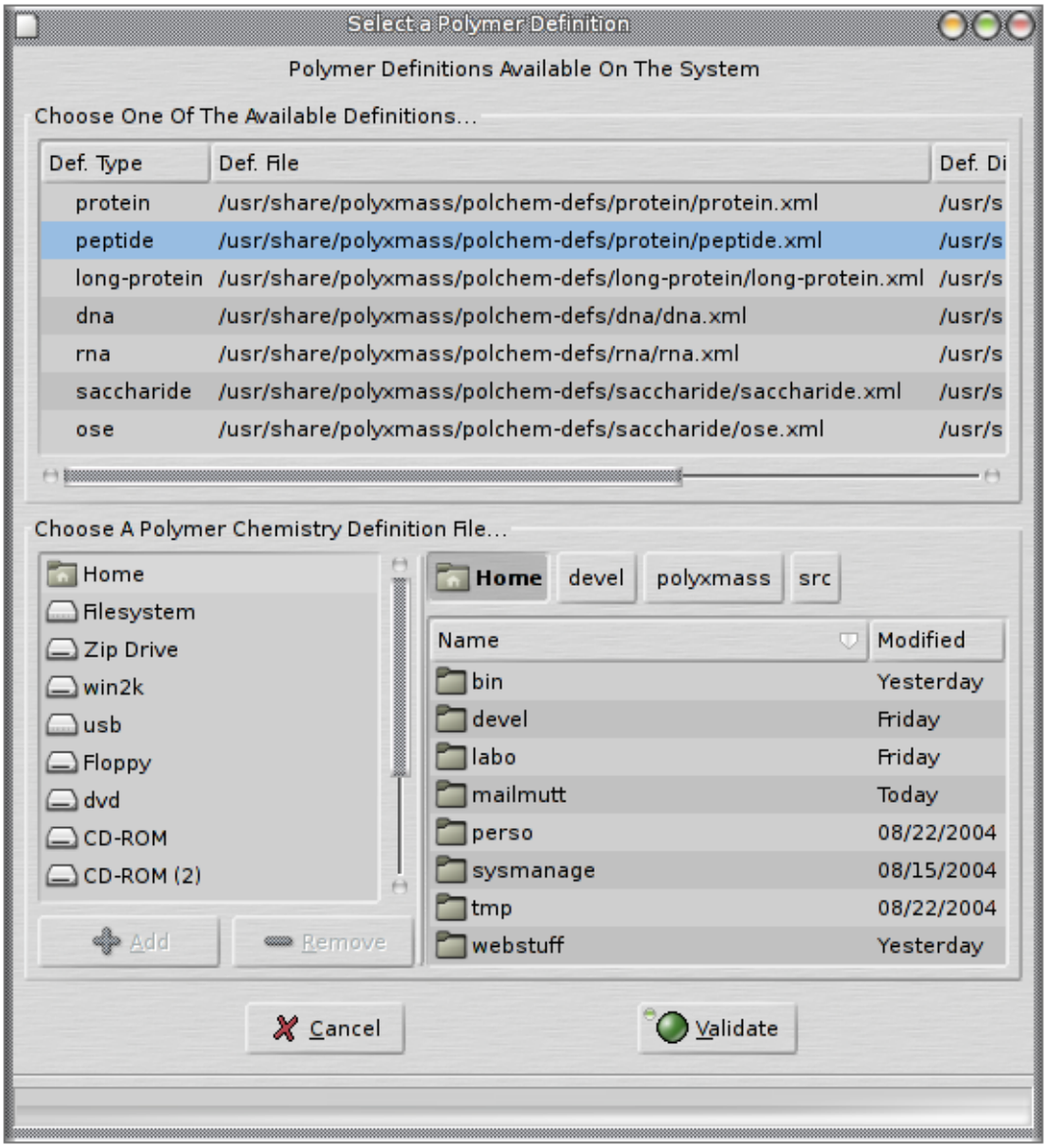

Figure 6.6: **polyxdef polymer chemistry definition choosing window** The user might either select a polymer chemistry definition already registered to the **GNU polyxmass** software suite (upper frame) or select a polymer chemistry definition that is not registered (lower frame).

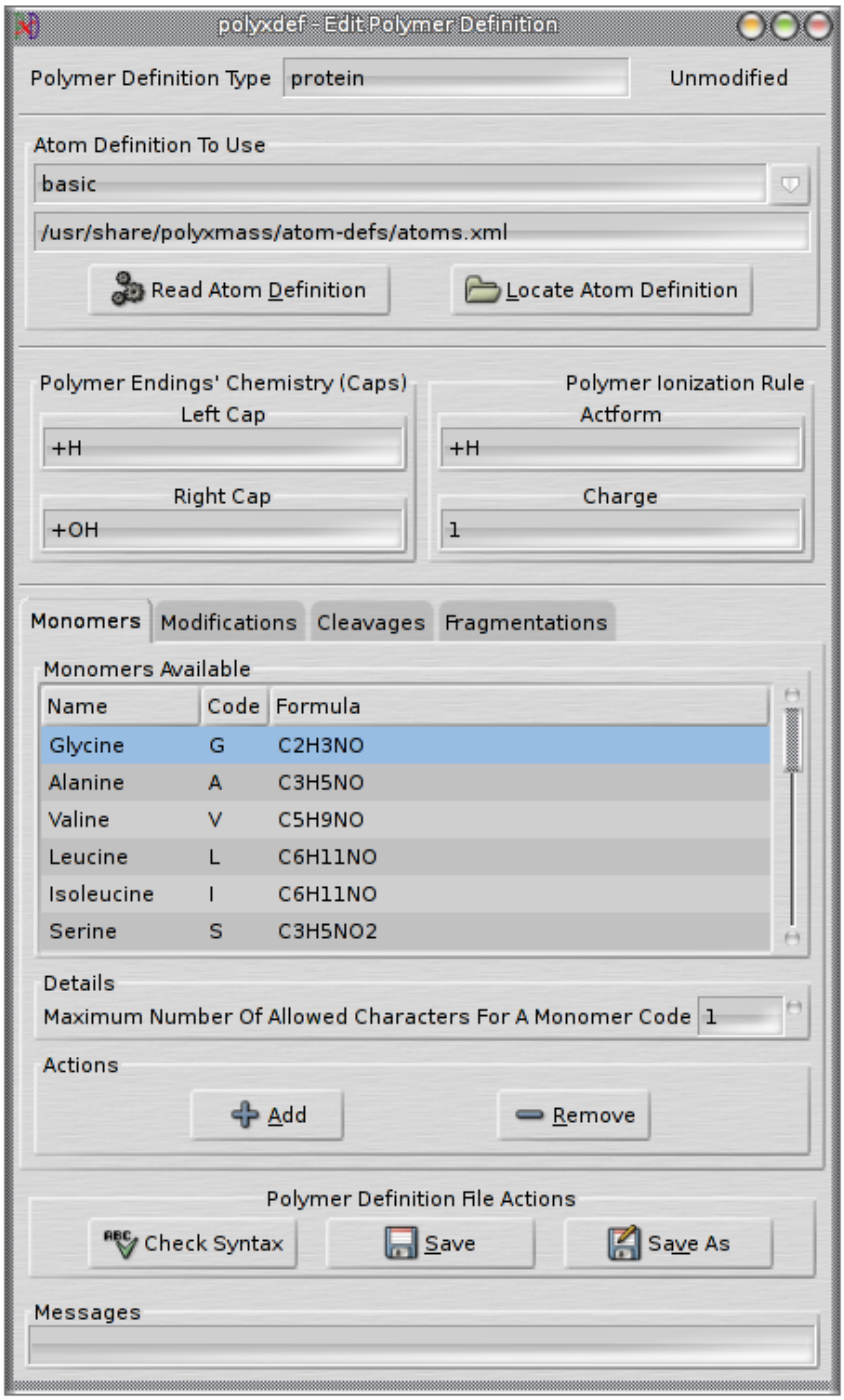

Figure 6.7: **polyxdef polymer chemistry definition window** The window lets the user define with great flexibility the chemical entities that characterize the polymer chemistry being defined. Here the monomer definition treeview is displayed.

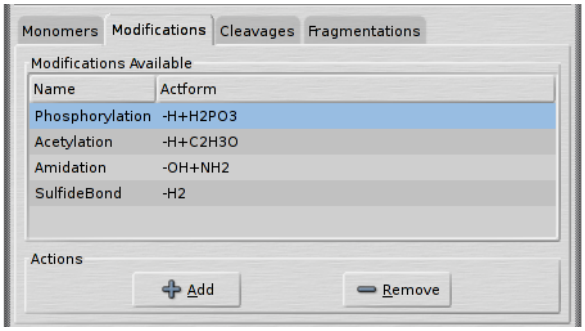

Figure 6.8: **polyxdef chemical modifications definition** The user may define any number of chemical modifications to be later applied to the whole polymer sequence or onto any individual monomer.

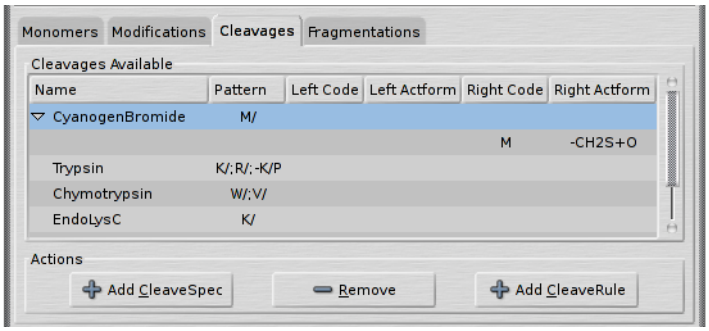

Figure 6.9: **polyxdef cleavages definition** The user may define any number of chemical/enzymatic cleavages to be later applied to the polymer sequence.

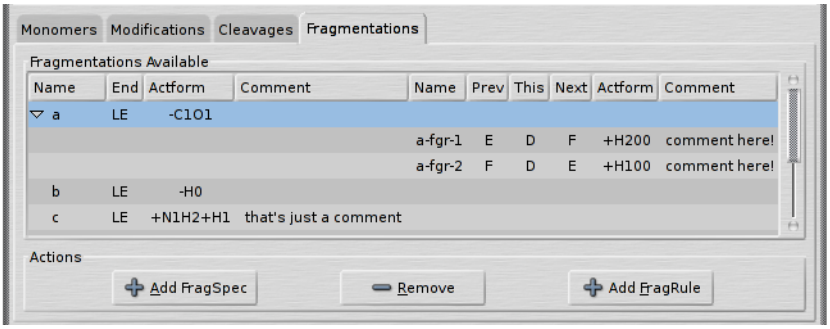

Figure 6.10: **polyxdef fragmentations definition** The user may define any number of gas-phase fragmentation patterns to be later applied to the whole polymer sequence or onto any polymer selection (oligomer).

like, we have to go through some details.

First off, we should immediately explain what the reference to an atom definition file is for, at the top of Figure 6.7 on page 56 (under the label Atom Definition To Use): **GNU polyxmass** is now able to cope with different atom definitions. Each polymer chemistry definition must unequivocally state what atom definition it has to work with. The combobox list item that is shown on the figure reads **basic**. This is where the user should mention what atom definition file is to be used for using with the polymer chemistry definition that is being worked on. The combobox list widget lists all the available atom definitions at the time the window was opened. In the figure it only lists one item: **basic**, which is the basic atom definition file that is installed by the **GNU polyxmass**-common essential **GNU polyxmass** package.

Note that the user is given the opportunity to select an atom definition file that is not yet registered to the **GNU polyxmass** system. To locate such a file on disk, the user should just use the Locate Atom Definition button. Once the user chooses a file on disk, its name will be shown in the text entry widget below the combobox list.

Telling what atom definition file is to be used by any given polymer chemistry definition is of primary importance, because any mass-related computation will be performed by looking at the formulae of each chemical entity in the polymer sequence; the transformation of a chemical formula to a molecular mass is based upon the lookup of what a given atom symbol should weigh. This lookup step is done by going into the atom definition file looking for the atom of the proper symbol, and checking what its isotope(s) is(are). *Thus, we say that the resolution of the **GNU polyxmass** mass spectrometric software suite is isotopic.*

It is necessary to let the **polyxdef** module know the contents of that selected atom definition file, so that they can be used by the polymer chemistry definition being elaborated in this session. This is achieved by clicking onto the Read Atom Definition button. Clicking that button triggers the parsing of the file whose name is displayed in the text entry widget sitting right above the buttons. If an error occurs while parsing the atom definition file, then a message is displayed to inform the user.

It is only when the **polyxdef** module has completed successfully the parsing of the atom definition window (the result will be displayed in a timeout manner in the messages text entry widget at the bottom of the window), that the user can start defining the new polymer chemistry. We will review that process in a detailed manner below.

The atom definition that is associated to the polymer chemistry must be registered to the **GNU polyxmass** software suite at the time the polymer chemistry definition is used. The way this association is performed will be described in a later chapter.

Various Identification And Singular Data

"Identification data" are pieces of information that should be defined in order to describe the polymer chemistry (these are non-chemical pieces of information). For example, an identification datum is the polymer chemistry definition type. "Singular data" are pieces of information that are not present in more than one copy in the polymer definition. An example of a singular datum is the string that describes how the elongating polymer sequence should be left- or right-capped so that it gets to its "finished state", after the polymerization has terminated.

Looking at Figure 6.7 while reading the following paragraphs will help. This and subsequent figures illustrate the process by which a polymer chemistry definition "protein" is defined.

As the reader can see, there are a number of identification and singular data to be entered at the top of the polymer chemistry definition window; these are described in the list below:

* Polymer Definition Type **protein** String describing the type of the new polymer chemistry definition being elaborated;
* Polymer Endings' Chemistry (Caps) Description of the chemical capping reaction that should happen either on the left end (Left Cap) or on the right end (Right Cap) of the polymer sequence, once it is successfully polymerized. As shown, this chemistry is divided into two pieces of information:
  - Left Cap **+H** String describing the actform that should be applied to the left end of the elongating polymer sequence;
  - Right Cap **+OH** String describing the actform that should be applied to the right end of the elongating polymer sequence;
* Maximum Number of Allowed Characters For A Monomer Code **1** This integer value indicates the maximum number of characters that may be used to describe monomer codes. See below for details about this critical value;
* Polymer Ionization Rule This rule describes the manner in which the polymer sequence should be ionized by default, when the mass is calculated. This rule actually holds two elements:
  - Actform **+H** String describing what chemical reaction should be applied to the polymer in order to ionize it. Here we ask that all the polymer sequences of polymer chemistry definition "protein" be protonated once by default;
  - Charge **1** Signed numerical value indicating what charge the polymer will hold once the ionization rule's actform has been applied to it. Here, it is asked that the proteins bear one positive charge after that the default mono-protonation mentioned above has taken place.

Now that we have defined the identification and singular data for the polymer, we will go on with another type of data: "plural data". Conversely to what said previously about singular data, plural data are pieces of information that can be present in more than one copy in the polymer chemistry definition. An example of plural data is the data pertaining to the monomers.

Various Plural Data

The Monomers

The monomers are the constitutive blocks of the polymer sequence. Their definition should be done with great care, as all the mass calculations are based on the formulae of the defined monomers. Remember that in our **GNU polyxmass**' jargon, "monomer" stands *not* for the molecule that you bought from the chemicals vendor in order to synthesize the polymer; it stands for this molecule *less* the chemical group(s) that left it when the polymerization occurred. If this sounds strange to you, you definitely should read chapter 3 on page 15 for a detailed explanation of the **GNU polyxmass** specialized words.

The lower part of Figure 6.7 on page 56 shows how easy it is to define a new monomer: this is as easy as entering three strings in each column of a row (that may be created by clicking onto the Add button). Note that none of the two Name and Formula strings are limited in size.

The case of the Code string is a bit more complicated and depends on the value that is entered in the Maximum Number of Allowed Characters For A Monomer Code field. In our example, this value is **1**, which means that we are allowed to use only one character to describe a monomer's code. Thus, we can see in the figure that all the monomers have a single-character code. It is possible however, to use another value, for example 3. In this case there is a general rule which is enforced in **polyxdef**:

> *"The first character of a monomer code must be uppercase, while the remaining characters (if any) must be lowercase."* That means that —in our example of 3-character codes— 'A', "Al", "Ala" would be perfectly fine, while "Alan", "AL", 'a', "AlA" would be wrong.

.

The mechanism here is highly sophisticated, contrary to what may look like, because you have to imagine what goes on in the different **GNU polyxmass** modules, in particular in the polymer sequence editor (**polyxedit**): how are monomer codes keyed-in if 'A' and "Ala" are valid monomer codes in a polymer chemistry definition? The magic is described in the chapter about **polyxedit**. Not conforming to the instructions above will yield unpredictable results.

The Modifications

Oft-times a polymer will be modified chemically by the user. This is especially true when the user tries to mimick polymer chemical modifications that arise in biochemical processes, in particular regulatory modifications, like protein phosphorylations, for example. Indeed, a biopolymer is modified more often than not. A modification can be a phosphorylation onto a protein residue (on an alcohol function-bearing residue) like a seryl residue, for example, or an acetylation onto a amino function-bearing residue, like a lysyl residue. The **GNU polyxmass** mass spectrometry framework gives the user the entire freedom to define any number of modifications. Let us see how; once again, looking at Figure 6.8 on page 57 will help. Indeed, this figure shows, amongst others, how a *Phosphorylation* modification is defined. Most evidently, a modification is defined by a Name string (of unlimited length) and by an Actform string (of unlimited length). The syntax of an actform should by now be somewhat familiar to the reader. In the *Phosphorylation* case, it can be read like this: —*"The polymer looses a proton and gains H2PO3"*. When the polymer is modified with this modification, its masses will change by the mass corresponding to this "reaction". Of course, the fact that the actform is written in this way is related to the fact that a chemist always thinks in terms of "leaving" and "entering" groups. However, a user might perfectly write "+HPO3" instead of "-H+H2PO3", or even more precisely "-H+H3PO4-OH". Any of these actforms are exactly identical from a molecular mass point of view (and thus also from the **GNU polyxmass**'s perspective).

The Cleavage Specifications

It is common practice —in biopolymer chemistry, at least— to cut a polymer into pieces using molecular scissors like the following:

* proteases, for proteins;
* nucleases, for nucleic acids;
* glycosidases, for saccharides...

For each different polymer type, the molecular scissors are specific. Indeed, a protease will not cleave a polysaccharide. The specificity of a cleaving enzyme is thus something that should be described in each polymer chemistry definition, since this specificity is indeed polymer chemistry-specific. Here we show the way that the user can define the cleavage specificity of a molecular scissor. As usual, looking at Figure 6.9 on page 57 might help in reading the following paragraphs.

By looking at this figure, it should be obvious that defining a cleavage specification gets a little more involved than what we saw earlier for modifications. This is true only for certain chemical reagents that modify the substrate they cleave, which is not that frequent. In the Figure 6.9 on page 57, the first cleavage specification is "CyanogenBromide" (note that there is no space between *Cyanogen* and *Bromide* in the Name column entry).

Let us analyze the data entered by the user in order to fully qualify this cleavage agent (which, conversely to the other ones listed in the Name column of the treeview shown in the figure, is not a protease but a chemical reagent):

* Name **CyanogenBromide** This is merely the name of the cleavage agent;
* Pattern **M/** This tells the **GNU polyxmass** framework where to cleave in the polymer sequence when a CyanogenBromide cleavage is asked. The syntax of the cleavage pattern is detailed below;
* Left Code and Left Actform (Empty) This is a special case for those cleavage agents that not only cut a polymer sequence (usually it is a hydrolysis) but that also modify the substrate in such a way that must be taken into account by **GNU polyxmass** so that it computes correct molecular masses for the resulting oligomers. These rules are optional. However, if Left Code is filled with something, then it is compulsory that Left Actform be filled with something valid also, and conversely;
* Right Code and Right Actform **M** and **-CH2S+O3**, respectively. Same explanation as above. Here, what we say is that each oligomer resulting from the cleavage of the polymer sequence at a 'M' monomer should be modified using the Right Actform actform. Since the cleavage occurs right of 'M', it is logical that a 'M' is found right of the oligomer that was generated upon a "CyanogenBromide" cleavage. A special case in which a 'M' may be found at the right end of an oligomer, without resulting from a polymer sequence cleavage, is if the 'M' was at the right end of the polymer sequence. Of course this case is evaluated and if it is found, the the actform is not applied.

In order to best explicate the cleavage specification pattern syntax I shall provide below some examples:

* **Trypsin = K/;R/;-K/P** *"Trypsin cuts right of a 'K' and right of a 'R'. But it does not cut right of a 'K' if this K is immediately followed by a P"*;
* **EndoAspN = /D** *"EndoAspN cuts left of a D"*;
* **Hypothetical = T/YS; PGT/HYT; /MNOP; -K/MNOP** *"Hypothetical cuts after 'T' if it is followed by YS and also cuts after 'T' if preceded by PG and followed by HYT. Also, Hypothetical cuts prior to 'M' if 'M' is followed by NOP and if 'M' is not preceded by K"*.

Please, *do* note that the letters above correspond to monomer codes and *not* to monomer names. If, for example, we were defining a "Trypsin" cleavage specification pattern —in a protein polymer chemistry definition with the standard 3-character monomer codes— we would have defined it this way: "Trypsin = Lys/;Arg/;-Lys/Pro".

Now comes the time to explain in more detail what the Left Code and Left Actform (along with the Right siblings) are for. For this, we shall consider that we have the following polymer sequence (1-character monomers codes):

THIS**M**WILL**M**BECUT**M**ANDTHAT**M**ALSO

If we cleave this polymer using "CyanogenBromide" and if the cleavage is total,[1] we shall get the following oligomers:

THIS**M** WILL**M** BECUT**M** ANDTHAT**M** ALSO

But if there is a partial cleavage, we would *also* get one or more of these oligomers:

THISMWILL**M** BECUTMANDTHAT**M** ALSO WILLMBECUT**M** ANDTHATMALSO

and so on...

Now, the biochemist knows that when a protein is cleaved with cyanogen bromide, the cleavage occurs effectively right of monomer 'M' (this we also know already) *and* that the 'M' monomer that underwent the cleavage is changed from a methionyl residue to an homoseryl residue (this chemical change involves this actform: "-CH2S+O"). The following two lines of oligomers should definitely "undergo the actform", one time only for each oligomer:

THIS**M**, WILL**M**, BECUT**M**, ANDTHAT**M**

and

THISMWILL**M**, BECUTMANDTHAT**M**, WILLMBECUT**M**

while the two oligomers shown below should not "undergo the actform" because (even if one of them does contain a 'M' monomer) the cleavage *did not occur* at a this 'M' monomer:

ALSO ANDTHATMALSO

This example should clarify why we clearly indicate –in the cleavage specification for "CyanogenBromide"– that the oligomers resulting from this cleavage should "undergo the '-CH2S+O' actform" *only if they have a 'M' as their right end monomer code.*

This would be of crucial importance, if we had a cleavage agent that would cleave not only right of 'M' but at some other places: we really would need to specify these rules in a careful way. For example, imagine you had noted –in your many cyanogen bromide experiments– that more often than rarely cyanogen bromide would cleave right of 'C' (cysteine) residues, but with no chemical modification of the 'C' monomer.[2] In this case, you would be glad that the possibility is given to you to specify that the generated oligomers should "undergo the '-CH2S+O' actform" only if they have a 'M' as their right end monomer, so that 'C'-terminated oligomers are not chemically modified. You would thus safely define this pattern: "M/;C/"...

[1] Cleavage occurs at every possible position, right of each monomer 'M'.

[2] This is a purely hypothetical situation that I never observed personally!

The logical conditions that the user can set forth for a cleavage reaction are called (in an intuitive manner) *cleavage rule*s.

Now that we got trained to think in an abstract way with these cleavage rules, we can proceed to yet meatier stuff: the fragmentation specifications. A polymer chemistry definition can hold as many fragmentation specifications as necessary. A fragmentation specification holds a number of pieces of information, amongst which there is a compound datum describing logical conditions similar to, but more complex than cleavage rules: *fragmentation rules*. Each fragmentation specification might have zero or more (with no limitation) fragmentation rules. We review this complex matter in the next section.

The Fragmentation Specifications

As you might have noticed reading page 26, the fragmentation specification is a tricky business. Figure 6.10 shows examples of protein fragmentation specifications for fragment types *a, b, c, z, y, x, imm*.

Let's concentrate on the fragmentation specification of type *a*. While the first row of this fragmentation specification is effectively valid (for a "protein" polymer chemistry definition, at least), the lower two rows (describing fragmentation rules named *a-fgr-1* and *a-fgr-2*) are fake, only to show the way fully qualified fragmentation specifications can be created.

Let us analyze the data that the user entered to fully qualify this *a* fragmentation specification:

* Name **a** This is the name of the fragmentation specification. Fragments obtained with this specification will be named according to the following naming scheme: "a-*i*", with *a* being the fragmentation name and *i* being the position –in the precursor polymer ion– of the monomer at which the fragmentation occurred (see page 26);

* End **LE** This is the end of the precursor polymer that is to be found in the fragment. Accepted values are "LE" (left end), "RE" (right end) and "NE" (no end). We have previously seen –for proteins and nucleic acids– that fragments *a, b, c* include the left end ("LE") of the precursor polymer, while "RE" applies to fragmentation specifications that lead to fragments that contain the right end of the precursor polymer (for example, fragments *x, y, z*). Special cases, like proteinaceous immonium ions, do not bear any end of the precursor polymer, in which case "NE" (for no end) should be written here instead of "LE").

  This End piece of information is important for two reasons: 1) because it tells the fragmentation engine from which end it should iterate (in the precursor polymer sequence) when making all the fragments of a given fragment ion series and 2) because it guides **GNU polyxmass** to apply the conventional naming scheme using *i* with the proper value. Therefore, the smallest fragment of the *a* series is *a-**1*** (note subscript 1), which is the left end monomer of the precursor polymer. The smallest fragment of the *x* series is *x-**1*** (note that subscript is also 1). This time, the *x-**1*** fragment, however, corresponds to the right end monomer of the polymer sequence. This is because the numbering of the fragments always starts at the precursor polymer's end that was specified by the End piece of data from the polymer chemistry definition;

* Actform **-C1O1** Optional. This is the chemical reaction that will actually change a monomer chain into the proper fragment. Indeed, the mass calculation of the fragment's mass is performed by summing the mass of the monomers running from the *end* of the precursor polymer up to the position where the fragmentation occurs, plus

adding the mass of the end's cap as specified in the polymer chemistry definition. But, for the *a* fragments, this is not enough, as it does not lead to a correct mass. It is required that the actform "-C1O1" be applied to the monomer chain so that it is of the correct mass (after having added the mass corresponding to the left cap; see below). This actform is optional, because for some fragments (for example, fragments *b* in the protein polymer chemistry) there is no need for any actform besides adding the masses of the monomers and adding the mass corresponding to the left cap of the polymer chemistry definition. As can be seen on the picture, "-H0" is set as an actform for *b* fragments. Again, see page 26;

* Comment (Empty) Optional. This is simply a comment, if the user wants to set any. *Ad libitum*.

A fragmentation specification can include zero or more fragmentation rule(s) that help model –in a highly detailed manner– complex fragmentation patterns. Let's see what it takes to define a fragmentation rule:

* Name **a-fgr-1** This is the name of the fragmentation rule. It should be self-explanatory and should somehow provide a hint to the fact that this fragrule belongs to the *a* fragmentation specification;

* Prev **E** Optional. This is one of the logical conditions that can be set to be verified so that the actform can be applied to the fragment currently generated. In our example, we are saying that if –in the precursor ion sequence– the monomer preceeding the one that is currently fragmented is of code 'E', then this condition is verified and the **+H200** actform should be applied to the resulting fragment;

* This **D** Optional. This is an analogous condition as the one above, unless the monomer onto which this condition applies is the monomer being actually fragmented;

* Next **F** Optional. This is similar condition, unless that it applies to the monomer that is one position forward in the precursor ion sequence, with respect to the presently fragmented position;

* Actform **+H200** This is the chemical action with which the fragment will actually be challenged if the set of logical conditions above is verified. This actform is the *raison d'être* of the fragmentation rule, so it is compulsory;

* comment **comment here!** Optional. *Ad libitum*.

A fragmentation rule is a set of one or more logical conditions that (if verified) determine a user-specified chemical actform to be applied to the fragment that was generated in the first place by fragmenting the precursor polymer using the fragmentation specification to which the fragrule itself belongs. As can be seen in the example figure, the fragmentation specification for fragments *a* (fragmentation specification *a*) contains two fragmentation rules, but it could have contained as many of them as necessary to finely describe experimentally observed fragmentation events...

The following paragraph will explain thoroughly how fragmentation rules modify the way fragments are generated, for a given fragmentation pattern.

We have seen, in our example of a fragmentation specification named *a* (Figure 6.10), that it should generate fragments starting from the left end of the precursor polymer. Now we see that the fragmentation specification includes a fragmentation rule: This is set to 'D',

which means that this fragmentation rule is evaluated further *only* if the monomer currently fragmented is indeed a 'D'. If not, the whole fragmentation rule is skipped. If Prev is set to something (for us: 'E'), then the fragmentation rule is evaluated further only if the monomer at position [current -1] is a 'E'. If not, the fragmentation rule is skipped. If Next is set to something (for us: 'F'), then the fragmentation rule is evaluated further only if the monomer at position [current +1] is a 'F'. If not, the fragmentation rule is skipped.

What is called a position **[current +1]** and a position **[current -1]** depends on the kind of fragmentation specification: if the fragmentation specification states that End (seen earlier) is "LE" (or "NE"), then the position [current +1] refers to the position right of the currently fragmented monomer (in the standard left-to-right polar horizontal representation of a polymer); if the fragmentation specification states that End is "RE", then the position [current +1] refers to the position left of the currently fragmented monomer. This has to do with the way the fragmentations are normally described: the fragment numbering scheme starts at the right end of the precursor polymer for "RE" fragments and at the left end of the precursor polymer for "LE" fragments. This is also true here: for a fragment of the series *a*, the fragmentation rule that we have described would effectively be applied to the following sequence:

MYNAMEIS**EDF**FIL

only upon generation of the MYNAMEIS**ED** fragment.

If we were using the same fragmentation rule for a fragment of the series *x* (for which End is "RE"), the fragmentation rule would never have been evaluated. Instead, for the following sequence:

MYNAMEIS**FDE**FIL

it would have, and thus would have generated the fragment **EDF**IL.

Now, what about internal fragment specifications, like the immonium ions' case, where the End is defined to be "NE" in the polymer chemistry definition? **GNU polyxmass** evaluates the conditions from left to right; so the conditions are evaluated like for "LE" cases.

Another important thing to figure out: how are the logical conditions tested? The main condition (entered as This) is evaluated first, because this is the simplest evaluation: the value of the This monomer can be compared with the currently fragmented monomer code without depending on the End value. If the monomer context complies with this condition (in our example that would mean that we are actually fragmenting at a 'D' monomer), other conditions (if any) are evaluated. Thus, in logic terminology the conditions are *AND*ed one with the other: as soon as a condition is stated it must be verified. If *any* condition is not verified, no fragment is created and the other fragmentation rules are analysed (if any).

If there are more than one fragmentation rule in a fragmentation specification, each fragmentation rule is evaluated separately. If the monomeric context (previous/this/next monomer codes) complies with the logical conditions stated in the evaluated fragmentation rule, a new fragment is generated. When a fragmentation rule is found not to comply with the monomer context, then it is simply skipped (no fragment is generated).

It should be noted that the presence of a fragmentation rule in a fragmentation specification is not exclusive, in the sense that if the fragmentation rule contains never satisfied logical condition(s),[3] a single fragment is indeed generated, which corresponds to the fragmentation specification without taking into account any framentation rule.

[3]Such as if "this monomer's code" is 'Y', "next monomer's code" is 'Y' and "previous monomer's code" is 'Y' and there is no "YYY" sequence element in the polymer, for example.

The fact that each fragmentation rule –that has logical conditions which are verified in the sequence– yields a new fragment implies that the fragmentation rules are not summative: a fragment is not generated by applying onto it the actform of each validated fragmentation rule in a fragmentation specification. Each fragmentation rule, in a given fragmentation specification, gives rise to a fragment that is a fragment ion resulting from the application of both the actform specified in the fragmentation specification (if any) and the actform specified in the fragmentation rule (this one is compulsory). Next, when another fragmentation rule of the same fragmentation specification is evaluated, a brand new fragment is generated according to the same process as the one just described.

Saving A Polymer Chemistry Definition

Once the polymer chemistry definition is completed, the user can save it to a file. Prior to actually writing to the file, the program checks the syntax validity of the elements that the user has entered in the window. This check can be triggered manually by clicking onto the Check Syntax button. If an error is found in the polymer chemistry definition being worked on, that error is displayed in a window so that the user may identify the problem and fix it.

When saving a polymer chemistry definition to a file, if no error is detected, the program proceeds with writing the polymer chemistry definition to an *XML* file.

The location where the file should be saved, and the manner that it may be made available to the whole **GNU polyxmass** framework is to be described in a later chapter.

In that chapter, the user will be instructed on how to insure that the newly-made polymer chemistry definition uses the proper atom definition.

Indeed, **GNU polyxmass** is a rather powerful framework, wholly designed to be modular. But this modularity and power have a cost: complexity. A well configured system is the key to a powerful program running smoothly. It is thus very important to grasp the **GNU polyxmass** framework configuration data hierarchy so that the program knows at each instant where to find the configuration and data files required to perform properly both the polymer sequence display and the mass calculations.

But for now go on with the polymer chemistry definition-aware calculator: **polyxcalc**!

7

polyxcalc: A Powerful Mass Calculator

After having completed this chapter you will be able to perform sophisticated mass computations in a polymer chemistry-aware manner.

polyxcalc Invocation

The **polyxdef** module is easily called by pulling down the "**polyxcalc**" menu item from the **GNU polyxmass** program's menu. The user may accomplish two different tasks in the **polyxcalc** module:

* Use the calculator in a polymer chemistry-unaware manner;
* Use the calculator in a polymer chemistry-aware manner.

polyxcalc Operation: An Easy Task

When the user elects to open a **polyxcalc** module with a polymer chemistry definition, he is provided a window where to choose an already registered polymer chemistry definition

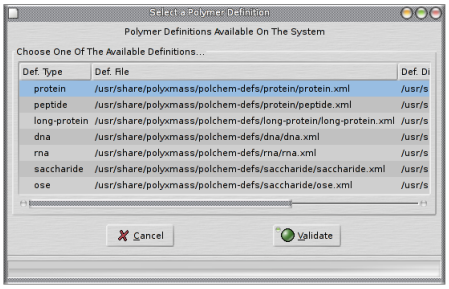

Figure 7.1: **Selecting a polymer chemistry definition for use with polyxcalc** This figure shows that the user can only select one already registered polymer chemistry definition with the **polyxcalc** module. Choosing a polymer chemistry definition will allow to take advantage of all the chemical entities defined therein during the mass computations.

out of the list of all the polymer chemistry definitions available to the **GNU polyxmass** framework (Figure 7.1).

When the user selects one polymer chemistry definition, the calculator window comes preloaded with that definition, which allows the user to take advantage of the chemical entities defined in that definition when performing mass computations.

The way **polyxcalc** is operated is very easy. This is partly due to the very self-explanatory graphical user interface of the module, which is illustrated in Figure 7.2. As the reader can see, there are a number of items that **polyxcalc** can handle. We are going to review these one by one:

* Initial Masses This is the place where the mass calculator may be seeded so as to start computations on pre-existing molecules of which masses are known already. The user may enter either a Mono Mass or an Avg Mass or both masses. When any of these masses is set and the Result Masses are empty, they are taken into account (**polyxcalc** considers that the system is seeded with them) in the first mass calculation that is elicited by clicking onto the Apply button. Once the Result Masses are no more empty, these masses are no more taken into account, and instead will be updated to reflect the previous mass calculation results. Thus, each time a calculation is performed, the previous results are stored in the Initial Masses text entry widgets. This way, the user has the ability to always "undo" the last calculation step;

* Atom This is a drop-down list widget that contains all the atoms available in the polymer chemistry definition-associated atom definition.[1] The user may select any of these atoms and enter any number (positive or negative) in the related Count text entry widget. Entering a positive value means that the selected chemical entity must be added to the masses, while a negative value will remove this entity from the masses;

* Formula/Actform This is a text entry widget where the user may enter as complicated actforms (or a formula) as she wishes. Same as above applies for the Count text entry widget;

[1] If no polymer chemistry definition is loaded into the calculator, then the 'basic' atom definition is used. That 'basic' atom defintion is distributed along the **GNU polyxmass-common** package that is essential to the proper functioning of the **GNU polyxmass** mass spectrometry software framework.

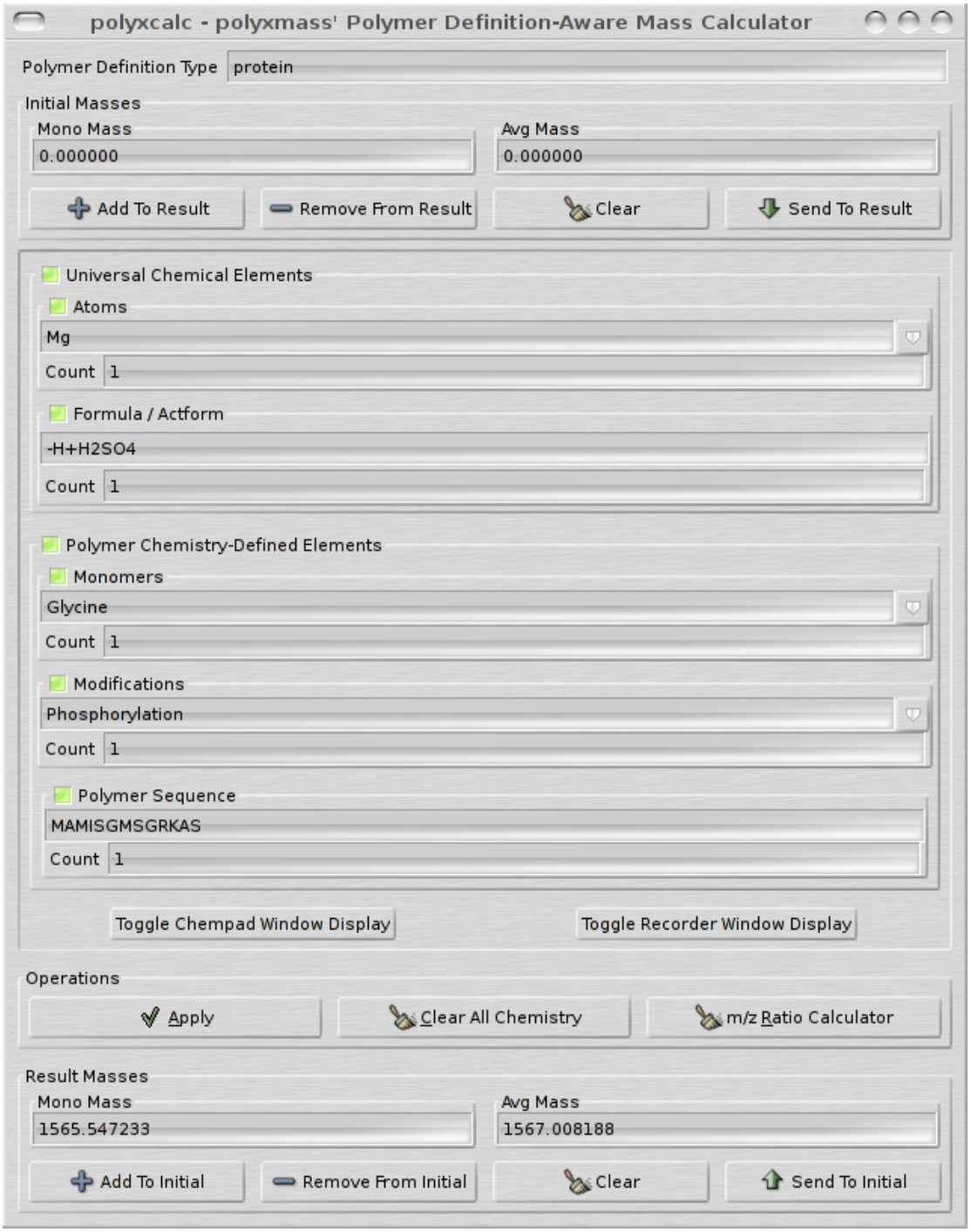

Figure 7.2: **Interface of the polyxcalc module.** This figure shows that the **polyxcalc** polymer definition-aware module can handle atoms, actforms, monomers, modifications and even polymer sequence... for computing masses.

* Monomers If a polymer chemistry definition file was loaded when bringing up this **polyxcalc** calculator module, this drop-down list widget contains all the monomers listed in the chosen polymer chemistry definition. For example, if the "protein" polymer chemistry definition file had been opened in **polyxcalc**, then this drop-down list widget would have contained the twenty names of all the naturally-occurring most common monomers (amino-acids). Same as above applies for the Count text entry widget;
* Modifications This is exactly the same as for the Monomers drop-down list widget, unless the "chemical elements" listed here are the modifications described in the polymer chemistry definition file, such as "Acetylation" or "Phosphorylation", for example. Same as above applies for the Count text entry widget;
* Polymer Sequence This is a text entry widget were the user may enter a polymer sequence complying with the polymer definition currently opened in **polyxcalc**. A "protein" sequence may be this "MAMISGMSGRKASPTSPINADK", for example, which is the N-terminal end of the chicken gizzard telokin.[2] Same as above applies for the Count text entry widget;

Noteworthy, when **polyxcalc** is launched without specifying a polymer chemistry definition, the polymer chemistry definition-specific widgets (monomers, modifications, polymer sequence; all described above) are made insensitive. This is to make sure that the user cannot enter data that would not make sense because the chemistry definition is loaded.

Also, interesting from a graphical user interface point of view, the fact that the user might "collapse" parts of the calculator window widgets. For example, if the user does not make use of monomers, she might click onto the Monomers checkbutton to have the whole monomers-related frame collapse. This is true for all the other frames on that window. More radically, if no polymer chemistry definition was loaded when launching the **polyxcalc** module, the whole series of widgets pertaining to the polymer chemistry definition may be collapse in one step by clicking onto the Polymer Chemistry-Defined Elements checkbutton.

Note also, that a series of buttons, located both in the Initial Masses and Result Masses frames are made available to the user in order to perform sophisticated combinations of calculations... The reader is invited to experiment with these buttons'-triggered actions.

The mass calculation operation is actually triggered by clicking onto the Apply button. When this button is pressed, then the program goes through all the valid widgets an applies all the requirements that are listed in the window. From the Figure 7.2, we can see that the user asked a number of chemical operations to be performed in one Apply button click:

* Add one **Mg** atom;
* Add the net mass corresponding to the **+H2SO4-H** action-formula;
* Add one **Glycine** residue;
* Modifiy the substrate using one **Phosphorylation** chemical modification entity;
* Add one protein sequence: **MAMISGMSGRKAS**;

When all these operations are performed, starting from empty Initial Masses, the result is displayed in the text entry widgets located in the Result Masses frame: **1565.547233** as the monoisotopic mass, and the **1567.008188** as the average mass.

As a final point, the reader may have noticed that, with this interface, any possibly imaginable molecule can be constructed since the "granularity" of the **polyxcalc** module is atomic.

[2] If I remember well my PhD experimental work...

polyxcalc Contains A m/z Ratio Calculator

It very often happens that the massist doing electrospray analyzes is faced with a challenging task: to compute by mind all the m/z ratios for a given family of charge peaks. To ease that daunting task, **polyxcalc** contains a m/z ratio calculator that is called by clicking onto the m/z Ratio Calculator button. This action pops up a window that is shown in Figure 7.3.

In order to compute the m/z ratios requested by the user, the program needs to have some seeding data, which have to be entered in the Initial Data frame widget. Of course, initial m/z values have to be entered (both monoisotopic and average m/z values need to be entered). The user must inform the calculator about how these m/z values were calculated, that is what was the ionization status of the analyte when these m/z values were measured (either virtually or effectively). These ionization data are entered in the Ionization Status frame, which is subdivided into two frames, where the user enters how the ionization is performed (in our example, the ionization is a protonation, thus bringing one charge per protonation event (Ionization Unitary Charge is **1**)) and what's the level of the ionization, that is how many times the ionization was performed (in our example, the ionization level is **1**, that is the analyte was mono-protonated). With all these data, the m/z ratio calculator can compute the molecular mass of the analyte. That mass will be used to perform the requested m/z ratio calculations (Requested Ionization Status frame, which behaves identically to the one described above). The computed m/z ratios are displayed in a treeview widget.

polyxcalc Is A Programmable Calculator

For the scientists who work on molecules that are often modified in the same usual ways, **polyxcalc** features a built-in mechanism by which they can easily program their calculator. This programming involves the definition of how a *chemical pad* (or *chempad*) may be arranged, exactly the same way as a usual calculator would display its numerical keypad.

An example of such a chemical pad is shown in Figure 7.4, where a "protein" polymer chemistry definition-associated chempad is featured. As shown, the user has programmed a number of chemical reactions that may be applied to the masses in the **polyxcalc** calculator window by simply clicking on their respective item.

The configuration of the chemical pad is very easy, as shown in the code below (excerpt taken from file `/usr/share/polyxmass/polchem-defs/protein/chempad.conf`):

```
chempad_columns$3

chempadkey=protonate%+H1%adds a proton
chempadkey=hydrate%+H2O1%adds a water molecule
chempadkey=OH-ylate%+O1H1%adds an hydroxyl group
chempadkey=acetylate%-H1+C2H3O1%adds an acetyl group
chempadkey=phosphorylate%-H+H2PO3%add a phosphate group
chempadkey=sulfide bond%-H2%oxydizes with loss of hydrogen
```

What this text file says is very simple:

* That the buttons should be arranged in rows of three columns;
* Follows the description of a number of buttons (chempad keys) to be laid out in the chempad (each line describes one button).

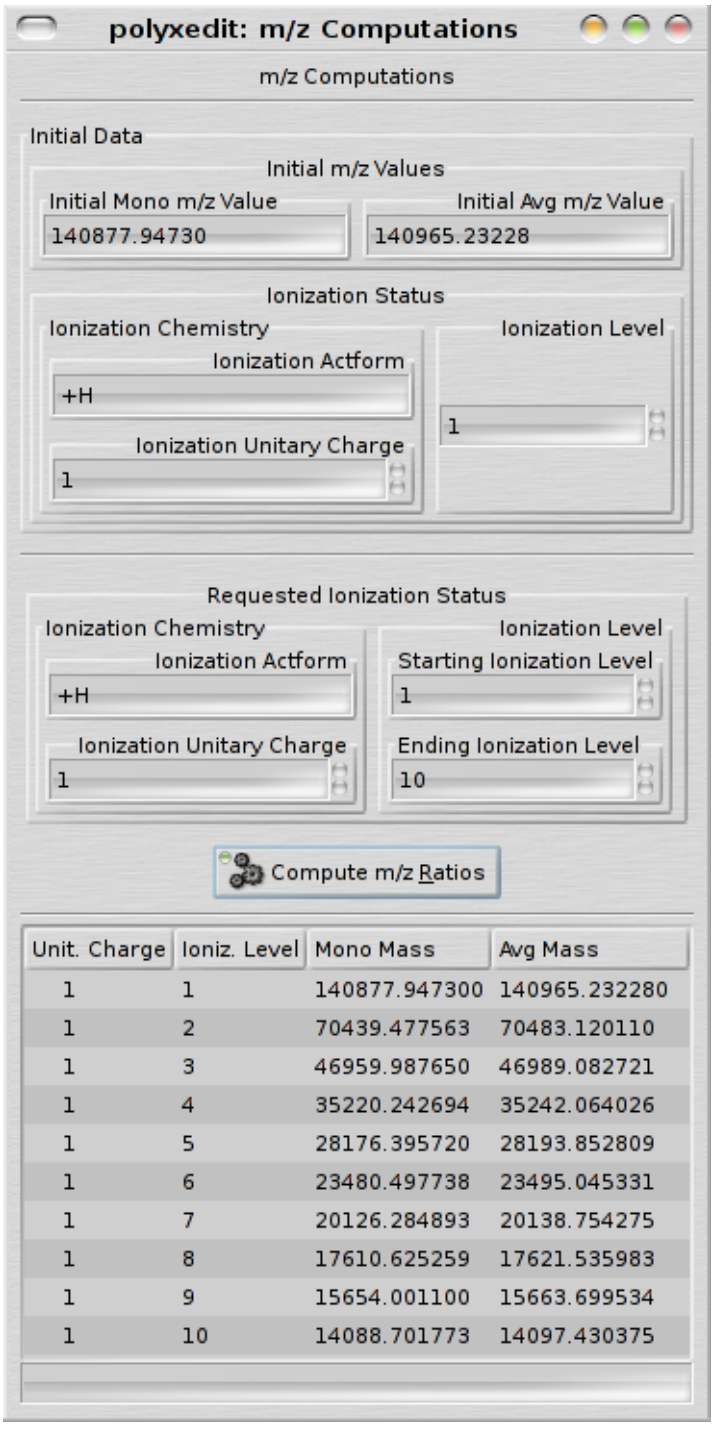

Figure 7.3: **The m/z ratio calculator.** The m/z ratio calculator is rather straight forward to use. Given initial m/z values with details on how they were computed, the program computes m/z ratios as requested.

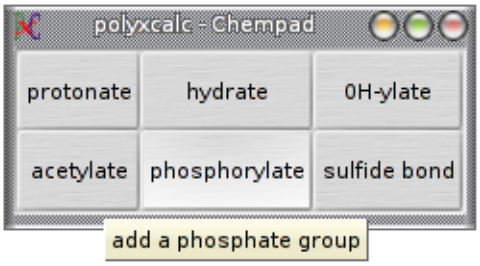

Figure 7.4: **Interface of the chemical pad.** This figure shows that the chemical pad is very similar to what a numerical calculator would display. Here, the user has programmed a number of chemical reactions.

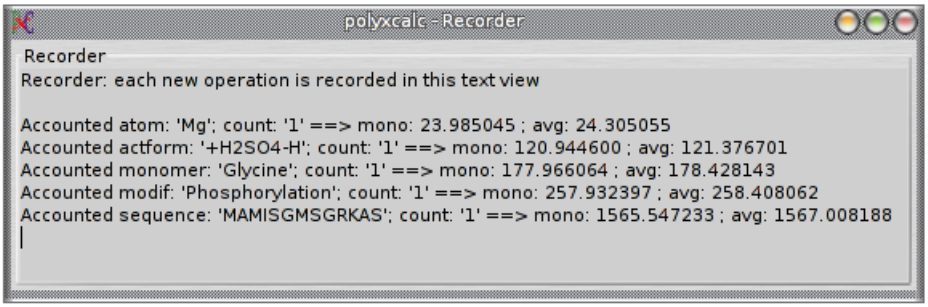

Figure 7.5: **The polyxcalc recorder window.** This figure shows that the recorder window is a simple textview widget that records all the mass-significant operations in the **polyxcalc** calculator. The text in the recorder may be selected and later used in an electronic logbook or printed.

Each button is defined in a line that begins with the text `chempadkey=`. Let's look at one button definition, the "phosphorylate" button. The `phosphorylate` text string after the `=` character is the label that will decorate the button that is being configured. The `-H+H2PO3` text string is the actform that should be applied to the result masses in the **polyxcalc** main window when this button is clicked; that's a chemical reaction, in fact. The `add a phosphate group` is a text string that is displayed as a tooltip when the mouse cursor stays for a number of milliseconds over the button.

From a geometrical layout point of view, the user is allowed to set either a number of rows (`chempad_rows$3`, in our example) or a number of columns (`chempad_columns$3`, in the example). The program then chooses the best layout corresponding to the user's requirement.

polyxcalc Is LogBook-Friendly

Each time an action that is chemically relevant —from a molecular mass perspective— is performed, the program dumps the calculations to the **polyxcalc** recorder window.

This recorder window is shown in Figure 7.5. The text in the recorder window is editable

for the user to customize the **polyxcalc** output, and selectable so that pasting to text editors or word processors is easy.

8

polyxedit: A Powerful Simulator

After having completed this chapter you will be able to perform sophisticated polymer chemistry simulations on polymer sequences —that can be edited in place— along with automatic mass recalculations.

polyxedit Invocation

The **polyxedit** module is easily called by pulling down the "**polyxedit**" menu item from the **GNU polyxmass** program's menu. The user may start the **polyxedit** module by:

* Ask that a polymer sequence be loaded from disk;
* Ask that a new polymer sequence be started *ex nihilo*.

polyxedit Operation: *In Medias Res*

When starting a new polymer sequence from scratch, the first thing the program does is to provide the user with a window (Figure 8.1 on the following page) where the user is invited

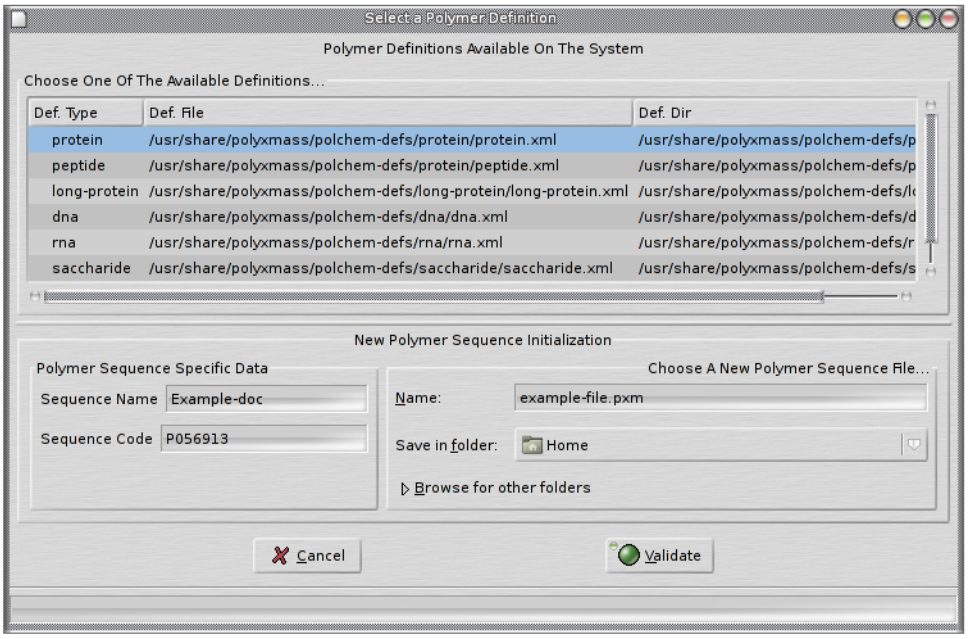

Figure 8.1: **Initializing a new polymer sequence in polyxedit** When starting a new sequence from scratch, it is necessary to seed the program with a number of data that the user is invited to give in this window.

to:

* Select the polymer chemistry definition (Def. Type) to be used to interpret the polymer sequence (compulsory datum);
* Enter a Sequence Name for the polymer sequence (non-compulsory datum);
* Enter a Sequence Code for the polymer sequence (non-compulsory datum);
* Choose a file Name for the polymer sequence file.

Once all the data have been selected/entered, then the user clicks onto the Validate button and the program open an empty sequence window as shown on Figure 8.2 on the next page.

At this point, when the user starts editing a sequence, the characters entered at the keyboard, or pasted from the clipboard, will be interpreted using the polymer chemistry definition that was selected in the initialization window described above.

Now, of course, editing a polymer sequence is not enough for a mass spectrometric-oriented software suite; what we want is to compute masses! When the **GNU polyxmass** software program is started, the window displaying the masses of the sequence being edited is not displayed. Go to the main menu of the program and select the item ***polyxedit*** $\longrightarrow$*View* and activate the checkbutton menu Display Masses Window.

The window that displays the masses for the currently edited polymer sequence is show in Figure 8.3 on the facing page, where the reader can see that two different types of masses are displayed:

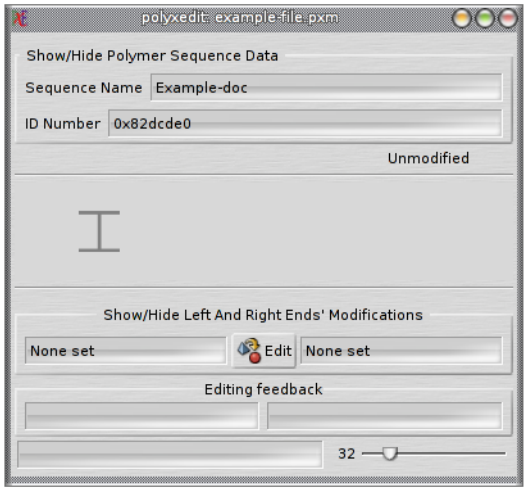

Figure 8.2: **An empty polyxedit window** This figure shows an empty **polyxedit** window, waiting for the user to either paste a sequence from the clipboard or edit one from the keyboard.

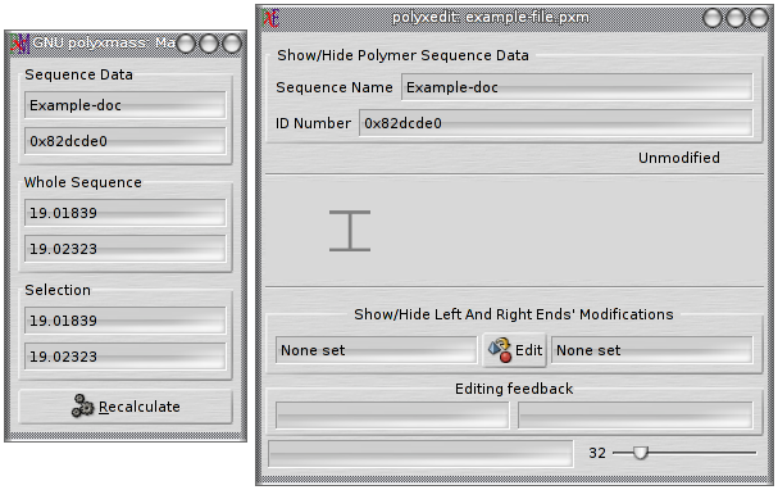

Figure 8.3: **The window displaying the masses** This figure shows the window that displays masses for the currently edited polymer sequence. As can be seen the identity of the polymer sequence is shown along with masses computed for the sequence.

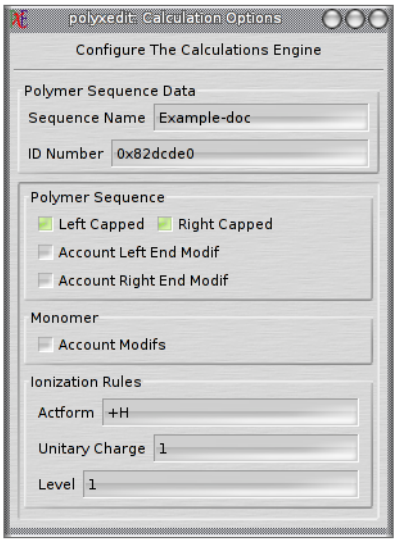

Figure 8.4: **Configuring the mass calculation engine** This figure shows the detail in which the mass calculation engine can be configured. See the text for details.

* Whole Sequence These are the monoisotopic and average masses computed for the whole polymer sequence;

* Selection These are the monoisotopic and average masses computed for the selected portion of the polymer sequence;

As the user can see, the protein sequence that we did initialize earlier is empty (the only visible item is the cursor), and the masses displayed correspond to an empty protein. But if there is no polymer sequence, then how come *nihil* weighs some 19 mass units? Well that's because we still have to show how polymer sequence masses are computed: by adding the masses of each monomer in the sequence, that's for sure. But also —depending on the configuration set by the user— on other parameters. Figure 8.4 shows to what extent the way masses are computed can be configured. The window that is shown in this figure was shown as a result of right-clicking in the polymer sequence editor, selecting —from the contextual menu that pops up— the *View*⟶*Calc. Options* menu.

We'll review the different items in this window:

* Sequence Name: This entry widget holds the name of the polymer sequence for which the mass computations are being configured;

* ID Number: Unambiguous identification of the polymer sequence (this is useful in case the same identical polymer sequence file is loaded twice in **polyxedit** since this ID number will differ);

* Left Capped: If checked, the left cap of the polymer definition corresponding to this polymer sequence will be taken into account when computing masses

* Right Capped: Same as for Left Capped but for the right end of the polymer;

* Account Left End Modif: If checked, take into account the modification that might be set to the left end of the polymer sequence;
* Account Right End Modif: If checked, take into account the modification that might be set to the right end of the polymer sequence;
* Monomer – Account Modifs: If checked, take into account the chemical modifications that might be set to monomers in the polymer sequence (or selection portion of it);
* Ionization Rules – Actform: What action-formula to apply to the polymer sequence when ionization is computed;
* Ionization Rules – Unitary Charge: What is the charge that is brought by the action-formula mentioned above;
* Ionization Rules – Level: How many times the polymer sequence should be ionized according to the two data elements above.

The fact that the user can specify ionization rules should make it clear that the masses that are displayed are actually $\frac{m}{z}$ ratios, as long as one ionization occurs... Also, note that the masses that are displayed in the window shown in Figure 8.3 on page 77, are updated automatically anytime something "ponderable" happens with the polymer sequence (Whole Sequence masses) or anytime the cursor is moved in the sequence (this is equivalent to selecting from the beginning of the sequence up to the cursor point) or a selection is modified (Selection masses).

For the moment that should be enough. Let's delve more into the capabilities of the **polyxedit** module of the **GNU polyxmass** mass spectrometric software suite.

polyxedit The Polymer Sequence Menu

There are two menus available to the user in the polymer sequence editor window. The first menu is a conventional menu sitting on top of the sequence editor window. The second menu pops up when the user right-clicks onto the sequence-displaying area onto a monomer icon. The general rule of thumb is rather simple: whenever a menu item allows to perform an action onto a specific sequence graphical rendering item (I mean a specific sequence as displayed in a specific canvas), the menu to explore first is the popup menu. Conversely, if the action to be triggered more about the sequence itself, and less about its actual graphical rendering, then the menu to explore first is the main window menu.

The sequence editor window main menu comprises the items described below:

* *File*
  - ⟶*Save...* Save the polymer sequence;
  - ⟶*Save As...* Save the polymer sequence with a new name;
  - ⟶*Close...* Close the polymer sequence;
* *Edit*
  - ⟶*Polymer Sequence Properties...* Edit the polymer sequence properties, such as sequence name, sequence code, for example. Note that the annotation process will let you enter as many notes as required to the polymer sequence;

- *View*
  - ⟶*Calc. Options...* View/Modify the way calculations are performed, be them mass calculations or elemental composition calculations;
- *Chemistry*
  - ⟶*Cleave...* Open a window so that a polymer sequence can be cleaved;
  - ⟶*Fragment...* Open a window so that a polymer sequence can be fragmented;
  - ⟶*Compositions*
    - ⟶*Elemental...* Open a window so that options can be set for the program to compute the elemental composition of the polymer sequence or a region of it;
    - ⟶*Monomeric...* Open a window so that options can be set for the program to compute the monomeric composition of the polymer sequence or a region of it;
  - ⟶*pKa-pH-pI*
    - ⟶*(Re)Load The Data...* Ask that the `acidobasic.xml` file be read (or re-read) from disk;
    - ⟶*Calculations...* Open a window so that options can be set for the program to compute the charges of the polymer sequence (or its isoelectric point);
  - ⟶*m/z Ratio Calculations...* Open a window to perform m/z ratio calculations;
  - ⟶*Search Mass(es)...* Open a window so that options can be set for the program to search arbitrary oligomers in the polymer sequence that have the same mass as the one(s) searched for.
- *Reports*
  - ⟶*Make Reports...* Open the window management facility to let the user choose windows to make reports of their contents;
  - ⟶*Report Options...* Configure the way reports are prepared.

Note that each action undertaken as the response to choosing one menu item is performed onto the polymer sequence being edited in the polymer sequence editor from which the menu was selected.

When the user right-clicks onto a monomer icon, an Edit contextual menu pops-up that has the following menu structure:

- *Edit*
  - ⟶*Copy...* Copy to the clipboard the currently selected sequence;
  - ⟶*Cut...* Copy to the clipboard the currently selected sequence and remove it from the sequence;
  - ⟶*Paste...* Paste the sequence from the clipboard to the current location of the cursor in the polymer sequence editor;
  - ⟶*Find Replace...* Extremely flexible Find/Replace functionality;

- ✦ *Annotation*
  - ★ ⟶*Monomer*... Edit (add/remove/modify) the notes for the monomer lying below the cursor when the menu was elicited;
  - ★ ⟶*Polymer*... Edit (add/remove/modify) the notes for the polymer being edited in the polymer sequence editor;
- ✦ ⟶*List Completions*... Show the list of available monomer code completions according to what is already typed in the sequence editor and the monomer codes defined in the polymer chemistry definition;
- ✦ *Select All*... Selects the whole sequence in the polymer sequence editor;

✱ *Chemistry*

- ✦ ⟶*Monomer Modifications*... Open a window so that a monomer (or any combination of monomers) can be modified or unmodified;
- ✦ ⟶*Polymer Modifications*... Open a window so that the polymer sequence can be modified or unmodified either on its left end or on its right end (or both);

✱ ⟶*Self Read Sequence To File*... Write to file a configurable list of sound files to play the sequence aloud;

Editing Polymer Sequences

As we have seen in the **polyxdef** module, the user may stipulate that a polymer chemistry definition allows more than one character in order to define the codes of the different monomers of this same polymer chemistry (see section 6 on page 59). Remember that it is not because the number of allowed characters is 3, for example, that all your monomer codes must be defined using three characters. 3 is the *max* number of characters that you may use. This means that you are perfectly entitled, in this case, to have single-character or bi-character monomer codes in this polymer chemistry definition. Let's start by looking at how the polymer sequence editor window behaves when the user tries to enter multi-character monomer codes. Next, we'll see that whatever the length of a monomer code, if its very first character is unambiguous, the behaviour of the polymer sequence editor is flexible and powerful.

Multi-Character Monomer Codes

In this section we will describe the editing of a polymer sequence for which monomers can be described using more than one character.

The Figure 8.5 on the following page shows the case of a polymer sequence that is of a polymer chemistry definition that allows three characters to define monomer codes. Let's now assume that the user wants to edit the sequence by insertion —at the cursor point— of a new monomer "Aspartate", of which the user knows only that its code starts with an 'A'. The cursor is located between the two "Ala" monomers at positions 15 and 16 (panel 1).

The user keys-in A (panel 2). To her dismay, nothing happens in the polymer sequence, but she sees an 'A' character now displayed in the left text widget under the label Editing Feedback. The reason why we have this behaviour is related to the fact that we are allowed up to 3 characters to describe a monomer code. If no monomer icon is displayed in the polymer sequence, that may simply mean that more than one monomer code start with an

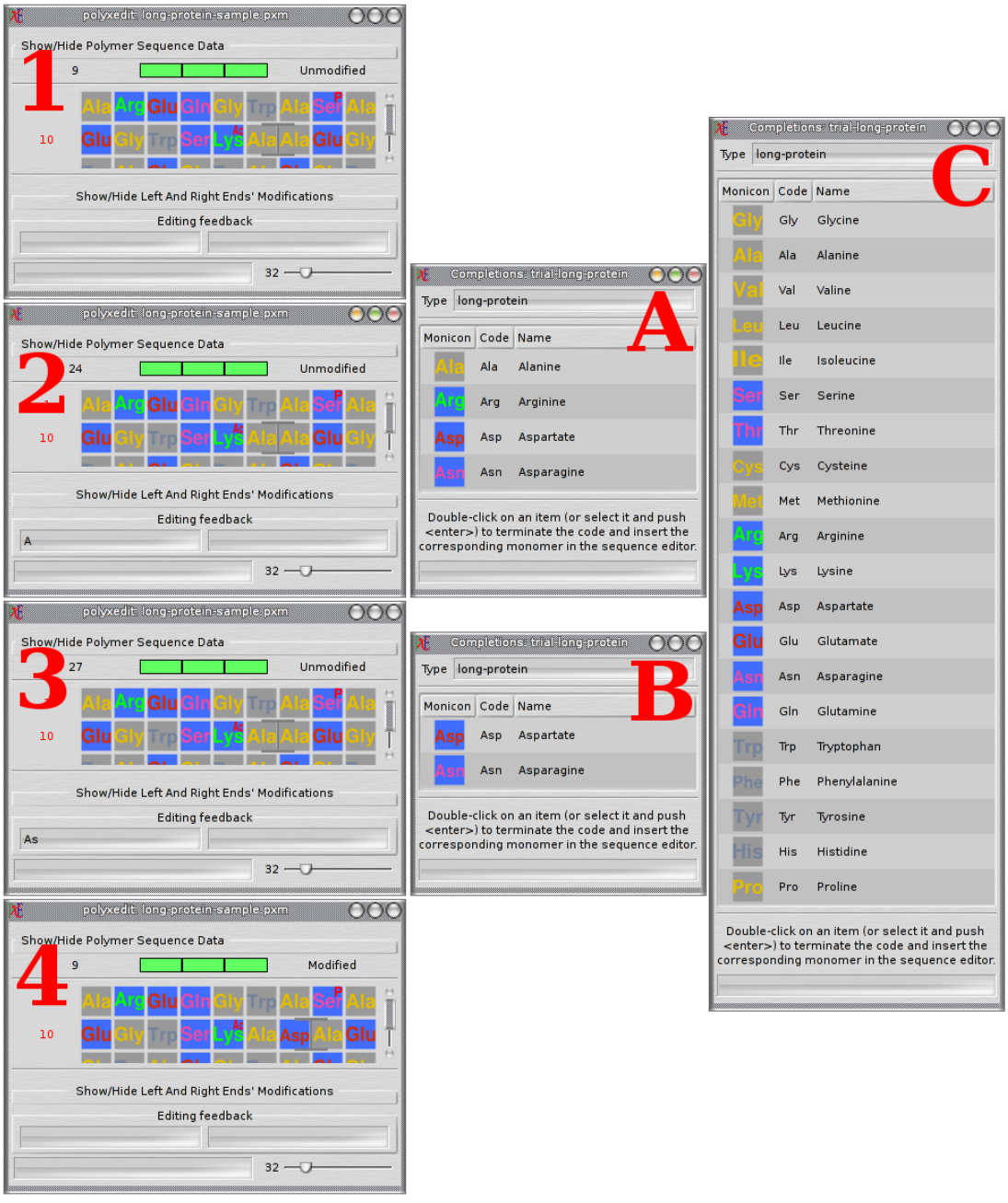

Figure 8.5: **Multi-character code sequence editing in polyxedit.** This figure shows the process by which it is made possible to edit polymer sequences with a code set that allows more than one character per code.

'A' character: **polyxedit** cannot figure out which monomer code the user actually means when keying-in [A].

There is a way, called *code completion*, to know which monomer code(s) —in the current polymer chemistry definition— do start with the keyed-in character(s) ('A' for us, now). The user can always enter the *code completion mode* by hitting the tabulation [TAB] key. This is what is shown in the panel A. We see that, in the current polymer chemistry definition, four monomer codes start with an 'A' character, and these are "Ala", "Arg", "Asp" and "Asn". We could be selecting the monomer of choice by double-clicking onto the proper list item, which would insert the corresponding monomer icon ("monicon") in the polymer sequence at the cursor location. But, since this is a manual, we are going through another step.

Let's continue editing the polymer sequence and key-in a [s] (we did not forget that we wanted to enter an "Asp" monomer code in the first place, did we?). The result is shown in panel 3. What we see here is that, this time also, nothing changed in the polymer sequence. What changed is that there is now a "As" character string in the left text widget under the label Editing Feedback. Let's key-in once more the [TAB] key, and we get the small window show in panel B. This time, only two items are listed: "Asp" and "Asn". This is easy to understand: there are only two monomer codes that start with the two letters 'A' and 's' ("As") that we have keyed-in so far. At this time, we either select one of the items (we wanted to enter the "Aspartate" monomer, so we'll double-click onto the first item of the list), or we just key-in a last character: [p]. At this point, the monomer is effectively inserted in the polymer sequence, as the "Asp" monomer left of the cursor, as shown in panel 4.

Unambiguous Single-/Multi-Character Monomer Codes

Let's imagine that we have a polymer chemistry definition that allows up to 3 characters for the definition of monomer codes, but that we have one monomer code (let's say the one for the "Glutamate" monomer) that is 'E'. This monomer code 'E' is the only one of the polymer chemistry definition that starts (and ends, since it is mono-character) with an 'E'. In this case, when we key-in [E], we'll observe that the monomer code is immediately validated and that its corresponding monomer icon is also immediately inserted in the polymer sequence. This is because, *if there is no ambiguity,* ***polyxedit*** *will immediately validate the code being edited.* This means that you are absolutely free to define *only single-character monomer codes* in your polymer chemistry definition, so that you are not even conscious that the powerful multi-character feature exists! Indeed, in this 1-character monomer code configuration, each time you'll key-in an uppercase character, you'll be inserting its corresponding monomer into the polymer sequence immediately.

Displaying All The Available Monomer Codes

Equally interesting is the fact that if you key-in the [TAB] key while no monomer code is being edited (that is: the left text widget under the label Editing Feedback is empty), all the monomer codes available in the polymer chemistry definition currently in use are displayed, exactly as shown in the panel C, Figure 8.5 on the preceding page.

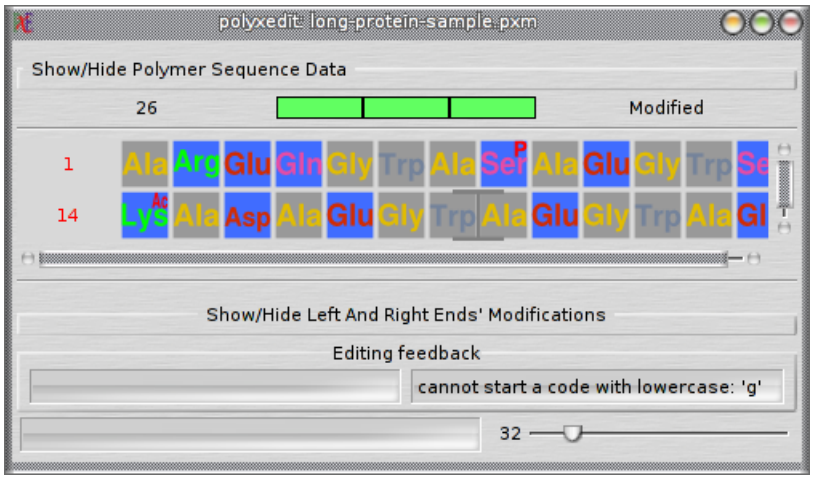

Figure 8.6: **Bad code character in polyxedit sequence editor.** This figure shows the feedback that the user is provided by the code editing engine, when a bad character code is keyed-in.

Erroneous Monomer Codes

Let's see now what happens when the user keys-in bad characters in the polymer sequence editor window. This is described in the Figure 8.6 on the following page. If the user enters a lowercase character as the first character of a monomer code, the program immediately complains in the right text widget under the label Editing Feedback. In this case, the monomer code is not put into the left text widget, which means it is simply ignored.

If the user starts keying-in valid monomer character codes, like for example we did earlier with "As", and that she wants to erase these characters because she changed her mind, she *must not* use the BACKSPACE key, because this key will erase the monomer left of the cursor point in the polymer sequence! The way that the user has to remove the characters currently displayed in the left text widget under the label Editing Feedback, is to key-in the Esc key once for each character. For example, let's say I've already keyed-in A and s. In this case the left text widget, under label Editing Feedback, displays these two characters: "As". Now, *I change my mind* and do not want to enter the "Asp" monomer code anymore. I want to enter the "Gly" code. All I have to do is key-in the Esc key once for the 's' character (which disappears) and once more to remove the remaining 'A' character which disappears also. At this point I can start fresh with the "Gly" monomer code by keying-in sequentially G, l and finally y.

Clipboard-Importing Of Sequences

Very often, the user will make a sequence search on the web and be provided with a polymer sequence that is crippled with non-code characters. The user typically selects all the text provided by the remote site, pastes that sequence in the **GNU polyxmass** polymer sequence editor window and finally encounters invalid codes in it. It might be uncomfortable to have to trigger —prior to pasting a correct sequence in **polyxedit**— a text editor only to "purify"

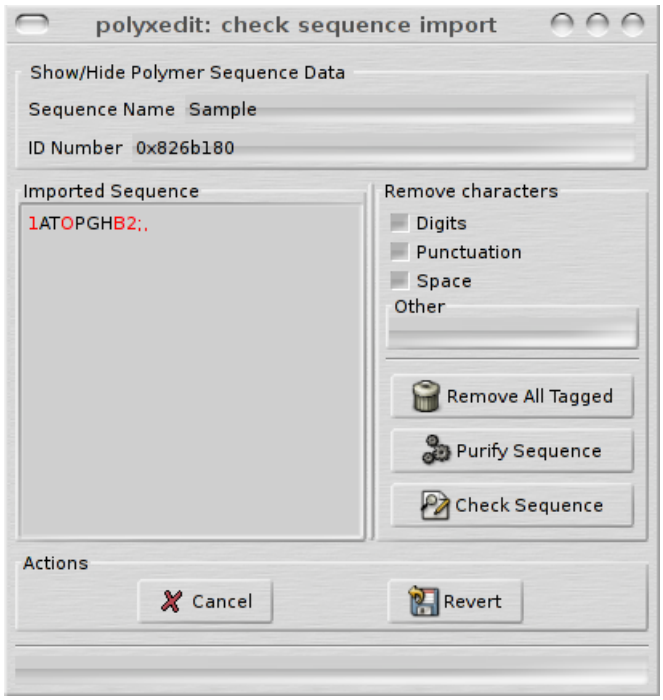

Figure 8.7: **Clipboard-imported sequence error-checking.** If a sequence that is imported through the clipboard to the **polyxedit** sequence editor contains invalid characters, the user is provided with a facility to "purify" the sequence. This facility is provided to the user through the window depicted in this figure.

that sequence...

GNU polyxmass provides a convenient way to spot non-valid characters from a polymer sequence and to let the user "purify" the imported sequence. A clipboard-imported sequence is systematically parsed. When invalid characters are found the window depicted in Figure 8.7 on the next page is presented to the user for her to make appropriate adjustments. The sequence is presented to the user in a textview widget (Imported Sequence) with the improper characters tagged in red color. The rationale for tagging characters in red colour is by comparing the imported sequence with the monomer codes available in the current polymer chemistry definition. As soon as a character does not correspond to any valid monomer code, it is tagged in red. At that point, if the user clicks onto the Remove All Tagged button, all the red-tagged characters will be automatically removed.

Also, the user is provided with an automatic "purification" procedure whereby it is possible to remove one or more classes of characters from the imported sequence (Remove Characters frame widget). Checking one or more of the Digits or Punctuation or Space checkbuttons, or even entering other user-specified characters in the Other text entry widget, will elicit their removal from the imported sequence after the user clicks the Purify Sequence button.

When the user is confident that almost all the erroneous characters have been removed, she can click the Check Sequence button, which will trigger a "re-reading" of the sequence in the Imported Sequence textview widget. If erroneous characters are still found, they are presented to the user in red color.

Note that, for maximum flexibility, the user is allowed an immediate and direct editing of the imported sequence in the textview widget (that is, the textview widget is *not* read-only).

Once the sequence if finally depured from all the invalid characters, the user can select it in the textview on the left of the window and can paste it in the **polyxedit** sequence editor. This time, the paste operation will be error-free.

Sequence Selections: The Various X Mechanisms

As any text editor, the **polyxedit** polymer sequence editor can perform the usual clipboard operations. In the `X window` world, there is another process to copy text and paste it into another place: the `X window` primary selection mechanism. That process is easy: text is first selected (either using the keyboard or the mouse; that makes the *selection*), and when that selected text needs to be pasted, the user just clicks the mouse's middle button at the destination location. The copy/cut/paste process, much usual in the *MS Windows* system, is implemented also. Thus, the users of **polyxedit** get the best features of selection and pasting.

When the user tries to paste a sequence element from the clipboard (say, after copying it from a web browser), the program checks that sequence very thoroughly. If an invalid character is found, the whole process is stopped with a message logged to the console; the sequence is not modified in any way and the user may verify that sequence so that she removes the invalid characters or codes.

When the users copies/cuts a sequence from the **polyxedit** sequence editor window to the clipboard, what is actually copied in the clipboard is a text string that is made with all the monomer codes of the polymer sequence that was selected the copying/cutting operation was performed.

Visual Feedback In The Editor

The polymer sequence editor provides a number of widgets to inform in real time the user about what is going on in it. These widgets are briefly reviewed below, and the user is invited to look at Figure 8.8:

- The Un/Modified label informs the user if the polymer sequence was modified or not since it was either last written to a file on disk or last read from a file;
- The monomer status flag (here it is red-green-red) is supposed to inform the user about the status of the monomer onto which the mouse cursor is positioned (in the image example, that is monomer 'S', at position 22). The flag is interpreted in the following manner:
  - The first flag element (red in the example) tells if the monomer contains properties. That is a flag about the internal status of the monomer. This flag is mainly interesting to the power user who goes in the source code and modifies it to adapt it to her specific needs. Red means that the monomer has at least one "prop" object in it. Green means that it has no such "prop" in it. If this flag element is green, then the two remaining flags are necessarily green. This is because the two other flag elements tell the presence or the absence of monomer characteristics that are subsets of the "prop" object;

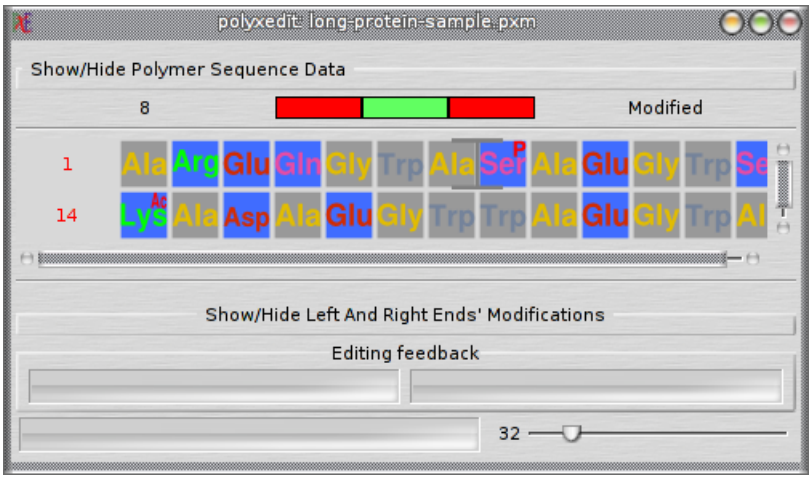

Figure 8.8: **Visual feedback in the polyxedit sequence editor.** This figure shows the feedback that the user is provided when moving the mouse cursor over monomer icons. See the text for details.

- ✦ The second flag element (green in our example) tells if the monomer has been *annotated* at least once. The green color indicates that no note is found in the monomer. That flag would be red if the monomer had been annotated at least once;
- ✦ The third flag element (red in our example) tells if the monomer has undergone a *chemical modification*. In our example that flag is red, because as the reader can see, the 'S' monomer at position 22 is indeed modified: it is a phosphorylated seryl residue! If the monomer had not been modified, then that flag element would have been green.

* The label that is located left of the monomer status flag (it indicates **8** on the figure) tells the sequence position of the monomer onto which the cursor is positioned at any given time[1].

Sequence Annotation: The Various Mechanisms

The annotation of polymer sequences is very often required in projects for which a number of scientist-made observations are to be "connected" in a time-lasting manner either to a polymer sequence (as a whole object *per se*) or to any monomer in a polymer sequence.

polyxedit allows the annotation of the whole polymer and/or of any (and any number) of monomers in the polymer sequence. There is no limitation on the number of notes that can be set to the polymer or any given monomer. Further, the user is provided with two mechanisms by which she can set notes to monomers (annotate): *single-mode* monomer annotation and *range-mode* monomer annotation. All these polymer/monomer note-setting processes are described in detail below.

[1] The cursor is not visible because the screen dump function in `The Gimp` removes it to clean the image.

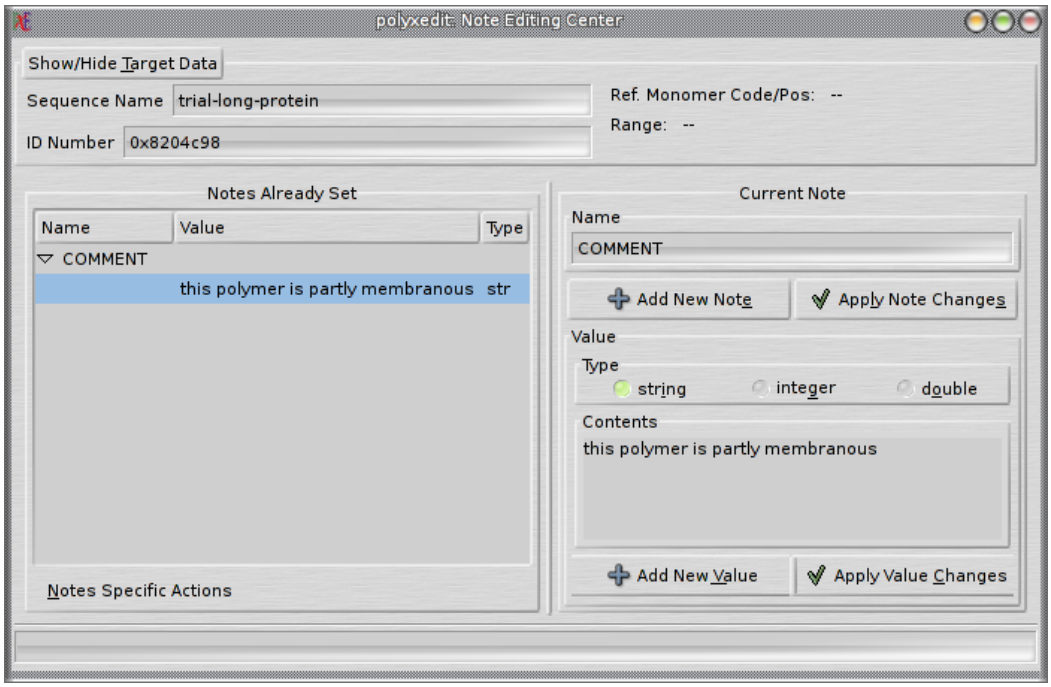

Figure 8.9: **Annotating polymer sequences.** This figure shows the graphical interface to the annotation of polymer sequences.

First, however, I should tell you, respected reader, that a note is basically an envelope that contains a number of elements:

- A textual element that is the *name of the note*;
- Any number of paired data, called *noteval* objects (like "note value"). A noteval is made of two data:
  - A datum describing the type of the noteval: either *string*, or *integer*, or *double*;
  - The contents of the noteval object.

The notes are stored in the polymer sequence file and are easily managed graphically, as we'll describe now.

Managing Polymer Notes

The user may set/modify/remove polymer notes using the following contextual menu:

Edit⟶*Annotation*⟶*Polymer*

The Figure 8.9 shows the window that pops up to let the user perform a number of note-related actions that are rather self-explanatory.

A note that is set to a polymer sequence is set to that sequence as a whole, and not to any specific monomer or monomer range. If all the monomers in the annotated polymer sequence were removed, that (empty) polymer sequence would still bear the annotation. In order to add notes, the user must first fill-in the note Name field. Once this field is filled,

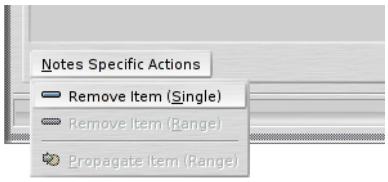

Figure 8.10: **The menu governing actions on note items.** This figure shows the menu that the user may use in order to remove any item currently selected in the treeview. When the window is opened in single-mode, the range-mode actions are inactive.

the user clicks the Add New Note button. The note name will be listed in the Name column of the Notes Already Set treeview.

It is only once a note (name) has been added, as described above, that the user can add notevalue objects to that note. Remember, we said earlier that a note was made of a name and of any number of [value type+value contents] noteval pairs. The note, or one of its noteval objects, has to be selected in the treeview on the left hand side of the window, so that the user can add a noteval object, by:

* Choosing the type of the value (string, integer or double) by selecting the radiobutton of choice in the Type widget;

* Entering the data proper in the Contents textview widget;

* Clicking onto the Add New Value button.

Accomplishing the tasks above will create a new subitem in the treeview: a new noteval object will be listed under the node corresponding to the note name under which a new noteval [type-contents] pair has been defined.

It is possible to change the note name of a note that is selected in the treeview or to change the type or contents of a noteval object that is currently selected in the treeview. Most intuitively, these changes are done by editing the data in their respective widgets, and then clicking either Apply Note Changes or Apply Value Changes.

It is also possible to remove any item that is currently selected in the treeview. The menu entitled Notes Specific Actions will popup when clicked, to show the menu items shown on the Figure 8.10.

Setting notes to the polymer sequence as a whole is conceptually simpler than what we are about to visit: the annotation of any monomer in either single-mode or range-mode.

Managing Monomer Notes

As stated earlier, monomer notes can be set in two distinct modes: *single-mode* and *range-mode*. Setting notes to a monomer is as easy as setting notes to a polymer sequence. However, before starting doing any annotation work, it should be understood what kind of note is appropriate for the specific annotation task. Let's first see the simplest mode of monomer annotation: *single-mode*.

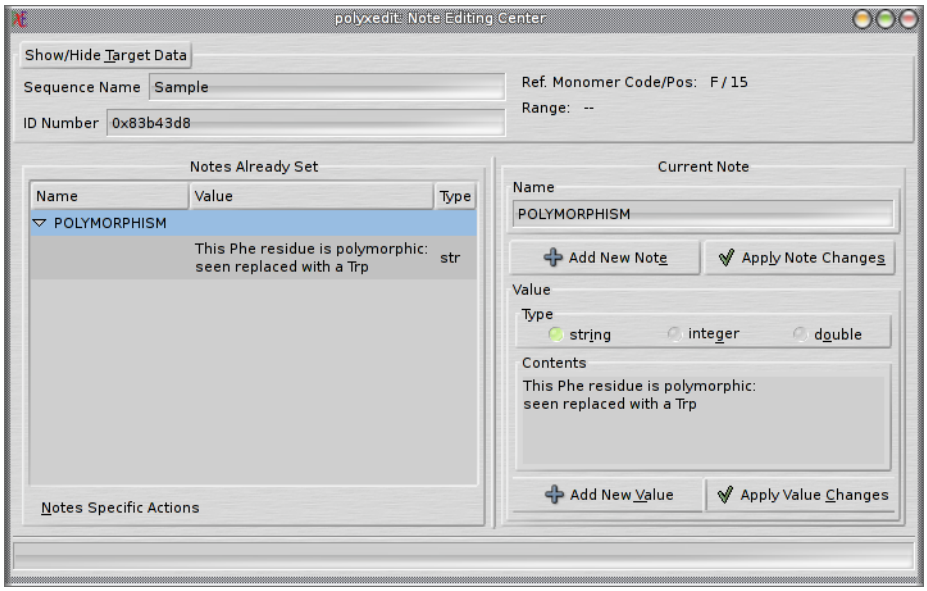

Figure 8.11: **Annotating monomers in single-mode.** This figure shows the graphical interface to the annotation of monomers in single-mode.

Managing Monomer Notes In Single-Mode

If the annotation pertains to a single monomer in the sequence,[2] the user should hit the corresponding monomer icon with the mouse and right-click onto it so that the following menu item can be selected out of the contextual menu that pops up:

Edit⟶*Annotation*⟶*Monomer*⟶*Single*

The precise mouse-clicking of that specific monomer icon will trigger internal calculations that will lead to the proper initialization of the popped up window, as shown in Figure 8.11, where the Ref. Monomer Code/Pos. label indicates **F/15**. That example means that the user wanted to annotate a phenylalanine residue located at position 15 of the polymer (protein) sequence. Note, by the way, that the Range label indicates no specific value (- -). We'll see later that this bit of information is useful in other cases.

Once the window shown in that example is displayed, the managing of monomer notes is identical to the managing of polymer notes (as was previously described).

Managing Monomer Notes In Range-Mode

Sometimes it is desirable to be able to set an identical note to a range of consecutive monomers. For example, one user might want to set to a range of residues in a protein a note (with a name *"TRYPSIN"* and a number of notevalue objects describing scientific observations (either text or numerical) and interrogations, for example). That note will be set in each monomer of the range of monomers. Once the range-mode annotation has been performed, each note in each monomer will behave exactly the same way as notes set using the *single-mode* annotation procedures. See Figure 8.12 on the next page for a good example of such note.

[2] Like indicating that this specific residue is polymorphic, for example, or entering any kind of comment.

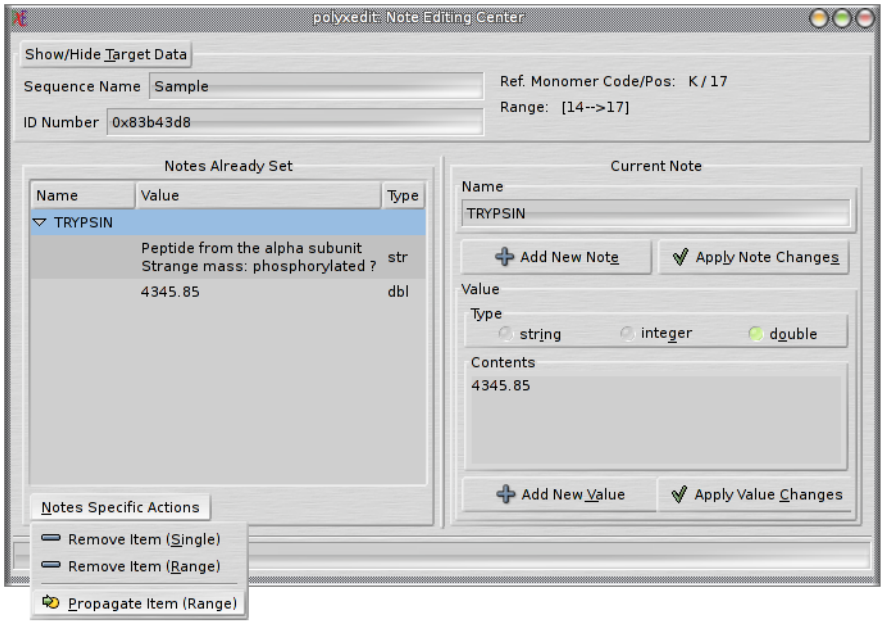

Figure 8.12: **Annotating monomers in range-mode.** This figure shows the graphical interface to the annotation of monomers in range-mode.

So, how are range-mode annotations actually carried out by the program? The very first thing is to select –in the polymer sequence editor– the range of monomers to be annotated. Once that range of monomers is effectively selected, the user can mouse-click with the right button one specific monomer, in that range of selected monomers. In order to elicit the displaying of a window like the one represented in Figure 8.12 on the facing page, the user must select the following menu item from the contextual menu:

Edit⟶*Annotation*⟶*Monomer*⟶*Range*

As can be seen on that figure, this time the Range label gives an indication in the form **[xx->yy]**. This means that the user wanted to edit a note for all the monomers comprised in that range (from position **xx** to position **yy**). That makes a range-mode annotation action that is taken on three monomers.

One interesting question is: —*"Given the fact that the user is performing a range-mode annotation, to which monomer do belong the notes shown in the Notes Already Set list on the left hand side of the window?"* That's undoubtedly a good question. The answer is that the notes that are listed there belong to the *reference monomer*, that is the monomer that was actually pointed while right-clicking the sequence (to elicit the popping up of the contextual menu). This *reference monomer* is very important, as we'll see in a moment.

The Figure 8.12 shows that range-mode annotations are performed much like monomer single annotations or polymer annotations (same window, in fact, with same widgets). The big difference comes with the notes menu, that lists menu items that are specific to the *range-mode* actions (Figure 8.12):

- The menu item ⟶*Remove Item (Range)* will remove the selected item (note item) from all the monomers in the range;
- The menu item ⟶*Propagate Item (Range)* will make a copy of a newly created note

into all the other monomers in the range. Note that, if a note by the same name exists already in any of the monomers in the range, the note is not added to it. The user will be informed by a dialog window that a given monomer was skipped.

Note that the single-mode menu item (Remove Item (Single) will perform the action, when in range-mode, on the reference monomer, that is the one that was right-clicked upon when the note editing process was triggered (see above for the definition of the *reference monomer*.

> It is important to grasp that in the range-mode annotations, when an action cannot be performed in one of the monomers in the selected range of monomers, then this does not prevent the process from trying to accomplish the task on the other monomers of the range. For example, the user selects a stretch of twenty monomoners in a polymer sequence, and then elicits a range-mode annotation process (namely the addition of a note) onto these twenty monomers. Let's say that the to-be-added note is identical to a note present in the fifth monomer of the monomer range. The note addition –for this monomer– is going to fail. That does not mean that the whole process is stopped: if the to-be-added note is not found identical in any other monomer, it is going to be successfully added into all the remaining monomers. In other words, one failure does not abort the whole range-mode annotation process.

Without bothering the reader with more descriptions, I would suggest that she experiments with the features described here. The design has been conceived as the most flexible possible. Notheworthy is that flexibility sometimes goes with risky programmatic behaviours: the user must know what she does when clicking onto a button! The Save As menu item is your friend *before* experimenting that annotation feature.

Chemically Modifying Polymer Sequences

It very much often happens that the (bio) chemist uses chemical reactions to modify the polymer sequence she is working on. Mass spectrometry is then often used to check if the reaction proceeded properly or not. Further, in nature, chemical modifications of biopolymer sequences are very often encountered. For example, protein sequences get often modified as a means to regulate their function (phophorylations, namely). Nucleic acid sequences are very often and extensively modified with modifications such as methylation...

It is thus crucial that **GNU polyxmass** be able to model with high precision and flexibility the various chemical reactions that can be either made in the chemistry lab or found in nature. The **GNU polyxmass** program provides two different chemical modification processes:

* A process by which monomers in the polymer sequence can be individually modified;

* A process by which the whole polymer sequence can be modified, either on its left end or on its right end or even on both ends.

We shall review these two processes separately in the two sections below.

Chemical Modification Of Monomers

Modification Of Monomers

There are a number of manners in which monomers can be modified in a polymer sequence. The Figure 8.13 on page 94 shows the simplest manner: the user first selects the monomer icon to modify, next calls the *Chemistry*⟶*Modifications*⟶*Monomer* menu and –as a result– is provided with a window where all the modifications currently available in the polymer chemistry definition are listed. Since a monomer icon was initially selected in the editor window, the Selected Monomer target radiobutton is on by default. It is then simply a matter of choosing the right modification from the Available Modifications list and clicking onto the Modify button.

The modified seryl residue is shown in the polymer sequence editor window: a transparent graphics object (a red 'P') was overlaid onto the corresponding seryl monicon.

While the Modification Target(s) frame widget contains radiobuttons the signification of which is rather easy to understand, we want to detail one of these: the Specific Monomer Locations frame. If the user selects the radiobutton inside that specific frame (labelled Positions Should Be Separated With ';'), she also has to write the locations in the text entry widget below it. This text entry widget receives textual strings that should describe what locations on the polymer sequence should be modified. The syntax of the descriptive string allows logical positions to be indicated. The user is invited to experiment, maybe using variations on the themes described below as examples:

- **ALL** That would mean that the currently selected modification in the Available modifications list is to be applied to all the monomers in the polymer sequence. This is equal to selecting the radiobutton labelled All Monomers;
- **EVEN** or **even** This will modify all monomers at even positions: 2, 4, 6...
- **ODD** or **odd** This will modify all monomers at odd positions: 1, 3, 5...
- **EVEN;ODD** is identical to **ALL**;
- **[1-10];[20-30,odd]** This will modify all the monomers from position 1 to position 10 inclusive, and all the odd-positioned monomers between position 20 and position 30 inclusive;

The user is responsible for correctly reading the results that are published in the paned textview lying between the upper pane (labelled Monomer Modification Rules) and the two buttons at the bottom of the window. Further, when a modification or un-modification is performed, the count of successful events and of failed events is displayed in the messages' text widget at the very bottom of the window. The messages that are displayed in this widget are not permanent, they last some seconds and disappear. Care should be taken at what is displayed in this messages' text widget.

> Attention should be paid to the fact that the user is responsible for applying chemical modifications to monomers that are listed as modifiable with the modification used. For example, if a phosphorylation modification is applied to a monomer that is not listed as phosphorylatable in the relevant configuration file, then the modification is applied to it (which means that –internally– the monomer is modified) but its corresponding monicon is not graphically changed because no graphical rule is associated with the phosphorylation of this monomer (see section 9 on page 132, the file of interest is `monicons.dic`).

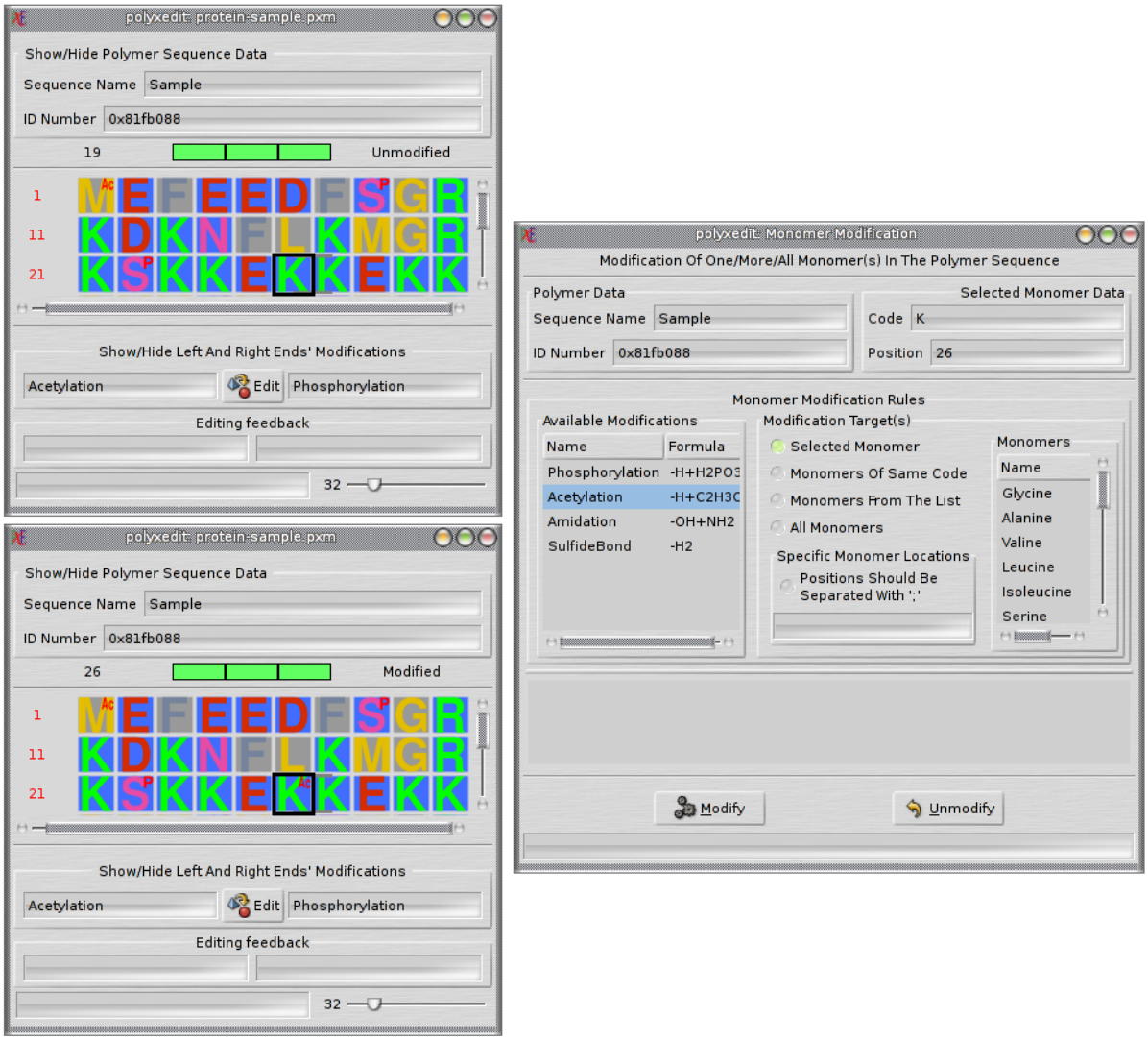

Figure 8.13: **Modification of a monomer in a polymer sequence.** This figure shows the graphical rendering of a phosphorylation of a seryl residue in a protein polymer sequence.

It is important to understand that, when a monomer is modified, its previous modification (if any) is overwritten with the new one. The user is invited to experiment a bit with the monomer modification process, so as to be confident of the results that she is going to obtain when real polymer chemistry work is to be modelled in **GNU polyxmass**.

Un-Modification Of Monomers

If a monomer is modified, then it also should be possible to revert the chemical reaction: to un-modify it. There is, however, a subtlety here, that we ought to put into the limelight: an example will do.

Let's say that all the seryl residues of our protein polymer sequence are phosphorylated.[3] Only seryl residues are phosphorylated in this polymer sequence. We thus see all their corresponding monicons overlaid with a small 'P' on them (see the example above). Other monomers are acetylated, like lysyl residues, for example. What we want to do is un-modify all the phosphorylated seryl monomers in one go. We thus open the monomer modification window, select the monomer code corresponding to the seryl residue in the Monomers list, select the rabiobutton labelled Monomers From The List, we select "Phosphorylation" in the Available Modifications list and finally we click the Unmodify button. All the seryl residues currently phosphorylated are un-modified. This is OK.

Now, let's assume that we had not selected "Phosphorylation" in the list of available modifications, but "Acetylation", for example: no phosphorylated seryl residue would have been un-modified. This is a foolproof feature: if you select a modification name from the list of available modifications, and next click onto the Unmodify button, that means that your un-modifying action has –as targets– monomers that are currently modified with the modification that you selected.

That means that if, in our example, you had selected, as monomer targets to the un-modification, the All Monomers radiobutton, selected the "Phosphorylation" modification and clicked onto the Unmodify button, *only* the phosphorylated monomers[4] would have been un-modified.

Now, if you un-select all the items in the list of available modifications[5], that you select the All Monomers radiobutton and next click onto the Unmodify button, then you'll un-modify absolutely *all* the monomers, because you are not restricting the monomer targets neither by their code, neither by the identity of their potential modification.

The user is encouraged to play with these features... Also of great importance is to understand that the modifications that can be set to the monomers do disappear when the monomer is removed from the polymer sequence. These modifications are *monomer modifications*, they belong to the monomer that is modified. We say that these modifications are *intrinsic*.

Chemical Modification Of The Polymer Sequence

We have seen above that it is possible to modify any monomer in the polymer sequence and that when the modified monomer is removed, the modification associated to it disappears also.

[3] That's protein chemistry stuff.

[4] Whatever they be, because the All Monomers radiobutton was selected.

[5] You may need to maintain the Ctrl key pressed while clicking onto the currently selected item to unselect it.

The modifications that we describe here are not of this kind. They can be applied to either the left end of the polymer sequence or its right end. But these modifications do belong to the polymer sequence *per se* and are not removed from it even if the polymer sequence is edited by removing the left end monomer or the right end monomer. We say that these *polymer modifications* are *permanent*.

The way in which a polymer sequence is modified using *polymer modifications* is much easier than the previous *monomer modifications* case. The modification window is opened by choosing the *Chemistry⟶Modifications⟶Polymer* menu or the Edit button below the polymer sequence rendering area. The Figure 8.14 on the preceding page shows that window.

The modification is absolutely easy to perform, with a clear feedback provided to the user (by listing the permanent modifications in two convenient text widgets located under the polymer sequence graphical rendering area, under label Left and Right Ends' Modifications). In the example (Figure 8.14 on the facing page), the top polymer sequence is not yet modified. By using the window on the right, the polymer sequence is modified on its left end using the "Acetylation" modification. The newly modified polymer sequence is shown in the window below, with the left text widget displaying the name of the left end modification.

The Unmodify button is responsible for the un-modification of the selected polymer sequence end (left/right), so that reverting a modification is perfectly feasible.

Finding and Replacing Sequence Motifs

It is very much often the case that one wants to find a given sequence motif quickly. **GNU polyxmass** allows this easily by selecting in the contextual menu the following menu item:

Edit⟶Find Replace

Using that menu item will provide an options window, as described in Figure 8.15 on the next page.

What is interesting with this Figure 8.15 on the following page is that it shows how flexible the functionality is: the user has two sequence editor widgets at hand. The left one Find Motif is where the motif to find should be entered. The right one Replace Motif is where the motif to be used in order to replace the found motif is edited. As visible on the right hand widget, the monomers entered in these two widgets might be modified (by chemical modification) or annotated (by monomer annotation) exactly in the same way as the user is used to do in the polymer sequence editor. The sequence editor widgets in Figure 8.15 on the next page are actually *the same* as the ones that are located in the polymer sequence editor windows.

Let us see some of the available options:

* Start At Point The find operation will not start from the very first monomer in the polymer sequence, but at the position where the cursor is located (*the point*);

* Backward Normally, the find operation is performed downstream of the current location; thus the next found motif will necessarily occur at positions in the polymer sequence greater than the current. With this option, however, it is possible to reverse the direction of the search. Backward instructs the search engine to look for motifs in the upstream sequence with respect to the current location ; thus any found motif will be at a position lesser than the current position;

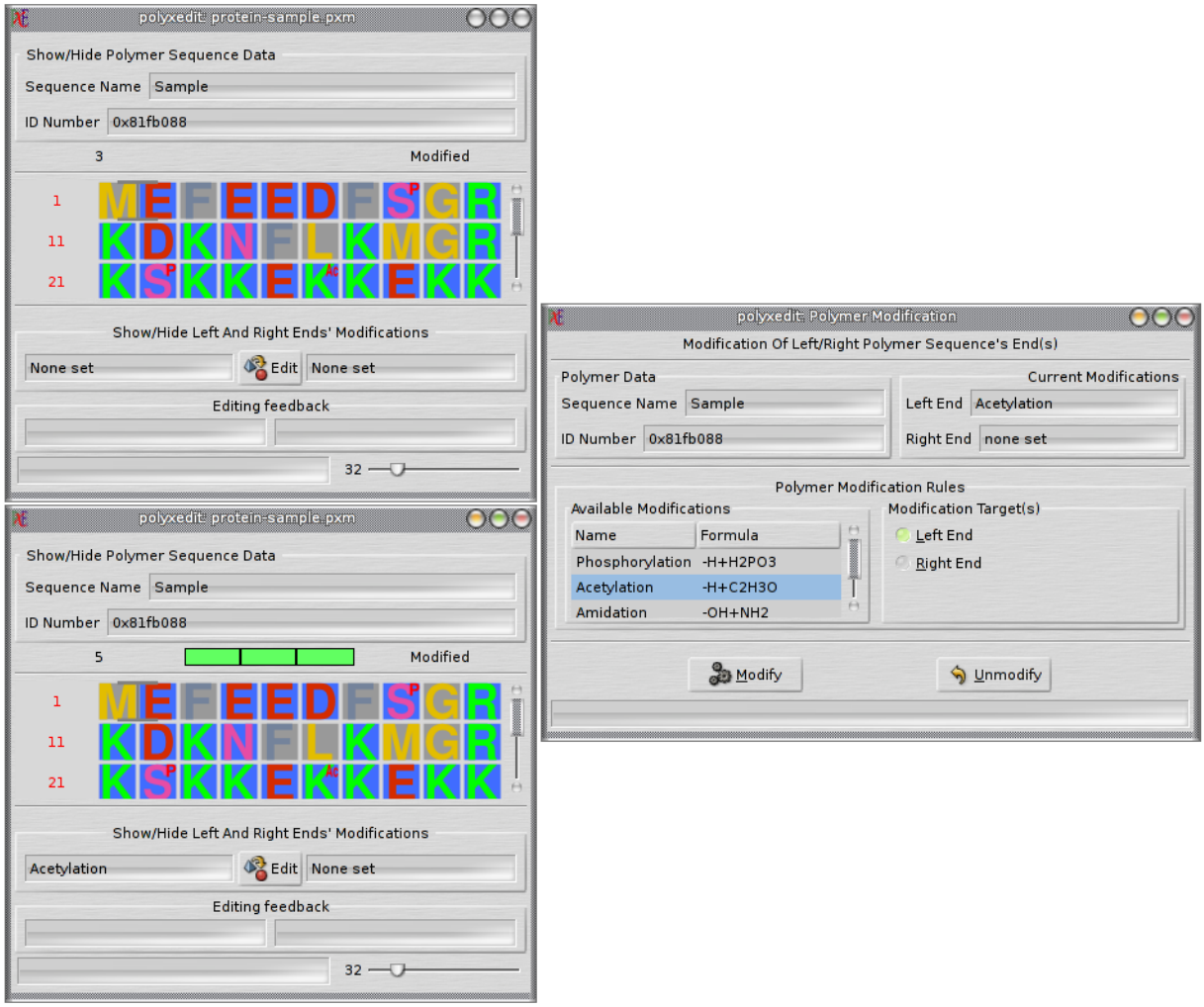

Figure 8.14: **Modification of the left end of a polymer sequence** This figure shows how simple it is to permanently modify a polymer sequence on either or both its left/right ends. The permanent modifications currently set to a polymer sequence are conveniently listed in two text widgets located under the polymer sequence rendering area.

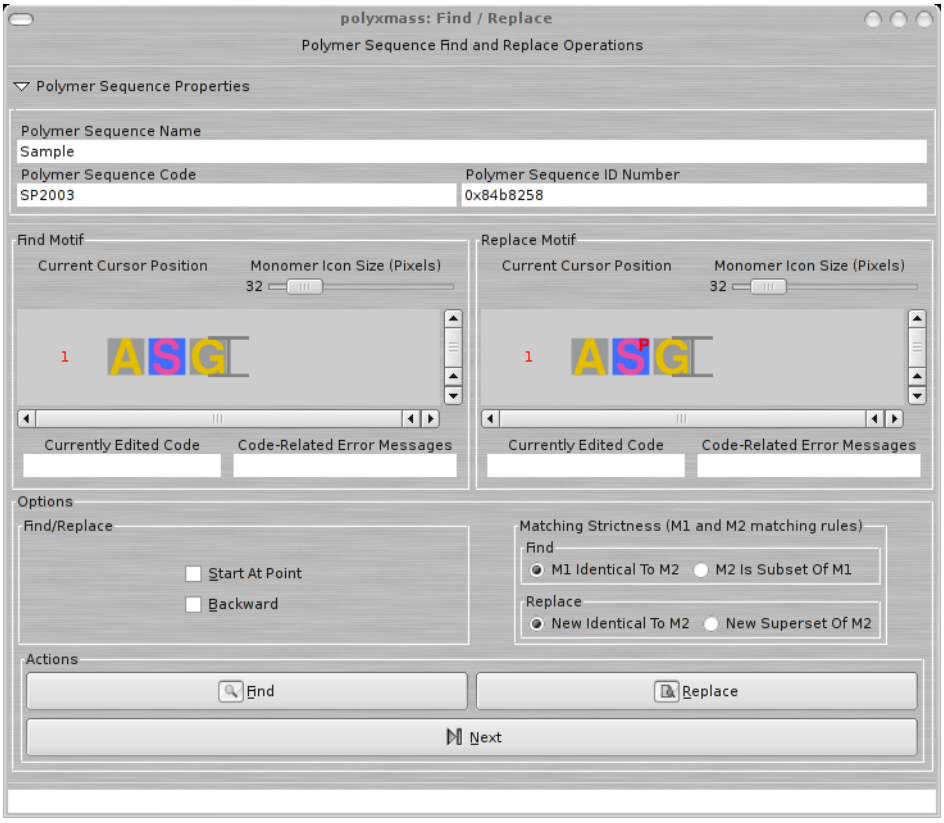

Figure 8.15: **Find/Replace options window.** This figure shows the window with which the user is provided when she performs a polymer sequence find/replace operation. The two sequence editing regions are full blown sequence editor widgets in which the user edits sequence motifs exactly the same way she edits a sequence in the polymer sequence editor. This allows for flexible find/replace operations.

- Matching Strictness (M1 and M2 matching rules) These matching rules will govern the way monomers in the polymer sequence are considered as matching the monomers in the Find Motif motif sequence or how stringent the replacement using Replace Motif should be:
  - The Find matching rules:
    - M1 Identical To M2: M1 is a given monomer in the polymer sequence and M2 is a monomer in the Find Motif motif sequence; both monomer are being compared, and will be considered to actually match only if both are absolutely *identical*;
    - M2 Is Subset of M1: M1 is a given monomer in the polymer sequence and M2 is a monomer in the Find Motif motif sequence; both monomer are being compared, and will be considered to actually match if all the modification and/or note(s) present in M2 are found in M1, *even if* M1 might contain other modification and/or note(s);
  - The Replace matching rules:
    - New Identical To M2: New is the monomer that will be in the polymer sequence after the replacement is performed and M2 is the monomer from the Replace Motif sequence that was used to guide the replacement process; the new monomer will be identical to M2;
    - New Superset Of M2: New is the monomer that will be in the polymer sequence after the replacement is performed and M2 is the monomer from the Replace Motif sequence that was used to guide the replacement process; upon replacement all the modification and/or notes from M2 will be present in the New monomer, but if the original monomer in polymer sequence had modification and/or notes not present in M2, then these will be retained; thus, New will be a superset of M2;

It is obvious that the Replace Motif sequence might be empty when performing Find or Replace operations.

The way Replace operations are performed is sequential: first the user clicks onto the Find button. If a sequence element is found to match the Find Motif sequence it is selected in the polymer sequence editor window. At this time the user might click onto the Replace button. Once the replacement is performed, the search engine is automatically asked to find a new occurrence of the Find Motif sequence, and so on...

The user is invited to experiment with the series of options described above as these render the operations rather flexible.

Cleavage Of Polymer Sequences

It happens very often that polymer sequences get cleaved in a sequence-specific manner. These specific cleavages do occur very often in nature, and are made by enzymes that do cleave biopolymer sequences, like the glycosidases (cleaving saccharides), the proteases (cleaving proteins) or the nucleases (cleaving nucleic acids). But the scientist also uses purified enzymes to perform such cleavages in the test tube. **GNU polyxmass** must be able to perform thoses cleavages *in silico*. Let's see how a polymer sequence can be cleaved using **GNU polyxmass**.

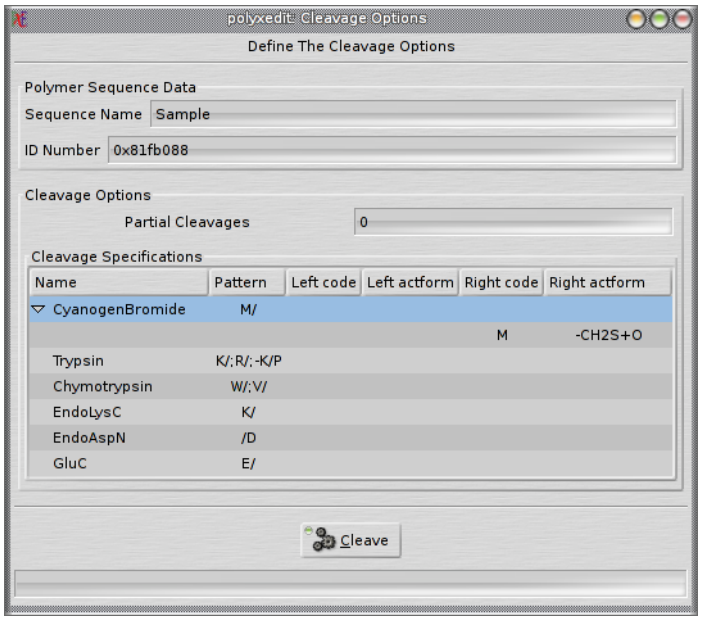

Figure 8.16: **Cleavage options window.** This figure shows the window with which the user is provided when she performs a polymer sequence cleavage. The user can select one cleavage specification and specify what level of partial cleavage the chemical cleavage should perform.

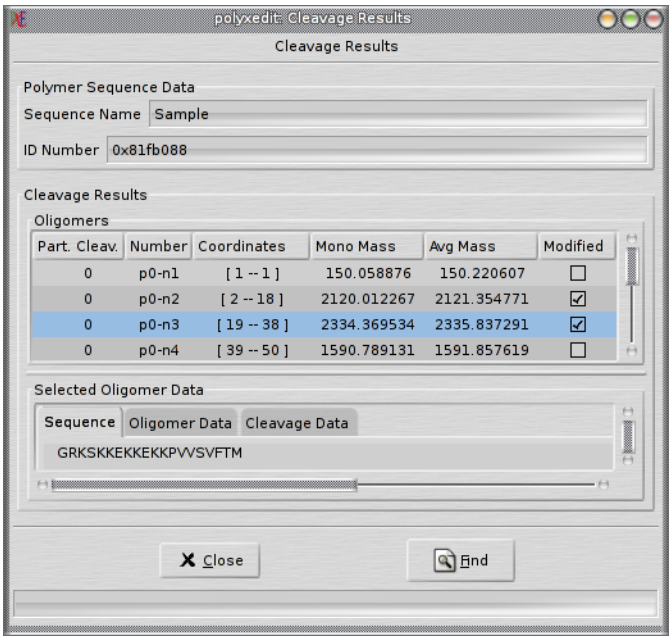

Figure 8.17: **Cleavage-generated oligomers window.** This figure shows the window that is opened so that the oligomers generated upon cleavage of a polymer sequence can be displayed. Other data are also displayed (see text for details).

It is a matter of having a polymer sequence opened in an editor window and selecting the *Chemistry*$\longrightarrow$*Cleave* menu. The user is provided with a window where a number of cleavage specifications are listed (Figure 8.16 on the following page). These cleavage specifications are listed by looking into the polymer chemistry definition corresponding to the polymer sequence to be cleaved. The program knows, for example, that the polymer sequence to be cleaved is of the "protein" chemistry type, and thus will list all the cleavage specifications that were defined in the "protein" polymer chemistry definition. The cleavage specifications are available for the user to select one of them to perform the cleavage.

The user selects the cleavage specification of interest and also sets the number of partial cleavages that the cleaving agent may yield. In our example, 2 was entered, which means that the cleavage reaction will yield the set of oligomers corresponding to a total cleavage (no missed cleavages=partial cleavages 0) along with the set of oligomers corresponding to 1 missed cleavage and to 2 missed cleavages. The calculating process is extremely rapid, so the user may enter rather high values here.

Upon successful termination of the cleavage reaction, the user is provided with a new window (Figure 8.17) in which all the oligomers that were generated are listed (upper pane). The listview widget on the upper pane sports a number of columns. Each row of this listview widget describes the properties of a single oligomer. The different columns are detailed below:

* Part. Cleav. This is the missed cleavage level for which the oligomer was generated;
* Number This is the number of the oligomer, so that the user may refer to it simply. The syntax is simple: px-ny means that this oligomer is the oligomer number y from the set of oligomers obtained in the x-missed cleavages series;

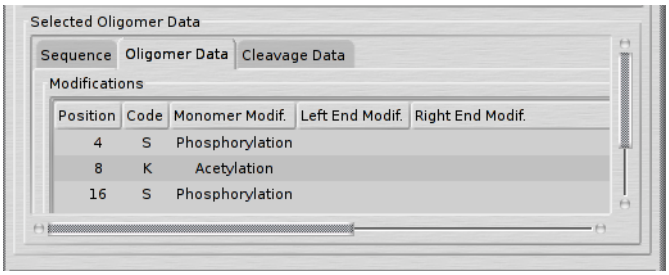

Figure 8.18: **Cleavage-generated oligomers' data.** This figure shows the notebook tab in which data pertaining to a selected oligomer are displayed. In particular, this tab contains a listview where monomer modifications of the selected oligomer (if any) are displayed.

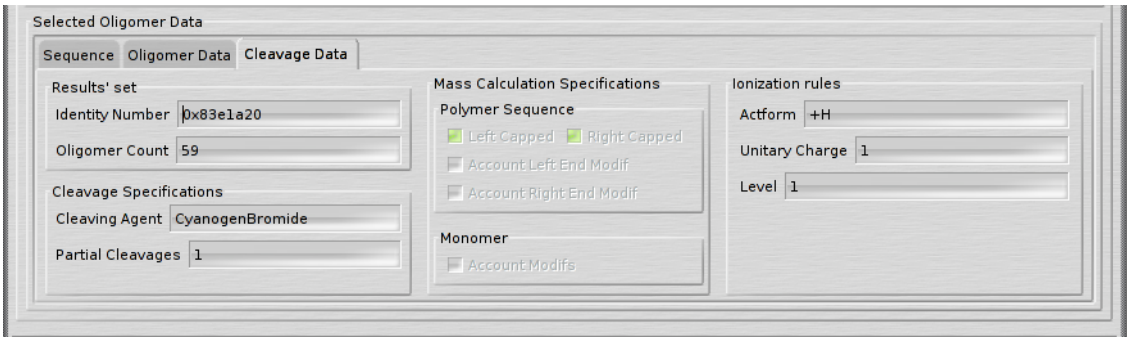

Figure 8.19: **Cleavage specification data.** This figure shows the notebook tab in which data pertaining to the cleavage operation are displayed.

* Coordinates These are the coordinates of the oligomer as it is occurring in the polymer sequence that was cleaved in the first place. For example, "[19-38]" would mean that the oligomer starts at position 19 and ends at position 38 of the polymer sequence, both values being inclusive;

* Mono Mass This is the monoisotopic mass of the oligomer, computed using the options that are set in the Calculation Options window (see above);

* Avg Mass Same as above, but for the average mass;

* Modified Indicates if the oligomer contains an intrinsically-modified monomer (it does not mean that the modification's mass was taken into account, it simply says that at least one monomer is modified in the oligomer. See below for details).

The lower pane of the Cleavage Results window contains a number of additional data, displayed in a set of pages belonging to the Selected Oligomer Data notebook widget:

* Sequence (Figure 8.17 on the preceding page) This is the sequence that is displayed when an oligomer is selected in the listview displaying the oligomers (in the upper pane);

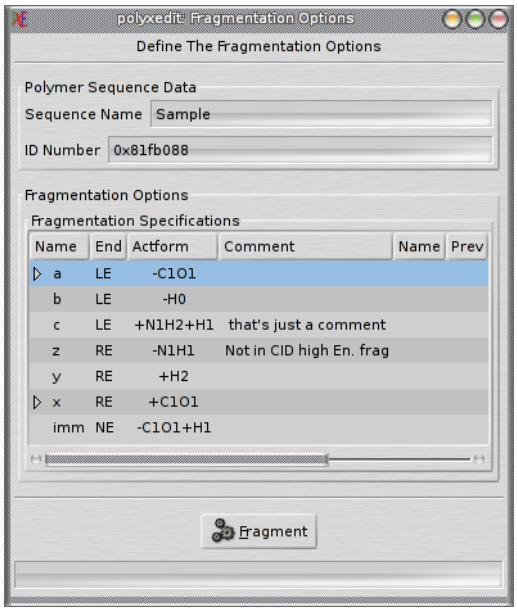

Figure 8.20: **Fragmentation options window.** This figure shows the window with which the user is provided when she performs a polymer sequence fragmentation. The user can select one or more fragmentation specifications (patterns).

* Oligomer Data (Figure 8.18) This is the place where monomer modifications are listed as soon as an oligomer that contains modified monomers is selected in the listview. Note that each modified monomer in the selected oligomer will show up as a row in this listview.

* Cleavage Data (Figure 8.19) This is the place where the cleavage operation configuration is reported, so that each cleavage results' displaying window is self-traceable to both the cleavage configuration and the polymer sequence that was cleaved in the first place.

The button labelled Find will allow the user to find masses in the oligomers that were generated upon the cleavage reaction simulation (see section 8 on page 105)

Fragmentation Of Polymer Sequences

It happens very often that polymer sequences need to be fragmented in the gas phase (in the mass spectrometer) so that structure characterizations may be performed. For protein chemistry, this happens very often in order to get sequence information for a given peptide ion selected in the gas phase. **GNU polyxmass** must be able to perform those fragmentations *in silico*. Let's see how a polymer sequence can be fragmented using **GNU polyxmass**.

It is a matter of having a polymer sequence opened in an editor window and selecting the sequence region to be fragmented. Once this is done, the user selects the *Chemistry*⟶*Fragment* menu. The user is provided with a window where a number of fragmentation specifications are listed (Figure 8.20). These fragmentation specifications are listed by looking into the polymer chemistry definition corresponding to the polymer sequence to

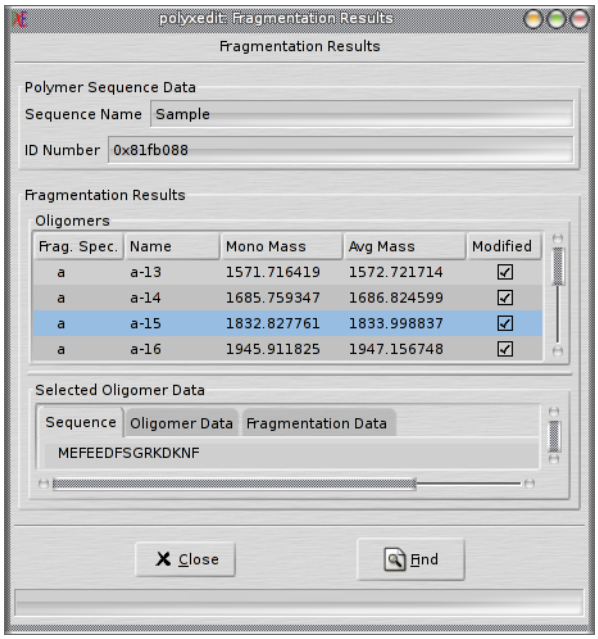

Figure 8.21: **Fragmentation-generated oligomers window.** This figure shows the window that is opened so that the oligomers generated upon fragmentation of a polymer sequence can be displayed.

be fragmented. The program knows, for example, that the polymer sequence to be cleaved is of the "protein" chemistry type, and thus will list all the fragmentation specifications that were defined in the "protein" polymer chemistry definition.

The user selects the fragmentation specification(s) of interest and clicks the Fragment button.

Upon successful termination of the fragmentation reaction, the user is provided with a new window (Figure 8.21 on the next page) in which all the oligomers that were generated are listed (upper pane). The listview widget on the upper pane sports a number of columns. Each row of this listview widget describes the properties of a single oligomer. The different columns are detailed below:

* Frag. Spec. This is the name of the fragmentation specification that was used to compute the corresponding fragment;
* Name This is the name of the oligomer, so that the user may refer to it simply. The syntax is simple: x-y means that this oligomer is the oligomer number y from the fragmentation specification x;
* Mono Mass This is the monoisotopic mass of the oligomer, computed using the options that are set in the Calculation Options window (see earlier explanations);
* Avg Mass Same as above, but for the average mass;
* Modified Indicates if the oligomer contains an intrinsically-modified monomer (it does not mean that the modification's mass was taken into account, it simply says that at least one monomer is modified in the oligomer. See below for details).

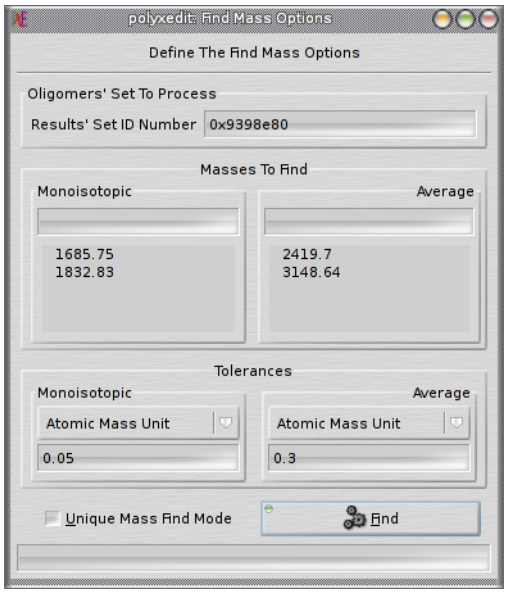

Figure 8.22: **Finding masses in a set of oligomers.** This figure shows how to ask that masses be found in a set of oligomers that result, for example, from the fragmentation of a polymer sequence.

The Sequence, Oligomer Data and Fragmentation Data pages of the notebook in the Selected Oligomer Data frame widget are conceptually identical to the ones described at the section 8 on page 99).

The button labelled Find will allow the user to find masses in the oligomers that were generated upon the fragmentation reaction simulation (see section 8 on the next page).

Finding Masses In The Results

It is often necessary to make sure that a mass –observed in the real mass spectrum– actually corresponds to an oligomer that was generated during a previous simulation experiment (like a cleaving of the polymer sequence with a given cleavage agent or a fragmentation of a simple mass searching operation –see section 8 on page 107). To allow this, and as shown in Figures 8.17 to 8.21 on pages 101–104, it is possible to ask that masses be found into the oligomers resulting from any previous simulation (cleavage or fragmentation of a polymer sequence or arbitrary mass search operations). Indeed, the button labelled Find will open a window where the user may enter masses to be found.

The Figure 8.22 illustrates how easy it is to defines the mass(es) to be found in a set of oligomers, either in the monoisotopic mass list or in the average mass list. There are two ways to actually trigger the mass finding operation:

* When the Unique Mass Find Mode checkbutton *is* checked: the user must enter one mass in the single-line text entry widget and hitting the Find button or the ENTER issues the "Find Mass" request. For this to happen properly, it is necessary that only one of the two single-line text entry widgets be filled with a mass (either monoisotopic

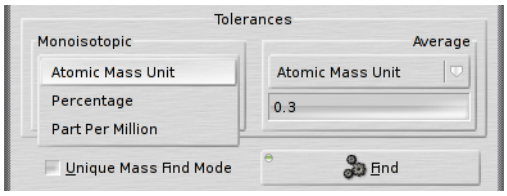

Figure 8.23: **Tolerances available in finding masses.** This figure shows the three different ways that tolerances can be configured.

or average). This is because if there are two masses entered in the widgets, the program would not know which one of the monoisotopic or average masses is to be found in the set of oligomers.

* When the Unique Mass Find Mode checkbutton is *not* checked: the user may enter masses in whatever the single- or multi-line widgets (either by keying-in one mass per line or by pasting a preformatted list of masses). In the present case, hitting the ENTER key will trigger the "multi-mass" mass finding operation only if the Find button has the focus. A click onto the Find button will do!

Prior to asking that masses be found, it is required that tolerances be entered for either monoisotopic or average masses (or both if both kinds of masses are of interest) in their respective text entry widget. In the example of Figure 8.22, the tolerance that is given to the mass finding operation on monoisotopic masses is of `0.1` amu, while the one for the average masses is greater (`1` amu). These values must be understood in a "broad" manner (*i.e.* ± tolerance): for example, if we searched for a mass `1000` with a `0.5` amu tolerance, we would get all the oligomers having masses ranging $[1000 - 0.5 \rightarrow 1000 + 0.5]$ (which is [999.5–1000.5] *and not [999.75–1000.25]*). The Figure 8.23 shows that there are two other means to define the tolerance with which masses should be found. They all are self-explanatory and should also be understood in the same "broad" manner described above.

The oligomers that were found to comply with the masses to find and with the tolerances defined are displayed in a window similar to the one shown in Figure 8.24.

Note that here also the traceability of the data is ensured using unambiguous identity numbers (Results' Set ID Number). This identity number is unique and describes the results window in which the user has asked that masses be found (see Figure 8.22 on the previous page).

Searching Masses In The Polymer Sequence

It may happen that the scientist needs to know if some polymer sequence region would have a given mass. **GNU polyxmass** allows for mass searching operations in the polymer sequence. This is done by using the menu *Chemistry⟶Search Mass(es)*. The window illustrated in Figure 8.25 on the facing page shows up and the user enters masses to search for (see section 8 on page 105 for details on the workings of a very similar window).

Once the masses have been searched, if results are found they are displayed in the window shown in Figure 8.26 on the following page. This window has very similar characteristics to the ones of the previously described results' windows (see section 8 on page 99, for example).

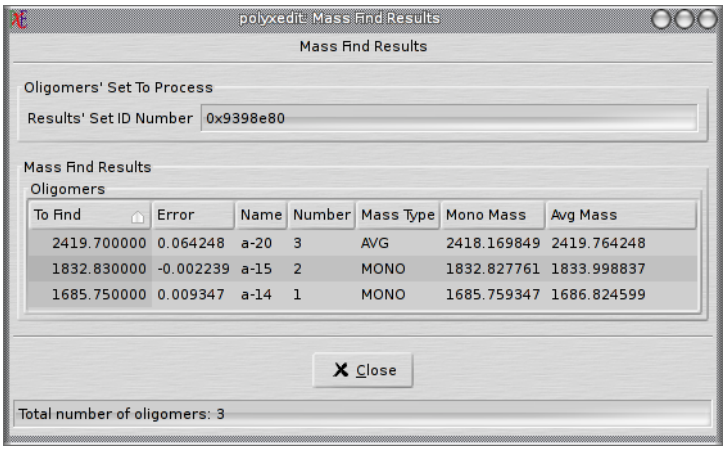

Figure 8.24: **Finding masses in a set of oligomers.** This figure shows oligomers that were found in a set of oligomers after a mass finding operation has been performed.

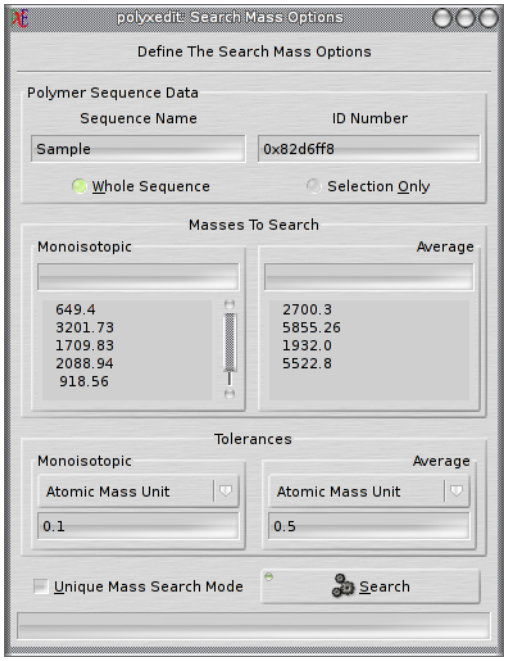

Figure 8.25: **Finding masses in a polymer sequence.** This figure shows how to ask that masses be searched in a polymer sequence.

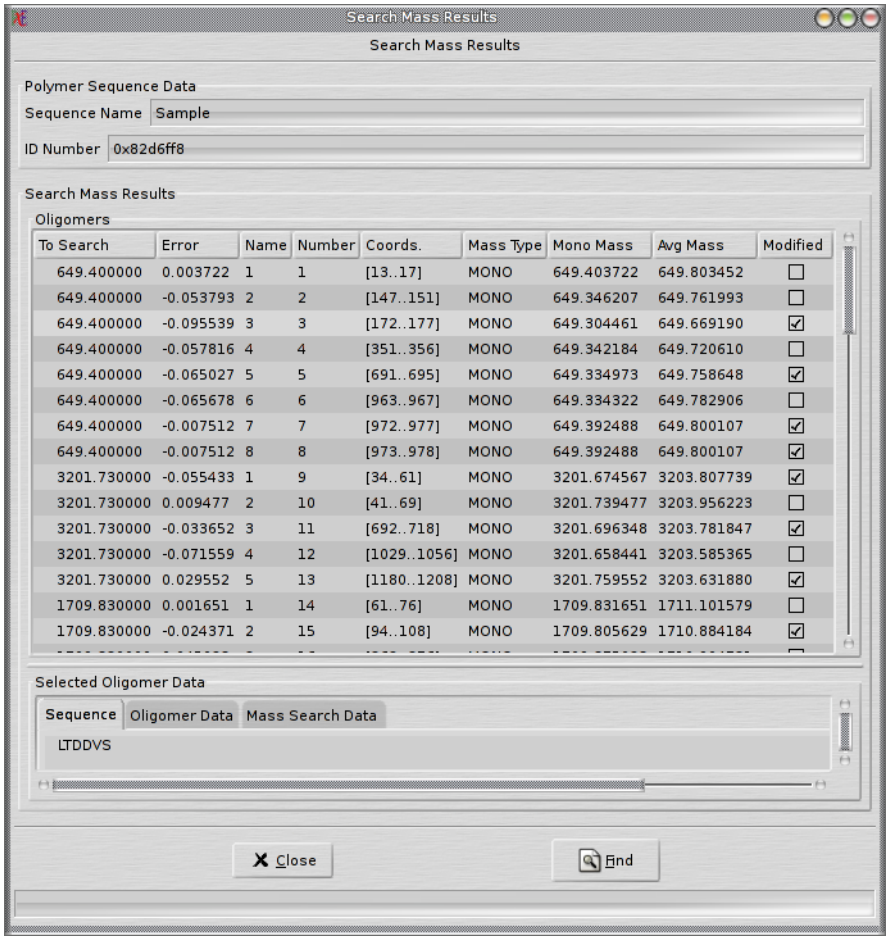

Figure 8.26: **Results window after searching masses in a a polymer sequence.** This figure shows the oligomers that were found upon a mass search operation.

The button labelled Find will allow the user to find masses in the oligomers that were generated upon the mass searching operation (see section 8 on page 105).

The acido-basic calculations: pH, pI and charges

When preparing biochemical experiments, very often users need to know how many charges a given polymer sequence will bear at any given pH. Equally important is the ability to know at which pH value the polymer sequence will have a net charge near to zero. The pH value for which a given polymer sequence has a net charge near to zero (typically this means that the number of positive charges equals the number of negative charges) is called the isoelectric point —the pI.

Such computations are pretty computer-intensive and require a very precise knowledge of the chemical structure of the different monomers that take part in the definition of the polymer chemistry. A file, called `acidobasic.xml` is located in the polymer chemistry definition directory. This file lists all the chemical groups that are possibly charged; each monomer of the polymer definition is represented by a `<mnm>` element in which data are defined for any chemical group of that monomer that might bear a charge at any given pH. You can find the listing of the `acidobasic.xml` file in chapter 11 on page 147. We'll discuss any aspect of this file's contents in the next sections with enough detail that the user will be able to write one such file for her specific polymer chemistry.

At the moment, two entities in the polymer chemistry definition might have chemical groups bearing charges: monomers and modifications. We will first review monomers, and modifications next.

Monomers might have ionized chemical group(s)

Some theory first

Monomers are the building blocks of polymer sequences. These blocks must have at least two reactive groups so that they can be polymerized into a polymer sequence thread. Reactive groups are often chargeable groups; for example, the amino group of amino-acids is such that it gets protonated (positively charged) at a pH inferior to its pKa (that is a physiological pH). Similarly, the carboxylate group —that is the other reactive group of amino-acids— is charged at physiological pH: it is in its carboxylate form (that is singly negatively charged; COO^-) instead of being in its carboxylic form (that is non-charged; COOH).

For the non-biochemist reader, amino-acids involved in the formation of proteins have always at least two chemical groups that are of inverted electrical charge, at physiological pH values (see Figure 8.27):

- The amino group (called αNH_2) has a typical pKa value of 9.6. This means that, at physiological pH values (between 6.5 and 7.5), the amino group will find the environment rather acidic, and will thus be protonated, leading to a positively-charged species (αNH_3^+);

- The carboxylic group (called $\alpha COOH$) has a typical pKa value of 2.35. This means that, at physiological pH values, the carboxylic group will be in a rather basic environment, and will thus be deprotonated, leading to a negatively-charged species (αCOO^-).

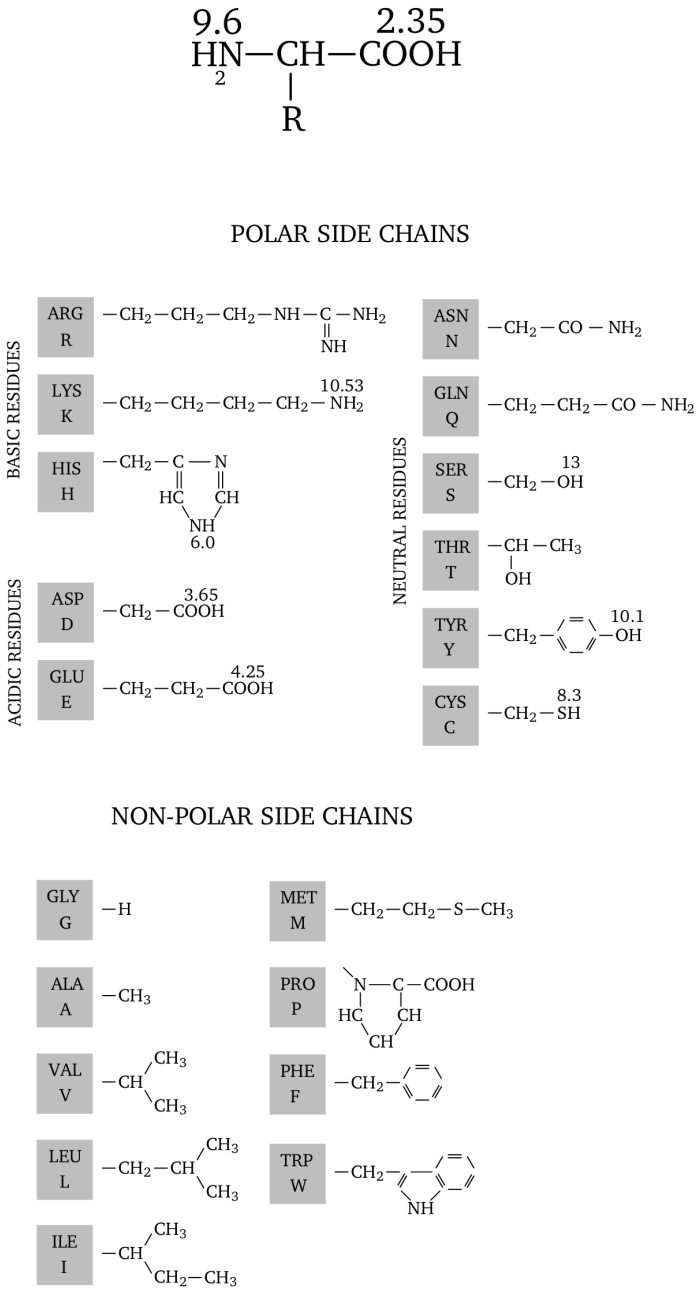

Figure 8.27: **Different pKa values for a number of amino-acids' chemical groups.** All of the twenty amino-acids are represented here, which each amino-acid's lateral chain fully represented. Above each chemical group —for which the value makes sense from a biological perspective— the pKa value is indicated.

It should be clear that, at physiological pH values the two α chemical groups have a net charge of 0. But proteins are charged, and this is because some of the twenty common amino-acids have other chemical groups beyond the two others already described.

Indeed, some amino-acids have lateral chains that bear groups that might be charged depending on the pH: seryl residues have an alcohol group that has a pKa of 13, for example; that means that it is almost always uncharged (form ROH at physiological pH values). The lateral chain of lysine has a pKa of 10.53, which means that at pH values below this pKa value, the ϵNH_2 gets protonated, introducing a positive charge in the protein. Similarly, amino-acids glutamate and aspartate do have a lateral chain ended with a γCOOH and a βCOOH, respectively. Their pKa values are below 4.5, and thus the groups are negatively charged a physiological pH values.

When the net charge of a polymer sequence has to be computed for a given pH condition, the program iterates in the sequence, and for each monomer will check which one of its chemical group(s) is possibly charged. For this to happen, it is required that a number of data be known for each monomer's chemical group that might play a role in the determination of the polymer sequence's electrical charge. Thus, for each chemical group a number of data should be listed in the `acidobasic.xml` file (please, see that file in the chapter 11 on page 147):

- the chemical group's <name> element is required. Examples: "αNH_2" or "ϵNH_2" or "αCOOH";
- the chemical group's <pka> element is optional, but is the basis for the charge calculation. Examples: 9.6 for the "αNH_2" or 2.35 for "αCOOH";
- the <acidcharged> element is required if the <pka> element is given. This element is responsible for telling if the chemical group is charged (positively) when the pH is lower than pKa (that is when the medium is acidic with respect to the pKa). Examples: an amine is positively charged when it is in its acidic form (protonated); a carboxylic acid is *not* charged when it is in its acidic form;
- there can be none, one or more <polrule> element(s) for each chemgroup. The <polrule> element gives informations about the way the chemical group at hand might be "trapped" (or not) in the formation of inter-monomer bonds (while the monomer is polymerized into the polymer sequence). The value "left_trapped" means that the chemical group ceases to be involved in charge calculations as soon as it has a monomer at its left end. The value "right_trapped" means the same as above, but when a monomer is polymerized at its right end. For a chemical group that is "left_trapped", we understand that it is only effectively evaluated if it is at the left end of the polymer sequence, since in this case it does not have a monomer at its left side. Conversely, a chemical group that has a <polrule> element with value "right_trapped", will be evaluated only if the monomer is actually the right end monomer in the polymer sequence. Finally, the typical lateral chains of amino-acids have a <polrule> element with a value "never_trapped", as these chemical groups do not take part in the formation of the inter-monomer bond;
- there can be none, one or more <chemgrouprule> element(s) for each chemgroup. A chemgrouprule element should contain the following:
  - there must be an <entity> element that indicates what is the chemical entity being dealt with in the current chemgroup element. Valid values for this element are "LE_PLM_MODIF", "RE_PLM_MODIF" or "MNM_MODIF";
  - there must be a <name> element naming the chemical entity properly;

- there must be an <outcome> element telling what action should be taken when encountering the <entity> on the chemgroup. Valid values are either “LOST” or “PRESERVED”.

Understanding by example

Let us take some examples in order to make sure we actually understand the process of describing how an electrical net charge is calculated for a given polymer sequence and at any given pH value.

Let us see the example of the aspartate amino-acid, of which the lateral chain is nothing but CH_2COOH:

```
<mnm>
  <code>D</code>
  <chemgroup>
    <name>N-term NH2</name>
    <pka>9.6</pka>
    <acidcharged>TRUE</acidcharged>
    <polrule>left_trapped</polrule>
    <chemgrouprule>
      <entity>LE_PLM_MODIF</entity>
      <name>Acetylation</name>
      <outcome>LOST</outcome>
    </chemgrouprule>
  </chemgroup>
  <chemgroup>
    <name>C-term COOH</name>
    <pka>2.36</pka>
    <acidcharged>FALSE</acidcharged>
    <polrule>right_trapped</polrule>
  </chemgroup>
  <chemgroup>
    <name>Lateral COOH</name>
    <pka>3.65</pka>
    <acidcharged>FALSE</acidcharged>
    <polrule>never_trapped</polrule>
    <chemgrouprule>
      <entity>MNM_MODIF</entity>
      <name>AmidationAsp</name>
      <outcome>LOST</outcome>
    </chemgrouprule>
  </chemgroup>
</mnm>
```

We see that the code of the monomer for which acid-basic data are being defined is ‘D’ and that this monomer has three chemical groups that might bring electrical charges. These chemical groups are described by three `<chemgroup>` elements that we will review in detail below (see Figure 8.27 on the preceding page).

The first `<chemgroup>` element is related to the αNH_2 amino group of the amino-acid:

- `<name>N-term NH2</name>` The name of the chemical group is not immediately useful, but will be used when reports are to be prepared for the calculation;
- `<pka>9.6</pka>` This element is optional. However, of course, if the chemical group might be electrically charged, the pKa value will be essential in order to compute the charge that is brought by this chemical group at any given pH;
- `<acidcharged>TRUE</acidcharged>` This element is also optional, however, if the previous element is given, then this one is compulsory. Telling if the conjugated acid form is charged (that is protonated) is essential in order to know what sign the charge has to be when the chemical group is ionized. The value "TRUE" indicates that when the pH is lower than the pKa, the chemical group is charged, thus protonated (in the form NH_3^+). Consequently, if the pH is higher than the pKa, then the chemical group is neutral (in the form NH_2);
- `<polrule>left_trapped</polrule>` This element indicates that the chemical group should only be taken into account in the eventuality that the monomer bearing it (code 'D') is the left end monomer of the polymer sequence. This can easily be understood, as this chemical group is responsible for the establishment of the inter-monomer bond towards the left end of the polymer sequence;
- `<chemgrouprule>` This element provides further details on the chemistry that the chemical group at hand (αNH_2) might be involved in:
  - `<entity>LE_PLM_MODIF</entity>` This element indicates that the supplementary data in the current `<chemgrouprule>` element are pertaining to the αNH_2 chemical group *only* in case the polymer sequence is left end-modified (that is with a permanent left end modification) and the monomer (code 'D') is located at the left end of the polymer sequence (that is: it is the first monomer of the sequence for which the electrical charge —or pI— computation is to be performed).
  - `<name>Acetylation</name>` This element goes further in the detail of the potential chemistry of the αNH_2 chemical group: if the left end permanent modification is "Acetylation", then the current chemgrouprule element can be further processed, otherwise it should be abandoned;
  - `<outcome>LOST</outcome>` This element actually indicates what should be done with the chemical group for which the chemgrouprule is being defined. What we see here is: —*"If the αNH_2 chemical group, belonging to a 'D' monomer located at the left end of a polymer sequence, is modified permanently with an "Acetylation" left end modification, it should not be taken into account when computing the charge that it could bring to the polymer sequence."*

The second `<chemgroup>` element is related to the $\alpha COOH$ carboxylic group of the amino-acid:

- `<name>C-term COOH</name>` Same remark as above;
- `<pka>2.36</pka>` Same remark as above;
- `<acidcharged>FALSE</acidcharged>` Same remark as above. However, as we can see, the value indicates that the acid conjugate (form COOH) does not bring any charge. This means that when the basic conjugate is predominant (that is when pH > pKa), it brings a negative charge: the form is COO^-;

* `<polrule>right_trapped</polrule>` The chemical group should not be evaluated if a monomer is linked to it at its right side. That means that the current chemical group is only evaluated if the monomer bearing it is located at the right end of the polymer sequence. This is easily understood, as the αCOOH chemical group is involved in the formation of the inter-monomer bond towards the right end of the polymer sequence.

The third `<chemgroup>` element is related to the βCOOH carboxylic group of the amino-acid:

* `<name>Lateral COOH</name>`;
* `<pka>3.65</pka>`;
* `<acidcharged>FALSE</acidcharged>`;
* `<polrule>never_trapped</polrule>` This element indicates that, whatever the position of the monomer bearing the chemical group in the polymer sequence (left end, right end or middle), the chemical group is to be evaluated;
* `<chemgrouprule>` This element provides further details on the chemistry that the chemical group at hand (βCOOH) might be involved in:
  - `<entity>MNM_MODIF</entity>` This element indicates that the supplementary data in the current `<chemgrouprule>` element are pertaining to the βCOOH chemical group *only* in case the monomer bearing the chemical group is chemically modified;
  - `<name>AmidationAsp</name>` This is the modification by which the monomer should be modified in order to have the `<chemgrouprule>` element effectively evaluated;
  - `<outcome>LOST</outcome>` This element actually indicates that if the monomer bearing the chemical group is modified with an "AmidationAsp" chemical modification, then the chemical group should not be evaluated any more for the electrical charge —or pI— calculations, since reacting a carboxylate group with an amino group produces an amide group which is not easily chargeable at physiological pH values.

At this point we should have made it clear how the charge calculations can be configured for the different monomers in the polymer chemistry definition. As usual, the more the polymer chemistry definition is sophisticated, the more sophisticated the computations allowed.

Modifications might have ionized chemical group(s)

In the excerpt from the `acidobasic.xml` file below, we see that chemical modifications can also bring charges. The example of the chemical modification "Phosphorylation" shows that when a monomer is phosphorylated, two chemical groups are brought in: the first has a pKa value of 12 (that is it will always be protonated at physiological pH values), the second has a pKa value of 7 (that is it will be divided by half in a protonated (not charged) form and in an un-protonated (negatively charged) form, leading to a net electrical charge of -0.5.

```
<modifs>
  <mdf>
```

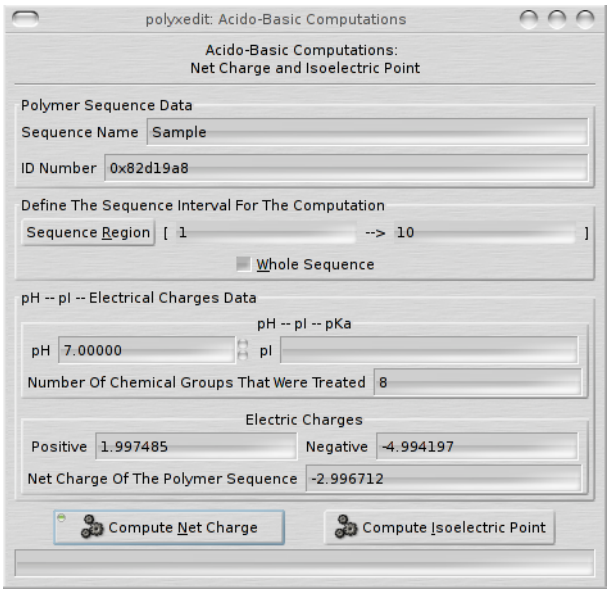

Figure 8.28: **Acido-basic computations: pI, pH, pKa.** This figure shows the options that can be set for the calculations related to the charges beared by the polymer sequence.

```
      <name>Phosphorylation</name>
      <chemgroup>
        <name>none_set</name>
        <pka>12</pka>
        <acidcharged>FALSE</acidcharged>
      </chemgroup>
      <chemgroup>
        <name>none_set</name>
        <pka>7</pka>
        <acidcharged>FALSE</acidcharged>
      </chemgroup>
    </mdf>
  </modifs>
```

At this point we should be able to study the way computations are actually performed in the **polyxedit** module.

Performing pH, pI and charges computations

The user willing to compute charges (positive, negative, net) or the isoelectric point for the current polymer sequence uses the contextual menu $\longrightarrow$*pKa-pH-pI*$\longrightarrow$*Computations* which triggers the appearance of the window shown in Figure 8.28 on page 115.

This figure shows that the user might either compute the charges (positive, negative and net) for the polymer sequence by setting the pH value at which the computation should take place and clicking onto the Compute Net Charge button, or ask that the isoelectric point be computed *ex nihilo* by clicking onto the Compute Isoelectric Point button (in which case the

pI text entry widget will display the pH at which the Net Charge Of The Polymer Sequence will be near to **0**.

Clicking onto the Compute Isoelectric Point will trigger computations that are lengthy, and the user is advised to be patient. As an example, on my computer,[6] the pI computation for a protein of 10201 residues took 10 seconds (no modifications taken into account). If the user asks that the different modifications (permanent polymer modifications and monomer modifications) be taken into account, the duration of the computation is twice as long (23 seconds).

Note that the computations might involve the permanent left/right modifications of the polymer sequence, as well as the monomer chemical modifications. To configure the way net charge —or pI— computations are performed, please use the calculations engine configuration window, as described in Figure 8.4 on page 78.

The m/z Ratio Calculator

When requiring m/z ratio calculations the user might use the

Chemistry⟶m/z Ratio Calculations

contextual menu that shows up when the user right-clicks onto the polymer sequence. Note that the process of using the calculator was described in Section 7 on page 71. When the calculator is used in **polyxedit**, the initial ionization status data are set from the currently defined ionization rules (see the Ionization Rules frame in the window displayed in Figure 8.4 on page 78) of the polymer sequence for which the computations are to be performed.

The Self-Read Feature Of Polymer Sequences

It happens some times that the user needs somebody to read a sequence while he double-checks the sequence being read. I have been confronted to that situation a number of times (in particular when having to confirm oligonucleotidic sequences), and finally decided that I would give polymer sequences a "self-reading" ability.

The basis of the self-reading framework is as simple as the writing of (yet) another dictionary that makes —for each polymer chemistry definition— the correspondence between a chemical entity and a sound file that should be played in order to "read out" a polymer sequence. Two chemical entities are able to read themselves out:

- Monomers: the user may define two sound files for each monomer of the polymer chemistry definition: a sound file vocalizing the monomer name ("alanine", for example), and another sound file vocalizing the monomer code ("A" or "Ala", for example).
- Modifications: the user may define only one sound file for each modification of the polymer chemistry definition.

Selecting the

Edit⟶Export Sound Playlist

menu item in the contextual menu that pops up when the user right-clicks onto the polymer sequence will trigger the window displayed in Figure 8.29 on the facing page to show up.

[6] My `/proc/cpuinfo` and `/proc/meminfo` say "Intel(R) Pentium(R) M processor 1400MHz; cpu family: 6; model: 9; 1024 KB cache size; 774376 kB of memory; bogomips: 2768.89".

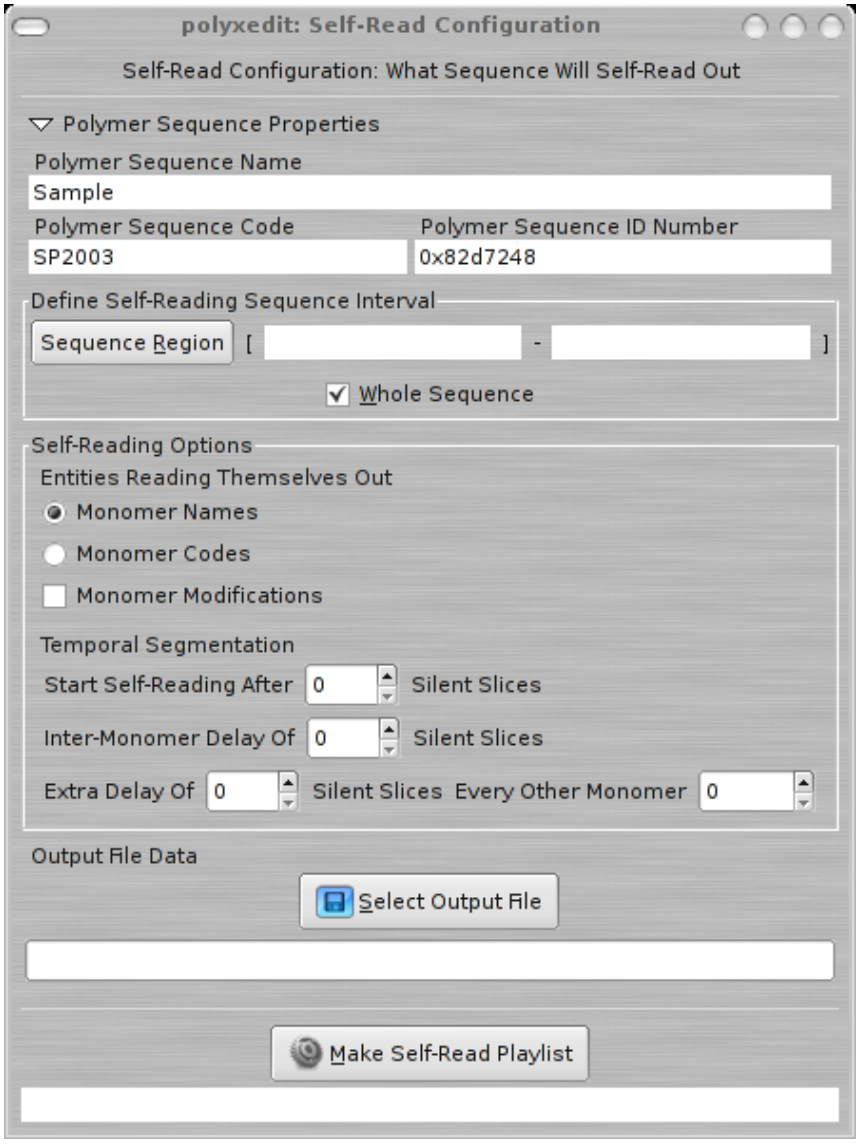

Figure 8.29: **Polymer Sequence Self-Read Options.** This figure shows the options that can be set for the polymer sequence to read itself out to a playlist file.

If a polymer sequence region is selected when the menu above is selected, then the positions of the monomers delimiting that region are displayed in the Define Self-Reading Sequence Interval frame. If the user changes the selection in the sequence editor, these values can be updated by clicking onto the Sequence Region button. It is, however, possible to ask that the whole polymer sequence be self-spoken out by clicking onto the Whole Sequence checkbutton.

The polymer sequence self-reading feature allows to select if monomer codes or monomer names should be vocalized in the sequence, and if the monomer modifications should be vocalized also.

Finally, the Temporal Segmentation frame lets the user define how the files corresponding to the monomers' code/name (and modifications' name, if so is required) are played. Specifically, it is possible to ask that silences be interspersed between the sounds corresponding to the chemical entities being self-spoken out. Silent delays are played exactly in the same manner as the other chemical entities' sounds (that is: a silent delay is played as a "silence sound" file...). The user might ask that the sequence read-out be interspersed with the following silent delays:

* Start Self-Reading After [x] Silent Slices: the "silent sound" file is played the specified number of times before the sequence starts to read itself out. If the "silent sound" file is 300 milliseconds-long, and the user wants a 1 second delay before the sequence actually begins to read itself out, the number asked would be 3;
* Inter-Monomer Delay Of [x] Silent Slices: a silent delay will be inserted between each monomer sound;
* Extra Delay Of [x] Silent Slices Every Other Monomer [y]: it might be useful, sometimes, to insert a silent delay each time a given number of monomers have been spelled. This is particularly interesting when nucleic acids sequences read themselves out, so that a "reading frame" is conserved all along. One would thus set the silent delay to be inserted every three monomers...

The user indicates the name of the file where the playlist is to be written. It is advisable to use the *m3u* file extension so that the sound player will recognize that file as a playlist file.

Indeed, **GNU polyxmass** does not generate sounds on the sound card. All it does is write a sound playlist that the user later hands out to a sound player, like `xmms` or `winamp`.

The *m3u* file format is pretty easy: it is a list of files to be played in succession. Note that for the sequence to be properly spoken-out at that step, the "shuffle" feature of the player should be disabled.

The following is the contents of the `sequence.m3u` playlist file that was obtained by having a protein sequence read itself out:

```
/usr/share/polyxmass/polchem-defs/protein/sounds/glutamate.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/glutamate.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/aspartate.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/silence.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/phenylalanine.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/serine.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/phospho.ogg
```

Note that the last serine monomer is phosphorylated and that the user asked that an interval be played every three monomers.

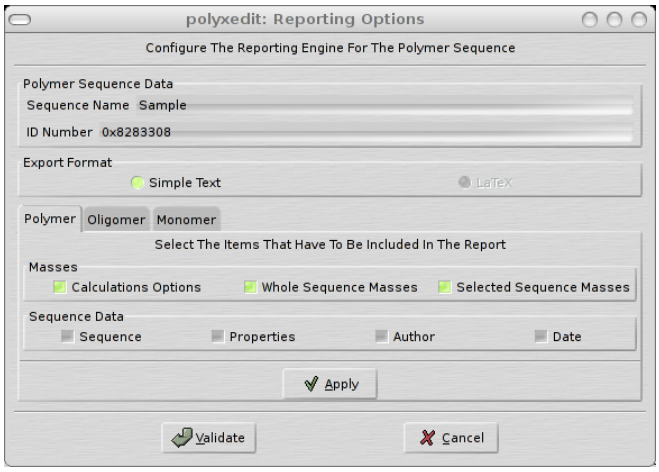

Figure 8.30: **The reporting options configuration.** The configuration of the way window contents are reported is highly configurable. The configuration will affect the way the polymer sequence's data are reported, but also the way oligomers' data are reported and monomers'. Each tab of the depicted window deals with each one of these configuration options..

The correspondence between a given monomer (or modification) and the sound it should use to read itself out is performed in a text file (`sounds.dic`) located in the `sounds` directory itself located in the polymer chemistry definition data directory. See the chapter about **GNU polyxmass-common** for details.

Results Reporting

polyxedit allows the user to perform a great number of different simulations on any number of polymer sequences opened at any given time. While the simultaneity of simulations (for example having at one given time different enzymatic cleavages on a set of different proteins) is necessary, as a simple matter of flexibility and power, it is necessary to perform well-organized results reporting.

The reports might be asked for any window that displays results. For example, a window that displays a polymer sequence (the polymer sequence editor, in fact) is a results window as it displays a sequence. A window displaying the oligomers obtained upon cleavage of a polymer sequence with a chemical cleavage agent is also a results window. As we have seen earlier, each results window is registered to the program and its specifics are stored in items visible in the Available Windows treeview of the window management window shown in Figure 5.4 on page 48.

The configuration of the way reports are prepared takes place in the polymer sequence context. The polymer sequence editor window menu

Reporting⟶*Reporting Options*

will open a window as depicted in Figure 8.30.

Once the reporting options are configured in a polymer sequence editing context, they automaticall apply for all the results windows in the same polymer sequence editing context.

The reporting options are always modifiable using the same menu as above. Once the configuration of the reporting options is performed, the user might use the

Reporting⟶Make Reports

menu to elicit the opening of the window management window, where the following reporting actions are made avaiblable through button widgets.

After selecting a particular window item from the Available Windows treeview, it becomes possible to ask that the selected window exports a report about its contents. The report can be sent to the clibpoard or to a file (in append or overwrite mode) by using the corresponding button widgets in the window management window:

* Report To Clipboard: ask that the window contents be exported to the clipboard;
* Overwrite To File: ask that the window contents be exported to a file. Overwrite the file if it exists already;
* Append To File: ask that the window contents be exported to a file. The new contents report data are appended to a preexisting file.

9

GNU polyxmass-common: The Configuration and Data Files Hierarchy

The **GNU polyxmass** software suite is designed to be compatible with any polymer chemistry that the user cares to define. To be that flexible, **GNU polyxmass** has to be able

to store polymer chemistry definition files and related data files in a very clearly-designed "file-system". The structure of this "file-system" is what this chapter is all about.

When the user defines a polymer chemistry definition (typically using **polyxdef**), the contents of that definition are saved in a text file (using the *xml* format). Once a polymer chemistry definition is saved and registered in the **GNU polyxmass** system, the user can create a new polymer sequence of that polymer chemistry:[1] when entering monomer codes at the keyboard, the user sees monomer icons (small graphical images) being displayed in the sequence editor (we call that process the "graphical rendering" of the polymer sequence). So, here are a number of typical questions:

* Where is defined the correspondence between any monomer code (as keyed-in during a polymer sequence editing session) and the monomer icon[2] that is displayed in the **polyxedit** sequence editor?
* Where are located all the graphics files that are used to graphically render a sequence in the editor? And the sound files used to let a polymer sequence speak itself out?
* Where is located the atom definition that should be used with this specific polymer chemistry definition, and were is defined the correspondence between a polymer chemistry and the atom definition file that is required?

Within **GNU polyxmass**, there are two different kinds of data/configuration data files:

* Compulsory data/configuration files that *must* be on the system at precise locations whatever the chemical definitions being used on that system;
* Optional data/configuration files that are installed by users (or system administrators) so as to comply with requirements specific of each installation.[3]

The present chapter is about the first item in the above bulleted list: compulsory data/configuration files that are all shipped with the **GNU polyxmass-common** package. We will review the locations where data/configuration files are installed and the mechanics that make **GNU polyxmass** work on any kind of polymer chemistry.

Overview Of The Files Installed

Let us first review all the files that are installed by the **GNU polyxmass-common** package:

```
/usr/share/polyxmass/atom-defs/atoms.xml

/usr/share/polyxmass/polchem-defs/protein
/usr/share/polyxmass/polchem-defs/protein/acetyl.png
/usr/share/polyxmass/polchem-defs/protein/acetyl.svg
/usr/share/polyxmass/polchem-defs/protein/acetyl-text.svg
/usr/share/polyxmass/polchem-defs/protein/alanine.png
/usr/share/polyxmass/polchem-defs/protein/alanine.svg
/usr/share/polyxmass/polchem-defs/protein/alanine-text.svg
```

[1] Editing of polymer sequences is typically performed in **polyxedit**.

[2] We call that icon a "monicon."

[3] For example, a synthetic polymer lab will almost certainly not install data packages about proteins or nucleic acids, while a biochemistry lab will almost certainly not install packages about polymethylmetacrylate...

```
/usr/share/polyxmass/polchem-defs/protein/monicons.dic
/usr/share/polyxmass/polchem-defs/protein/sounds/alanine.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/a.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/methyl.ogg
/usr/share/polyxmass/polchem-defs/protein/sounds/sounds.dic

/usr/share/polyxmass/polchem-defs/protein/chempad.conf

/usr/share/polyxmass/polchem-defs/protein/acidobasic.xml

/usr/share/polyxmass/polchem-defs/protein/cursor.svg

/usr/share/polyxmass/polchem-defs/protein/protein.xml
/usr/share/polyxmass/polchem-defs/protein/peptide.xml

/usr/share/polyxmass/pol-seqs/protein-sample.pxm
/usr/share/polyxmass/pol-seqs/protein-fragments-sample.pxm

/etc/polyxmass/atom-defs/polyxmass-common-atom-defs-cat
/etc/polyxmass/polchem-defs/polyxmass-common-polchem-defs-cat
/etc/polyxmass/polchem-defs/polyxmass-common-polchem-defs-atom-defs-dic

/etc/polyxmass/chempad.conf

/usr/share/doc/polyxmass-common/AUTHORS
/usr/share/doc/polyxmass-common/COPYING
/usr/share/doc/polyxmass-common/INSTALL
/usr/share/doc/polyxmass-common/NEWS
/usr/share/doc/polyxmass-common/README
/usr/share/doc/polyxmass-common/TODO
/usr/share/doc/polyxmass-common/THANKS

/usr/share/man/man7/polyxmass-common.7

/usr/lib/pkgconfig/polyxmass-common.pc
```

All the text above is the output (edited for clarity) of the `make install` command that is performed to install the **GNU polyxmass-common** package on the system. It is taken for granted that the user did not change the `--sysconfdir=/etc` option to the `configure` script and that he passed the following option to that same `configure` script: `--prefix=/usr`. If the **GNU polyxmass-common** package is installed as a binary package, then the user needs not worry: the packager did choose the best installation directories. Let us review each file that is installed one by one, telling what it is meant for:

* Files located in `/etc/polyxmass`:
  - `atom-defs/polyxmass-common-atom-defs-cat` This file is the catalog file corresponding to the **GNU polyxmass-common** package. It contains the list of the atom definition files that are brought by the **GNU polyxmass-common** package.
  - `polchem-defs/polyxmass-common-polchem-defs-cat` This file is the catalog file corresponding to the **GNU polyxmass-common** package. It contains the list of the polymer chemistry definition files that are brought by the **GNU polyxmass-common** package.
  - `polchem-defs/polyxmass-common-polchem-defs-atom-defs-dic` This file is the dictionary file corresponding to the **GNU polyxmass-common** package. It contains the relations between each polymer chemistry definition file shipped with the package and its cognate atom definition file.

- `chempad.conf` This file describes the layout of the chemical pad of the **polyxcalc** module, in case the polymer chemistry definition does not have one and the user does not have one neither. This file can thus be called the "default" layout definition file for the **polyxcalc**'s chemical pad.

* Files located in `/usr/share/polyxmass`:

- `polyxmass/atom-defs/atoms.xml` This file is the *"basic"* atom definition file. The **GNU polyxmass** software suite mandates that one atom definition file be present in the system.
- `polyxmass/polchem-defs/protein/acetyl.png` This is one of the raster files that are used in the polymer chemistry definition to graphically render the "Acetylation" chemical modification. Note that also installed is a file by the same name but with extension `.svg`, instead of `.png`. This file is a scalar vector graphics version from which the `.png` file was generated.
- `polyxmass/polchem-defs/protein/alanine.png` One of the graphics files that are used to render graphically the monomers defined in the polymer chemistry definition (in this case the monomer is "alanine"). Same remark as above for the `.svg` extension file.
- `polyxmass/polchem-defs/protein/chempad.conf` This file describes the layout of the chemical pad of the **polyxcalc** module. Each polymer chemistry definition might have a `chempad.conf` file associated to it. This file is optional.
- `polyxmass/polchem-defs/protein/acidobasic.xml` This file describes the chemistry of all the monomers and modifications in the polymer chemistry definition that might bring charges. The data contained in this file are used by the functions that compute either the charge level of a polymer sequence at a given pH value, or the isoelectric point of a polymer sequence (that is the pH value at which the net charge of the protein is near zero).
- `polyxmass/polchem-defs/protein/monicons.dic` This file is the one that lists the correspondence between the monomer codes/modifications and the files used to render the monomers/modifications graphically in the sequence editor. The lines in this file look like:
  `monomer;A=alanine.svg|alanine.png` for a monomer, and like:
  `modif;Phosphorylation%T%phospho.svg|phospho.png` for a modification. The latter line indicates that when a monomer is modified using the "Phosphorylation" modification, the to-be-modified monomer icon get modified by transparently pasting onto it the monicon contained in the file `phospho.svg` (see the `%T%`).
- `polyxmass/polchem-defs/protein/sounds/sounds.dic` This file is the one that lists the correspondence between the monomer codes (or names) and their corresponding sound files. The same is true for modifications. The file contains lines in the form:
  `monomer;Y=tyrosine.ogg|y.ogg` for monomers and in the form:
  `modif;Phosphorylation=phospho.ogg` for modifications. The "monomer;" line indicates that the monomer 'Y' has its name vocalized in the `tyrosine.ogg` file, while its monomer is vocalized in the `y.ogg` file. The "modif;" line indicates that that the "Phosphorylation" modification is vocalized in the `phospho.ogg` file.
- `polyxmass/polchem-defs/protein/cursor.svg` This file is a graphics file that describes how the cursor should be rendered graphically in the polymer sequence editor. Each polymer chemistry definition must provide this file.
- `polyxmass/polchem-defs/protein/protein.xml` This is the actual polymer chemistry definition file. This file is a text file formatted according to the *xml* standard. It contains a description of all the chemical entities that make up the polymer chemistry definition.
- `polyxmass/pol-seqs/protein-sample.pxm` This is an example polymer sequence file. It can be used by the user to learn how to use the **polyxedit** module. This protein sequence

is of polymer chemistry definition "protein", that is defined in the file that we described above (`protein.xml`).

- ✦ `man/man7/polyxmass-common.7` This file is the manual page that accompanies the **GNU polyxmass-common** package.

✱ Files located in `/usr/lib`:

- ✦ `pkgconfig/polyxmass-common.pc` This file is the `pkg-config` configuration file that will allow other packages to check if **GNU polyxmass-common** is installed correctly and what is its version.

Detailed Explanations About Installed Files

Now that we have an overview of what each one of the files that get installed does, we may want to take a closer look at some of the files that were listed above.

File polyxmass-common-atom-defs-cat

Each package that brings atom definition files, should list —in a similar file (ending with the `atom-defs-cat` suffix)— all the atom definitions that are made available to the system. The **GNU polyxmass-common** package installs
`polyxmass-common-atom-defs-cat` in
`/etc/polyxmass/atom-defs`. Its contents are:

```
basic=/usr/share/polyxmass/atom-defs/atoms.xml
```

Thus, we see that **GNU polyxmass-common** brings one atom definition file (`atoms.xml`), installed in `/usr/share/polyxmass/atom-defs` and made available to the **GNU polyxmass** system under the name "basic". This latter name is the one used by polymer definitions to specify with what atom definition file they should be working. We'll see that later.

File polyxmass-common-polchem-defs-cat

Each package that brings polymer chemistry definition files, should list —in a similar file (ending with the `polchem-defs-cat` suffix)— all the polymer chemistry definitions that are made available to the system. The **GNU polyxmass-common** package installs
`polyxmass-common-polchem-defs-cat` in
`/etc/polyxmass/polchem-defs`. Its contents are (each polymer chemistry definition name and its corresponding data *must* be on a single line without space; here, for clarity the line was broken, as symbolised with the "\\" characters that are absent in the file):

```
protein=/usr/share/polyxmass/polchem-defs/protein/protein.xml\\
%/usr/share/polyxmass/polchem-defs/protein
peptide=/usr/share/polyxmass/polchem-defs/protein/peptide.xml\\
%/usr/share/polyxmass/polchem-defs/protein
```

Thus, we see that **GNU polyxmass-common** brings two polymer chemistry definition files (`protein.xml` and `peptide.xml`), installed in `/usr/share/polyxmass/polchem-defs` and made available to the **GNU polyxmass** system under the names "protein" and "peptide", respectively. As can be seen by the example above, the polymer chemistry definition file names are absolute file names (that means that they are preceded by the whole path leading to the file in question) and are separated —by a `%` character— from the absolute name of the directory where corresponding data reside.

Thus, the "protein" polymer chemistry definition file is `protein.xml`, that is located at `/usr/share/polyxmass/polchem-defs`.

The directory where all the "protein" polymer chemistry definition-related data are located is:
`/usr/share/polyxmass/polchem-defs/protein`.

We will see later how this catalogue file is used, in order to create a main catalogue file that is use to read the proper polymer chemistry definition file that is needed when the user asks, for example, that a "protein" sequence be displayed in **polyxedit**.

File polyxmass-common-polchem-defs-atom-defs-dic

Each package that brings polymer chemistry definition files, should list —in a similar file (ending with the `polchem-defs-atom-defs-dic` suffix)— all the relations that govern the use of a determinate atom definition file by any given polymer chemistry definition. The **GNU polyxmass-common** package installs
`polyxmass-common-polchem-defs-atom-defs-dic` in
`/etc/polyxmass/polchem-defs`. Its contents are:

```
protein=basic
peptide=basic
```

The first line of this file stipulates that —*"When working on polymer sequences of polymer chemistry "protein", the atom definition to be used is the one having name "basic."* Since the system knows what actual file corresponds to the atom definition "basic", as we already have seen above, it is not difficult to load that specific atom definition file from disk.

File chempad.conf

This file is responsible for governing the chemical pad layout in the **polyxcalc** module. Each polymer chemistry definition may have one such file in its directory (for example, for the "protein" polymer chemistry definition, we have the
`/usr/share/polyxmass/polchem-defs/protein/chempad.conf` file.

When a polymer chemistry definition with no `chempad.conf` is used in **polyxcalc**, the program automatically tries to read the file from the user's `.polyxmass/chempad.conf` file. If that user's file is not found, the last resort is to read the
`/etc/polyxmass/chempad.conf` file.

This file contains lines like the following:

```
chempad_columns$3

chempadkey=protonate%+H1%adds a proton
```

```
chempadkey=hydrate%+H2O1%adds a water molecule
chempadkey=OH-ylate%+O1H1%adds an hydroxyl group
chempadkey=acetylate%-H1+C2H3O1%adds an acetyl group
chempadkey=protonate%+H1%adds a proton
chempadkey=hydrate%+H2O1%adds a water molecule
```

The first line tells that the chemical pad buttons should be laid out in three columns. Each following line configures one button, that will sit on the chemical pad. Thus, the syntax of a line is the following:

`chempadkey=button_label%action-formula%button_tooltip`

The first button-defining line, for example, configures the creation of a button with the label **"protonate"** which —when mouse-clicked— will elicit the addition of the contents of the action-formula **"+H1"** in the **polyxcalc** module. The string **"adds a proton"** is the text that will appear as a tooltip when the mouse cursor sits on the button.

File acidobasic.xml

This file contains all the pKa data about all the different chemical groups beared either by monomers or modifications defined in the polymer chemistry definition. This file is used when computations about net charges of polymer sequence at a given pH value are asked. Also, this file is used when isoelectric point calculations are performed. See section 8 on page 107.

File monicons.dic

This file is obligatory for each polymer chemistry definition. So, for our example of the "protein" polymer chemistry, it would be found in that polymer chemistry definition directory: `/usr/share/polyxmass/polchem-defs/protein/monicons.dic`.

See below for detailed explanations of its contents.

File atoms.xml

This file, that is located at

`/usr/share/polyxmass/atom-defs`,

is obligatory for **GNU polyxmass** to operate normally. Indeed, if there were no atom definitions, we would be in trouble to compute masses for any chemical entity that is represented by its formula (or action-formula). There might be other atom definitions files, located in that same directory, but with other names. As we have seen above, there is one atom definition, called "basic", that is used by the "protein" and "peptide" polymer chemistry definitions. This "basic" atom definition is actually this `atoms.xml` file.

In more details: this file is an atom definition file, where atoms are defined by defining their individual data. An atom is the resultant of the isotope(s) that it is comprised of. Some atoms only have one isotope, other atoms have as much as seven or eight different isotopes. An isotope is characterized by its mass and its abundance. Hence, the structure of an atom definition, in this file:

```
<atom>
  <name>Carbon</name>
  <symbol>C</symbol>
  <isotope>
    <mass>12.0000000000</mass>
    <abund>98.9300000000</abund>
  </isotope>
  <isotope>
    <mass>13.0033548390</mass>
    <abund>1.0700000000</abund>
  </isotope>
</atom>
```

There might be as many such atom definitions —in this atom definition file— as required for the polymer chemistry definition with which it is to be used. Indeed, we already have mentioned that any polymer chemistry definition must specify the atom definition with which it must work specifically for things to behave properly (that association is specified in the `polyxmass-common-polchem-defs-atom-defs.dic` file (for the polymer chemistry definitions brought by the **GNU polyxmass-common** package; see above).

Directory protein

This directory is the directory where the example "protein" polymer chemistry definition data are located (`/usr/share/polyxmass/polchem-defs/protein`). Indeed, **GNU polyxmass-common** comes with a full polymer chemistry definition; that is: a polymer chemistry definition file (`protein.xml`) and all the data files that permit a polymer sequence of that polymer chemistry to be rendered graphically in the **polyxedit** editor module. Also, comes with the "protein" polymer chemistry data, a `chempad.conf` file that describes —for this specific polymer chemistry— how to lay out the chemical pad used in the **polyxcalc** module. Let's review all the files that make up the "protein" polymer chemistry definition as a functional set of data.

File protein.xml

This file is located in the "protein" polymer chemistry definition directory:
`/usr/share/polyxmass/polchem-defs/protein`.

It is the file where the "protein" polymer chemistry definition is detailed. Its contents look like this (omitting the DTD of the *xml*-format file):

```
<polchemdefdata>
<type>protein</type>
<leftcap>+H</leftcap>
<rightcap>+OH</rightcap>
<codelen>1</codelen>
<ionizerule>
<actform>+H</actform>
<charge>1</charge>
<level>1</level>
```

```
</ionizerule>
<monomers>
<mnm>
<name>Glycine</name>
<code>G</code>
<formula>C2H3NO</formula>
</mnm>
<mnm>
<name>Alanine</name>
<code>A</code>
<formula>C3H5NO</formula>
</mnm>
⋮
</monomers>
<modifs>
<mdf>
<name>Phosphorylation</name>
<actform>-H+H2PO3</actform>
</mdf>
<mdf>
<name>Acetylation</name>
<actform>-H+C2H3O</actform>
</mdf>
<mdf>
<name>Amidation</name>
<actform>-OH+NH2</actform>
</mdf>
</modifs>
<cleavespecs>
<cls>
<name>CyanogenBromide</name>
<pattern>M/</pattern>
<clr>
<re-mnm-code>M</re-mnm-code>
<re-actform>-CH2S+O</re-actform>
</clr>
</cls>
⋮
<cls>
<name>Trypsin</name>
<pattern>K/;R/;-K/P</pattern>
</cls>
</cleavespecs>
<fragspecs>
<fgs>
<name>a</name>
<end>LE</end>
<actform>-C1O1</actform>
<fgr>
```

```
<name>a-fgr-1</name>
<actform>+H2O0</actform>
<prev-mnm-code>E</prev-mnm-code>
<this-mnm-code>D</this-mnm-code>
<next-mnm-code>F</next-mnm-code>
<comment>comment here!</comment>
</fgr>
<fgs>
<name>z</name>
<end>RE</end>
<actform>-N1H1</actform>
<comment>Not in CID high En. frag</comment>
</fgs>
⋮
<fgs>
<name>imm</name>
<end>NE</end>
<actform>-C1O1+H1</actform>
</fgs>
</fragspecs>
</polchemdefdata>
```

As can be seen, the chemical entities that make up the "protein" polymer chemistry definition are listed here in a very structured way. This file is written by the **polyxdef** module, described in another chapter of this manual. Note that some data shown here are fake —as far as the "protein" polymer chemistry is concerned— and are only listed as examples of the fine-grain with which chemical data can be defined in this file.

When a polymer sequence is either loaded from disk, or created *ex nihilo*, the **GNU polyxmass** program will manage to know of what polymer chemistry definition it is. Once it knows what polymer chemistry definition is involved for the polymer sequence at hand, the program loads the corresponding file from disk (if it has not already done so; no polymer chemistry definition file is read from disk more than once, to preserve the smallest memory footprint for the whole **GNU polyxmass** software suite).

Files alanine.svg and alanine.png

These two files are located in the "protein" polymer chemistry definition directory:

`/usr/share/polyxmass/polchem-defs/protein`.

There are two such files for any monomer that is defined in the polymer definition file (for our example that is the `protein.xml` file).

These two files are responsible for the graphical rendering —in the polymer sequence editor, the **polyxedit** module— of the monomers that constitute a polymer sequence. For each monomer in a polymer sequence, its graphical representation is performed by the graphical rendering of a "monomer icon" file (that we call "monicon"). These two files are "monicon files" (see the chapter 8 on page 75).

It should be noted right now that the user may ask that the rendering of the monomers in a polymer sequence be performed at a given size (in pixel units). Thus the size of the

monicons has to be regulatable —preferably without loss of resolution: we will see now how this is achieved.

Of these two files, the first has a name ending with the `.svg` extension: it is a *scalar vector graphics* file (*`svg`*-format file) that describes vector-graphically how the corresponding monomer should be displayed in the **polyxedit** sequence editor. The fact that this file is of that *`svg`* format is interesting because it makes it possible to render in the editor the monicon at any size asked by the user without loosing the image resolution.

The second of these two files has a `.png` extension: it is a *portable network graphics* file (*`png`*-format file) that describes raster-graphically how the corresponding monomer should be displayed in the **polyxedit** sequence editor. Since this file describes the rendering of a monomer icon in a "static" raster/bitmap graphics format, it cannot scale properly without loss of resolution.

It is noteworthy that in theory, if all the scalar vector graphics files (*`svg`* files) were correctly interpreted by the polymer sequence editor, the raster vector graphics files (*`png`* files) should be totally redundant and useless. However, the *`png`* file-reading libraries are much more robust than the *`svg`* file-reading libraries (*`svg`* is a rather recent standard). This is why it is required to always provide the polymer sequence editor with a fall-back solution in the form of a raster graphics *`png`* file to be used in case the monicon rendering from the scalar vector graphics file failed.

Finally, we should mention that because the user may draw himself these small graphics files, the graphical rendering of a polymer sequence is totally customizable. For the user to be guided in this process, I would simply mention that the *`svg`* files were all drawn using the `sodipodi` software program, and that the raster *`png`* files were obtained using the "export" function in this same program.

We will see later how the correspondence between a monomer in the polymer chemistry definition and its corresponding graphics files is established, so that when the user edits a polymer sequence —by typing the monomer codes at the keyboard— the proper monicon is displayed in the **polyxedit** sequence editor.

Files acetyl.svg and acetyl.png

According to the same token as above, for monomer icon files, these two files are respectively the *`svg`* and *`png`* versions of the file that is used to graphically render the "Acetylation" monomer modification. These files are with a transparent background and the small "Ac" red text that appears on them is the only graphical element that will be visible when the files are used for compositing their contents *onto* the monomer icon file that is used to render the monomer being chemically modified using the "Acetylation" modification.

We will see later how the correspondence between a chemical modification and its graphical file is performed, so that when the user selects a monomer in the sequence editor and modifies it, the proper graphical modification of its monicon is performed in order to give the user a proper feedback that the monomer has effectively been modified.

File cursor.svg

This file is responsible for the representation of the editing cursor in the **polyxedit** module. Depending on the color of the monicons, it might be necessary to modify the graphical rendering of the cursor in the polymer sequence editor. This is necessary so that the graphical rendering of polymer chemistries during polymer sequence editing can be totally themeable. The cursor graphics file is necessarily a *`svg`* file because it *has to scale up/down properly*

when the user changes the dimension of the monicons that render the polymer sequence in the editor. The cursor always scales with the monomer icons and adopts the same dimensions as theirs.

File chempad.conf

We have already explained what this file is for. It might exist in the polymer chemistry definition directory, in which case it is used to lay out the chemical pad in the **polyxcalc** module. Note that this file is used only if **polyxcalc** is run with specifying that a polymer chemistry definition be loaded in it.

File monicons.dic

This file, also located in the "protein" polymer chemistry definition directory, contains critical correspondences between monomer codes and the graphics files used to render these monomers in the sequence editor. Also, this file lists the correspondences between the chemical modifications that might be set to monomers and the graphical operations to perform so that the user is provided with a visual feedback. Its contents are:

```
monomer;A=alanine.svg|alanine.png
monomer;C=cysteine.svg|cysteine.png
monomer;D=aspartate.svg|aspartate.png
monomer;E=glutamate.svg|glutamate.png
monomer;F=phenylalanine.svg|phenylalanine.png
⋮

modif;Phosphorylation%T%phospho.svg|phospho.png
modif;Acetylation%T%acetyl.svg|acetyl.png
modif;AmidationAsp%O%asparagine.svg|asparagine.png
modif;AmidationGlu%O%glutamine.svg|glutamine.png
```

The first line of this file is saying —"*Whenever the user wants to insert —in the polymer sequence— a monomer by keying-in* 'A'*, that monomer should be rendered using the* `alanine.svg` *file or, if that rendering fails, using the* `alanine.png` *file*". The same wording is true for all the monomers in the polymer chemistry definition.

The sixth line indicates that the monomers that are chemically modified using a modification called "Phosphorylation" should have their monicon graphically altered by compositing `%T%`ransparently (onto the monicon of the monomer being modified) either the transparent scalar vector graphics `phospho.svg` file, or —if something is wrong with this file— the raster `phospho.png` file (see the chapter 8 on page 75).

The eighth line shows another graphical compositing rule. The rule is not `%T%`ransparency, but involves an `%O%`paque graphical compositing. This line says that when a monomer is modified using an "AmidationAsp" modification, its monomer icon should be *replaced* using a monomer icon rendered *ex novo* by reading either the scalar vector graphics file `asparagine.svg`, or —if something is wrong with this file— by using the raster graphics file `asparagine.png`.

File sounds.dic

This file, located in the "protein" polymer chemistry definition directory (in the `sounds` sub-directory), contains critical correspondences between monomers' code/name or modifications' names and their corresponding sound files. The format of the file is very simple, as shown below:

```
silence-sound$silence.ogg

monomer;A=alanine.ogg|a.ogg
monomer;C=cysteine.ogg|c.ogg
monomer;D=aspartate.ogg|d.ogg
monomer;E=glutamate.ogg|e.ogg
monomer;F=phenylalanine.ogg|f.ogg
⋮

modif;Phosphorylation=phospho.ogg
modif;AmidationAsp=amidation.ogg
modif;Acetylation=acetyl.ogg
modif;AmidationGlu=amidation.ogg
```

The first line of this file is saying —*"Whenever the user wants to insert —in the polymer sequence self-speak playlist— a silent delay, that file `silence.ogg` is to be used"*.

The second line indicates that when a sequence that is speaking itself out encounters a monomer of code 'A', then the file to be used should be either:

* `alanine.ogg` if the user asks that the monomer names be vocalized in the playlist;
* `a.ogg` if the user asks that the monomer codes be vocalized in the playlist.

The same wording is true for all the other monomers in the polymer chemistry definition (see the chapter 8 on page 75).

The seventh line states that if modifications are to speak themselves out, the "Phosphorylation" modification should use the sound file `phospho.ogg`.

Polymer Sequence Sample Files

There are two polymer sequence sample files that are shipped with **GNU polyxmass-common**. We'll detail one of the two in this section.

File protein-sample.pxm

This file is a sample "protein"-polymer chemistry polymer sequence. It is shipped with **GNU polyxmass-common** in order to let the user experiment with the **GNU polyxmass** software package right after installation. This polymer sequence file is of polymer chemistry definition "protein" as can be seen from part of its contents:

```
<polseqdata>
<polseqinfo>
<type>protein</type>
```

```
<name>Sample</name>
<code>SP2003</code>
<author>rusconi</author>
<date>
<year>2004</year>
<month>01</month>
<day>19</day>
</date>
</polseqinfo>
</polseqinfo>
<polseq>
<monomer>
<code>M</code>
<prop>
<name>MODIF</name>
<data>Acetylation</data>
</prop>
</monomer>
<codes>EFEEDF</codes>
⋮
<monomer>
<code>V</code>
<prop>
<name>NOTE</name>
<data>SAMPLE-NOTE</data>
<data type="str">This monomer belongs to the KPVV
                 peptide [30-->33]</data>
</prop>
</monomer>
</polseq>
<prop>
<name>NOTE</name>
<data>COMMENT</data>
<data type="str">this polymer is partly membranous.</data>
</prop>
<prop>
<name>LEFT_END_MODIF</name>
<data>Acetylation</data>
</prop>
<prop>
<name>NOTE</name>
<data>COMMENT</data>
<data type="str">This protein is responsible
                 for the multi-drug resistance effect.</data>
</prop>
</polseqdata>
```

As can be seen here, one *xml* element, tagged "<type>" contains a datum "protein", that tells `polyxmass` that, when this polymer sequence file is loaded, it should ensure that the

"protein" polymer chemistry definition file is used to interpret its data.

Other data follow that detail what the user has put in this polymer sequence (monomer modifications —see element "<name>MODIF</name>"—, polymer modifications —see element "<name>LEFT_END_MODIF</name>"—, etc...)

Example Of A New Atom Definition

The **GNU polyxmass** package has to ensure that users can either develop their own polymer chemistry definitions or install packages that ship polymer chemistry definition files (along with their configuration files and data files; the whole set of files is collectively called the "polymer chemistry definition"). To achieve that goal, the **GNU polyxmass** software suite needs to be able to screen catalogue files on the system in search for these atom/polymer chemistry definitions.

> The user willing to understand the process that leads to the creation of a polymer chemistry definition package, can study the **GNU polyxmass-data** package that is part of the **GNU polyxmass** software suite. This package brings a number of new polymer chemistry definitions, like the "dna", "rna", "saccharide" polymer chemistry definitions. When installed, this package will make sure that catalogue files are installed in the configuration directory of the **GNU polyxmass** software suite. This way, when polyxmass is executed, it can parse these catalogue files in search for all the available polymer chemistry/atom definitions. The **GNU polyxmass-data** package is an excellent tutorial for the user willing to learn how to package polymer chemistry definitions.

Packages can either bring atom definition files or polymer chemistry definition files or both. In each case, different catalogue files are to be installed in different configuration directories. In this section we are exploring the ways to install a new atom definition. Users are — once again — invited to peruse the chapter about **GNU polyxmass** customization for detailed instructions about creating and installing new polymer chemistry definition packages.

When an atom definition package brings one or more atom definition files to the **GNU polyxmass** software suite, it should bring the equivalent of the

`polyxmass-common-atom-defs-cat`

file that is brought by the **GNU polyxmass-common** package.

Let's see an example where a new atom definition file might be of great use. Imagine that we are using mass spectrometry to fully characterize bacterially-synthesized polypeptides for use in nuclear magnetic resonance studies. These polypeptides were *almost fully* [^{15}N]-labelled by growing the bacteria in [^{15}N]-saturated culture medium. Of course, the way masses should be computed is very different than the usual way, because the isotopic $\frac{[^{14}\mathrm{N}]}{[^{15}\mathrm{N}]}$ ratio of the nitrogen element has changed dramatically from the naturally-occurring one.

How would this situation be dealt with in **GNU polyxmass**? The first action would be to create a new atom definition file, say `atoms-n-nmr.xml`, for example. This `atoms-n-nmr.xml` atom definition file would list —amongst all the other atoms— only one isotope for the nitrogen atom, the [^{15}N] isotope: its mass would thus be 15.0001089780 and its abundance would be set to 100. We may give that new atom definition the following name: "n-nmr" (see below).

How to let **GNU polyxmass** know that we may want to use this new atom definition file? We would make a package, put that file into it, name the package in a **GNU polyxmass**-consistent way, like "polyxmass-n-nmr" for example. We also would have to put in that package a file listing the name of the atom definition that should be correlated to the shipped atom definition file. This file should look like the file that we already described earlier, which is shipped with **GNU polyxmass-common**: `polyxmass-common-atom-defs-cat` (see above about the requirement that atom definition catalogues *must* have a filename ending with the `atom-defs-cat` suffix. Typically, the prefix should be the name of the package that brings that catalogue file, such as `polyxmass-common`, which thus yields the `polyxmass-common-atom-defs-cat` catalogue name). Thus we may ship a catalogue file that should be named `polyxmass-n-nmr-atom-defs-cat` listing these contents:

```
n-nmr=/usr/share/polyxmass/atom-defs/polyxmass-n-nmr-atom-def.xml
```

This `polyxmass-n-nmr-atom-defs-cat` catalogue file should be installed in the `/etc/polyxmass/atom-defs` directory, thus its absolute file name should be: `/etc/polyxmass/atom-defs/polyxmass-n-nmr-atom-defs-cat`.

When our atom definition package is installed, and polyxmass is executed, its catalogue file will be parsed and the atom definition will automatically be made available for use in the whole **GNU polyxmass** software suite.

At this point, we have to make sure that this new atom definition is used to compute masses when we are working on the polypeptides of interest (the ones that are [^{15}N]-rich); that is, we must let **GNU polyxmass** know that there exists a new notion of a polymer chemistry definition, say "n-nmr-protein", for example. As the system administrator, we can create a new polymer chemistry catalogue file, like the one we described earlier: `polyxmass-common-polchem-defs-cat`, but naming it this way, for example: `polyxmass-n-nmr-polchem-defs-cat`. These files are located in `/etc/polyxmass/polchem-defs`. The new file should contain this line (each polymer chemistry definition name and its corresponding data *must* be on a single line without space; here, for clarity the line was broken, as symbolised with the "\\" characters that are absent in the file):

```
n-nmr-protein=/usr/share/polyxmass/polchem-defs/protein/protein.xml\\
% /usr/share/polyxmass/polchem-defs/protein
```

What this line says is that there now exists a new polymer chemistry definition, named "n-nmr-protein", that uses a pre-existing polymer chemistry definition named "protein".

When our new polymer chemistry definition catalogue file is installed, and that polyxmass is run, it will parse that catalogue file along with all the other ones and will thus aknowledge that the "n-nmr-protein" polymer chemistry definition should use the polymer chemistry definition data located in the directory mentioned on the line above.

Now comes the really interesting configuration: we have to let **GNU polyxmass** know that, whenever a polymer chemistry definition "nmr-protein" is used, the atom definition "n-nmr" is to be used in order to compute masses.

To do that we have to create, as root, a new dictionary file, similar to the one that we have described earlier:

`polyxmass-common-polchem-defs-atom-defs-dic`, but naming it this way, for example: `n-nmr-protein-polchem-defs-atom-defs-dic`. These files are located in `/etc/polyxmass/polchem-defs`. That file should contain this line:

```
n-mnr-protein=n-nmr
```

When our new polymer chemistry definition/atom definition dictionary file is installed, this new dictionary file will be parsed by polyxmass and it will thus be known that when using the "n-nmr-protein" polymer chemistry definition, the atom definition "n-nmr" should be used for any computation.

The last action that we should take in order to automatically compute the masses in the "n-nmr"-specialized way we want, is to tell the polypeptide sequences we are working on that they are of polymer chemistry definition "n-nmr-protein". To that end, make a copy of the polymer sequence of interest and change, using a text editor, the contents of the `<type>` element (that is "protein") to "n-nmr-protein". Open that new sequence file in a freshly started **GNU polyxmass** program, and the masses should be computed with the new atom definitions.

That's the end of the story here.

Conclusion

In this chapter, we have described what file-system hierarchy governs the **GNU polyxmass** understanding of different polymer chemistries. The described set of data/configuration files (and directories) is the minimal set of information that is required for **GNU polyxmass** to operate.

The user willing to learn how to create brand new packages that bring to the user new atom definitions and/or new polymer chemistry definitions is invited to carefully study the **GNU polyxmass-data** package, that is optional in the **GNU polyxmass** software suite.

The data/configuration files that are brought by both packages (**GNU polyxmass-common** and **GNU polyxmass-data**) are installed —by default— in system directories, like `/usr`, `/etc`, `/usr/local`... and are thus available to all the users gaining access to the system. **GNU polyxmass** manages a creativity space for the individual user to add atom definitions and/or polymer chemistry definitions for his own use *exclusively*.

The next chapter describes in detail the process leading to the customization of the **GNU polyxmass** software suite, by going through the example of adding the "saccharide" polymer chemistry definition to the **GNU polyxmass** software suite for individual use (not globally available to the system).

10

GNU polyxmass Customization: Adding Atom and Chemistry Definitions

This chapter will guide the user through a step-by-step procedure to learn how to customize the **GNU polyxmass** software suite by adding atom definitions and polymer chemistry definitions for a *local and individual* use.

Getting The Substrate Of Our Experiment

The example that we will make will be based on part of the contents of the **GNU polyxmass-data** package (the "saccharide" polymer chemistry definition), that the user is invited to download *without installing it*. The best way to go is to download the "source tarball": the file `polyxmass-data-0.8.5.tar.gz` and perform the following procedure to unpack it in a suitable place:

* Copy that file to a directory where you have write permissions:[1]

```
mkdir /home/rusconi/tmp ↩
cp polyxmass-data-0.8.5.tar.gz /home/rusconi/tmp ↩
```

* Unpack the *tar.gz* archive into the current working directory:

```
tar xvzf polyxmass-data-0.8.5.tar.gz ↩
```

* Now go to the newly created directory:

```
cd polyxmass-data-0.8.5 ↩
```

* Check that there is effectively the directory with the "saccharide" polymer chemistry definition data:

```
ls polchem-defs/saccharide ↩
```

  You should get a rather large listing of files, and if you look well at them you'll see two files of interest: `saccharide.xml` and `monicons.dic`. Also, a `sounds` directory should contain empty sound files (filename extensions are `.ogg`) and a dictionary file (`sounds.dic`). Great, we have the substrate with which we will demonstrate how to bring new polymer chemistry definitions to the **GNU polyxmass** software suite!

Creating A New Polymer Chemistry Definition

At this point we know we can go on with our procedure. We should immediately create a directory where we will install the "saccharide" polymer chemistry definition-related files:

```
mkdir /home/rusconi/pxm-custom ↩
```

Now we can copy the "saccharide" polymer chemistry definition directory in our juste-made directory (we are still in the `polyxmass-data-0.8.5` directory):

[1] I usually work by making a tmp directory in my home directory and copying there the files that correspond to temporary tasks; thus our tarball file would have the following absolute name: `/home/rusconi/tmp/polyxmass-data-0.8.5.tar.gz`.

```
cp -rpf polchem-defs/saccharide /home/rusconi/pxm-custom ↩
```

Right, we now have our new "saccharide" polymer chemistry definition data; we could have made them from scratch, but we happen to be lazy so we copied them from an available package... We now have to start the configuration process that will let **GNU polyxmass** know that we have brought one more polymer chemistry definition. To do that, we'll have to create (if it does not exist already) the `.polyxmass` directory in our home directory.[2]

Also, we should make the `polchem-defs` directory inside it. So these are the command lines:

```
mkdir /home/rusconi/.polyxmass ↩
mkdir /home/rusconi/.polyxmass/polchem-defs ↩
```

At this point, we should mimick the file-system hierarchy that we have already described for the `/etc/polyxmass` directory, by creating a `rusconi-`*`polchem-defs-cat`* where we'll add the line corresponding to the new "saccharide" polymer chemistry definition, as we have seen in the chapter about **GNU polyxmass-common** (note the compulsory *polchem-defs-cat* suffix in the catalogue filename):

```
cd /home/rusconi/.polyxmass/polchem-defs ↩
touch rusconi-polchem-defs-cat ↩
```

Now use an editor to put the following line in this file (each polymer chemistry definition name and its corresponding data *must* be on a single line without space; here, for clarity the line was broken, as symbolised with the "\\" characters that are absent in the file). *Remember to terminate the text line with a carriage return, symbolized with '↩' below!*:

```
saccharide=/home/rusconi/pxm-custom/saccharide/saccharide.xml\\
% /home/rusconi/pxm-custom/saccharide ↩
```

If we started **GNU polyxmass** now, we would be able to partially enjoy the result: when trying to load the new polymer chemistry definition in **polyxdef**, we would get the file-chooser window shown on Figure 10.1.

However, if we were courageous enough to select "saccharide" from the listview and click the Validate button, we would find an error because **GNU polyxmass** does not yet know what atom definition should be used with this polymer chemistry definition. The error message that is sent to the terminal window is the following:

```
** (polyxmass:14734): CRITICAL **: polyxdef-ui-polchemdef.c@225:
failed setting the proper atom definition name for current
polymer definition to combo list
```

[2]The home directory, in *UNIX* systems, is the directory where the user finds himself once logged onto the system. Usually, it is something like `/home/`*`logname`*, with *logname* being the username used to log onto the system (for me that's *rusconi*). Furthermore, the home directory of a user is visible in the `/etc/passwd` file, which lists it for the user *rusconi* in this way (try for yourself):

```
rusconi:x:1000:1000:Filippo Rusconi:/home/rusconi:/bin/bash
```

.

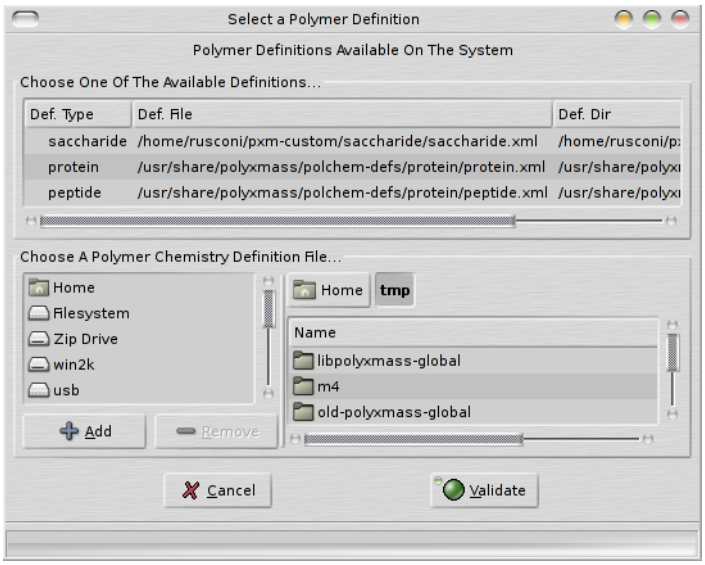

Figure 10.1: **The new polymer chemistry definition** The new polymer chemistry definition "saccharide" is now known to the system, since we have edited the `/home/rusconi/.polyxmass/polchem-defs/rusconi-polchem-defs-cat` catalogue file to that precise end.

Creating A New Atom Definition

What this error message is saying, is that the program could not tell what atom definition was to be loaded for us to correctly display (and supposedly edit) the polymer chemistry definition that was selected. That means that we still have some work to do: we still have to provide an atom definition and register it to **GNU polyxmass**.

We could —of course— just tell **GNU polyxmass** that the atom definition to be used when working with the "saccharide" polymer chemistry definition is the "basic" atom definition that is shipped with the **GNU polyxmass-common** package. But since we are making a tutorial hands-on procedure, we'll fake the system into thinking that the "saccharide" polymer chemistry definition requires a specific atom definition called "mysugar". But we do not want to write that atom definition file *ex nihilo*, so we'll just make a copy of the `atoms.xml` atom definition file that was shipped with the **GNU polyxmass-common** package. On my system, the **GNU polyxmass-common** package was installed in the `/usr` system directory tree, so I'll find that `atoms.xml` file there, and copy it to `/home/rusconi/pxm-custom` , like this (on a *single* line, please):

```
cp /usr/share/polyxmass/atom-defs/atoms.xml
            /home/rusconi/pxm-custom/for-sugar.xml ↩
```

At this point we'll have to inform the **GNU polyxmass** software suite that we have a new atom definition, named "mysugar", of which the file is
`/home/rusconi/pxm-custom/for-sugar.xml`.

For this, we'll have to continue mimicking the file-system hierarchy that we have already

described for the `/etc/polyxmass` directory, by creating an `atom-defs` directory in the user's `/home/rusconi/.polyxmass` directory. Inside this directory, we will have to create the `rusconi-`*`atom-defs-cat`* file where we'll add the line corresponding to our new "mysugar" atom definition, as we have seen in the chapter about **GNU polyxmass-common** (note the compulsory *atom-defs-cat* suffix in the catalogue filename):

```
mkdir /home/rusconi/.polyxmass/atom-defs ↩
cd /home/rusconi/.polyxmass/atom-defs ↩
touch rusconi-atom-defs-cat ↩
```

Now we have to use an editor, so that we can put the following line in this file. *Remember to terminate the line with a carriage return, symbolized with '↩' below!*

```
mysugar=/home/rusconi/pxm-custom/for-sugar.xml ↩
```

The Polymer Chemistry Definition–Atom Definition Dictionary

At this point, we have told **GNU polyxmass** that we have a new atom definition file, but we still have to make sure this atom definition is actually loaded each time a "saccharide" polymer chemistry definition is used. For this, we already know that we have to edit a dictionary file so that we can make the correspondence between the polymer chemistry definition and the atom definition. This dictionary file should be located, along with the polymer chemistry definition files, in `/home/rusconi/.polyxmass/polchem-defs`, and be named, as in `/etc/polyxmass/polchem-defs`, `rusconi-`*`polchem-defs-atom-defs-dic`* (note the compulsory *polchem-defs-atom-defs-dic* suffix in the dictionary filename):

```
cd /home/rusconi/.polyxmass/polchem-defs/
touch rusconi-polchem-defs-atom-defs-dic ↩
```

Now use an editor to put the following line in this file. *Remember to terminate the line with a carriage return, symbolized with '↩' below!*:

```
saccharide=mysugar ↩
```

Enjoying The New Polymer Chemistry Definition

At this point, the user has successfully configured the new polymer chemistry definition and can start enjoying it: if we try to open, in a new `polyxmass` session (`polyxmass` needs to be restarted because all the files we have been configuring are read at startup), the "saccharide" polymer chemistry definition, we can do it successfully, as shown on Figure 10.2.

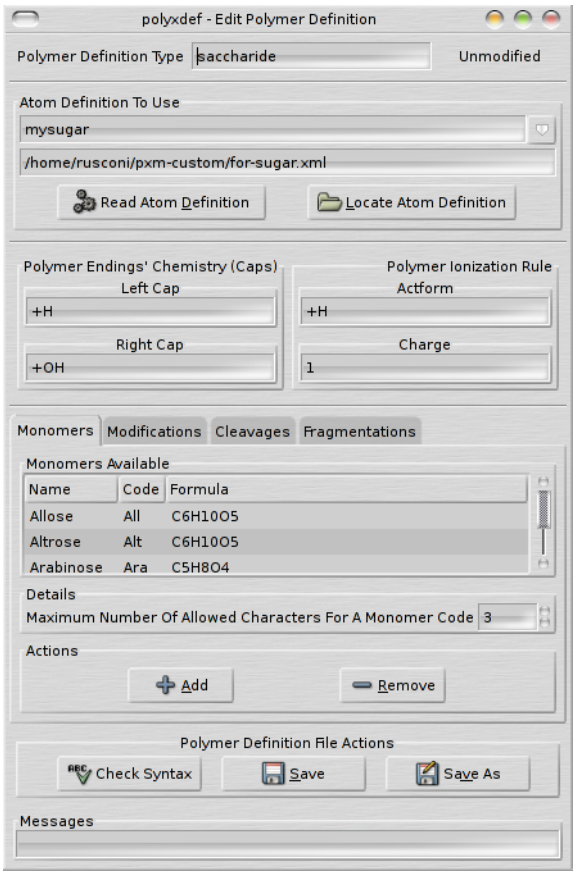

Figure 10.2: **Loading the newly installed polymer chemistry definition** Finally the polymer chemistry definition "saccharide" can be opened from disk using the **polyxdef** module, as all the configurations were performed. Note the Atom Definition To Use data at the top of the window, which indicate with what atom definition this polymer chemistry definition will work.

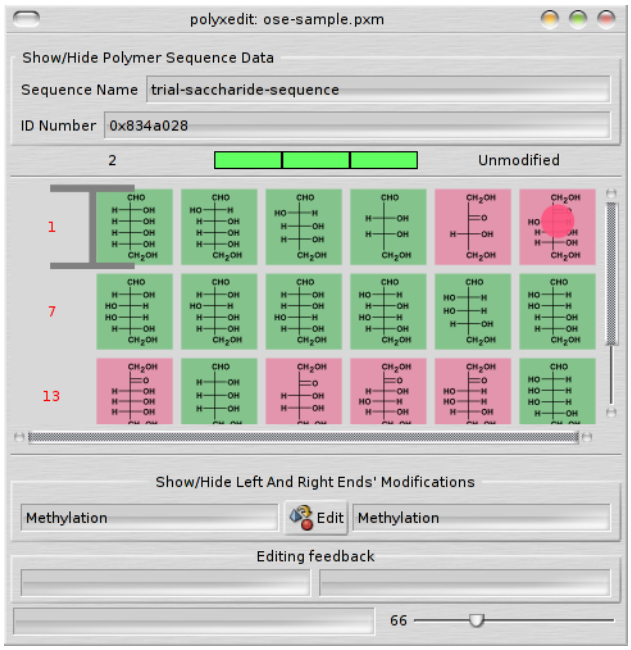

Figure 10.3: **Loading a "saccharidic" polymer sequence** Since the "saccharide" polymer chemistry definition is fully configured, we can load from disk a polymer sequence of that chemistry definition.

Finally, since we have an example "saccharide" sequence in the data that we initially unpacked in `/home/rusconi/tmp/polyxmass-data-0.8.5`, we can try to open the "saccharidic" polymer sequence file `pol-seqs/ose-sample.pxm`. That works perfectly, as shown on Figure 10.3.

11

Appendices

The "basic" Atom Definition File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- DTD for atom definitions, used by the
'GNU polyxmass' suite of mass spectrometry applications.
Copyright 2003, 2004 Filippo Rusconi - Licensed under the GNU GPL -->
<!DOCTYPE atomdefdata [
<!ELEMENT atomdefdata (atom+)>
<!ELEMENT atom (name,symbol,isotope+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT symbol (#PCDATA)>
<!ELEMENT isotope (mass , abund)>
<!ELEMENT mass (#PCDATA)>
<!ELEMENT abund (#PCDATA)>
]>
<atomdefdata>
  <atom>
    <name>Hydrogen</name>
    <symbol>H</symbol>
    <isotope>
      <mass>1.0078250370</mass>
      <abund>99.9885000000</abund>
    </isotope>
    <isotope>
      <mass>2.0141017870</mass>
      <abund>0.0115000000</abund>
    </isotope>
```

```
</atom>
<atom>
  <name>Helium</name>
  <symbol>He</symbol>
  <isotope>
    <mass>4.0026032500</mass>
    <abund>99.9998600000</abund>
  </isotope>
  <isotope>
    <mass>3.0160292970</mass>
    <abund>0.0001400000</abund>
  </isotope>
</atom>
<atom>
  <name>Lithium</name>
  <symbol>Li</symbol>
  <isotope>
    <mass>7.0160045000</mass>
    <abund>92.4100000000</abund>
  </isotope>
  <isotope>
    <mass>6.0151232000</mass>
    <abund>7.5900000000</abund>
  </isotope>
</atom>
<atom>
  <name>Beryllium</name>
  <symbol>Be</symbol>
  <isotope>
    <mass>9.0121825000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Bore</name>
  <symbol>B</symbol>
  <isotope>
    <mass>11.0093053000</mass>
    <abund>80.1000000000</abund>
  </isotope>
  <isotope>
    <mass>10.0129380000</mass>
    <abund>19.9000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Carbon</name>
  <symbol>C</symbol>
  <isotope>
    <mass>12.0000000000</mass>
```

```
    <abund>98.9300000000</abund>
  </isotope>
  <isotope>
    <mass>13.0033548390</mass>
    <abund>1.0700000000</abund>
  </isotope>
</atom>
<atom>
  <name>Nitrogen</name>
  <symbol>N</symbol>
  <isotope>
    <mass>14.0030740080</mass>
    <abund>99.6320000000</abund>
  </isotope>
  <isotope>
    <mass>15.0001089780</mass>
    <abund>0.3680000000</abund>
  </isotope>
</atom>
<atom>
  <name>Oxygen</name>
  <symbol>O</symbol>
  <isotope>
    <mass>15.9949146400</mass>
    <abund>99.7570000000</abund>
  </isotope>
  <isotope>
    <mass>16.9991306000</mass>
    <abund>0.0380000000</abund>
  </isotope>
  <isotope>
    <mass>17.9991593900</mass>
    <abund>0.2050000000</abund>
  </isotope>
</atom>
<atom>
  <name>Fluorine</name>
  <symbol>F</symbol>
  <isotope>
    <mass>18.9984032500</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Neon</name>
  <symbol>Ne</symbol>
  <isotope>
    <mass>19.9924391000</mass>
    <abund>90.4800000000</abund>
  </isotope>
```

```
    <isotope>
      <mass>20.9938453000</mass>
      <abund>0.2700000000</abund>
    </isotope>
    <isotope>
      <mass>21.9913837000</mass>
      <abund>9.2500000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Sodium</name>
    <symbol>Na</symbol>
    <isotope>
      <mass>22.9897697000</mass>
      <abund>100.0000000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Magnesium</name>
    <symbol>Mg</symbol>
    <isotope>
      <mass>23.9850450000</mass>
      <abund>78.9900000000</abund>
    </isotope>
    <isotope>
      <mass>24.9858392000</mass>
      <abund>10.0000000000</abund>
    </isotope>
    <isotope>
      <mass>25.9825954000</mass>
      <abund>11.0100000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Aluminium</name>
    <symbol>Al</symbol>
    <isotope>
      <mass>26.9815413000</mass>
      <abund>100.0000000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Silicon</name>
    <symbol>Si</symbol>
    <isotope>
      <mass>27.9769284000</mass>
      <abund>92.2297000000</abund>
    </isotope>
    <isotope>
      <mass>28.9764964000</mass>
```

```
    <abund>4.6832000000</abund>
  </isotope>
  <isotope>
    <mass>29.9737717000</mass>
    <abund>3.0872000000</abund>
  </isotope>
</atom>
<atom>
  <name>Phosphorus</name>
  <symbol>P</symbol>
  <isotope>
    <mass>30.9737634000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Sulfur</name>
  <symbol>S</symbol>
  <isotope>
    <mass>31.9720718000</mass>
    <abund>94.9300000000</abund>
  </isotope>
  <isotope>
    <mass>32.9714591000</mass>
    <abund>0.7600000000</abund>
  </isotope>
  <isotope>
    <mass>33.9678677400</mass>
    <abund>4.2900000000</abund>
  </isotope>
  <isotope>
    <mass>35.9670790000</mass>
    <abund>0.0200000000</abund>
  </isotope>
</atom>
<atom>
  <name>Chlorine</name>
  <symbol>Cl</symbol>
  <isotope>
    <mass>34.9688527290</mass>
    <abund>75.7800000000</abund>
  </isotope>
  <isotope>
    <mass>36.9659026240</mass>
    <abund>24.2200000000</abund>
  </isotope>
</atom>
<atom>
  <name>Argon</name>
  <symbol>Ar</symbol>
```

```
    <isotope>
      <mass>35.9675456050</mass>
      <abund>0.3365000000</abund>
    </isotope>
    <isotope>
      <mass>37.9627322000</mass>
      <abund>0.0632000000</abund>
    </isotope>
    <isotope>
      <mass>39.9623831000</mass>
      <abund>99.6003000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Potassium</name>
    <symbol>K</symbol>
    <isotope>
      <mass>38.9637079000</mass>
      <abund>93.2581000000</abund>
    </isotope>
    <isotope>
      <mass>39.9639988000</mass>
      <abund>0.0117000000</abund>
    </isotope>
    <isotope>
      <mass>40.9618254000</mass>
      <abund>6.7302000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Calcium</name>
    <symbol>Ca</symbol>
    <isotope>
      <mass>39.9625907000</mass>
      <abund>96.9410000000</abund>
    </isotope>
    <isotope>
      <mass>41.9586218000</mass>
      <abund>0.6470000000</abund>
    </isotope>
    <isotope>
      <mass>42.9587704000</mass>
      <abund>0.1350000000</abund>
    </isotope>
    <isotope>
      <mass>43.9554848000</mass>
      <abund>2.0860000000</abund>
    </isotope>
    <isotope>
      <mass>45.9536890000</mass>
```

```
    <abund>0.0040000000</abund>
  </isotope>
  <isotope>
    <mass>47.9525320000</mass>
    <abund>0.1870000000</abund>
  </isotope>
</atom>
<atom>
  <name>Scandium</name>
  <symbol>Sc</symbol>
  <isotope>
    <mass>44.9559136000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Titanium</name>
  <symbol>Ti</symbol>
  <isotope>
    <mass>45.9526327000</mass>
    <abund>8.2500000000</abund>
  </isotope>
  <isotope>
    <mass>46.9517649000</mass>
    <abund>7.4400000000</abund>
  </isotope>
  <isotope>
    <mass>47.9479467000</mass>
    <abund>73.7200000000</abund>
  </isotope>
  <isotope>
    <mass>48.9478705000</mass>
    <abund>5.4100000000</abund>
  </isotope>
  <isotope>
    <mass>49.9447858000</mass>
    <abund>5.1800000000</abund>
  </isotope>
</atom>
<atom>
  <name>Vanadium</name>
  <symbol>V</symbol>
  <isotope>
    <mass>49.9471613000</mass>
    <abund>0.2500000000</abund>
  </isotope>
  <isotope>
    <mass>50.9439625000</mass>
    <abund>99.7500000000</abund>
  </isotope>
```

```
</atom>
<atom>
  <name>Chromium</name>
  <symbol>Cr</symbol>
  <isotope>
    <mass>49.9464630000</mass>
    <abund>4.3450000000</abund>
  </isotope>
  <isotope>
    <mass>51.9405097000</mass>
    <abund>83.7890000000</abund>
  </isotope>
  <isotope>
    <mass>52.9406510000</mass>
    <abund>9.5010000000</abund>
  </isotope>
  <isotope>
    <mass>53.9388822000</mass>
    <abund>2.3650000000</abund>
  </isotope>
</atom>
<atom>
  <name>Manganese</name>
  <symbol>Mn</symbol>
  <isotope>
    <mass>54.9380463000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Iron</name>
  <symbol>Fe</symbol>
  <isotope>
    <mass>53.9396121000</mass>
    <abund>5.8450000000</abund>
  </isotope>
  <isotope>
    <mass>55.9349393000</mass>
    <abund>91.7540000000</abund>
  </isotope>
  <isotope>
    <mass>56.9353957000</mass>
    <abund>2.1190000000</abund>
  </isotope>
  <isotope>
    <mass>57.9332778000</mass>
    <abund>0.2820000000</abund>
  </isotope>
</atom>
<atom>
```

```
  <name>Cobalt</name>
  <symbol>Co</symbol>
  <isotope>
    <mass>58.9331978000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Nickel</name>
  <symbol>Ni</symbol>
  <isotope>
    <mass>57.9353471000</mass>
    <abund>68.0769000000</abund>
  </isotope>
  <isotope>
    <mass>59.9307890000</mass>
    <abund>26.2231000000</abund>
  </isotope>
  <isotope>
    <mass>60.9310586000</mass>
    <abund>1.1399000000</abund>
  </isotope>
  <isotope>
    <mass>61.9283464000</mass>
    <abund>3.6345000000</abund>
  </isotope>
  <isotope>
    <mass>63.9279680000</mass>
    <abund>0.9256000000</abund>
  </isotope>
</atom>
<atom>
  <name>Copper</name>
  <symbol>Cu</symbol>
  <isotope>
    <mass>62.9295992000</mass>
    <abund>69.1700000000</abund>
  </isotope>
  <isotope>
    <mass>64.9277924000</mass>
    <abund>30.8300000000</abund>
  </isotope>
</atom>
<atom>
  <name>Zinc</name>
  <symbol>Zn</symbol>
  <isotope>
    <mass>63.9291454000</mass>
    <abund>48.6300000000</abund>
  </isotope>
```

```
    <isotope>
      <mass>65.9260352000</mass>
      <abund>27.9000000000</abund>
    </isotope>
    <isotope>
      <mass>66.9271289000</mass>
      <abund>4.1000000000</abund>
    </isotope>
    <isotope>
      <mass>67.9248458000</mass>
      <abund>18.7500000000</abund>
    </isotope>
    <isotope>
      <mass>69.9253249000</mass>
      <abund>0.6200000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Gallium</name>
    <symbol>Ga</symbol>
    <isotope>
      <mass>68.9255809000</mass>
      <abund>60.1080000000</abund>
    </isotope>
    <isotope>
      <mass>70.9247006000</mass>
      <abund>39.8920000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Germanium</name>
    <symbol>Ge</symbol>
    <isotope>
      <mass>69.9242498000</mass>
      <abund>20.8400000000</abund>
    </isotope>
    <isotope>
      <mass>71.9220800000</mass>
      <abund>27.5400000000</abund>
    </isotope>
    <isotope>
      <mass>72.9234639000</mass>
      <abund>7.7300000000</abund>
    </isotope>
    <isotope>
      <mass>73.9211788000</mass>
      <abund>36.2800000000</abund>
    </isotope>
    <isotope>
      <mass>75.9214027000</mass>
```

```
    <abund>7.6100000000</abund>
  </isotope>
</atom>
<atom>
  <name>Arsenic</name>
  <symbol>As</symbol>
  <isotope>
    <mass>74.9215955000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Selenium</name>
  <symbol>Se</symbol>
  <isotope>
    <mass>73.9224771000</mass>
    <abund>0.8900000000</abund>
  </isotope>
  <isotope>
    <mass>75.9192066000</mass>
    <abund>9.3700000000</abund>
  </isotope>
  <isotope>
    <mass>76.9199077000</mass>
    <abund>7.6300000000</abund>
  </isotope>
  <isotope>
    <mass>77.9173040000</mass>
    <abund>23.7700000000</abund>
  </isotope>
  <isotope>
    <mass>79.9165205000</mass>
    <abund>49.6100000000</abund>
  </isotope>
  <isotope>
    <mass>81.9167090000</mass>
    <abund>8.7300000000</abund>
  </isotope>
</atom>
<atom>
  <name>Bromine</name>
  <symbol>Br</symbol>
  <isotope>
    <mass>78.9183361000</mass>
    <abund>50.6900000000</abund>
  </isotope>
  <isotope>
    <mass>80.9162900000</mass>
    <abund>49.3100000000</abund>
  </isotope>
```

```
</atom>
<atom>
  <name>Krypton</name>
  <symbol>Kr</symbol>
  <isotope>
    <mass>77.9203970000</mass>
    <abund>0.3500000000</abund>
  </isotope>
  <isotope>
    <mass>79.9163750000</mass>
    <abund>2.2800000000</abund>
  </isotope>
  <isotope>
    <mass>81.9134830000</mass>
    <abund>11.5800000000</abund>
  </isotope>
  <isotope>
    <mass>82.9141340000</mass>
    <abund>11.4900000000</abund>
  </isotope>
  <isotope>
    <mass>83.9115064000</mass>
    <abund>57.0000000000</abund>
  </isotope>
  <isotope>
    <mass>85.9106140000</mass>
    <abund>17.3000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Rubidium</name>
  <symbol>Rb</symbol>
  <isotope>
    <mass>84.9117996000</mass>
    <abund>72.1700000000</abund>
  </isotope>
  <isotope>
    <mass>86.9091836000</mass>
    <abund>27.8300000000</abund>
  </isotope>
</atom>
<atom>
  <name>Strontium</name>
  <symbol>Sr</symbol>
  <isotope>
    <mass>83.9134280000</mass>
    <abund>0.5600000000</abund>
  </isotope>
  <isotope>
    <mass>85.9092732000</mass>
```

```
    <abund>9.8600000000</abund>
  </isotope>
  <isotope>
    <mass>86.9088902000</mass>
    <abund>7.0000000000</abund>
  </isotope>
  <isotope>
    <mass>87.9056249000</mass>
    <abund>82.5800000000</abund>
  </isotope>
</atom>
<atom>
  <name>Yttrium</name>
  <symbol>Y</symbol>
  <isotope>
    <mass>88.9058560000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Zirconium</name>
  <symbol>Zr</symbol>
  <isotope>
    <mass>89.9047080000</mass>
    <abund>51.4500000000</abund>
  </isotope>
  <isotope>
    <mass>90.9056442000</mass>
    <abund>11.2200000000</abund>
  </isotope>
  <isotope>
    <mass>91.9050392000</mass>
    <abund>17.1500000000</abund>
  </isotope>
  <isotope>
    <mass>93.9063191000</mass>
    <abund>17.3800000000</abund>
  </isotope>
  <isotope>
    <mass>95.9082720000</mass>
    <abund>2.8000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Niobium</name>
  <symbol>Nb</symbol>
  <isotope>
    <mass>92.9063780000</mass>
    <abund>100.0000000000</abund>
  </isotope>
```

```
</atom>
<atom>
  <name>Molybdenum</name>
  <symbol>Mo</symbol>
  <isotope>
    <mass>91.9068090000</mass>
    <abund>14.8400000000</abund>
  </isotope>
  <isotope>
    <mass>93.9050862000</mass>
    <abund>9.2500000000</abund>
  </isotope>
  <isotope>
    <mass>94.9058379000</mass>
    <abund>15.9200000000</abund>
  </isotope>
  <isotope>
    <mass>95.9046755000</mass>
    <abund>16.6800000000</abund>
  </isotope>
  <isotope>
    <mass>96.9060179000</mass>
    <abund>9.5500000000</abund>
  </isotope>
  <isotope>
    <mass>97.9054050000</mass>
    <abund>24.1300000000</abund>
  </isotope>
  <isotope>
    <mass>99.9074730000</mass>
    <abund>9.6300000000</abund>
  </isotope>
</atom>
<atom>
  <name>Rutenium</name>
  <symbol>Ru</symbol>
  <isotope>
    <mass>95.9075960000</mass>
    <abund>5.5400000000</abund>
  </isotope>
  <isotope>
    <mass>97.9052870000</mass>
    <abund>1.8700000000</abund>
  </isotope>
  <isotope>
    <mass>98.9059371000</mass>
    <abund>12.7600000000</abund>
  </isotope>
  <isotope>
    <mass>99.9042175000</mass>
```

```
      <abund>12.6000000000</abund>
    </isotope>
    <isotope>
      <mass>100.9055808000</mass>
      <abund>17.0600000000</abund>
    </isotope>
    <isotope>
      <mass>101.9043475000</mass>
      <abund>31.5500000000</abund>
    </isotope>
    <isotope>
      <mass>103.9054220000</mass>
      <abund>18.6200000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Rhodium</name>
    <symbol>Rh</symbol>
    <isotope>
      <mass>102.9055030000</mass>
      <abund>100.0000000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Palladium</name>
    <symbol>Pd</symbol>
    <isotope>
      <mass>101.9056090000</mass>
      <abund>1.0200000000</abund>
    </isotope>
    <isotope>
      <mass>103.9040260000</mass>
      <abund>11.1400000000</abund>
    </isotope>
    <isotope>
      <mass>104.9050750000</mass>
      <abund>22.3300000000</abund>
    </isotope>
    <isotope>
      <mass>105.9034750000</mass>
      <abund>27.3300000000</abund>
    </isotope>
    <isotope>
      <mass>107.9038940000</mass>
      <abund>26.4600000000</abund>
    </isotope>
    <isotope>
      <mass>109.9051690000</mass>
      <abund>11.7200000000</abund>
    </isotope>
```

```
</atom>
<atom>
  <name>Silver</name>
  <symbol>Ag</symbol>
  <isotope>
    <mass>106.9050950000</mass>
    <abund>51.8390000000</abund>
  </isotope>
  <isotope>
    <mass>108.9047540000</mass>
    <abund>48.1610000000</abund>
  </isotope>
</atom>
<atom>
  <name>Cadmium</name>
  <symbol>Cd</symbol>
  <isotope>
    <mass>105.9064610000</mass>
    <abund>1.2500000000</abund>
  </isotope>
  <isotope>
    <mass>107.9041860000</mass>
    <abund>0.8900000000</abund>
  </isotope>
  <isotope>
    <mass>109.9030007000</mass>
    <abund>12.4900000000</abund>
  </isotope>
  <isotope>
    <mass>110.9041820000</mass>
    <abund>12.8000000000</abund>
  </isotope>
  <isotope>
    <mass>111.9027614000</mass>
    <abund>24.1300000000</abund>
  </isotope>
  <isotope>
    <mass>112.9044013000</mass>
    <abund>12.2200000000</abund>
  </isotope>
  <isotope>
    <mass>113.9033607000</mass>
    <abund>28.7300000000</abund>
  </isotope>
  <isotope>
    <mass>115.9047580000</mass>
    <abund>7.4900000000</abund>
  </isotope>
</atom>
<atom>
```

```
  <name>Indium</name>
  <symbol>In</symbol>
  <isotope>
    <mass>112.9040560000</mass>
    <abund>4.2900000000</abund>
  </isotope>
  <isotope>
    <mass>114.9038750000</mass>
    <abund>95.7100000000</abund>
  </isotope>
</atom>
<atom>
  <name>Tin</name>
  <symbol>Sn</symbol>
  <isotope>
    <mass>111.9048230000</mass>
    <abund>0.9700000000</abund>
  </isotope>
  <isotope>
    <mass>113.9027810000</mass>
    <abund>0.6600000000</abund>
  </isotope>
  <isotope>
    <mass>114.9033441000</mass>
    <abund>0.3400000000</abund>
  </isotope>
  <isotope>
    <mass>115.9017435000</mass>
    <abund>14.5400000000</abund>
  </isotope>
  <isotope>
    <mass>116.9029536000</mass>
    <abund>7.6800000000</abund>
  </isotope>
  <isotope>
    <mass>117.9016066000</mass>
    <abund>24.2200000000</abund>
  </isotope>
  <isotope>
    <mass>118.9033102000</mass>
    <abund>8.5900000000</abund>
  </isotope>
  <isotope>
    <mass>119.9021990000</mass>
    <abund>32.5800000000</abund>
  </isotope>
  <isotope>
    <mass>121.9034400000</mass>
    <abund>4.6300000000</abund>
  </isotope>
```

```
    <isotope>
      <mass>123.9052710000</mass>
      <abund>5.7900000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Antimony</name>
    <symbol>Sb</symbol>
    <isotope>
      <mass>120.9038237000</mass>
      <abund>57.2100000000</abund>
    </isotope>
    <isotope>
      <mass>122.9042220000</mass>
      <abund>42.7900000000</abund>
    </isotope>
  </atom>
  <atom>
    <name>Tellurium</name>
    <symbol>Te</symbol>
    <isotope>
      <mass>119.9040210000</mass>
      <abund>0.0900000000</abund>
    </isotope>
    <isotope>
      <mass>121.9030550000</mass>
      <abund>2.5500000000</abund>
    </isotope>
    <isotope>
      <mass>122.9042780000</mass>
      <abund>0.8900000000</abund>
    </isotope>
    <isotope>
      <mass>123.9028250000</mass>
      <abund>4.7400000000</abund>
    </isotope>
    <isotope>
      <mass>124.9044350000</mass>
      <abund>7.0700000000</abund>
    </isotope>
    <isotope>
      <mass>125.9033100000</mass>
      <abund>18.8400000000</abund>
    </isotope>
    <isotope>
      <mass>127.9044640000</mass>
      <abund>31.7400000000</abund>
    </isotope>
    <isotope>
      <mass>129.9062290000</mass>
```

```
    <abund>34.0800000000</abund>
  </isotope>
</atom>
<atom>
  <name>Iodine</name>
  <symbol>I</symbol>
  <isotope>
    <mass>126.9044770000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Xenon</name>
  <symbol>Xe</symbol>
  <isotope>
    <mass>123.9061200000</mass>
    <abund>0.0900000000</abund>
  </isotope>
  <isotope>
    <mass>125.9042810000</mass>
    <abund>0.0900000000</abund>
  </isotope>
  <isotope>
    <mass>127.9035308000</mass>
    <abund>1.9200000000</abund>
  </isotope>
  <isotope>
    <mass>128.9047801000</mass>
    <abund>26.4400000000</abund>
  </isotope>
  <isotope>
    <mass>129.9035095000</mass>
    <abund>4.0800000000</abund>
  </isotope>
  <isotope>
    <mass>130.9050760000</mass>
    <abund>21.1800000000</abund>
  </isotope>
  <isotope>
    <mass>131.9041480000</mass>
    <abund>26.8900000000</abund>
  </isotope>
  <isotope>
    <mass>133.9053950000</mass>
    <abund>10.4400000000</abund>
  </isotope>
  <isotope>
    <mass>135.9072190000</mass>
    <abund>8.8700000000</abund>
  </isotope>
```

```
</atom>
<atom>
  <name>Caesium</name>
  <symbol>Cs</symbol>
  <isotope>
    <mass>132.9054330000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Barium</name>
  <symbol>Ba</symbol>
  <isotope>
    <mass>129.9062770000</mass>
    <abund>0.1060000000</abund>
  </isotope>
  <isotope>
    <mass>131.9050420000</mass>
    <abund>0.1010000000</abund>
  </isotope>
  <isotope>
    <mass>133.9044900000</mass>
    <abund>2.4170000000</abund>
  </isotope>
  <isotope>
    <mass>134.9056680000</mass>
    <abund>6.5920000000</abund>
  </isotope>
  <isotope>
    <mass>135.9045560000</mass>
    <abund>7.8540000000</abund>
  </isotope>
  <isotope>
    <mass>136.9058160000</mass>
    <abund>11.2320000000</abund>
  </isotope>
  <isotope>
    <mass>137.9052360000</mass>
    <abund>71.6980000000</abund>
  </isotope>
</atom>
<atom>
  <name>Lanthanium</name>
  <symbol>La</symbol>
  <isotope>
    <mass>137.9071140000</mass>
    <abund>0.0900000000</abund>
  </isotope>
  <isotope>
    <mass>138.9063550000</mass>
```

```
    <abund>99.9100000000</abund>
  </isotope>
</atom>
<atom>
  <name>Gold</name>
  <symbol>Au</symbol>
  <isotope>
    <mass>196.9665600000</mass>
    <abund>100.0000000000</abund>
  </isotope>
</atom>
<atom>
  <name>Mercury</name>
  <symbol>Hg</symbol>
  <isotope>
    <mass>195.9658120000</mass>
    <abund>0.1500000000</abund>
  </isotope>
  <isotope>
    <mass>197.9667600000</mass>
    <abund>9.9700000000</abund>
  </isotope>
  <isotope>
    <mass>198.9682690000</mass>
    <abund>16.8700000000</abund>
  </isotope>
  <isotope>
    <mass>199.9683160000</mass>
    <abund>23.1000000000</abund>
  </isotope>
  <isotope>
    <mass>200.9702930000</mass>
    <abund>13.1800000000</abund>
  </isotope>
  <isotope>
    <mass>201.9706320000</mass>
    <abund>29.8600000000</abund>
  </isotope>
  <isotope>
    <mass>203.9734810000</mass>
    <abund>6.8700000000</abund>
  </isotope>
</atom>
<atom>
  <name>Lead</name>
  <symbol>Pb</symbol>
  <isotope>
    <mass>203.9730370000</mass>
    <abund>1.4000000000</abund>
  </isotope>
```

```
    <isotope>
      <mass>205.9744550000</mass>
      <abund>24.1000000000</abund>
    </isotope>
    <isotope>
      <mass>206.9758850000</mass>
      <abund>22.1000000000</abund>
    </isotope>
    <isotope>
      <mass>207.9766410000</mass>
      <abund>52.4000000000</abund>
    </isotope>
  </atom>
</atomdefdata>
```

The Protein Chemistry Definition File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- DTD for polymer definitions, used by the
'GNU polyxmass' suite of mass spectrometry applications.
Copyright 2003, 2004 Filippo Rusconi - Licensed under the GNU GPL -->
<!DOCTYPE polchemdefdata [
<!ELEMENT polchemdefdata (type,leftcap,rightcap,codelen,ionizerule,
                          monomers,modifs,cleavespecs,fragspecs)>
<!ELEMENT ionizerule (actform,charge,level)>
<!ELEMENT monomers (mnm*)>
<!ELEMENT modifs (mdf*)>
<!ELEMENT cleavespecs (cls*)>
<!ELEMENT fragspecs (fgs*)>
<!ELEMENT mnm (name,code,formula)>
<!ELEMENT mdf (name,actform)>
<!ELEMENT cls (name,pattern,clr*)>
<!ELEMENT fgs (name,end,actform,comment?,fgr*)>
<!ELEMENT clr ((le-mnm-code,le-actform)?,(re-mnm-code,re-actform)?)>
<!ELEMENT fgr (name,actform,prev-mnm-code?,this-mnm-code?,next-mnm-code?,comment?)>
<!ELEMENT type (#PCDATA)>
<!ELEMENT leftcap (#PCDATA)>
<!ELEMENT rightcap (#PCDATA)>
<!ELEMENT codelen (#PCDATA)>
<!ELEMENT actform (#PCDATA)>
<!ELEMENT charge (#PCDATA)>
<!ELEMENT level (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT formula (#PCDATA)>
<!ELEMENT pattern (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT le-mnm-code (#PCDATA)>
<!ELEMENT re-mnm-code (#PCDATA)>
<!ELEMENT le-actform (#PCDATA)>
<!ELEMENT re-actform (#PCDATA)>
<!ELEMENT comment (#PCDATA)>
<!ELEMENT prev-mnm-code (#PCDATA)>
<!ELEMENT this-mnm-code (#PCDATA)>
<!ELEMENT next-mnm-code (#PCDATA)>
]>
<polchemdefdata>
  <type>protein</type>
  <leftcap>+H</leftcap>
  <rightcap>+OH</rightcap>
  <codelen>1</codelen>
  <ionizerule>
    <actform>+H</actform>
    <charge>1</charge>
```

```
    <level>1</level>
  </ionizerule>
  <monomers>
    <mnm>
      <name>Glycine</name>
      <code>G</code>
      <formula>C2H3NO</formula>
    </mnm>
    <mnm>
      <name>Alanine</name>
      <code>A</code>
      <formula>C3H5NO</formula>
    </mnm>
    <mnm>
      <name>Valine</name>
      <code>V</code>
      <formula>C5H9NO</formula>
    </mnm>
    <mnm>
      <name>Leucine</name>
      <code>L</code>
      <formula>C6H11NO</formula>
    </mnm>
    <mnm>
      <name>Isoleucine</name>
      <code>I</code>
      <formula>C6H11NO</formula>
    </mnm>
    <mnm>
      <name>Serine</name>
      <code>S</code>
      <formula>C3H5NO2</formula>
    </mnm>
    <mnm>
      <name>Threonine</name>
      <code>T</code>
      <formula>C4H7NO2</formula>
    </mnm>
    <mnm>
      <name>Cysteine</name>
      <code>C</code>
      <formula>C3H5NOS</formula>
    </mnm>
    <mnm>
      <name>Methionine</name>
      <code>M</code>
      <formula>C5H9NOS</formula>
    </mnm>
    <mnm>
      <name>Arginine</name>
```

```
  <code>R</code>
  <formula>C6H12N4O</formula>
</mnm>
<mnm>
  <name>Lysine</name>
  <code>K</code>
  <formula>C6H12N2O</formula>
</mnm>
<mnm>
  <name>Aspartate</name>
  <code>D</code>
  <formula>C4H5NO3</formula>
</mnm>
<mnm>
  <name>Glutamate</name>
  <code>E</code>
  <formula>C5H7NO3</formula>
</mnm>
<mnm>
  <name>Asparagine</name>
  <code>N</code>
  <formula>C4H6N2O2</formula>
</mnm>
<mnm>
  <name>Glutamine</name>
  <code>Q</code>
  <formula>C5H8N2O2</formula>
</mnm>
<mnm>
  <name>Tryptophan</name>
  <code>W</code>
  <formula>C11H10N2O</formula>
</mnm>
<mnm>
  <name>Phenylalanine</name>
  <code>F</code>
  <formula>C9H9N1O</formula>
</mnm>
<mnm>
  <name>Tyrosine</name>
  <code>Y</code>
  <formula>C9H9N1O2</formula>
</mnm>
<mnm>
  <name>Histidine</name>
  <code>H</code>
  <formula>C6H7N3O</formula>
</mnm>
<mnm>
  <name>Proline</name>
```

```
      <code>P</code>
      <formula>C5H7N1O1</formula>
    </mnm>
  </monomers>
  <modifs>
    <mdf>
      <name>Phosphorylation</name>
      <actform>-H+H2PO3</actform>
    </mdf>
    <mdf>
      <name>Acetylation</name>
      <actform>-H+C2H3O</actform>
    </mdf>
    <mdf>
      <name>Amidation</name>
      <actform>-OH+NH2</actform>
    </mdf>
    <mdf>
      <name>SulfideBond</name>
      <actform>-H2</actform>
    </mdf>
  </modifs>
  <cleavespecs>
    <cls>
      <name>CyanogenBromide</name>
      <pattern>M/</pattern>
      <clr>
        <re-mnm-code>M</re-mnm-code>
        <re-actform>-CH2S+O</re-actform>
      </clr>
    </cls>
    <cls>
      <name>Trypsin</name>
      <pattern>K/;R/;-K/P</pattern>
    </cls>
    <cls>
      <name>Chymotrypsin</name>
      <pattern>W/;V/</pattern>
    </cls>
    <cls>
      <name>EndoLysC</name>
      <pattern>K/</pattern>
    </cls>
    <cls>
      <name>EndoAspN</name>
      <pattern>/D</pattern>
    </cls>
    <cls>
      <name>GluC</name>
      <pattern>E/</pattern>
```

```
  </cls>
</cleavespecs>
<fragspecs>
  <fgs>
    <name>a</name>
    <end>LE</end>
    <actform>-C1O1</actform>
    <fgr>
      <name>a-fgr-1</name>
      <actform>+H2O0</actform>
      <prev-mnm-code>E</prev-mnm-code>
      <this-mnm-code>D</this-mnm-code>
      <next-mnm-code>F</next-mnm-code>
      <comment>comment here!</comment>
    </fgr>
    <fgr>
      <name>a-fgr-2</name>
      <actform>+H1O0</actform>
      <prev-mnm-code>F</prev-mnm-code>
      <this-mnm-code>D</this-mnm-code>
      <next-mnm-code>E</next-mnm-code>
      <comment>comment here!</comment>
    </fgr>
  </fgs>
  <fgs>
    <name>b</name>
    <end>LE</end>
    <actform>-H0</actform>
  </fgs>
  <fgs>
    <name>c</name>
    <end>LE</end>
    <actform>+N1H2+H1</actform>
    <comment>that's just a comment</comment>
  </fgs>
  <fgs>
    <name>z</name>
    <end>RE</end>
    <actform>-N1H1</actform>
    <comment>Not in CID high En. frag</comment>
  </fgs>
  <fgs>
    <name>y</name>
    <end>RE</end>
    <actform>+H2</actform>
  </fgs>
  <fgs>
    <name>x</name>
    <end>RE</end>
    <actform>+C1O1</actform>
```

```
      <fgr>
        <name>x-fgr-1</name>
        <actform>+H100</actform>
        <prev-mnm-code>E</prev-mnm-code>
        <this-mnm-code>D</this-mnm-code>
        <next-mnm-code>F</next-mnm-code>
        <comment>comment here!</comment>
      </fgr>
      <fgr>
        <name>x-fgr-2</name>
        <actform>+H200</actform>
        <prev-mnm-code>F</prev-mnm-code>
        <this-mnm-code>D</this-mnm-code>
        <next-mnm-code>E</next-mnm-code>
        <comment>comment here!</comment>
      </fgr>
    </fgs>
    <fgs>
      <name>imm</name>
      <end>NE</end>
      <actform>-C1O1+H1</actform>
    </fgs>
  </fragspecs>
</polchemdefdata>
```

The acidobasic.xml File

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- DTD for polymer elements' pka data, used by the
'GNU polyxmass' suite of mass spectrometry applications.
Copyright 2003, 2004 Filippo Rusconi - Licensed under the GNU GPL -->
<!DOCTYPE acidobasicdata [
<!ELEMENT acidobasicdata (monomers*,modifs*)>
<!ELEMENT monomers (mnm*)>
<!ELEMENT modifs (mdf*)>
<!ELEMENT mnm (code,chemgroup*)>
<!ELEMENT mdf (name,chemgroup*)>
<!ELEMENT chemgroup (name,(pka,acidcharged)?,polrule*,chemgrouprule*)>
<!ELEMENT chemgrouprule (entity,name,outcome)>
<!ELEMENT pka (#PCDATA)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT code (#PCDATA)>
<!ELEMENT outcome (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT entity (#PCDATA)>
<!ELEMENT acidcharged (#PCDATA)>
```

```
<!ELEMENT polrule (#PCDATA)>
]>
<acidobasicdata>
  <monomers>
    <mnm>
      <code>A</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>C</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral SH2</name>
<pka>8.3</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
      </chemgroup>
    </mnm>
```

```
    <mnm>
      <code>D</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral COOH</name>
<pka>3.65</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>AmidationAsp</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>E</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
```

```
      <chemgroup>
        <name>Lateral COOH</name>
<pka>4.25</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>AmidationGlu</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>F</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>G</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
```

```
      </chemgroup>
    </mnm>
    <mnm>
      <code>H</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>In-ring NH+</name>
<pka>6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>never_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>I</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>K</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
```

```
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral NH2</name>
<pka>10.53</pka>
<acidcharged>TRUE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>L</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>M</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
```

```
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>N</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>P</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
```

```
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>Q</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>R</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral guanidinium</name>
<pka>12.48</pka>
<acidcharged>TRUE</acidcharged>
<polrule>never_trapped</polrule>
      </chemgroup>
    </mnm>
```

```
    <mnm>
      <code>S</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral alcohol</name>
<pka>13</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>Phosphorylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>T</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
```

```
      <chemgroup>
        <name>Lateral alcohol</name>
<pka>13</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>Phosphorylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
   <mnm>
      <code>V</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
    </mnm>
    <mnm>
      <code>W</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.35</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
```

```
      </chemgroup>
    </mnm>
    <mnm>
      <code>Y</code>
      <chemgroup>
        <name>N-term NH2</name>
<pka>9.6</pka>
<acidcharged>TRUE</acidcharged>
<polrule>left_trapped</polrule>
<chemgrouprule>
  <entity>LE_PLM_MODIF</entity>
  <name>Acetylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
      <chemgroup>
        <name>C-term COOH</name>
<pka>2.36</pka>
<acidcharged>FALSE</acidcharged>
<polrule>right_trapped</polrule>
      </chemgroup>
      <chemgroup>
        <name>Lateral phenol</name>
<pka>10.1</pka>
<acidcharged>FALSE</acidcharged>
<polrule>never_trapped</polrule>
<chemgrouprule>
  <entity>MNM_MODIF</entity>
  <name>Phosphorylation</name>
  <outcome>LOST</outcome>
</chemgrouprule>
      </chemgroup>
    </mnm>
  </monomers>
  <modifs>
    <mdf>
      <name>Phosphorylation</name>
      <chemgroup>
        <name>none_set</name>
<pka>12</pka>
<acidcharged>FALSE</acidcharged>
      </chemgroup>
      <chemgroup>
        <name>none_set</name>
<pka>7</pka>
<acidcharged>FALSE</acidcharged>
      </chemgroup>
    </mdf>
  </modifs>
</acidobasicdata>
```

GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original

authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent

infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software

Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

> <one line to give the program's name and a brief idea of what it does.>
> Copyright (C) <year> <name of author>
>
> This program is free software; you can redistribute it and/or modify
> it under the terms of the GNU General Public License as published by

```
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
   Gnomovision version 69, Copyright (C) year name of author
   Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
   This is free software, and you are welcome to redistribute it
   under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

```
  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  ‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.