GNU polyxmass User Manual

(Version *0.7.0*)

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

Revision History

- 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 **GNU 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 **GNU 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 **GNU 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** filesystem 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 **GNU 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 (**GNU 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**' filesystem 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 "filesystem-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** filesystem. 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 **GNU polyxdef** module);
- Allowing simple yet powerful mass computations to be made in a polymer chemistry definition-specific manner (in the **GNU polyxcalc** module);
- Allowing the highly sophisticated editing of polymer sequences on a polymer chemistry definition-specific basis along with chemical reaction simulations, finely configured mass spectrometric computations... (in the **GNU polyxedit** module);
- Allowing customization of the way each monomer will show up graphically during the program operation (in the **GNU polyxedit** module);
- Allowing polymer sequence editing with immediate visualization of the mass changes elicited by the editing activity (in the **GNU polyxedit** module);
- Unlimited number of polymer sequences opened at any given time and of any given polymer chemistry definition type (in the **GNU 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 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

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

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. For information on this 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 documents 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
- `bash-2.04 $` shows the prompt at which a command should be entered as non-root
- `bash-2.04 #` 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 primarily 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]).

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 the *Free/true Open Source Movement* 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`);
  - `GTK-Doc`, a system for automating the documentation of the source code and making readable developer's documentation in *`HTML`* format
    (`http://www.gtk.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`);

[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`

  - `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 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 **GNU polyxmass** modules;

- The basics of polymer chemistry;

- The basics of mass spectrometry;

- Generalities about the **GNU polyxmass** software suite;

- The **GNU polyxdef** module (definition of a new polymer chemistry);

- The **GNU polyxcalc** module (polymer chemistry-aware calculator);
- The **GNU polyxedit** module (the main module of the suite, where actual simulations are performed);
- The **GNU polyxmassdata** module describing the **GNU polyxmass**' complex 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

The **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. We will review this ordered installation procedure below.

Each module of the **GNU polyxmass** software suite may be developed independently, which means that it is not required that they have the same version/package 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 system, all the packages should be installed with no difficulty.

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, in the manner that suits her needs, the individual packages.

Each package comes in a number of different flavors:

- Uncompiled source package
  - *tar.gz* files which need to be compiled using the `GNU make` program;
  - *dsc* plus *tar.gz* files which need to be compiled using the *Debian GNU/Linux* packaging system;
  - *src.rpm* files which need to be compiled using the `rpm` tool;
- Binary ready-to-install package

- *_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 `deb` Binary Package

On the *Debian GNU/Linux* distribution, the packaging system makes the installation of software incredibly easy. The user is provided with two different mechanisms to get the software modules installed efficiently (although one method of the two is more convenient as it is automated by the software repository manager, which is not you, the reader).

The harderst way: using `dpkg` package installer

Using this method is less automated, and requires that the user first downloads all the packages from the following directory:

`http://www.polyxmass.org/debian-packages/binary-i386`

For the 0.7.0 release, that would mean that the following packages should be downloaded manually:

- `polyxmassdata_0.7.0-1_all.deb`
- `libpxmutils3_0.7.0-1_i386.deb`
- `libpxmchem4_0.7.0-1_i386.deb`
- `polyxdef_0.7.0-1_i386.deb`
- `polyxcalc_0.7.0-1_i386.deb`
- `polyxedit_0.7.0-1_i386.deb`

Once all these packages have been sucessfully downloaded, the user should start installing them (as superuser) using –in the order of the files described above– the following command line:

```
bash-2.04 # dpkg -i polyxmassdata_0.7.0-1_all.deb ↩
```

and so on for the other files.

Once, these files are successfully installed, the **GNU polyxmass** software suite is finally usable!

The easiest way: using `apt-get` package manager

The `apt-get` package manager is particularly clever in determining the dependencies between packages in an interrelated 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.

To let the `apt-get` package manager know what are your sources of software are located in some specific place on the network, you need to instruct it of these locations by editing the file `/etc/apt/sources.list`. To be able to automatically download **GNU polyxmass** software packages, you should add the two following lines at the end of said file:

```
deb http://www.polyxmass.org/debian-packages binary-i386/
deb-src http://www.polyxmass.org/debian-packages source/
```

Once this file is edited according to the above, the user can perform an update of all the packages available in the "sources" locations:

```
bash-2.04 # apt-get update ↩
```

The program will fetch all informations available at the locations listed in the `/etc/apt/sources.list` file and crunch the informations in order to be able later to resolve eventual package dependencies. This is the output that I get on my system:

```
Hit http://security.debian.org stable/updates/main Packages
Hit http://security.debian.org stable/updates/main Release
Get:1 http://ftp.fr.debian.org sid/main Packages [2857kB]
Hit http://www.aditel.org binary/ Packages
Ign http://www.aditel.org binary/ Release
Hit http://www.polyxmass.org binary-i386/ Packages
Ign http://www.polyxmass.org binary-i386/ Release
Hit http://www.polyxmass.org source/ Sources
Ign http://www.polyxmass.org source/ Release
Get:2 http://ftp.fr.debian.org sid/main Release [82B]
Get:3 http://ftp.fr.debian.org sid/contrib Packages [78.1kB]
Get:4 http://ftp.fr.debian.org sid/contrib Release [85B]
Hit http://ftp.fr.debian.org sid/non-US/main Packages
Hit http://ftp.fr.debian.org sid/non-US/main Release
Hit http://ftp.fr.debian.org sid/non-US/contrib Packages
Hit http://ftp.fr.debian.org sid/non-US/contrib Release
Get:5 http://ftp.fr.debian.org sid/main Sources [1126kB]
Get:6 http://ftp.fr.debian.org sid/main Release [84B]
Get:7 http://ftp.fr.debian.org sid/contrib Sources [33.8kB]
Get:8 http://ftp.fr.debian.org sid/contrib Release [87B]
Hit http://ftp.fr.debian.org sid/non-US/main Sources
Hit http://ftp.fr.debian.org sid/non-US/main Release
Hit http://ftp.fr.debian.org sid/non-US/contrib Sources
Hit http://ftp.fr.debian.org sid/non-US/contrib Release
Fetched 4095kB in 7s (536kB/s)
Reading Package Lists... Done
```

Now that the system has been made aware that there are packages at a new `http://www.polyxmass.org/debian-packages` location, the installation of the packages is straightforward, just select the package to install:

```
bash-2.04 # apt-get install polyxedit ↩
```

and the following output will be provided to you:

```
Reading Package Lists... Done
Building Dependency Tree... Done
The following extra packages will be installed:
  libpxmchem4 libpxmutils3 polyxmassdata
The following NEW packages will be installed:
  libpxmchem4 libpxmutils3 polyxedit polyxmassdata
0 upgraded, 4 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 polyxedit, it is necessary to first install packages onto which polyxedit actually depends. These packages are listed:

```
The following extra packages will be installed:
  libpxmchem4 libpxmutils3 polyxmassdata
```

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

```
The following NEW packages will be installed:
  libpxmchem4 libpxmutils3 polyxedit polyxmassdata
0 upgraded, 4 newly installed, 0 to remove and 9 not upgraded.
```

If you accept to continue (key-in `Y` ↩), this is what you get: all four packages get automatically downloaded, installed and configured. There is no available higher standard for such package managing tasks nowadays.

Installing From The `rpm` Binary Package

Installing any *rpm* package using the `rpm` program[1] is as easy as entering the following command, as root:

```
bash-2.04 # rpm -ivh polyxcalc-1.i386.rpm ↩
```

What this command does is read the `polyxcalc-1.i386.rpm` file contents (this package file probably contains a number of files packed in it) and copy them to their destination directories. The *rpm* file format allows to tell to which directory each file that it contains is to be copied.

Note that when the installation is performed using the `rpm` binary package, the installation directory is in the `/usr` standard tree. Once the package is installed, do not move the files

[1] 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`.

from their installation directory, because each **GNU polyxmass** module relies on these precise directories to locate the files needed to operate correctly. The binary files (program files, like the `polyxedit` or the `polyxcalc` program files) are installed in the `/usr/bin` directory.

Indeed, it is noteworthy that the package is *not relocatable*, which means that the user is strongly urged not to use the `--prefix` option (or the installation will be messed up). The only way to install the software through a `rpm` binary package in a customized directory is by recompiling the sources with the *`src.rpm`* file package. See below for instructions on how to cheat with the `rpm` utility.

To see all the files that are provided by a given `rpm`-based package file, issue the following command:

```
bash-2.04 $ rpm -qpl polyxcalc-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.

Installing From The `rpm` Source Package

We assume here that the system is a **Red Hat** *GNU/Linux* system, but the directions provided here might also be useful for other `rpm`-based systems (like **Mandrakesoft**'s or **SuSe**'s?). Installing any `rpm`-based package file from the *`src.rpm`* source package[2] is actually simply one more step (the building of the software) than in the previous binary package installation case.

The *`src.rpm`* package simply contains two files: the source archive file (in the form of a **GNU** classical *`tar.gz`* tarball (see next section), and a corresponding `rpm` *`spec`* file. The *`tar.gz`* tarball is very similar to the one described in detail later, while the *`spec`* file is a very simple text file that gives the `rpm` software directions on how to build a binary *`i386.rpm`* package out of the *`src.rpm`* file.

It is important to understand the functioning of the `rpm` program (read its documentation) before using a *`src.rpm`* package. After installing a *`src.rpm`* package,[3] the program itself is not available to the user (while with a binary package, the program is immediately available to the user). Apparently nothing happened, but what happened is that the *`src.rpm`* package's contents are unpacked by the `rpm` program into two different `/usr/src/redhat` subdirectories. In this `/usr/src/redhat` directory, indeed, the `SPECS` subdirectory now contains the *`spec`* file, and the `SOURCES` subdirectory now contains the *`tar.gz`* source tarball file. To build a binary package after having installed the `rpm` source package, it is necessary to first change directory to the `SPECS` subdirectory and next ask `rpm` to build the package. All these steps are described below with the **GNU polyxcalc** module package as an example:

```
bash-2.04 # rpm -ivh polyxcalc-1.src.rpm ↩
bash-2.04 # cd /usr/src/redhat/SPECS ↩
bash-2.04 # rpmbuild -ba polyxcalc.spec ↩
```

The `-ba` option tells the `rpmbuild` program to "build all" the package. After a while,[4]

[2] The filename has "src" in it, contrary to the binary package that has the platform name; for **Intel** platforms this is "i386" in most cases.

[3] Using the same command as for a binary package.

[4] Automatic sources unpacking, configuration, building of the program, packaging into a binary package.

the process stops. If the displayed result is 0, then that means that everything went correctly. Further, near the end of the `rpm` output, there must be a line indicating that a *`i386.rpm`* package file has been written. Change directory (from `/usr/src/redhat/SPECS`) to `/usr/src/redhat/RPMS/i386` and see that the `rpm` binary package (in our example that would be the `GNU polyxcalc-1.i386.rpm` file) has been correctly produced. This binary package is nothing but the `rpm` *`i386.rpm`* file package that was described in the previous section. Just install this package as described above for the binary packages.

This newly built package is guaranteed to be compatible with your processor and with your pre-installed programs and shared libraries, since the compilation completed without trouble. Ideally, the process described here should be performed for any `rpm` package but, since it is time-consuming, it is only performed on special critical mission software...

Note that there is a shortcut to the procedure described above:
```
bash-2.04 # rpmbuild -rebuild polyxcalc-1.src.rpm
```

It is possible to cheat with the `rpm` software so that the installation directory is not the default one. This is done by editing manually the package's *`spec`* file. It is simply a matter of telling the `rpmbuild` program that the *prefix* to be used to construct the installation path is not the default one (`/usr`) but `/opt`, for example. In this case, just add the following line on top of the *`spec`* file:
```
%define _prefix           /opt
```
if you intended to install the software package in the `/opt` directory. For example, here is how the `polyxcalc.spec` file would read if you intended to install the **GNU polyxcalc** package in the `/opt` directory:
```
%define _prefix           /opt
%define name              polyxcalc
```

Once this slight modification is done, just run:
```
bash-2.04 # rpmbuild -ba polyxcalc.spec ↩
```

And once the package is recompiled, this time you can install it the usual way:
```
bash-2.04 # rpm -ivh polyxcalc-1.i386.rpm ↩
```

Installing From The *`tar.gz`* Sources

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

```
bash-2.04 $ cp polyxcalc.tar.gz /tmp ↩ copy the package into a safe place
bash-2.04 $ cd /tmp ↩
bash-2.04 $ tar -xvzf polyxcalc.tar.gz ↩ this unpacks the sources into a source tree in the
GNU polyxcalc directory
bash-2.04 $ cd polyxcalc ↩
bash-2.04 $ ./configure ↩
bash-2.04 $ make ↩
bash-2.04 $ su ↩ become root if it is possible
bash-2.04 # make install ↩
```

Unlike with the previous `rpm`-based installation, it is possible to specify an installation directory to the `./configure` command. Indeed, the user can modify the default installation directory (which is the `/usr/local` tree) by using a qualified `--prefix` option to the `./configure` command.

For example, by default (with no qualified `--prefix` option), the **GNU polyxcalc** module's configuration data and executable files would be installed respectively in these two directories:

- `/usr/local/share/polyxcalc`
- `/usr/local/bin`

To change the installation directory, the user may use the qualified `--prefix` option as shown in the following example:

```
bash-2.04 $ ./configure --prefix=/usr ↩
```

Note that you will need to have root priviledges to be able to install the program in system directories like `/usr` or `/usr/local`.

Interestingly, since version 4.0 of `rpm`, it is possible to build `rpm` files with a suitably made source *`tar.gz`* tarball. This source tarball is nothing than a file containing the source files of the software package along with the corresponding *`spec`* file (the same we discussed above). The *`tar.gz`* source tarballs in any of the **GNU polyxmass** software suite modules complies with this format, and thus it is possible to build `rpm`-based files running this very simple command as root, for package **GNU polyxcalc**, for example:

```
bash-2.04 $ rpmbuild -ta polyxcalc.tar.gz ↩
```

At the end of the package building process, the two source and binary files are ready in the `/usr/src/redhat` subdirectories (see 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 0.6.0 of the modules of the **GNU polyxmass** software suite could be installed using the *`info`* files provided by D^{r} Mark Tracy. These `Fink` *`info`* files are scripts much like the `rpm` *`spec`* files. The `Fink` packaging system relies on the usual *`tar.gz`* source files, which may be used without modification[5]. 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 patches should be applied to the original source tarballs so that these code modifications are recreated when intalling the packages on the *Mac OS-X*/`Fink` platform. In this case, the *`patch`* files would be distributed along with the source tarball files and the *`info`* files. Providing *`patch`*

[5]That's the case for the version 0.6.0 of all the **GNU polyxmass** modules.

files for the software to build correctly on any given platform is the task of the package maintainer.

Once you have downloaded all the required files (*info*, *patch*, *tar.gz*), the installation process is as easy as doing the following:

First, since these *info* scripts may not yet be available through the Fink server, you need to copy them to the right place and go there to continue (run command as superuser):

```
bash-2.04 # cp *.info /sw/fink/10.2/local/main/finkinfo
bash-2.04 # cd /sw/fink/10.2/local/main/finkinfo
```

Note that, in the future, the *info* scripts will be placed in the right directory by the Fink server. Now, install the packages –in order– by issuing the following commands:[6]

```
bash-2.04 # fink install polyxmassdata ↩
bash-2.04 # fink install libpxmutils0 ↩
bash-2.04 # fink install libpxmchem1 ↩
bash-2.04 # fink install polyxdef ↩
bash-2.04 # fink install polyxcalc ↩
bash-2.04 # fink install polyxedit ↩
```

If the software packager did everything right, Fink will calculate the dependencies, and ask you if you want to install the dependent packages. 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).

This is all is needed to understand how to perform the installation of any package in the **GNU polyxmass** mass spectrometric software suite. The next chapters will deal with each module separately. The software packages in the **GNU polyxmass** software suite should be installed in the following order:

1. `polyxmassdata`
2. `libpxmutils`
3. `libpxmchem`
4. `polyxdef`
5. `polyxcalc`
6. `polyxedit`

Each module is described in its own chapter, with all the details that are required so that the user gets an intimate knowledge of the way the whole **GNU polyxmass** mass spectrometric software suite works.

[6] While the rest of the process happens you can read the user documentation. During the installation of the libraries, Fink will ask if you want to install the `-shlibs` also: say yes.

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.

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

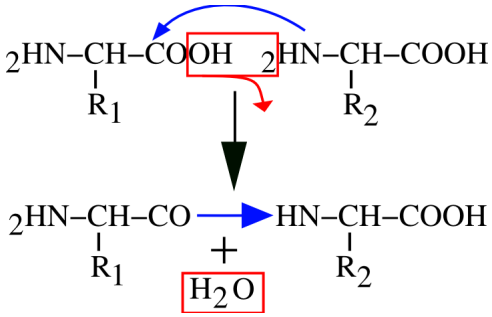

Figure 3.1: **Peptidic bond formation by condensation.** The left end monomer R_1 is condensed to the right end monomer R_2 to yield a peptidic bond. A water molecule is lost during the process.

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 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

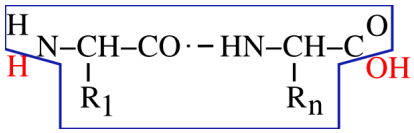

Figure 3.2: **End capping chemistry of the protein polymer.** A protein is made of a chain of residues and of two caps. The left cap is the N-terminal proton and the right cap is the C-terminal hydroxyl. Altogether, the residual chain (enclosed here in the blue polygon) and both red-colored caps (H and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

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 *nucleotide*s. 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 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

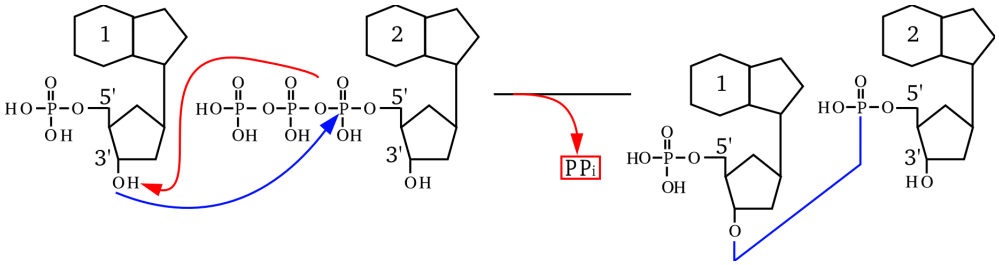

Figure 3.3: **Phosphodiester bond formation by esterification.** The arriving monomer (on the right) has its triphosphate on the 5' carbon of the sugar esterified by nucleophilic attack of the first phosphorus by the alcohol function beared by the 3' carbon of the (deoxy)ribose sugar ring of the left 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.

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 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).

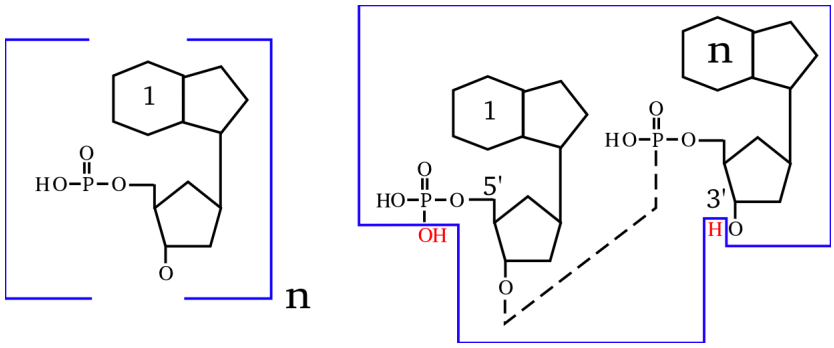

Figure 3.4: **End capping chemistry of the nucleic acid polymer.** A nucleic acid is made of a chain of nucleotides (left formula) and of two caps. The left cap is the hydroxyl group that belongs to the terminal phosphate of the 5' carbon of the sugar. The right cap is the proton that belongs to the 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).

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 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

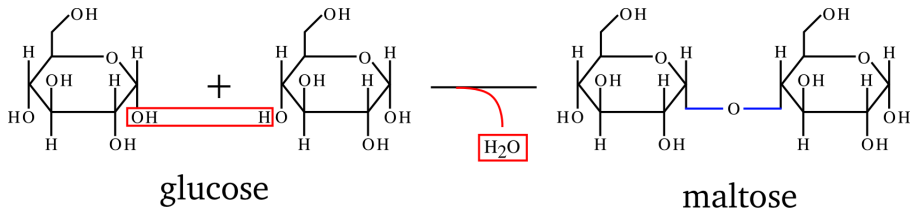

Figure 3.5: **Osidic bond formation by condensation.** The two monomers are subject to condensation with loss of one molecule of water.

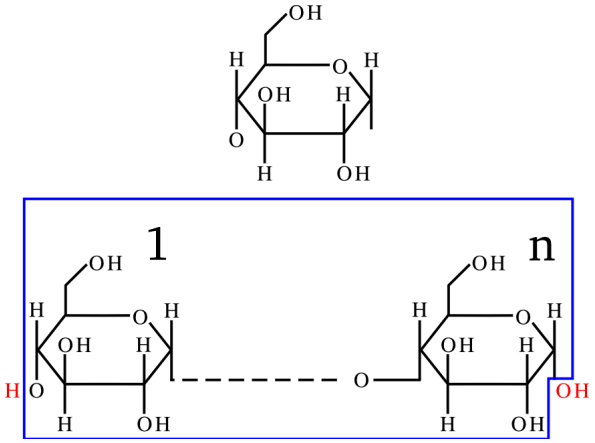

Figure 3.6: **End capping chemistry of the polysaccharidic polymer.** A polysaccharide is made of a chain of osidic residues (blue-boxed formula) and of two caps (red-colored atoms). The left cap is the proton group that belongs to the non-reducing end of the polymer. The right cap is the hydroxyl group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

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

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**

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

We have very 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 24 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.

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.

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:

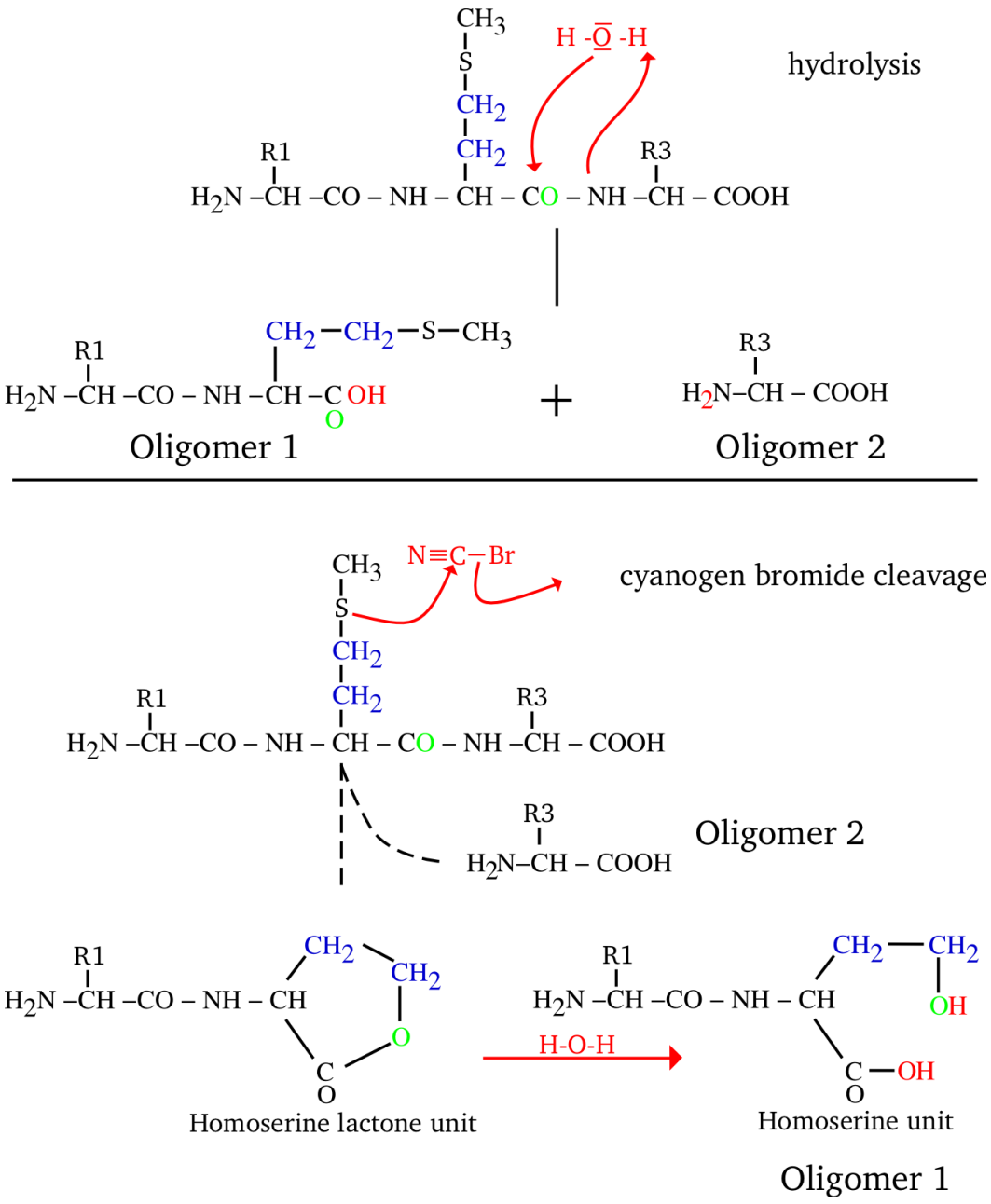

Figure 3.7: **Protein cleavage by water and cyanogen bromide.** A tripeptide (pretty small protein) is cleaved at position 1 either by hydrolysis (top) or by cyanogen bromide (bottom). Cyanogen bromide cleaves specifically on the right of a methionine monomer.

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 53).

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 53.

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 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;

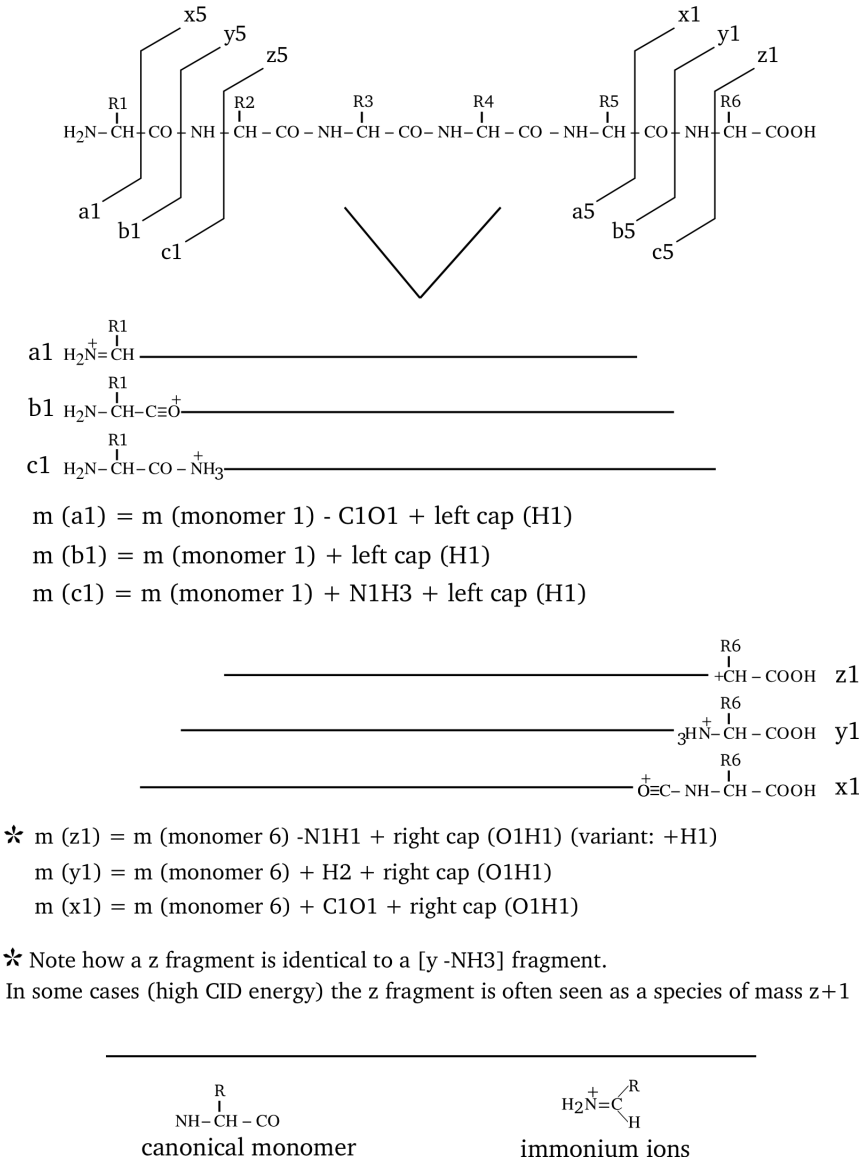

Figure 3.8: **Protein fragmentation patterns most widely encountered.** An hexapeptide is fragmented in the seven most widely encountered manners, such as to generate a, b, c, x, y, z and immonium fragment ions. The figure illustrates the position of the cleavage for each kind of fragment (exemplified 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.

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-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 56). 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 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**

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

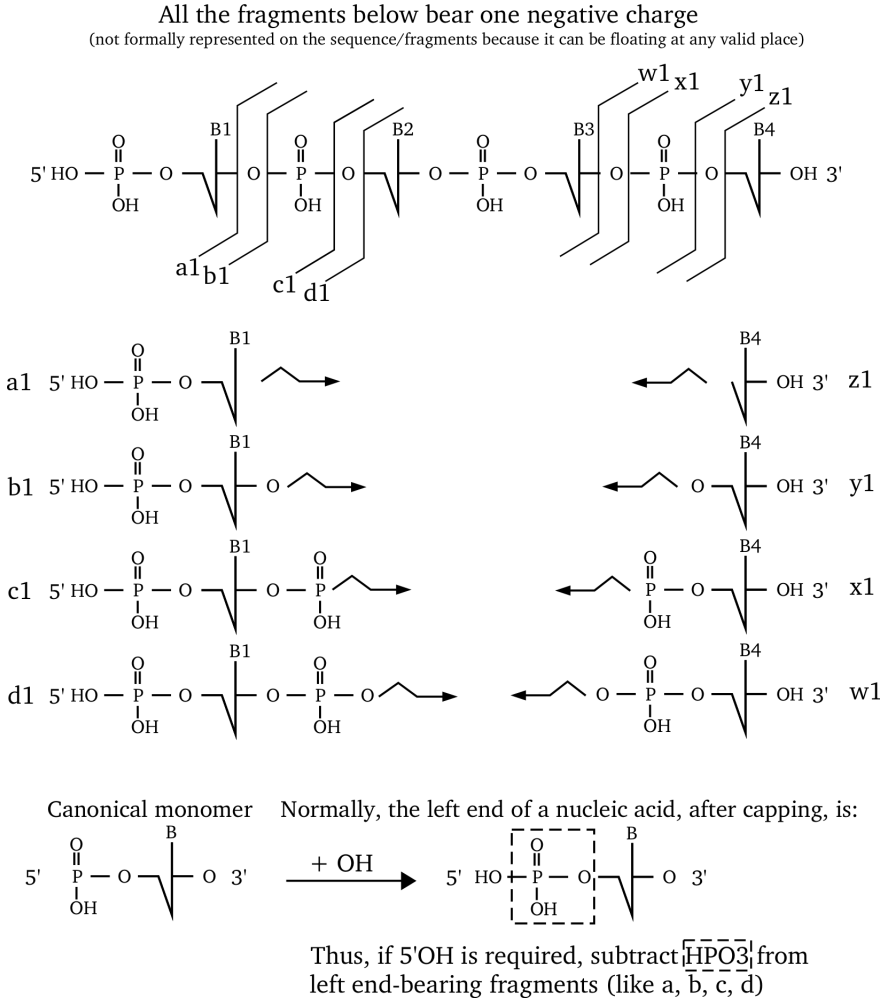

Figure 3.9: **DNA fragmentation patterns most widely encountered.** A short DNA sequence is fragmented in the eight most widely encountered manners, such as to generate a, b, c, d, w, x, y, z fragment ions. The figure illustrates the position of the cleavage for each kind of fragment (exemplified using 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).

***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 56.

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 executable programs):

1. **GNU libpxmutils** where housekeeping functions are implemented; this library is not graphical and of little interest to the pure chemist;
2. **GNU libpxmchem** where all the chemical intelligence of the **GNU polyxmass** software framework lies; this library is not graphical and may be interesting to the chemist so as to understand what a *monomer* or an *oligomer* is, from a programmatic standpoint;
3. **GNU polyxmassdata** where all the configuration files are stored, like the different sample polymer chemistry definition files, the little graphics files that are used in the polymer sequence editor to render and display graphically the polymer sequence in a polymer chemistry definition-specific manner...
4. **GNU polyxdef** where the user will easily define brand new polymer chemistries, which will produce a *polymer chemistry definition*, later saved in a *polymer chemistry definition file*; this module is an executable graphical user interface file;
5. **GNU polyxcalc** where the user will easily perform sophisticated mass calculations either using an available polymer chemistry definition or simply using the predefined set of atoms; this module is an executable graphical user interface file;
6. **GNU polyxedit** where the user will easily create/edit polymer sequence files that are of any available polymer chemistry definition, so that mass spectrometric simulations may be performed; this graphical user interface module is the core module for all the user-driven chemical reaction simulations, like modifying a monomer, cleaving the polymer sequence, gas phase fragmenting an oligomer...

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 a DNA sequence: "ATGC", 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. But, of course, 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 as flexibly as possible the required computations, she first defines two different polymer chemistries: the first named "protein" and the second named "dna". In each of the 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. This is performed in the **GNU polyxdef** module (here is modularity). 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 having all the polymer chemistry specifications in a polymer defintion file, the **GNU polyxmass** mass spectrometry software suite is able to deal with any polymer sequence that complies with the given polymer chemistry definition. This association between a polymer sequence and a polymer defintion is 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 graphical user interface modules, we'll review each of such modules in the 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"*. Simply because there are actions that 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"*.

The *formula* and *actform* chemical entities are *not* interchangeable in the **GNU polyxmass** framework.

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 Definitions*: 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 polyxmassdata** package (say, the `protein-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 by 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:

```
bash-2.04 $ 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:
`bash-2.04 $ chmod u+x polname-find.sh ↩`
Upon execution of this script, the output looks like this:
`bash-2.04 $ ./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 simply 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, page49).

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 of the fragmentation specifications... It is important not to cripple these polymer data for two main reasons:

- so that the program performs smoothly;
- 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*

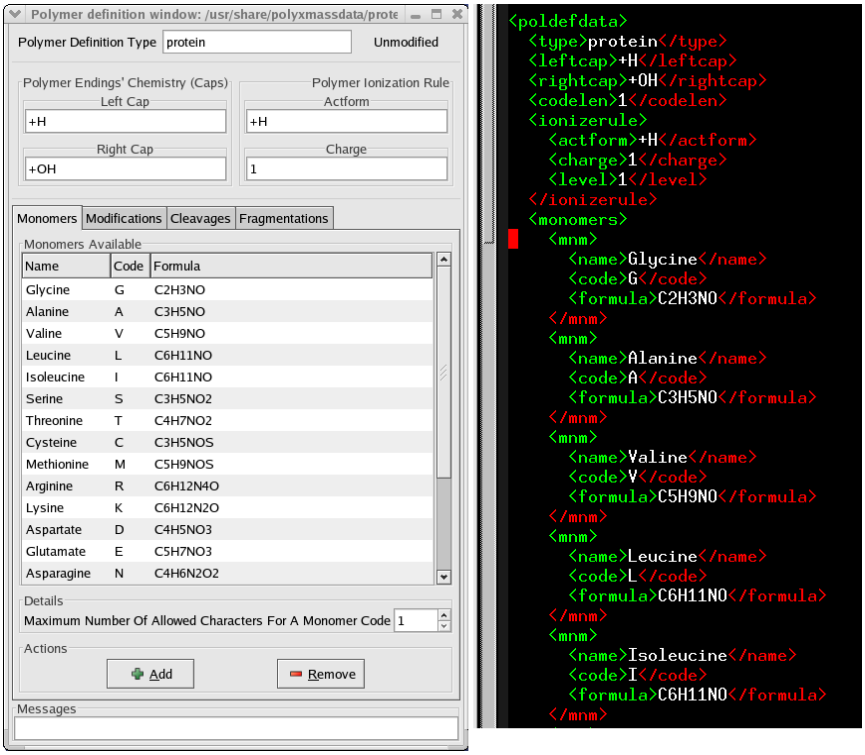

Figure 5.1: **Comparison of a graphical and a text way of editing a polymer chemistry definition file.** The left pane shows the graphical interface that is exposed to the user when defining a polymer. The right pane shows the same *XML* file opened in the `Emacs` editor with the *XML* editing mode switched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

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 other 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 of the polymer to which it is related. This identity is nothing else than the *unique* memory address of the polymer 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

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 in order to close the dialog window. **GNU polyxmass** has been implemented with the "console" philosophy in mind: every message that it wishes to "hand out" to the user is sent to the console window from which the program was started.

There are two levels of very important messages: the *CRITICAL* and the *ERROR* level messages. The CRITICAL messages indicate that time has come to make a quick save of all the data, because something bad might happen. ERROR 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.

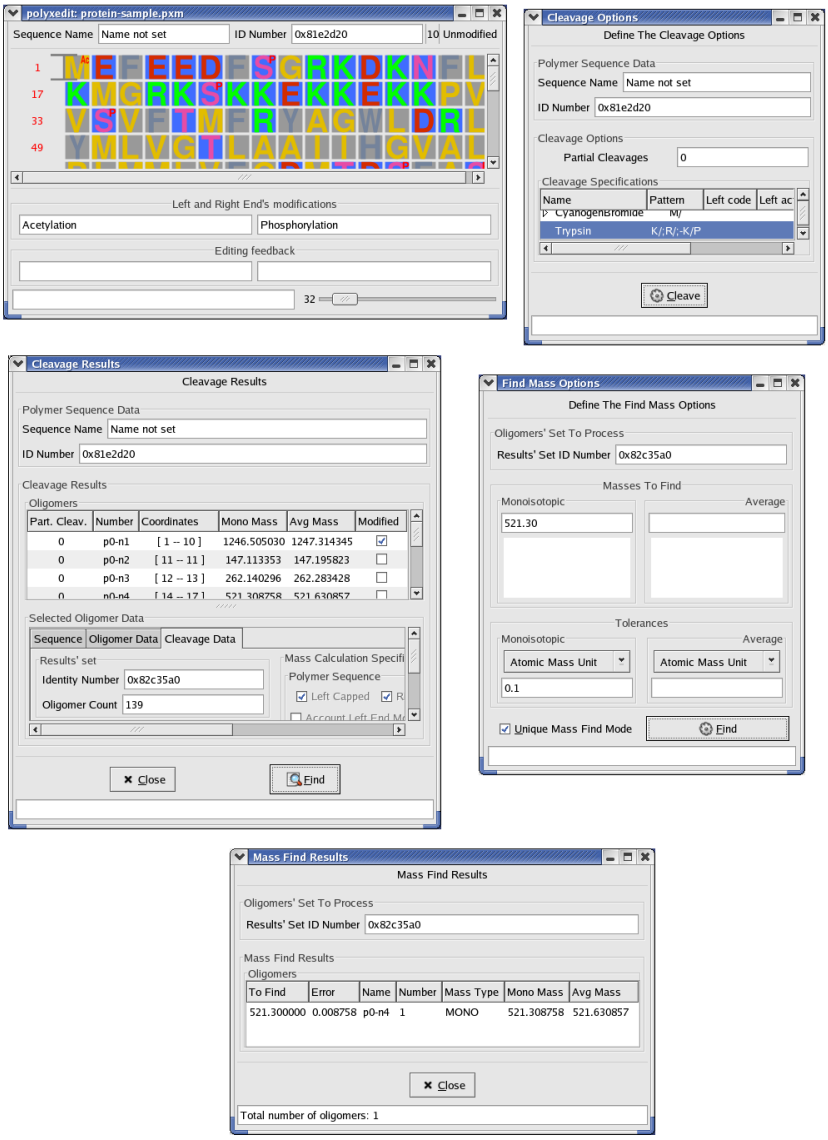

Figure 5.2: **Unambiguous identification of polymer sequences and related data.** When a polymer sequence is loaded/created, it is assigned a numeric value that unambiguously identifies it (for the programmer, this is the pointer to the polymer structure). Each time a window is displayed that contains 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 in this Figure, where the sequence has a number 0x81e2d20 and all the related windows display the same number. Note that the cleavage results data have another identifying number (0x82c35a0) that is later used to trace the mass find results data (last bottom window).

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 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.

GNU polyxdef Invocation

The **GNU polyxdef** module is simply called by its name: polyxdef from a command line. The user is invited to launch the following command and to inspect the various options that it accepts:

```
bash-2.04 $ polyxdef --help ↩
```

This command produces the following output:

```
Usage: polyxdef [OPTION...]
  -d, --details     prints the copyright owner and the licensing
                    of the program
```

```
  -l, --license     prints the license type of the program
  -v, --version     prints the version of the program

Help options:
  -?, --help        Show this help message
  --usage           Display brief usage message
```

Various Identification And Singular Data

"Identification data" are pieces of information that should be defined in order to describe the polymer (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 in its "finished state", after the polymerization has terminated.

Looking at Figure 6.1 while reading the following paragraphs might 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 Ending's Chemistry (Caps) Description of the chemical capping reaction that should happen on both the left and the right ends of the polymer sequence, once it is successfully polymerized and should be set to its "finished state". This chemistry is divided into two pieces of information:
  - Left Cap **+H** String describing the actform that should be applied to the elongating polymer on its left end;
  - Right Cap **+OH** String describing the actform that should be applied to the elongating polymer on its right end;
- 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 proteins 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.

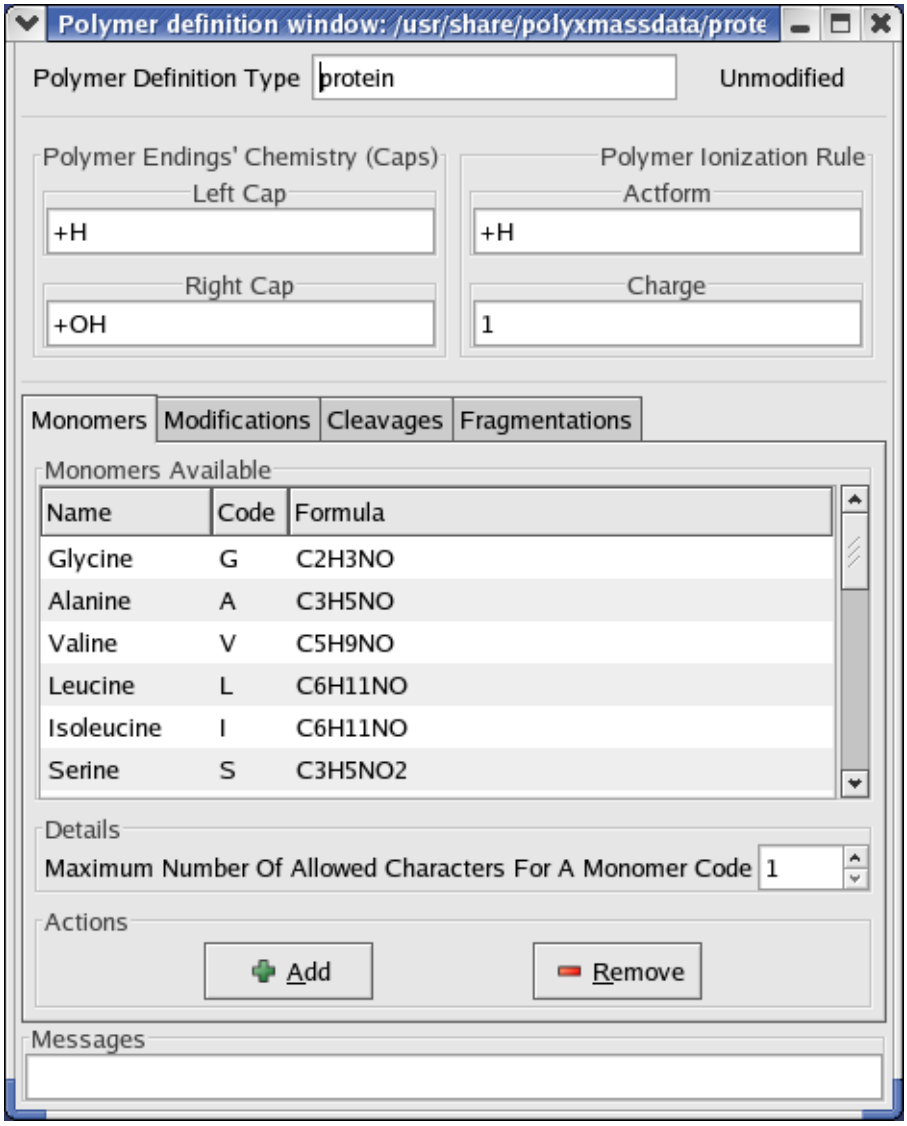

Figure 6.1: **Interface for the definition of the polymer, monomers.** This window shows two different parts, the top half is for the user to enter the polymer identification and singular data, and the bottom part is specialized for the plural data (here showing monomers definition tab). The figure shows how the data should be entered. For example, no double quotes are required when entering strings. Finally, a field is reserved for the number of characters allowed to describe a monomer's code (bottom of the window).

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. Of course, if your are working on polystyrene, you will almost certainly have one monomer in your polymer chemistry definition. But what if you work on DNA or proteins? Let us see what plural data are all about.

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, page 17 for a detailed explanation of the **GNU polyxmass** specialized words.

The lower part of Figure 6.1 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. You could give a monomer a name two gigabytes-long... Of course this would not make much sense.

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 **GNU 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 (**GNU 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 **GNU 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 serine, for example, or an

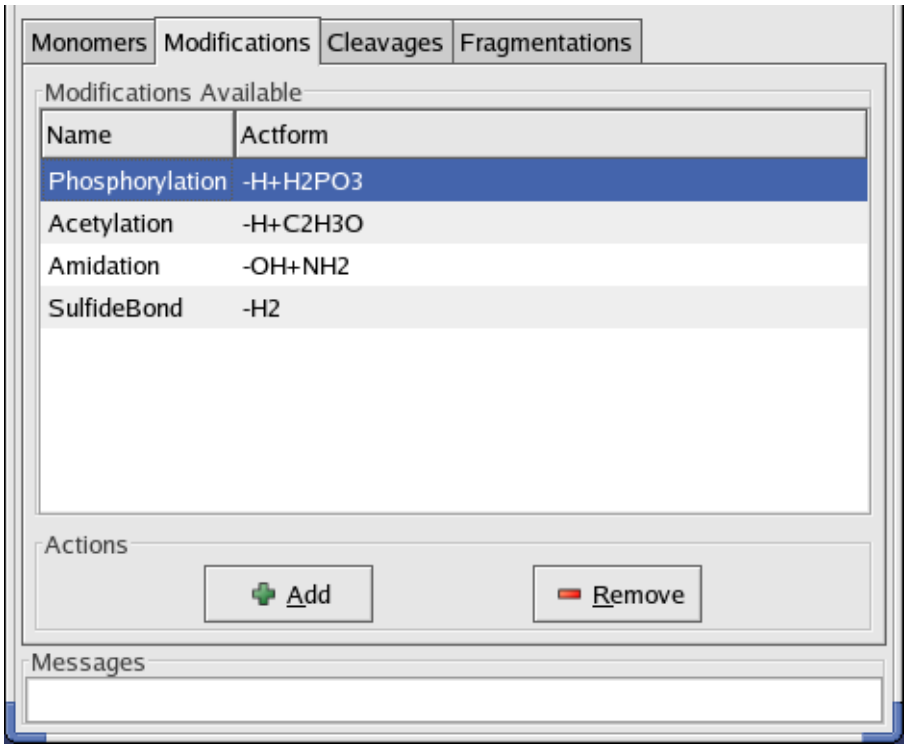

Figure 6.2: **Interface for the definition of the polymer modifications.** This is the same window as for Figure 6.1, but this time the modifications definition tab of it is shown. This figure shows that a modification is simply defined by two strings, a Name string and an Actform string.

acetylation onto a amino function-bearing 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 6.2 will help.

The Figure 6.2 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 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". Both actforms are exactly identical from a molecular mass point of view (and thus also from the **GNU polyxmass**' point of view).

The Cleavage Specifications

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

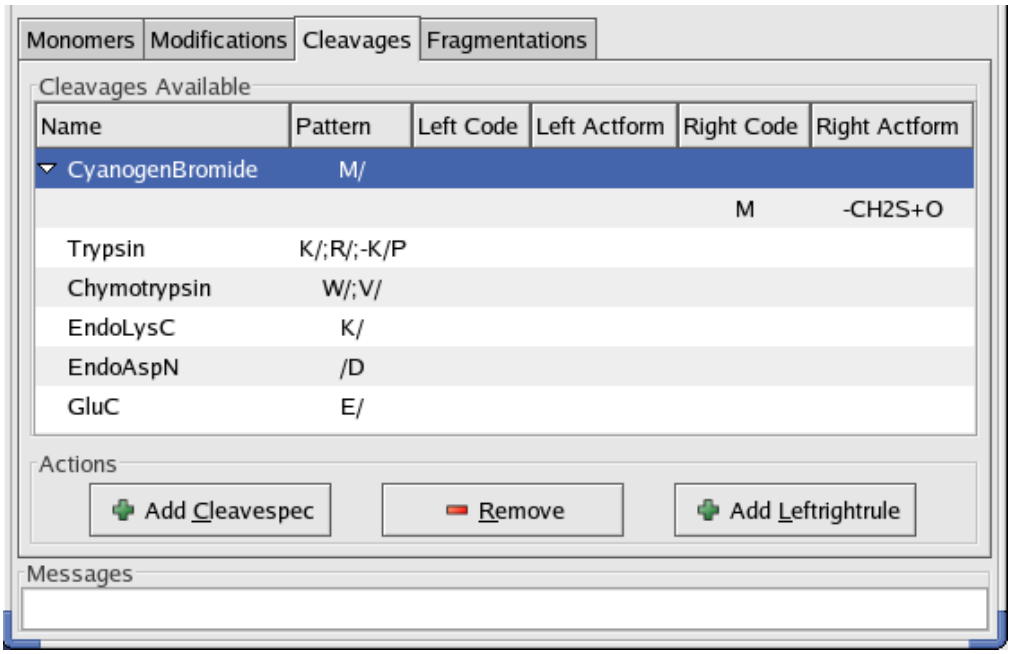

Figure 6.3: **Interface for the definition of the polymer cleavage specifications.** This is the same window as for Figure 6.1, but this time the cleavage specifications definition tab of it is shown. This figure shows that a cleavage specification is defined in a more complex way than previously described for monomers or modifications; see text for further details.

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

For each different polymer type, the molecular scissors are going to be somewhat specific. Indeed, a protease will almost certainly be unable to cleave whatever 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 6.3 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 6.3, 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**, respecively. 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):

THISMWILLMBECUTMANDTHATMALSO

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

THISM WILLM BECUTM ANDTHATM ALSO

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

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

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

Now that we got trained to think in an abstract way with these leftrightrules, 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 but more complex than leftrightrules: *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 27, the fragmentation specification is a tricky business. Figure 6.4 shows examples of protein fragmentation specifications for fragment types *a, b, c, z, y, x, imm.*

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

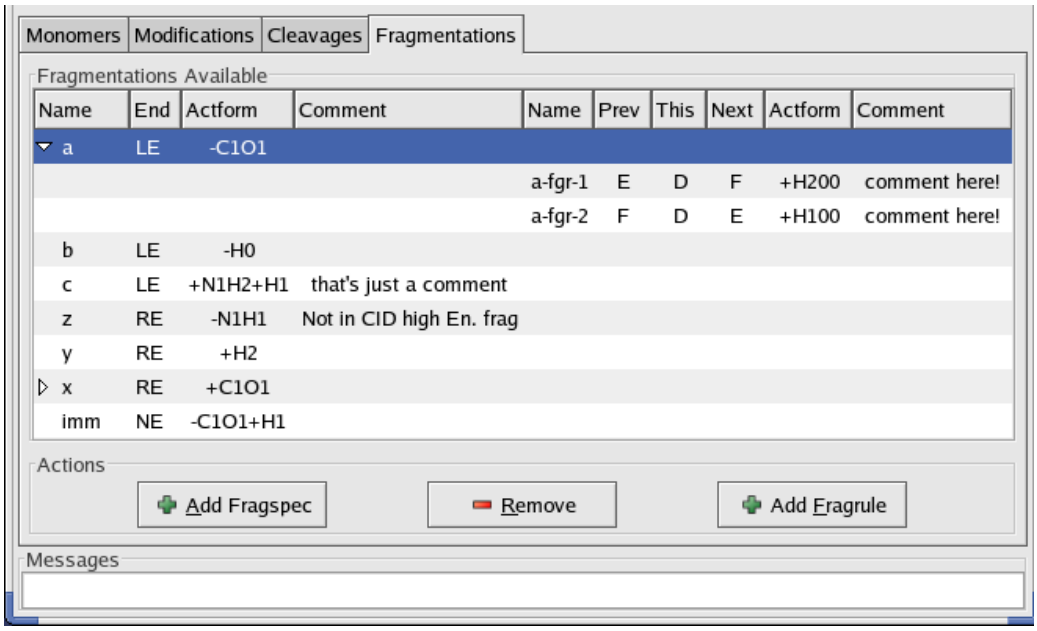

Figure 6.4: **Interface for the definition of the polymer fragmentation specifications.** This is the same window as for Figure 6.1, but this time the fragmentation specifications definition tab of it is shown. This figure shows that a fragmentation specification is defined in a more complex way than previously described for cleavage specifications; see text for further details.

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 27);
- 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 27;

- 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 6.4), 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.

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. 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. 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 described in another chapter. Indeed, **GNU polyxmass** is a very 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 data 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: **GNU polyxcalc**!

[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.

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.

GNU polyxcalc Invocation

The **GNU polyxcalc** module is simply called by its name: polyxcalc from a command line. The user is invited to launch the following command and to inspect the various options that it accepts:

```
bash-2.04 $ polyxcalc --help ↩
```

This command produces the following output:

```
Usage: polyxcalc [OPTION...]
  -d, --details          prints the copyright owner and the licensing of the
                         program
  -l, --license          prints the license type of the program
  -v, --version          prints the version of the program
  -t, --type=poltype     the type of the polymer chemistry definition
                         being requested (ie "protein", with no quotes)

Help options:
  -?, --help             Show this help message
```

```
--usage                Display brief usage message
```

GNU polyxcalc Operation: An Easy Task

The way **GNU 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 7.1.

As the reader can see, there are a number of items that **GNU 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 are set and the Result Masses are empty, they are taken into account (**GNU 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 **GNU polyxmass** mass spectrometry software framework. 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;

- Monomers If a polymer chemistry definition file was chosen by clicking onto the New toolbar menu button, 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 **GNU 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 **GNU polyxcalc**. A "protein" sequence may be this "MAMISGMSGRKASPTSPINADK", for example, which is the N-terminal end of the chicken gizzard telokin.[1] Same as above applies for the Count text entry widget;

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

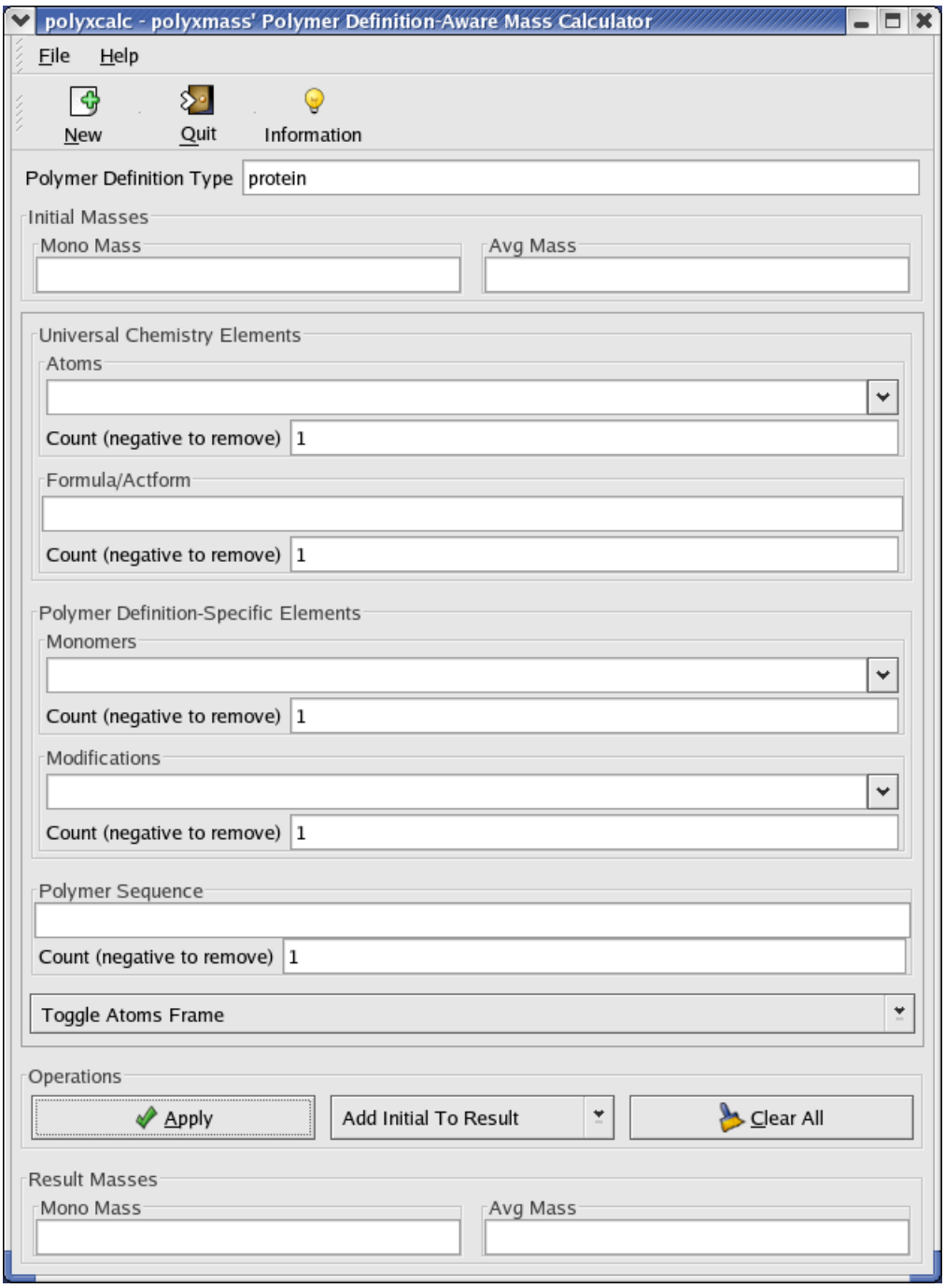

Figure 7.1: **Interface of the GNU polyxcalc module.** This figure shows that the **GNU polyxcalc** polymer definition-aware module can handle atoms, actforms, monomers, modifications and even polymer sequence for computing masses.

Noteworthy, when **GNU 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.

The multi-option menu button widget labeled Toggle Atoms Frame actually contains other similar menu items that unfold when the widget is clicked. The other menu items do exactly the same for all the other items that we have reviewed above: make them either disappear or reappear in a switch-like manner. This is so that the graphical user interface may be simplified if, for example, the user never needs to select individual atoms or modifications...

The Operations frame widget contains three widgets:

- Apply This button is the one through which all calculations are elicited. When this button is clicked, all the widgets detailed above are screened and checked for content. If content is found (let's say a Modification entity is found in the corresponding widget), the Count is checked. If this count is non-empty and non-0, the chemical entity is taken into account in the mass calculations;

- Multi-Option Menu Button This widget contains the following sub-menu items:

  - Add Initial To Result When this multi-option menu item is selected, the masses (if any) located in the Initial Masses text entry widgetsare added to the ones located in the Result Masses text entry widgets;

  - Remove Initial From Result This is a rather similar approach as above, unless the masses in the Initial Masses text entry widgets (if any) are removed from the ones in the Result Masses text entry widgets;

  - Send Result To Initial This is rather self-explanatory: masses from the Result Masses text entry widgets are sent to the Initial Masses text entry widgets thus overwriting the masses that could have been displayed there;

  - Clear Initial Masses Self-explanatory;

  - Clear Result Masses Self-explanatory;

  - Clear All Chem. Data This will erase all the data that are currently displayed in the chemistry-related widgets that we reviewed above. The Count text entry widgets are also reset to empty;

  - Cancel Operation This menu item is a "safe-conduit": if the multi-option menu was clicked and no available option needs to be chosen, the user *must* choose this cancellation option, otherwise any other option that gets selected upon releasing the widget will be executed.

- Clear All This button simply resets to empty all the mass-related and chemistry-related entry widgets along with all the Count text entry widgets.

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

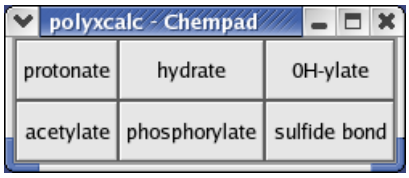

Figure 7.2: **Interface of the chemical pad.** This figure shows that the chemical pad is very similar to what a numerical calculator would display. Here, the user has programmed a number of chemical reactions.

GNU polyxcalc Is A Programmable Calculator

For the scientists who work on molecules that are usually modified in the same usual ways, **GNU polyxcalc** features a built-in mechanism by which they can easily program their polymer chemistry-aware 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 7.2, 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 **GNU polyxcalc** main window by simply clicking on their respective item.

The configuration of the Chempad is very easy, as shown in the code below (excerpt taken from file `polyxmassdata/protein/chempad.conf`):

```
#chempad_rows$3
chempad_columns$3

chempadkey=protonate%+H1%adds a proton
chempadkey=hydrate%+H2O1%adds a water molecule
chempadkey=0H-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).

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 **GNU 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.

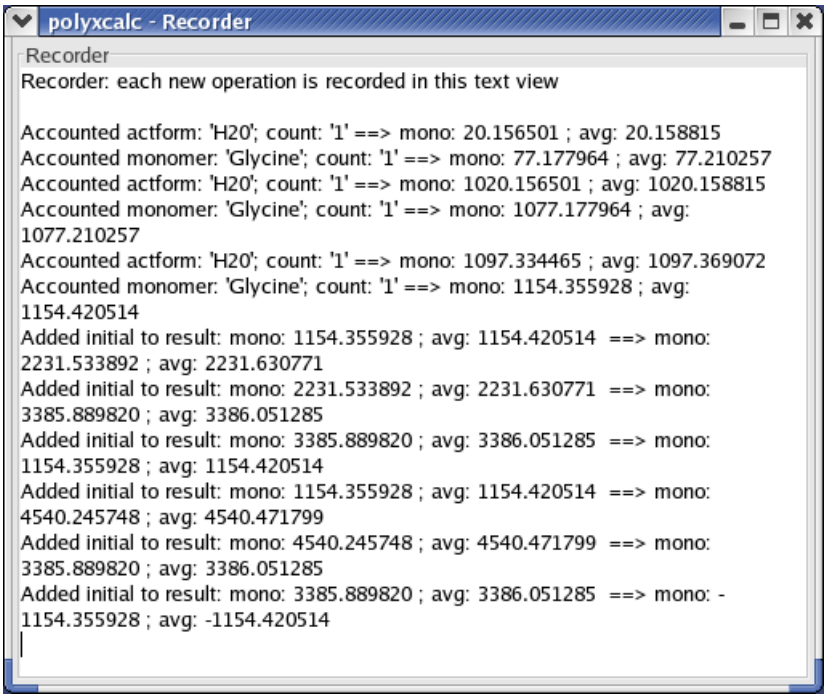

Figure 7.3: **The GNU polyxcalc recorder window.** This figure shows that the recorder window is a simple textview widget that records all the mass-significant operations in the **GNU polyxcalc** calculator. The text in the recorder may be selected and later used in an electronic logbook or printed.

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.

GNU polyxcalc Is LogBook-Friendly

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

This recorder window is shown in Figure 7.3. The text in the recorder window is editable for the user to customize the **GNU 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.

GNU polyxedit Invocation

The **GNU polyxedit** module is simply called by its name: polyxedit from a command line. The user is invited to launch the following command and to inspect the various options that it accepts:

```
bash-2.04 $ polyxedit --help ↩
```

This command produces the following output:

```
Usage: polyxedit [OPTION...]
  -d, --details     prints the copyright owner and the licensing of
                    the program
  -l, --license     prints the license type of the program
  -v, --version     prints the version of the program

Help options:
  -?, --help        Show this help message
  --usage           Display brief usage message
```

If the user passes to the command line strings that do not correspond to any of the options above, then **GNU polyxedit** considers them to be filenames of polymer sequences, and thus tries to open these files. If a polymer sequence file could not be found or opened, a warning message is logged to the console.

GNU polyxedit Operation: *In Medias Res*

A typical **GNU polyxedit** session looks like what is shown in Figure 8.1 on the facing page.

As the reader can see, there are a number of items that we need to describe and explain. This is the beginning of a journey in the guts of **GNU polyxedit**... Keep reading the next numerous sections!

GNU polyxedit Main Program Window: The Menu

In this short section we will review the different menus that are currently available in the **GNU polyxedit** main window. It is most probable that new menu items will be added when new features are added to the program. The menu items described below apply to any polymer sequence that may be currently visible in the program. There is another contextual menu that will be described later.

As with usual menus, the menus in **GNU polyxedit** are hierarchical, and we'll analyse each parent menu along with its submenus (and subsubmenus):

- *File*
  - ⟶*New*... Create a new polymer sequence;
  - ⟶*Open*... Open a polymer sequence;
  - ⟶*Close All*... Not yet implemented;
  - ⟶*Quit*... Quit the program;
- *View*
  - ⟶*Mass Display*... Open a window where the mono/avg masses are displayed both of the entire polymer sequence and of the currently selected region;
- *Help*
  - ⟶*Information*... Show a window with the version of the program and a short notice on the license of the program.

That is a fairly trivial menu. As I stated above, there is another menu that should be explained. That menu is displayed in two circumstances:

- When the user right-clicks onto the sequence-displaying area;
- When the user presses the F10 keyboard key while the focus is on a sequence editor window.

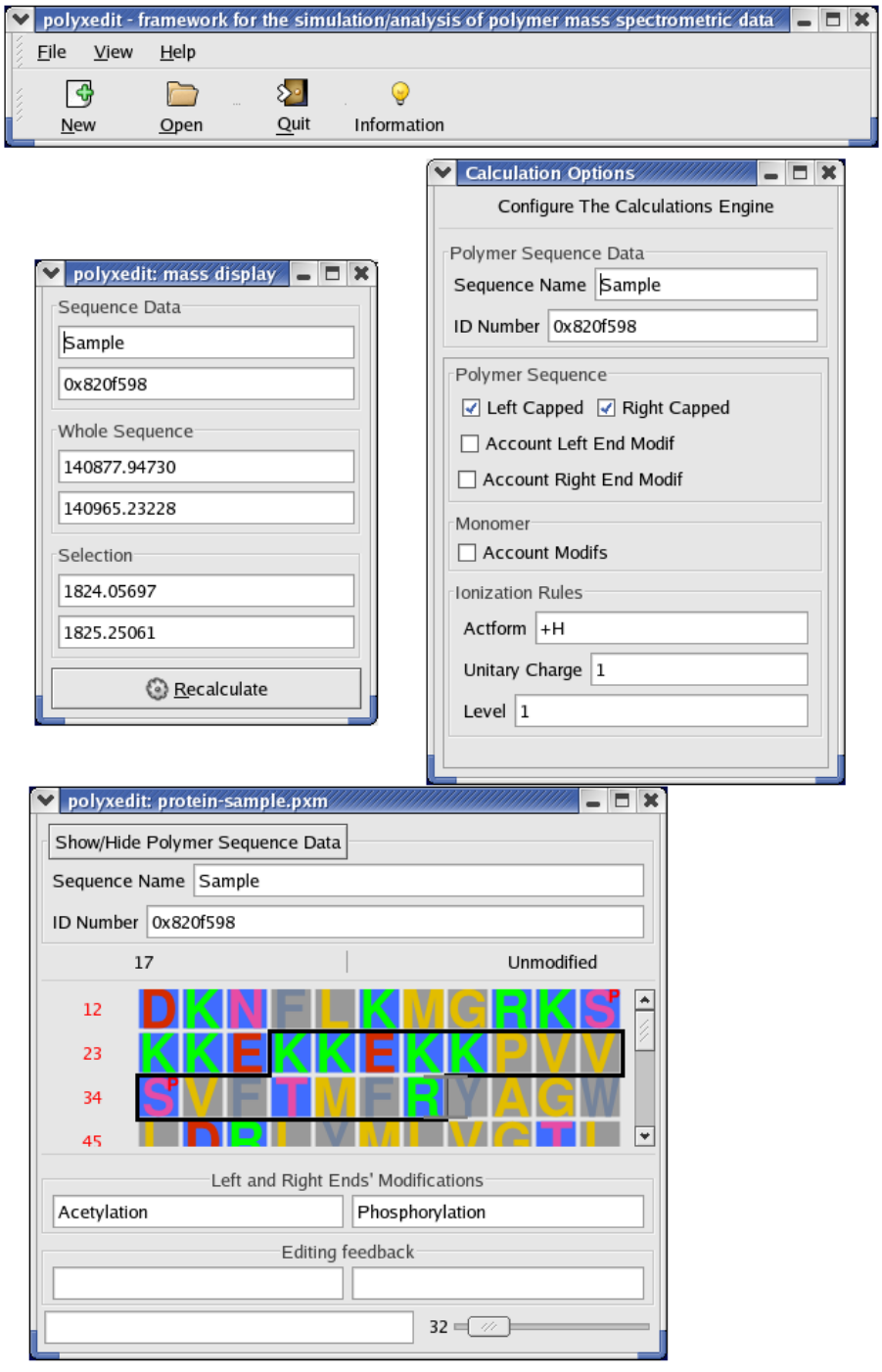

Figure 8.1: **Interface of the GNU polyxedit module.** This figure shows that the **GNU polyxedit**'s minimally useful environment is made of four different windows: the program's main window, with its menu and toolbar, the Calculation Options window, the Mass Display window and finally a polymer sequence editor with a sequence loaded in it.

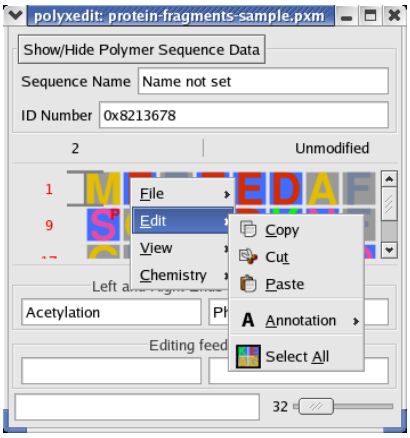

Figure 8.2: **The polymer sequence editor contextual menu in GNU polyxedit.** This figure shows the contextual menu that is popped up when the user right-clicks onto the polymer sequence editor in **GNU polyxedit**. Keying-in the F10 key elicits the same effect.

That contextual menu is visible in the Figure 8.2. We will review each of its items below:

- *File*
  - ⟶*Save*... Save the polymer sequence;
  - ⟶*Save As*... Save the polymer sequence with a new name;
  - ⟶*Close*... Close the polymer sequence;
- *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;
  - *Annotation*
    - ⟶*Monomer*... Edit (add/remove/modify) the notes for the monomer lying below the cursor when the menu was elicited (or the lastly right-clicked monomer if the menu was popped up by keying-in the F10 key;
    - ⟶*Polymer*... Edit (add/remove/modify) the notes for the polymer being edited in the polymer sequence editor;
  - *Select All*... Selects the whole sequence in the polymer sequence editor;
- *View*
  - ⟶*Calc. Options*... View/Modify the way calculations are performed, be them mass calculations or elemental composition calculations;
- *Chemistry*

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

This is a somewhat more involved menu, however it should be pretty easy to understand. Note that each action undertaken as the response to choosing one menu item is performed onto the polymer sequence onto which the contextual menu was popped up. That is a rather intuitive and standard way of doing things.

Creating A New Sequence

When a brand new polymer sequence file is created, the user is provided with a window where she is asked to choose the polymer chemistry type of that to-be created polymer sequence and where the user may enter both the name of the polymer sequence and a code for that new polymer sequence (both name and code are optional). That window is shown in Figure 8.3 on the following page.

There, we can see that the polymer chemistry definitions –currently available to the system– have been checked and are displayed in the Available Polymer Definition listview so that the user may select one.

Once the user has filled-in the required (with optional) data, she can click the Validate button. At that point a polymer sequence editor is shown (Figure 8.4 on the next page), where editing can take place, because the internals of the **GNU polyxmass** suite know the correspondence between the monomers and the codes that are keyed-in in the polymer sequence editor (thanks to knowing of what polymer chemistry definition the edited sequence is).

Displaying Masses

As soon as a polymer sequence is read from disk, the user may want to have masses displayed for it. The *View*⟶*Mass Display* menu from the main program window will open the polyxedit: mass display window that is shown in Figure 8.1 on page 69. As can be seen, there

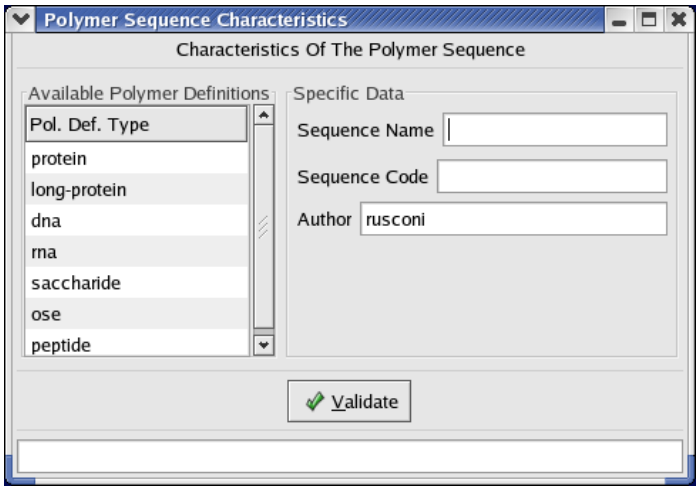

Figure 8.3: **Definition of the characteristics of a new polymer sequence.** This figure shows how the user can enter the data characterising the new polymer sequence to be created. The compulsory data to fill in the form is the polymer chemistry definition, to be chosen from the list of available polymer chemistry definitions.

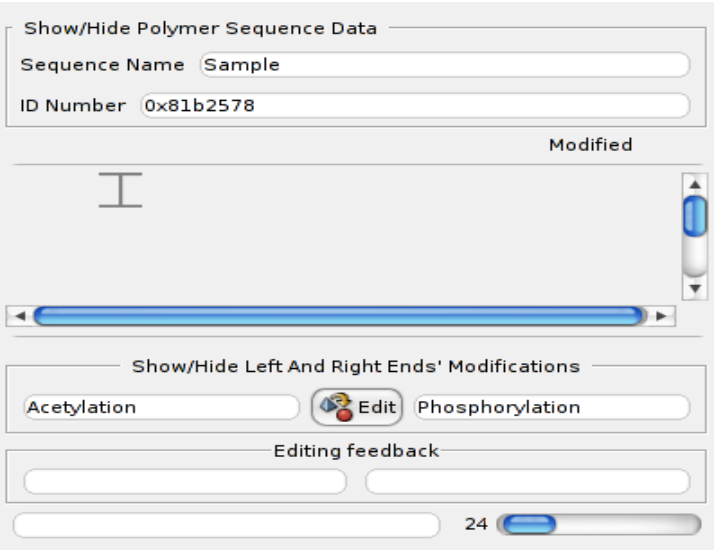

Figure 8.4: **A polymer sequence editor window with an empty sequence.** This figure shows how the polymer sequence editor looks like when it is empty (without sequence in it).

are three frames in this window, the first dealing with the polymer sequence data, the other two dealing with masses.

The first mass-related frame (Whole Sequence) contains two text entry widgets where masses are displayed. The first text entry contains the *monoisotopic* mass of the whole polymer sequence (for example, of the entire protein). The second text entry widget contains the *average* mass of this same polymer sequence.

The second mass-related frame (Selection) contains two text entry widgets where masses are displayed, exactly as above, unless this time the masses displayed correspond to the currently selected region of the polymer sequence. If no selection is currently made, then the virtual selection is considered to run from the first monomer in the polymer sequence up to the monomer left of the current cursor location.

If the words *monoisotopic* or *average* do not seem familiar to you, it might be useful to peruse the two chapters 3 (page 17 and 4 (page 35).

Try experimenting with an open polymer sequence and move the cursor around in the sequence. See how only the Selection masses do change. Also, try selecting some regions in the text and see how, here also, only the Selection masses do change. Check if, when the entire polymer sequence is selected, the masses displayed in both the Whole Sequence and Selection frames are identical or not.

There is *only one* polyxedit: mass display window in the **GNU polyxedit** module, even if more than one polymer sequences are open at any given time (which is actually the reason why this menu resides in the program's main window). The way masses are displayed for different polymer sequences is simply by updating the mass values in this window when the focus[1] moves from a polymer sequence to another. This system ensure that the masses that are displayed always pertain to the currently "active polymer sequence editor window".

Configuring The Calculations

As soon as the user wants to have masses displayed for a given polymer sequence, it is almost certain that she will want to configure the way these masses are computed. The *View⟶Calc. Options* menu will open a window entitled Calculation Options. This window is visible on the Figure 8.1 on page 69. As usual, there is a frame that deals with the name and identity of the polymer sequence for which these options are available.

Another frame, Polymer Sequence contains a number of widgets –mainly checkbuttons– so that the user can configure which chemical entities must be taken into account when masses are computed. The checkbuttons are rather self-explanatory.

Another frame, Monomer will let the user define if the monomer chemical modifications must be taken into account when computing masses.

The last frame, Ionization Rules, will let the user define the way both the whole polymer sequence and the selection must be ionized. The default ionization step is defined in the polymer chemistry definition, but the user can modify it in the text entry widgets made available to her here.

All the configurations that the user will perform in this Calculation Options window will apply to both the whole polymer sequence masses and to the selection masses (see above for the definition of these "whole sequence" and "selection" concepts).

[1]The focus is placed on a polymer sequence editor window when the mouse cursor hits it; as soon as at least one polymer sequence window is open, there is always a "last active polymer sequence window", which is the polymer sequence window that was hit last with the mouse cursor.

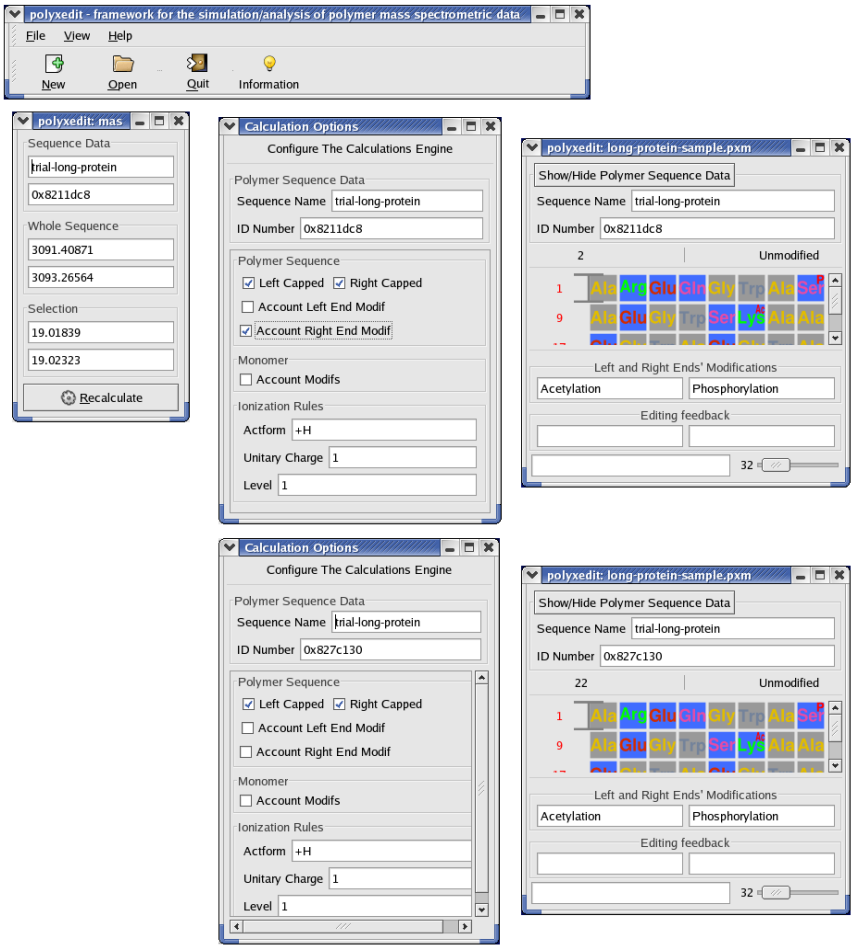

Figure 8.5: **The same polymer sequence opened twice in GNU polyxedit.** This figure shows the same polymer sequence being loaded twice in **GNU polyxedit**. The two polymer sequence editor windows are entirely separated in memory and thus totally independent.

Since it may be of great usefulness that two identical polymer sequences be either ionized differently, or that their masses be computed in different manners, *each polymer sequence window has its own Calculation Options window.* This is shown in Figure 8.5 on the facing page: a single polymer sequence file was opened twice as two entirely distinct objects in memory, so that changing either the sequence or the way masses are computed for one of the two polymer sequence editors will be relevant only to the corresponding polymer sequence.

This is clearly visible on the Figure 8.5 on the preceding page because, while both sequence editor windows display the same Sequence Name text widget's contents, the ID Number text widgets do contain different data: the two polymer sequence objects in memory are different and thus entirely distinct one from the other.

The top polymer sequence has its masses computed by taking into account the monomer modifications, while the bottom polymer sequence has its masses computed by *not* taking into account the monomer modifications. This mechanism gives the user a great flexibility in the manner any comparison may be performed in the way masses are computed, sequence differences are monitored real time, by editing one polymer sequence and not the other, for example...

Editing Polymer Sequences

As we have seen in the **GNU 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 52). 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 8.6 on the next 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' (panel 1, Figure 8.6 on the following page).

So, naturally, the user keys-in A (panel 2, Figure 8.6 on the next page). 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 due 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 'A' character: **GNU polyxedit** cannot figure out which monomer code the user actually means when keying-in A.

Figure 8.6: **Multi-character code sequence editing in GNU 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.

There is a way, called *completion*, to know which monomer codes –in the current polymer chemistry definition– start with the keyed-in character(s) ('A' for us now). The user can always enter the *completion mode* by using the tabulation [TAB] key. This is what is shown in the small window right of panel 2, Figure 8.6 on the facing page. 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 right monomer 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, Figure 8.6 on the preceding page. 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 right of the panel 3, Figure 8.6 on the facing page. 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 letters "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 seventh monomer, shown in panel 4, Figure 8.6 on the preceding page.

Unambiguous Single-/Multi-Character Monomer Codes

Let's imagine that we have now 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,* ***GNU 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 Monomer Codes

Equally interesting is the fact that if you key-in the [TAB] key while no monomer code is being edited (the left text widget under the label Editing Feedback is empty), all the monomer codes defined in your polymer chemistry definition are displayed, exactly as shown in the panel ALL, Figure 8.6 on the facing page.

Figure 8.7: **Bad code character in GNU 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 8.7. 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.

Sequence Selections: The Various X Mechanisms

As any text editor, the **GNU 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 **GNU polyxedit** get the best features of selection and pasting.

When the user tries to paste a sequence element from the clipboard (say, after copying

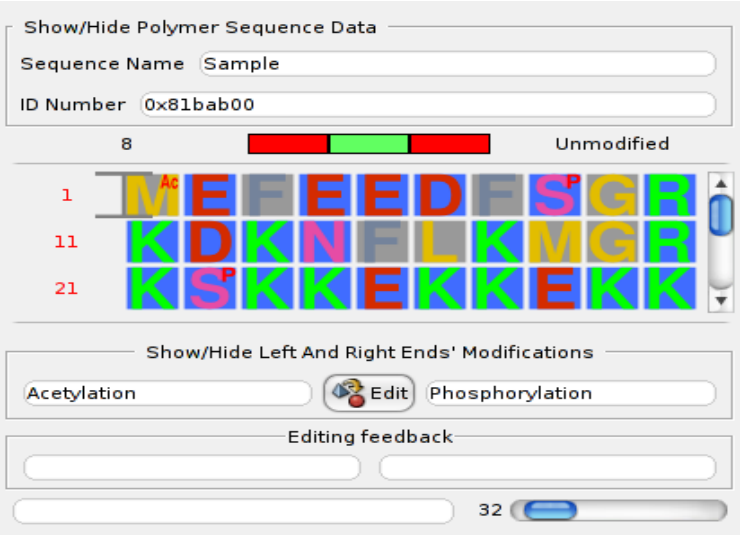

Figure 8.8: **Visual feedback in the GNU polyxedit sequence editor.** This figure shows the feedback that the user is provided when moving the mouse cursor over monomer icons. See the text for details.

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 **GNU 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 Unmodified label informs the user that the sequence was not modified since it was either last written to file on disk or last read from file. Upon modification of the polymer sequence, this label changes into Modified;
- The 22 label indicates the position in the polymer sequence of the monomer onto which the cursor is positioned at any given moment[2].
- 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

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

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.

GNU 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.

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 menu:

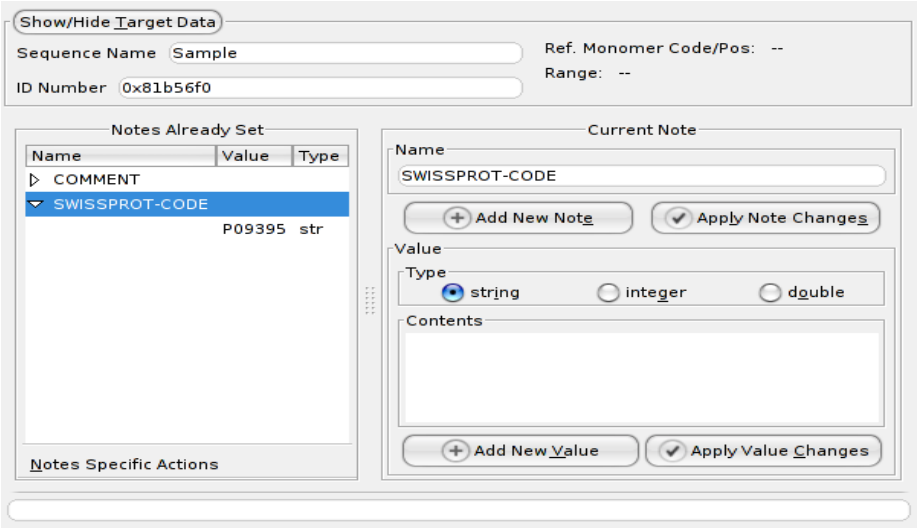

Figure 8.9: **Annotating polymer sequences.** This figure shows the graphical interface to the annotation of polymer sequences.

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, 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. In the Figure 8.9, we can see that the note **SWISSPROT-CODE** has a noteval object of type **str** (for "string") and of contents **P09395**. Adding an integer notevalue pair would be as simple as clicking onto the integer radiobutton widget and entering the corresponding number in the Contents textview widget. Clicking onto Add New Value will add one item under the node **SWISSPROT-CODE**. See Figure Figure 8.10 on the next page.

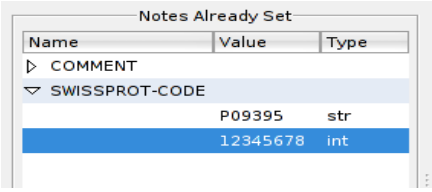

Figure 8.10: **Adding a noteval object to a note.** This figure shows the apparition of a new item under the relative note node in the treeview when a new integer noteval object is added (compare with previous Figure 8.9 on the page before).

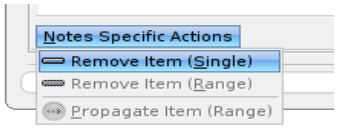

Figure 8.11: **The menu governing actions on note items.** This figure shows the menu that the user may use in order to remove any item currently selected in the treeview. When the window is opened in single-mode, the range-mode actions are inactive.

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 8.11.

Setting notes to the polymer sequence as a whole is conceptually simpler than what we are about to visit: the annotation of 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*.

Managing Monomer Notes In Single-Mode

If the annotation pertains to a single monomer in the sequence,[3] 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*

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

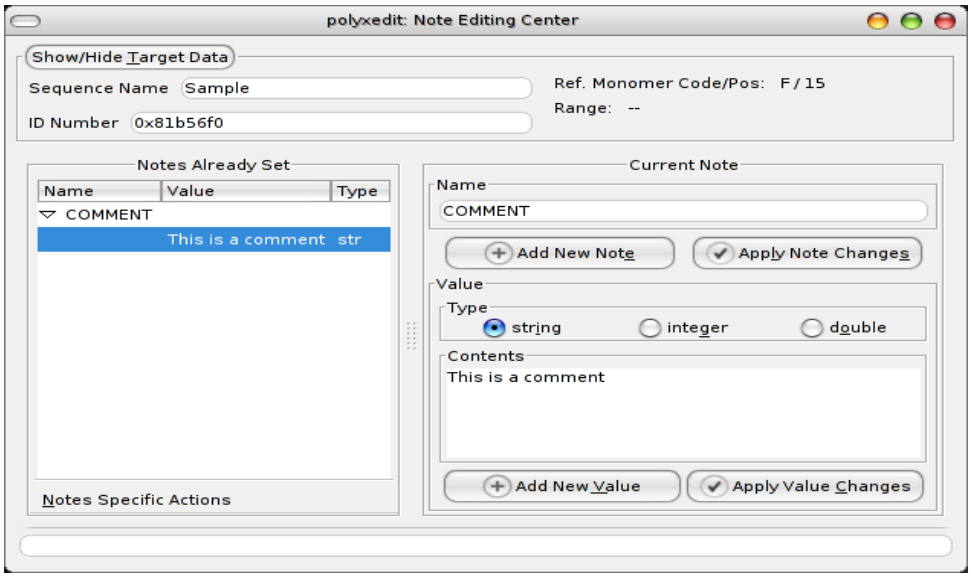

Figure 8.12: **Annotating monomers in single-mode.** This figure shows the graphical interface to the annotation of monomers in single-mode.

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 8.12, 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 8.13 on the following page for a good example of such note.

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 8.13 on the next page, the user must select the following menu item from the contextual menu:

Edit⟶*Annotation*⟶*Monomer*⟶*Range*

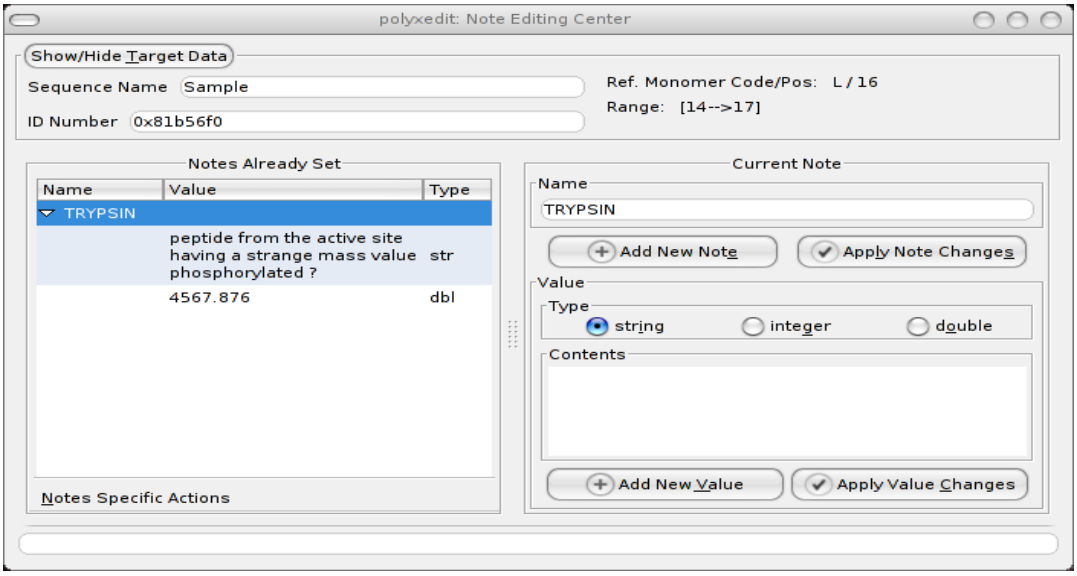

Figure 8.13: **Annotating monomers in range-mode.** This figure shows the graphical interface to the annotation of monomers in range-mode.

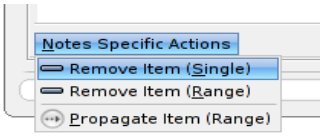

Figure 8.14: **Specific menu items available in range-mode.** This figure shows that the menu items that act in range-mode are made distinguishable from the single-mode menu item using the labelling (Single) or (Range).

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 right button mouse-clicked (to elicit the popping up of the contextual menu). This *reference monomer* is very important, as we'll see in a moment.

The Figure 8.13 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 8.14):

- 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 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.

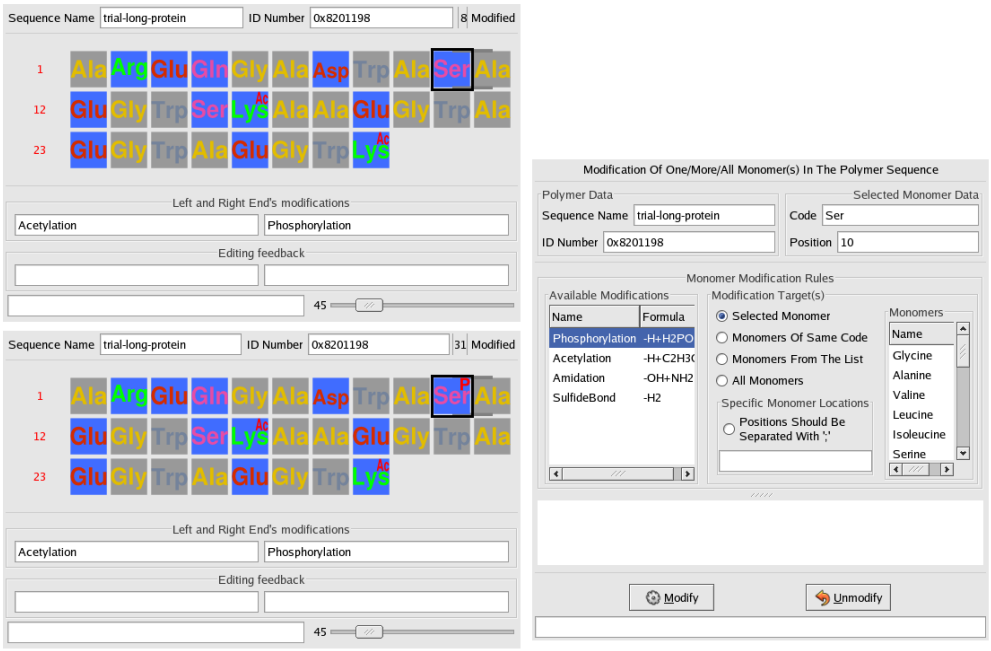

Figure 8.15: **Modification of a monomer in a polymer sequence.** This figure shows the graphical rendering of a phosphorylation of a seryl residue in a protein polymer sequence.

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 8.15 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 modifi-

cations 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 102, the file of interest is `monomer-modif.dic`).

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.[4] 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.

[4] That's protein chemistry stuff.

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[5] would have been un-modified.

Now, if you un-select all the items in the list of available modifications[6], 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.

The modifications that we describe here are not of this kind. They apply 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. The Figure 8.16 on the facing 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 8.16 on the next 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.

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

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

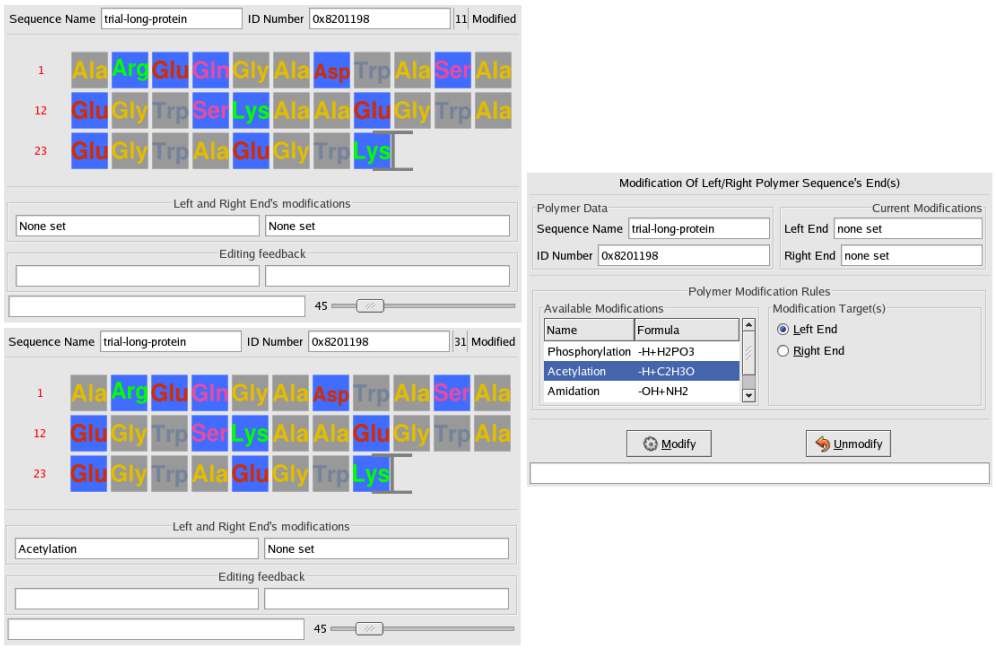

Figure 8.16: **Modification of a monomer in a polymer sequence.** This figure shows how simple it is to permanently modify a polymer sequence on either or both its left/right ends. The permanent modifications currently set to a polymer sequence are conveniently listed in two text widgets located under the polymer sequence rendering area.

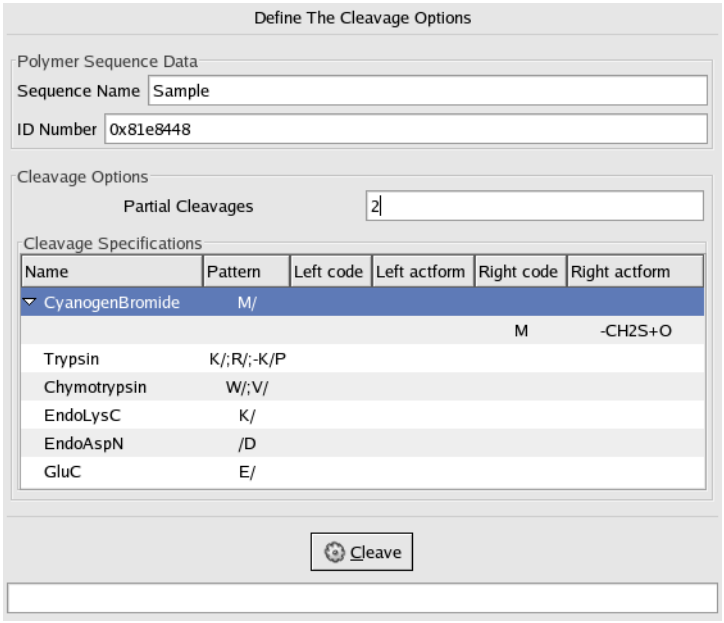

Figure 8.17: **Cleavage options window.** This figure shows the window with which the user is provided when she performs a polymer sequence cleavage. The user can select one cleavage specification and specify what level of partial cleavage the chemical cleavage should perform.

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**.

It is a matter of having a polymer sequence opened in an editor window and selecting the *Chemistry⟶Cleave* menu. The user is provided with a window where a number of cleavage specifications are listed (Figure 8.17). 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

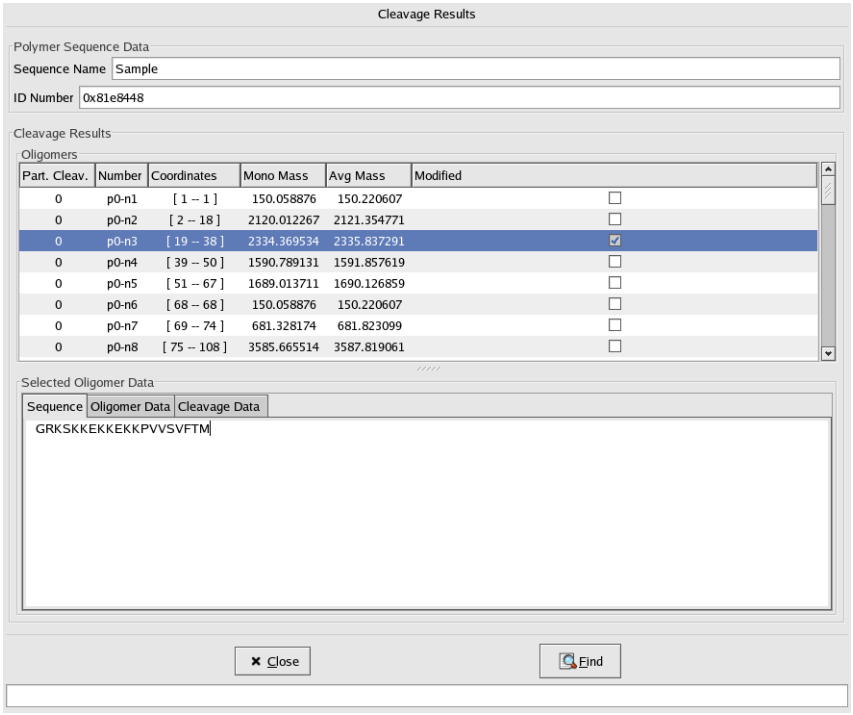

Figure 8.18: **Cleavage-generated oligomers window.** This figure shows the window that is opened so that the oligomers generated upon cleavage of a polymer sequence can be displayed. Other data are also displayed (see text for details).

the user may enter rather high values here.

Upon successful termination of the cleavage reaction, the user is provided with a new window (Figure 8.18) 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;
- 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 section 15 on page 73);
- Avg Mass Same as above, but for the average mass;

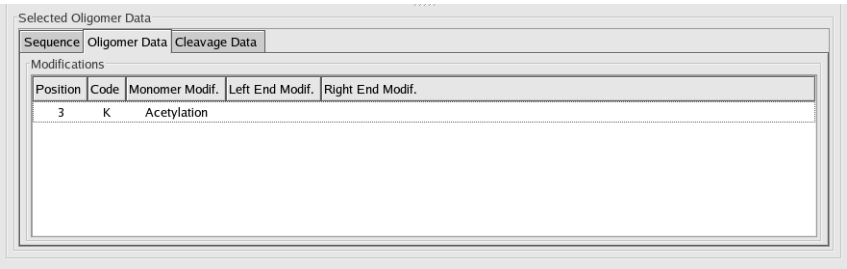

Figure 8.19: **Cleavage-generated oligomers' data.** This figure shows the notebook tab in which data pertaining to a selected oligomer are displayed. In particular, this tab contains a listview where monomer modifications of the selected oligomer (if any) are displayed.

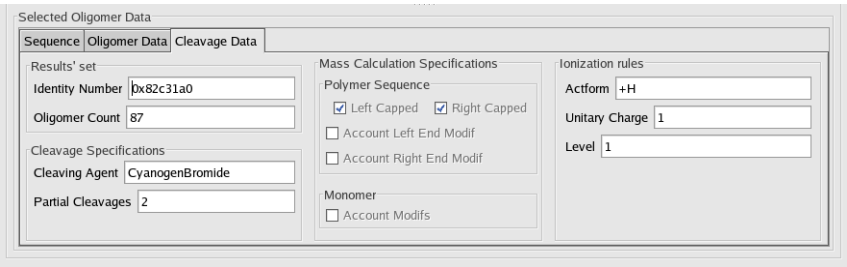

Figure 8.20: **Cleavage specification data.** This figure shows the notebook tab in which data pertaining to the cleavage operation are displayed.

- 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 8.18 on the page before) This is the sequence that is displayed when an oligomer is selected in the listview displaying the oligomers (in the upper pane);

- Oligomer Data (Figure 8.19) 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 8.20) 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 20 on page 95)

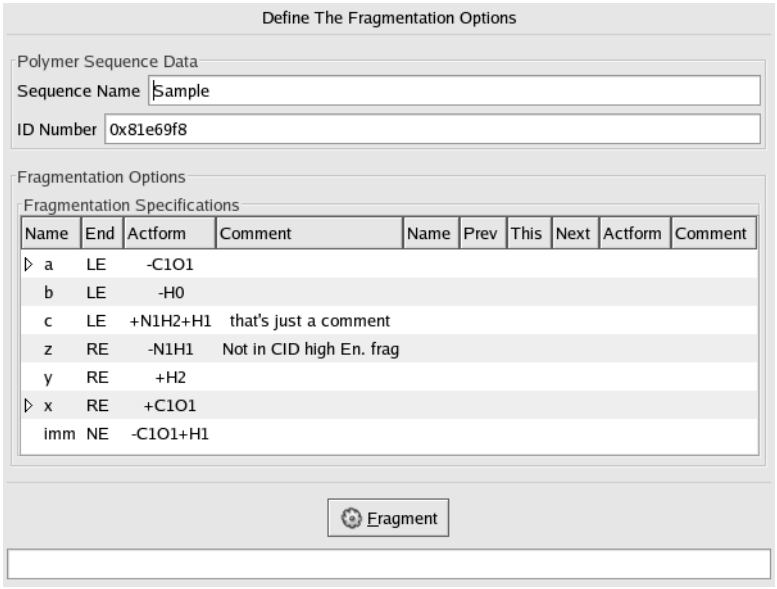

Figure 8.21: **Fragmentation options window.** This figure shows the window with which the user is provided when she performs a polymer sequence fragmentation. The user can select one or more fragmentation specifications (patterns).

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 8.21). These fragmentation specifications are listed by looking into the polymer chemistry definition corresponding to the polymer sequence to 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 8.22 on the following 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;

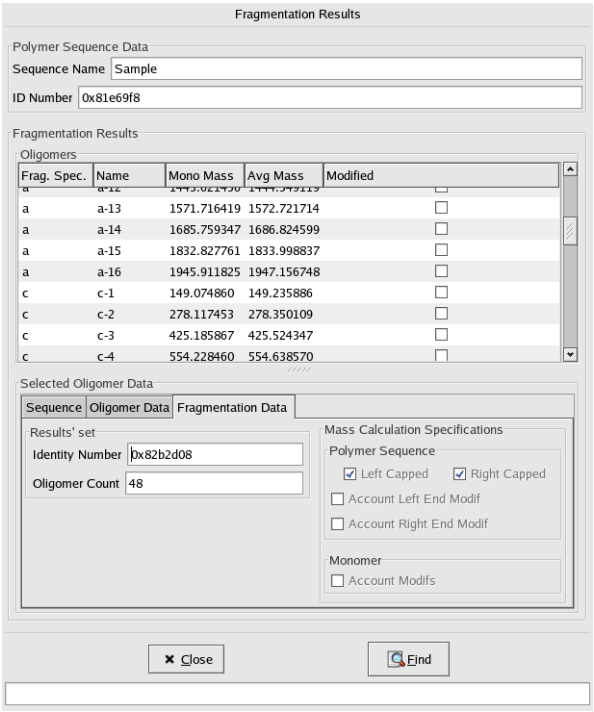

Figure 8.22: **Fragmentation-generated oligomers window.** This figure shows the window that is opened so that the oligomers generated upon fragmentation of a polymer sequence can be displayed.

- 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 section 15 on page 73);
- 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 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 20 on page 90).

The button labelled Find will allow the user to find masses in the oligomers that were generated upon the fragmentation reaction simulation (see section 20).

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 20 on page 97). To allow this, and as shown in Figures 8.18 to 8.22 on pages 91–94, 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 8.23 on the next page 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 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 8.23 on the following page, the

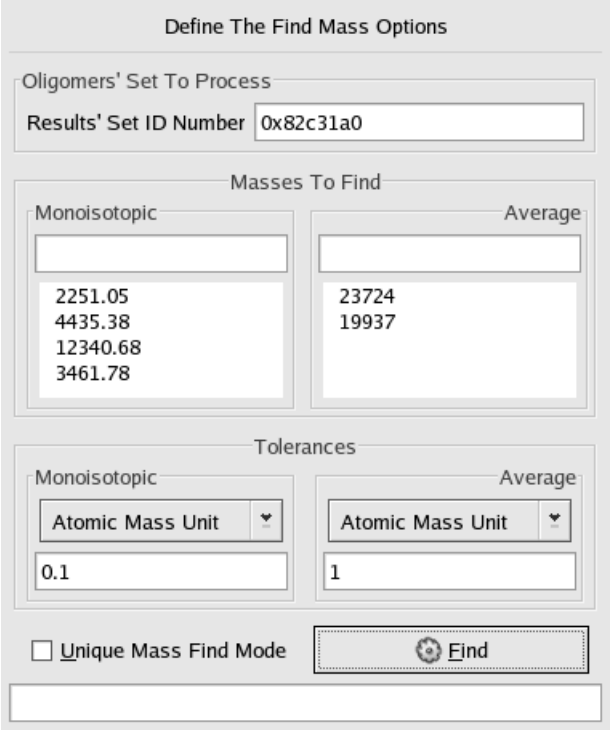

Figure 8.23: **Finding masses in a set of oligomers.** This figure shows how to ask that masses be found in a set of oligomers that result, for example, from the cleavage of a polymer sequence.

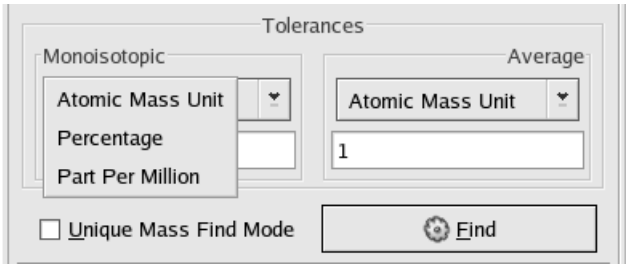

Figure 8.24: **Tolerances available in finding masses.** This figure shows the three different ways that tolerances can be configured.

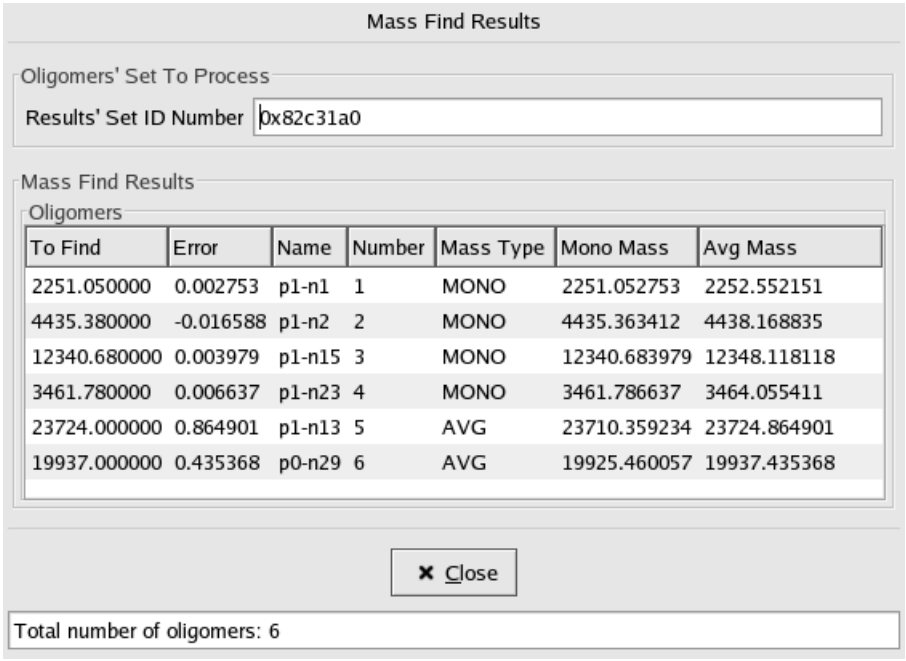

Figure 8.25: **Finding masses in a set of oligomers.** This figure shows oligomers that were found in a set of oligomers after a mass finding operation has been performed.

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 8.24 on the preceding page 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 8.25.

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 8.23 on the preceding 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 8.26 on the following page shows up and the user enters masses to search for (see section 20 on page 95 for details on the workings of a very similar window).

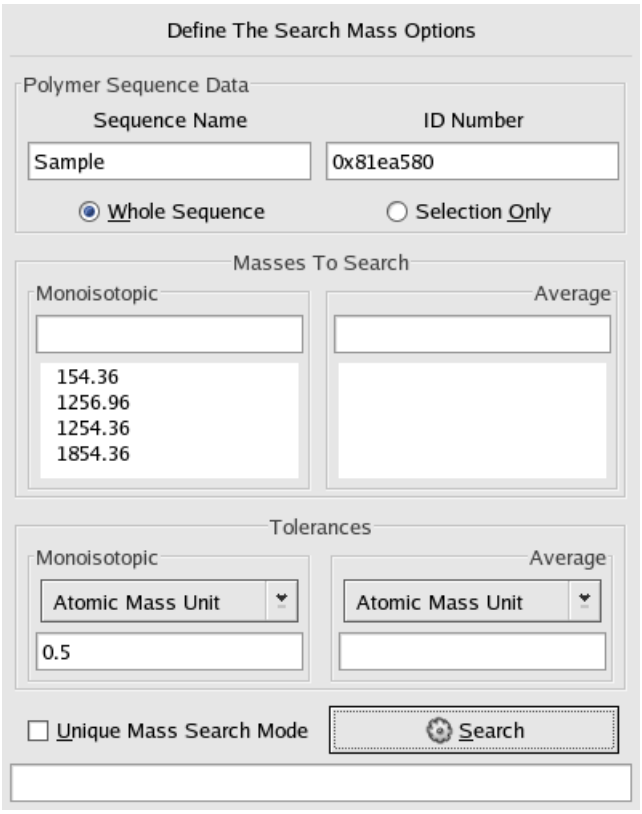

Figure 8.26: **Finding masses in a polymer sequence.** This figure shows how to ask that masses be searched in a polymer sequence.

Once the masses have been searched, if results are found they are displayed in the window shown in Figure 8.27 on the following page. This window has very similar characteristics to the ones of the previously described results' windows (see section 20 on page 90, for example).

The button labelled Find will allow the user to find masses in the oligomers that were generated upon the mass searching operation (see section 20 on page 95.

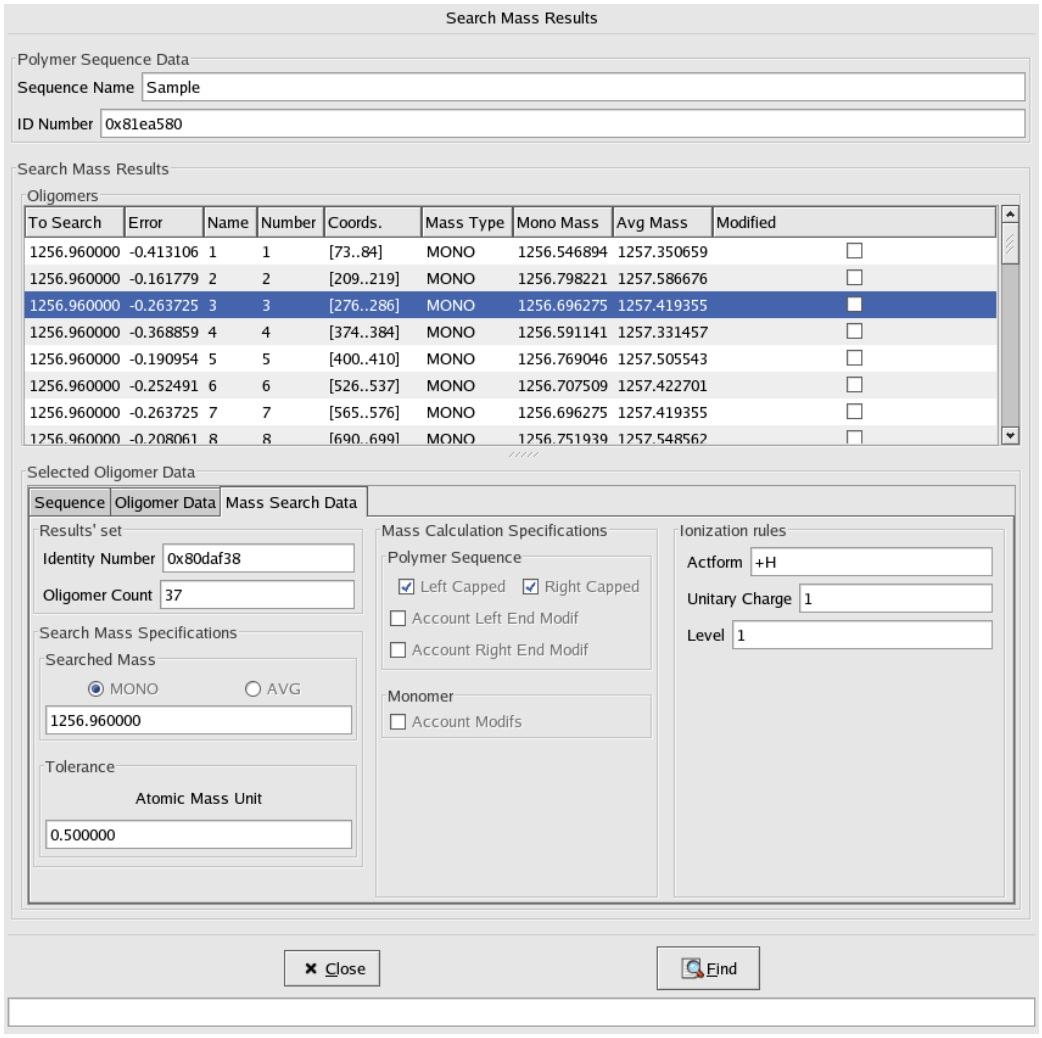

Figure 8.27: **Results window after searching masses in a a polymer sequence.** This figure shows the oligomers that were found upon a mass search operation.

9

GNU polyxmassdata: The Configuration Data Hierarchy

The **GNU polyxmass** software suite is designed to be compatible with any polymer chemistry that the user may want to define. To be that flexible, **GNU polyxmass** has to be able to store polymer chemistry definition-related data in a very clearly designed set of data directories and files. This configuration data hierarchy (which relates in some ways to a "filesystem") is what this chapter is all about.

After having read this chapter, the reader will be able to configure the **GNU polyxmass** configuration data filesystem hierarchy in such a way that a brand new polymer chemistry definition is made available to the **GNU polyxmass** software suite. The creation of a brand new polymer chemistry definition is typically performed using the **GNU polyxdef** module; see chapter 6 on page 49. This chapter will focus on how to register this new polymer chemistry definition with the **GNU polyxmass** software suite.

A polymer chemistry definition is only useful to edit any polymer sequence that complies with it (this editing is typically done with the **GNU polyxedit** module, see chapter 8 on

page 67), if it is associated with graphical files that are used by the sequence editor to render the sequence graphically. This chapter will teach the user to configure the **GNU polyxmass** filesystem in such a way that the graphical files can be automatically used by the polymer sequence editor when a given sequence is opened in **GNU polyxedit**. Further, the chemical modification of a polymer sequence, or chemical reactions simulations can only be performed if the polymer chemistry definition file is correctly integrated in the whole **GNU polyxmass** filesystem hierarchy.

What Gets Installed With **GNU polyxmassdata**

When the user installs the **GNU polyxmassdata** package, a number of files get installed in the destination directory that was chosen by the user if the package was installed by a method that allows using an option specifying in which directory the installation should be performed (see the chapter 2 on page 9).

The following listing lists the files that are installed when the **GNU polyxmassdata** package is installed (using an un-modified `polyxmassdata` `rpm` package), in the `/usr` system directory. This list was slightly edited to remove a number of lines that do not bring more information than what is needed to start figuring out the general architecture of the filesystem hierarchy of the **GNU polyxmass** software suite.

- Non polymer-specific configuration files:

  `/usr/share/polyxmass.d/polyxmassdata.conf`

  `/usr/lib/pkgconfig/polyxmassdata.pc`

  `/usr/share/polyxmassdata/polymer-definition.dtd`

  `/usr/share/polyxmassdata/atoms.xml`

  `/usr/share/polyxmassdata/chempad.conf`

  `/usr/share/polyxmassdata/cursor.svg`

  `/usr/share/polyxmassdata/poldefs-dictionary.dic`

- Polymer-specific configuration files:
  - "protein/peptide" polymer chemistry definition:

    `/usr/share/polyxmassdata/protein.xml`
    `/usr/share/polyxmassdata/peptide.xml`

    `/usr/share/polyxmassdata/protein/monomer-modif.dic`

    `/usr/share/polyxmassdata/protein/alanine-text.svg`
    `/usr/share/polyxmassdata/protein/alanine.png`

```
/usr/share/polyxmassdata/protein/alanine.svg
/usr/share/polyxmassdata/protein/acetyl-text.svg
/usr/share/polyxmassdata/protein/acetyl.png
/usr/share/polyxmassdata/protein/acetyl.svg
/usr/share/polyxmassdata/protein/arginine-text.svg
/usr/share/polyxmassdata/protein/arginine.png
/usr/share/polyxmassdata/protein/arginine.svg
⋮
/usr/share/polyxmassdata/protein/valine-text.svg
/usr/share/polyxmassdata/protein/valine.png
/usr/share/polyxmassdata/protein/valine.svg
```

– "dna" polymer chemistry definition:

```
/usr/share/polyxmassdata/dna.xml

/usr/share/polyxmassdata/dna/monomer-modif.dic

/usr/share/polyxmassdata/dna/adenine-text.svg
/usr/share/polyxmassdata/dna/adenine.png
/usr/share/polyxmassdata/dna/adenine.svg
⋮
/usr/share/polyxmassdata/dna/thymine-text.svg
/usr/share/polyxmassdata/dna/thymine.png
/usr/share/polyxmassdata/dna/thymine.svg
```

– "rna" polymer chemistry definition:

```
/usr/share/polyxmassdata/rna.xml

/usr/share/polyxmassdata/rna/monomer-modif.dic

/usr/share/polyxmassdata/rna/adenine-text.svg
/usr/share/polyxmassdata/rna/adenine.png
/usr/share/polyxmassdata/rna/adenine.svg
⋮
/usr/share/polyxmassdata/rna/methyl-text.svg
/usr/share/polyxmassdata/rna/methyl.png
/usr/share/polyxmassdata/rna/methyl.svg
/usr/share/polyxmassdata/rna/uracile-text.svg
/usr/share/polyxmassdata/rna/uracile.png
/usr/share/polyxmassdata/rna/uracile.svg
```

– "ose/saccharide" polymer chemistry definition:

```
/usr/share/polyxmassdata/ose.xml
/usr/share/polyxmassdata/saccharide.xml
```

```
/usr/share/polyxmassdata/saccharide/monomer-modif.dic

/usr/share/polyxmassdata/saccharide/allose-text.svg
/usr/share/polyxmassdata/saccharide/allose.png
/usr/share/polyxmassdata/saccharide/allose.svg
⋮
/usr/share/polyxmassdata/saccharide/xylose-text.svg
/usr/share/polyxmassdata/saccharide/xylose.png
/usr/share/polyxmassdata/saccharide/xylose.svg
/usr/share/polyxmassdata/saccharide/xylulose-text.svg
/usr/share/polyxmassdata/saccharide/xylulose.png
/usr/share/polyxmassdata/saccharide/xylulose.svg
```

- Example polymer sequence files:

```
/usr/share/polyxmassdata/polseqs/dna-sample.pxm
/usr/share/polyxmassdata/polseqs/long-protein-sample.pxm
/usr/share/polyxmassdata/polseqs/ose-sample.pxm
/usr/share/polyxmassdata/polseqs/protein-fragments-sample.pxm
/usr/share/polyxmassdata/polseqs/protein-sample.pxm
/usr/share/polyxmassdata/polseqs/rna-sample.pxm
```

- User manual files:

```
/usr/share/polyxmassdata/userman/*
```

Let's review these files and comment on them:

- The `/usr/share/polyxmass.d` directory will contain default configuration files for the different graphical modules that comprise the **GNU polyxmass** software suite. For example, when the user installs the **GNU polyxedit** package, this directory will contain the default configuration file for this package: `polyxedit.conf`. This directory also contains the most crucial configuration file of the whole **GNU polyxmass** software suite: `polyxmassdata.conf`, that is described below.

- `/usr/share/polyxmass.d/polyxmassdata.conf` This file contains one line:

```
polyxmassdata=/usr/share/polyxmassdata
```

  This line indicates what is the directory that contains all the **GNU polyxmass**' filesystem hierarchy, as it is installed by the **GNU polyxmassdata** package. In the present case, since the package was installed using an `rpm` package, the `/usr` system directory was the target installation directory and all the data were installed in the

```
/usr/share/polyxmassdata
```

directory.[1] This directory is the main **GNU polyxmass** configuration and polymer chemistry data files repository. The **GNU polyxmass**' modules very often query this file in order to know where to search for configuration data (typically to know where to search polymer chemistry definition files).

We will see that the user may duplicate this repository in one of his owned directories in order to modify the configuration and polymer chemistry definition files for her own use. **GNU polyxmass** can be made aware of this new user-owned location very easily, by simply editing one or more files.

- `/usr/share/polyxmassdata/atoms.xml` This file contains the definition of all the isotopes of all the chemical elements that a polymer scientist may have to use in her chemistry simulations. Each atom (chemical element) is actually defined using a set of isotopic mass/relative abundance pairs. The monoisotopic mass is by default (and that is the normal chemical situation) the mass of the lightest isotope. The average mass, as used by the **GNU polyxmass** simulations, is computed by taking into account all the *isotopic mass/relative abundance* ratios. There is only one file per system that describes the atoms.

- `/usr/share/polyxmassdata/poldefs-dictionary.dic` This file contains a number of lines, like the ones below:

```
protein=protein.xml%protein
long-protein=long-protein.xml%long-protein
dna=dna.xml%dna
rna=rna.xml%rna
saccharide=saccharide.xml%saccharide
ose=ose.xml%saccharide
peptide=peptide.xml%protein
```

  Each line is made of 3 parts. The part left of the '=' sign specifies a *polymer chemistry type*, exactly as it may be referenced in a polymer sequence file. The part left the '%' sign specifies in what file the polymer chemistry is defined. The last part of the line is the name of the directory where the polymer chemistry definition-specific files are located. This directory *must itself be located in the directory that is described in the `polyxmassdata.conf` main configuration file (see above).* In our example, that means that the "protein" chemistry-specific files should be located in the `/usr/share/polyxmassdata/protein` directory.

  Interestingly, as we can see, two different polymer chemistry definition types ("protein" and "peptide" may point to the same `protein` polymer chemistry-specific directory, while being defined in two different polymer chemistry definition files (respectively `protein.xml` and `peptide.xml`). The same applies for the "saccharide" and "ose" polymer chemistry types.

- The `/usr/share/polyxmassdata/protein.xml` file is the file where the "protein" polymer chemistry is defined. This file is the one that is associated to the "protein" polymer chemistry definition type in the `poldefs-dictionary.dic` file described above. These polymer chemistry definition files are typically produced by using the **GNU polyxdef** module.

[1] The `/usr/share` directory is a standard location for data installed by *`rpm`* packages, while source *`tar.gz`* packages install their data in `/usr/local/share` by default.

- The `/usr/share/polyxmassdata/protein/monomer-modif.dic` file is the file where a number of very important polymer chemistry-specific data are stored. This file contains a number of lines like the following (the numbers at the beginning of some lines were added to ease commenting below):

```
1 A=alanine.svg|alanine.png
C=cysteine.svg|cysteine.png
D=aspartate.svg|aspartate.png
4 S,T$Phosphorylation%T%phospho.svg|phospho.png
5 D$Amidation%O%asparagine.svg|asparagine.png
Y$Phosphorylation%T%phospho.svg|phospho.png
7 !$Acetylation%T%acetyl.svg|acetyl.png
E$Amidation%O%glutamine.svg|glutamine.png
```

  The first line tells that the monomer having a code 'A' should be graphically rendered –in the polymer sequence editor– using one of two graphics files: `alanine.svg` or `alanine.png`. Indeed, when graphically rendering a monomer code in a polymer sequence (by creating a "monomer icon", or "monicon"), the **GNU polyxedit** polymer sequence editor first tries to read the "scalar vector graphics" file (`alanine.svg`). This graphics format allows rendering the monomer icon at maximum quality whatever the monicon size requested by the user in the polymer sequence editor. If this rendering fails, for some reason, the program falls back to using the other graphics file (a "raster graphics" file). The raster graphics file cannot be resized without loss of image quality (see the chapter 8 on page 67).

  The fourth line indicates that the monomers having code 'S' or 'T' (in protein chemistry, these are seryl and threonyl residues) may be chemically modified using a modification called "Phosphorylation". The way the "Phosphorylation" modification should be rendered graphically is by compositing 'T'ransparently (see the %T%) the `phospho.svg` or the `phospho.png` transparent graphics files onto the monicon of the monomer being modified (see the chapter 8 on page 67).

  The fifth line shows another graphical compositing rule. This time the rule is not 'T'ransparency, but involves an 'O'paque graphical compositing (see the %O%). This line says that when a monomer of code 'D' is modified using an "Amidation" 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– the raster graphics file `asparagine.png`.

  The seventh line shows the use of the "joker" '!' character. This '!' character stands for *"all the monomer codes of the polymer chemistry definition"*. This line means that any monomer code in the "protein" polymer chemistry definition may be acetylated using the "Acetylation" modification, according to a graphical rendering rule of transparent compositing of the `acetyl.svg` (or the `acetyl.png`) graphics file onto the monomer icon of the modified monomer.

- The `/usr/share/polyxmassdata/protein/alanine.svg` is the file that contains the scalar vector graphics representation of the "Alanine" monomer. This file was created using the `Sodipodi` software (see chapter 1 on page 1, section 1 on page 4), by converting the textual representation of the monomer (that is a 'A' character) to curves prior to saving the file as a *`svg`*-formatted file.

- The `/usr/share/polyxmassdata/protein/alanine-text.svg` is provided as a convenience to the user. This is the file that was used to produce the previous one. But this file contains a textual representation of the 'A' character. This file is not correctly interpreted by the **GNU polyxedit** polymer sequence editor and should only be used as a model.

- The `/usr/share/polyxmassdata/protein/alanine.png` was prepared by exporting the contents of the `alanine.svg` file to a *png*-formatted file (all this from `Sodipodi`).

  It is noteworthy that in theory, if all the scalar vector graphics files (*svg* files) are 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 wise to always provide the polymer sequence editor with a fall-back solution in the form of a raster graphics file to be used in case the monicon rendering from the scalar vector graphics file fails.

- A number of files corresponding to a number of different polymer chemistry definitions are shown so that the user may grasp the way these polymer chemistry definitions are organized in the **GNU polyxmass** filesystem hierarchy.

- The `/usr/share/polyxmassdata/polseqs/protein-sample.pxm` is an example of a sequence of the polymer chemistry type "protein". There are other polymer sequences of other polymer chemistry types.

- The `/usr/share/polyxmassdata/userman` directory contains a number of files (including the one that I'm typing right now) that are used to compile the user manual file `polyxmass.pdf`. The command to issue so that the documentation file is compiled is (issue this command twice to resolve the cross-references):

  ```
  bash-2.04 $ pdflatex polyxmass.tex ↩
  bash-2.04 $ pdflatex polyxmass.tex ↩
  ```

Other files are both polymer chemistry-specific or not. For example, each polymer chemistry may have a **GNU polyxcalc** chemical pad configuration file. There is one `chempad.conf` file that is general (see the listing above). This `chempad.conf` file is located in the main **GNU polyxmass** configuration data directory. The user may define one such file for any polymer chemistry definition. In this case each polymer chemistry definition-specific directory (itself located in the **GNU polyxmass** main configuration directory, as we have seen above) may have its own `chempad.conf` file. **GNU polyxcalc** will thus read the `chempad.conf` file corresponding to the polymer chemistry definition that is currently loaded in the calculator. If no `chempad.conf` file is found in the polymer chemistry definition-specific directory, the default one is read.

Opening A Polymer Sequence: All The Events

In this section we'll review the internal mechanisms that make the **GNU polyxedit** module load the proper polymer chemistry definition file when a polymer sequence is loaded. This will enhance the reader's understanding of the reason why the filesystem hierarchy is that complex.

So, let's start the **GNU polyxedit** module of the **GNU polyxmass** mass spectrometric software suite, and open a file from the **GNU polyxmassdata** distribution. We see that the file selection window points directly to the `polseqs` subdirectory of the main **GNU polyxmass** data configuration directory. Let's select the `protein-sample.pxm` file, that is a polymer sequence of polymer chemistry *type* "protein". What does the **GNU polyxedit** program do in order to know how to render that sequence in the polymer sequence editor? Let's review that mechanics, but first I suggest that you use your favorite text editor (mine is `Emacs`) to open that same file. You'd see the following (only part of the file is reproduced below):

```
<polseqdata>
  <polseqinfo>
    <type>protein</type>
    <name>Sample</name>
    <code>SP2003</code>
    <author>rusconi</author>
    <date>
      <year>2003</year>
      <month>05</month>
      <day>13</day>
    </date>
  </polseqinfo>
  <polseq>
```

What you see here is that the `protein-sample.pxm` file contains the *type* of the polymer of which it is (`<type>protein</type>`). That *type* is –in our present case– "protein".

What we have done *right now*, the **GNU polyxedit** module does exactly the same way: it opens the file, reads it until it finds the `<type></type>` set of *XML* element tags. When it has found them it just reads the contents of the `<type>` element. That is the "protein" string.

By knowing of what polymer chemistry type the polymer sequence being opened is, **GNU polyxedit** can continue its work: it will first query the dictionary file responsible for making the correspondence between each available polymer chemistry type and its definition file. That dictionary file is `poldefs-dictionary.dic`, that we have described earlier. **GNU polyxedit** will thus search the line that starts with "protein". This line reads as follows:

```
protein=protein.xml%protein
```

What **GNU polyxedit** understands here is that the file that contains the polymer chemistry definition of the polymer chemistry type "protein" is `protein.xml` (check the Appendix to see what this file looks like). Now that **GNU polyxedit** knows what file contains the "protein" polymer chemistry definition, it has all the chemical "toolset" to compute masses and perform chemical simulations, like cleavages or fragmentations or whatever.

But what about the graphical rendering of the polymer sequence we are asking to open in the **GNU polyxedit**'s polymer sequence editor window? What the line above tells **GNU polyxedit** is that the "protein"-specific files are located in the `protein` directory, itself located in the **GNU polyxmass** main data configuration directory. We have briefly described above the contents of this `protein` directory. What **GNU polyxedit** now needs to know is what graphics file to use for any given monomer found in the polymer sequence that is described in the `protein-sample.pxm` file. The correspondence between each monomer code (in the polymer chemistry definition) and the graphics file to be used to render it graphically

in the polymer sequence editor is made in a dictionary file located in the polymer chemistry-specific directory (`protein`, for us now):

`monomer-modif.dic`

This file contains lines like the ones described above, that I reproduce here for convenience:

```
1 A=alanine.svg|alanine.png
C=cysteine.svg|cysteine.png
D=aspartate.svg|aspartate.png
4 S,T$Phosphorylation%T%phospho.svg|phospho.png
5 D$Amidation%O%asparagine.svg|asparagine.png
Y$Phosphorylation%T%phospho.svg|phospho.png
7 !$Acetylation%T%acetyl.svg|acetyl.png
E$Amidation%O%glutamine.svg|glutamine.png
```

When **GNU polyxedit** reads the `protein-sample.pxm` file it will get the sequence of that protein in the form of a "stream" of monomer codes. Each time it gets a new monomer code it will check what graphics file it should use to render this monomer graphically. If a monomer is described as being modified (in the polymer sequence file), it will perform the graphical operation described by the corresponding line (see lines 4–7 above). Of course there are implementation specifics that I do not describe that allow to make a tight memory management.[2]

Now that we know how **GNU polyxmass** copes with the flexibility required to handle any polymer chemistry, we can start thinking of ways to configure it in ways that suit the user's needs.

Configurability By The User

Any single bit of information in **GNU polyxmass** is modifiable by the user. That is a prerequisite for a powerful program designed in the most exquisite `GNU` tradition. When the **GNU polyxmassdata** package is installed, it comes with "default" configuration data[3]. These configuration data are simply examples and are not considered ready for publication of scientific work. The user is required to double-check all these example data before considering for publication any result yielded by the **GNU polyxmass** software suite.

In the section above, we have seen that all these configuration data are located in the **GNU polyxmass** main configuration directory (it was the

`/usr/share/polyxmassdata`

directory). That directory is a subdirectory of the `/usr/share` system directory, and the user needs system priviledges to modify files in these directories.

Let's imagine that the user wants to modify some files to suit her "chemical needs". Or that she wants to modify the way some monomers are rendered graphically in the polymer sequence editor. Unfortunately, it may happen that the user cannot have system priviledges, so she cannot modify the files that were installed by the system administrator. The solution

[2] By never loading twice in memory the same graphics file, for example (these files use up a lot of memory).

[3] Here, configuration is intended "at large", since it refers both to filesystem configuration and to all the polymer chemistry files which "configure" the chemistry of a polymer, or to the graphics files that "configure" the way monomers should be rendered graphically in the sequence editor.

is very simple: copy the entire (or part of) **GNU polyxmass** main data configuration directory into a location that she owns or can access with modification priviledges. For the sake of our example, let's consider that the user's login is "rusconi" and that his HOME directory is `/home/rusconi`. Let's continue considering that the "rusconi" user wants to duplicate the **GNU polyxmass** data configuration filesystem in `/home/rusconi/polyxmassdata` by issuing the following command:

```
bash-2.04 $ cp --rpf /usr/share/polyxmassdata /home/rusconi ↩
```

Once this configuration data duplication is done, it is necessary to let the **GNU polyxmass** software suite know that when any of its modules is run it should search for configuration data in her new location and not in the default installation location (described in detail above).

When the **GNU polyxdef**, **GNU polyxcalc** and **GNU polyxedit** packages belonging to the **GNU polyxmass** software suite are installed, their default configuration is stored in a file that is located in the `/usr/share/polyxmass.d` directory.[4]

When the user runs any one **GNU polyxdef**, **GNU polyxcalc** or **GNU polyxedit** program, the program in question checks the user's home directory (`/home/rusconi`) for a `.polyxmass.d` directory containing the corresponding package's configuration file (for example, the **GNU polyxedit** package has a configuration file named `polyxedit.conf`).

If the `.polyxmass.d` directory is not found, it is created and the

```
/usr/share/polyxmass.d/polyxmassdata.conf
```

file is copied in it. If the package's configuration file is not found it is copied in the user's `.polyxmass.d` directory straight from `/usr/share/polyxmass.d/`. For example, when the user "rusconi" runs **GNU polyxedit** for the first time ever, this program will check the existence of the

```
/home/rusconi/.polyxmass.d/polyxedit.conf
```

file. If this file is not found it is copied straight from the `/usr/share/polyxmass.d/` directory with its default contents.

Now, how does a user tell **GNU polyxmass** that the configuration data are not to be searched for in the default directory, but in one user-customized directory? That is simply done by editing the user's copy of the `polyxmassdata.conf` file.

Indeed, when any program of the **GNU polyxmass** software suite is executed, it first checks if there is a user's configuration directory named `.polyxmass.d`. If it finds that directory, the program checks if there is a `polyxmassdata.conf` file in it. If it finds that file, it reads the configuration directory from it. Since this file is copied from the main `/usr/share/polyxmass.d` directory in the first place, it contains the default data configuration directory, namely the following line:

```
polyxmassdata=/usr/share/polyxmassdata
```

But, we know that user "rusconi" wants **GNU polyxmass** to search the configuration files not in the default configuration directory, but in `/home/rusconi/polyxmassdata`. To achieve this, user "rusconi" would only have to change the line above to:

[4] Assuming that the packages were installed with the `--prefix=/usr` option, or with un-modified `rpm` packages.

```
polyxmassdata=/home/rusconi/polyxmassdata
```

in the `/home/rusconi/.polyxmass.d/polyxmassdata.conf` file.

As a summary, I can list the contents of the `/home/rusconi/.polyxmass.d` directory after having:

- Installed all the `rpm` packages in the usual order;
- Run the **GNU polyxdef**, **GNU polyxcalc** and **GNU polyxedit** programs one after the other.

Here are the contents of this directory:

```
polyxmassdata.conf
polyxdef.conf
polyxcalc.conf
polyxedit.conf
```

Let's now see the contents of each file in this directory:

- `polyxmassdata.conf`

```
polyxmassdata=/usr/share/polyxmassdata
```

- `polyxdef.conf`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<package_conf>

<package_name>polyxdef</package_name>

<gladedir>/usr/share/polyxdef/glade</gladedir>

</package_conf>
```

- `polyxcalc.conf`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<package_conf>

<package_name>polyxcalc</package_name>

<gladedir>/usr/share/polyxcalc/glade</gladedir>

</package_conf>
```

- `polyxedit.conf`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>

<package_conf>

<package_name>polyxedit</package_name>

<gladedir>/usr/share/polyxedit/glade</gladedir>

<numformats>
<numformat_atom>%.10f</numformat_atom>
<numformat_monomer>%.5f</numformat_monomer>
<numformat_oligomer>%.4f</numformat_oligomer>
<numformat_polymer>%.3f</numformat_polymer>
</numformats>

</package_conf>
```

Note that for the last `polyxedit.conf` file, the user may wish to configure a special directory where her polymer sequences are stored. This is performed by adding right after the `<gladedir>/usr/share/polyxedit/glade</gladedir>` line the following line, for example:

`<datadir>/home/rusconi/laboratory/sequences</datadir>`

What this line says it just that the polymer sequences for the user "rusconi" are stored in the directory that is set between the `<datadir></datadir>` tags. When the user will either open, or save, or save..as a polymer sequence, the file selection window will automatically open with this directory preset. This is merely a convenience feature.

If this line is not configured, then the default location where **GNU polyxmass** thinks that polymer sequences are stored is the `polseqs` directory that is itself located in the data configuration directory (in our examples it was either

`/usr/share/polyxmassdata/polseqs`

or the

`/home/rusconi/polyxmassdata/polseqs`

depending on wether we refer to the default data configuration directory of the one that user "rusconi" wanted to configure for himself (see above for the whole story).

Appendices

The Protein Chemistry Definition File

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE poldefdata SYSTEM
                  "/usr/share/polyxmassdata/polymer-definition.dtd">

<poldefdata>
  <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>
      <lr-rule>
        <re-mnm-code>M</re-mnm-code>
        <re-actform>-CH2S+O</re-actform>
      </lr-rule>
    </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>
</poldefdata>
```

GNU General Public License

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

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., 59 Temple Place, Suite 330, Boston, MA 02111-1307 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.
```

GNU Free Documentation License

GNU Free Documentation License
Version 1.1, March 2000

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the

publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either

commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers

   or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements"
   or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties--for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of

Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History"
in the various original documents, forming one section entitled
"History"; likewise combine any sections entitled "Acknowledgements",
and any sections entitled "Dedications". You must delete all sections
entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents
released under this License, and replace the individual copies of this
License in the various documents with a single copy that is included in
the collection, provided that you follow the rules of this License for
verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute
it individually under this License, provided you insert a copy of this
License into the extracted document, and follow this License in all
other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate
and independent documents or works, in or on a volume of a storage or
distribution medium, does not as a whole count as a Modified Version
of the Document, provided no compilation copyright is claimed for the
compilation. Such a compilation is called an "aggregate", and this
License does not apply to the other self-contained works thus compiled
with the Document, on account of their being thus compiled, if they
are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these
copies of the Document, then if the Document is less than one quarter
of the entire aggregate, the Document's Cover Texts may be placed on
covers that surround only the Document within the aggregate.
Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may
distribute translations of the Document under the terms of section 4.
Replacing Invariant Sections with translations requires special
permission from their copyright holders, but you may include
translations of some or all Invariant Sections in addition to the
original versions of these Invariant Sections. You may include a
translation of this License provided that you also include the

original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document 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.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation 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. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

> Copyright (c) YEAR YOUR NAME.
> Permission is granted to copy, distribute and/or modify this document
> under the terms of the GNU Free Documentation License, Version 1.1
> or any later version published by the Free Software Foundation;
> with the Invariant Sections being LIST THEIR TITLES, with the
> Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
> A copy of the license is included in the section entitled "GNU
> Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections"

```
instead of saying which ones are invariant.  If you have no
Front-Cover Texts, write "no Front-Cover Texts" instead of
"Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.
```