xps4xps Release V0.09

An "embedded" expert system (a reasoning inference engine) with rule sets for XPS/AES

http://xps4xps.sourceforge.net/

©János Végh (Janos_Vegh@users.sourceforge.net) MTA ATOMKI, Debrecen, Hungary

December 23, 2003

Contents

Та	able o	of contents	2
Li	st of	figures	4
Li	st of	tables	5
Co	opyri	ght notice	6
Fo	orewo	ord	7
I	Use	r's Guide	11
1	Bas	e terms	12
	1.1	Spectrum acquisistion and evaluation	12
		1.1.1 Acquisition	12
		1.1.2 Modeling	12
	1.2	XPS base terms	13
		1.2.1 Binding/kinetic energy	13
		1.2.2 Energy scale types	13
		1.2.3 Charging shift	13
	1.3	Objects	13
	1.4	Expert systems	14
	1.5	The 3-valued logic	15
	1.6	The inference engine	16
	1.7	The multi-platform feature	16
2	Inst	alling	18
	2.1	How to install	18
	2.2	Packages	18
		2.2.1 Executable for 32-bit Windows	18
		2.2.2 Sources for 32-bit Windows	19
		2.2.3 Static binary for Linux	19
		2.2.4 Sources for Linux	20

		2.2.5 Portable documentation	20
	2.3	What to install	20
		2.3.1 Developer	20
		2.3.2 Expert	21
		2.3.3 User	21
	2.4	The installed files	21
3	Usiı	ng rules	24
U	3.1	Coding a rule	<u>-</u> - 24
	3.2	Reasoning	26
	3.3	Wrapper rules	27
	3.4	Combined rules	 27
	0.1	3.4.1 Subrange-type rules	 27
		3.4.2 Multiple condition rule	28
	35	Verifying rules	28
	3.6	Rules vs wizards	20 29
	0.0		20
4	Dat	a acquisition	31
II	Re	ference Guide	33
5	The	base system	34
	5.1	Extended boolean logic	34
	5.2	Generic rule	34
	5.3	Agents	35
		5.3.1 Database	36
		5.3.2 User	36
	5.4	Extensions for spectroscopy	36
		5.4.1 Spectrum	36
		5.4.2 Sample	37
		5.4.3 Background	37
		5.4.4 Peak	37
6	Gen	eral XPS rules	39
Ū	61	Generic XPS rule	39
	6.2	Global variables	30
	0.2	6.2.1 E4Background	30
		6.2.2 EnergyTolerance	39 40
		6.2.2 vnShortoutMada	40
	6.0		40
	0.3 C 1		40
	ь.4		40
	6.5	ISENEROVBE	41

	6.6	IsEnergyKE	41
	6.7	IsEnergyAvailBE	41
	6.8	IsEnergyAvailKE	42
	6.9	IsPeakInRangeBE	42
	6.10	IsPeakInRangeKE	42
	6.11	IsRegionMeasuredBE	43
	6.12	IsRegionMeasuredKE	43
	6.13	IsXEnergyKnown	44
7	Carl	on contamination rules	45
	7.1	DoMarkCarbon1sPeak	45
	7.2	DoesSampleContainCarbon	45
	7.3	HasCarbon1sPeak	45
	7.4	IsCarbon1sPeak	46
	7.5	IsCarbonAngleRatioBiggest	46
	7.6	IsCarbonAugerPresent	46
	7.7	IsCarbonContaminationConsensus	46
	7.8	IsCarbonEnergySeparationOK	47
	7.9	IsCarbonPostPeakSlopeBiggest	47
	7.10	IsCarbonShirleyTailHigh	47
	7.11	IsCarbonXPresent	48
	7.12	IsRutheniumPresent	48
8	The	demo program	49
	8.1	The main window	50
	8.2	Verifying rules	50
		8.2.1 The spectrum page	52
		8.2.2 The peak page	53
		8.2.3 The sample page	54
	8.3	The menu system	55
		8.3.1 Help menu	55
		8.3.1.1 About	55
		8.3.2 Options menu	55
		8.3.2.1 Settings	55
		8.3.2.1.1 Background length	55
		8.3.2.1.2 Energy tolerance	55
		8.3.2.1.3 Use Local Numeric mode	55
		8.3.2.1.4 Shortcut mode \ldots	56
		8.3.2.1.5 Trial energy max	56
		8.3.2.1.5 Trial energy max	56 56
		8.3.2.1.5 Trial energy max	56 56 56

	8.3.3.1.1 DoMarkCarbon1sPeak	56
	8.3.3.1.2 HasCarbon1sPeak	56
	8.3.3.1.3 IsCarbonAugerPresent	56
	8.3.3.1.4 IsCarbonContaminationConsensus	57
	8.3.3.1.5 IsCarbonXPresent	57
	8.3.3.1.6 IsRutheniumPresent	57
	8.3.3.2 Sample	57
	8.3.3.2.1 Contains Carbon	57
	8.3.3.3 Truth table	57
	8.3.3.4 XPS	57
	8.3.3.4.1 HasPeakInRangeBE	57
	8.3.3.4.2 HasPeakInRangeKE	57
	8.3.3.4.3 IsEnergyAvailBE	57
	8.3.3.4.4 IsEnergyAvailKE	57
	8.3.3.4.5 IsRegionMeasuredBE	57
	8.3.3.4.6 IsRegionMeasuredKE	58
8.3.4	Spectrum menu	58
	8.3.4.1 Add peak	58
	8.3.4.2 Add sample	58
	8.3.4.3 Exit	58
	8.3.4.4 Load	58
8.3.5	Wizard menu	58
	8.3.5.1 Is Carbon present	60

III Appendix

Bibliography	62
Index	64

61

List of Figures

1.1	The Disney metaphor for objects	14
1.2	Access levels to the "embedded expert system"	15
5.1	The spectrum objects interdependence	37
8.1	The spectrum page of the demo application	50
8.2	The main page of the demo application	51
8.3	The spectrum page of the demo application	52
8.4	The peak page of the demo application	53
8.5	The sample page of the demo application	54
8.6	The spectrum setup page of the carbon wizard	59

List of Tables

5.1	Truth table for the extended boolean NOT operation	34
5.2	Truth table for the extended boolean AND operation	35
5.3	Truth table for the extended boolean OR operation	35
5.4	Truth table for the extended boolean EQUALS operation	36

Copyright notice

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

You should have received a copy of the GNU Library General Public Licence along with this software, usually in a file named COPYING.LIB. If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.

EXCEPTION NOTICE

- As a special exception, the copyright holders of this library give permission for additional uses of the text contained in this release of the library as licenced under the wxWindows Library Licence, applying either version 3 of the Licence, or (at your option) any later version of the Licence as published by the copyright holders of version 3 of the Licence document. (See http://wxwindows.sourceforge.net/licence3.txt)
- 2. The exception is that you may use, copy, link, modify and distribute under the user's own terms, binary object code versions of works based on the Library.
- 3. If you copy code from files distributed under the terms of the GNU General Public Licence or the GNU Library General Public Licence into a copy of this library, as this licence permits, the exception does not apply to the code that you add in this way. To avoid misleading anyone as to the status of such modified files, you must delete this exception notice from such code and/or adjust the licensing conditions notice accordingly.
- 4. If you write modifications of your own for this library, it is your choice whether to permit this exception to apply to your modifications. If you do not wish that, you must delete the exception notice from such code and/or adjust the licensing conditions notice accordingly.

Foreword

" To write a good software is hard. To write a good documentation is impossible."

About this document

R ecently, a renewed interest in deploying expert system in the data acquisition and evaluation process in the electron spectroscopy raised. Different approaches might be (and are!) taken to solve the problem. This document describes a particular one, an "embedded" expert system, which implements a simple "inference engine" [VJ-02], the most important and most mystic part of the expert systems. Based on this development, and using some widely known rules from the field of X-Ray Photoelectron Spectroscopy, the first published real-life rule set [CB-99] has been implemented as an example to demonstrate the feasibility of the inference engine. Also, although it is not "rule" in the sense above, some useful hints have been published [HH-92] to assist users in selecting data acquisition parameters to reach some well-defined analytical precision. The procedure has been implemented, too, and it can be used to demonstrate the principles and even in the everyday planning practice. Hopefully, based on the collective expertise and effort of the XPS community, more rule sets will be added soon.

The goal of developing the rule sets is to make this expert knowledge publicly available. Correspondingly, their implementation as well as source code of the base "inference engine" shall be made public. In terms of computer programming, the resulting system is intended to be part of the "open source" world [OS-01] and this is why it is hosted by SourceForge [SF-02].

The present document describes the elements of this project in detail. Where possible, it refers to the available publications. For legal reasons, those copyrighted materials must not be reproduced here, so only their short summary and some supplements can be contained here.

About xps4xps

The project described here comprises several components and is a result of collective efforts. It is running continuously, and the different rules are in state of different maturity. It is also possible that the inference engine and the rules will be implemented in different computer languages. Right now, the inference engine and the first rule sets are implemented in the C++ language. Both of these components are in "beta" stage (i.e. they are in course of final approval, minor beutifying corrections are still possible).

The development needs some general programming constructs (like lists, strings, vectors, etc.), as well as radioboxes, checkboxes, text input fields, etc. when testing/simulating the rules. Also, because the developed rules are to be built into data acquisition and evaluation applications, a program language feasible for making such applications has to be used. Because of this, the development has been started in a way using a base package, which poses no copyright, royalty, etc. problems, supports (practically) all popular platforms and its user community guarantees a long term usage. The multiplatform C++ macro package wxWindows [WX-92] has been chosen as the development platform.

How to use this manual

U sers are strongly advised to study the *User's Guide* first. It gives you an overview of principles, techniques and terms, used in this project and throughout this manual. Beside this, it makes you acquainted with handling the components of the project. The different versions (releases) of the results of the project might differ in that some new rules and demos/settings are added and (less frequently) some old rules are modified. Also, some bugs might be fixed between versions and also sometimes optimization takes place. So -especially after receiving a new release- it is worth to study this guide carefully.

The *Reference Guide* provides the most detailed information about the comprised components of the project. Here you find also the default values/settings and the notes about their usage. Most of the (non-trivial) operations, algorithms, as well as some parameter values, are known from scientific publications. These sources are referred to as usual with a [code], and under the correspondingly designated item in the Bibliography you will find the description of the real source, typically in form of widely available books or well-known journal articles or URLs.

The *Programmer's Guide* is generated now by Doxygen [HD-97] and is provided in HTML help file through the WEB site, as well as a Portable Document file, available as part of the distribution. It provides the complete reference to the package for the programmers, implementing further rules or building the existing ones into some software. A short but concise description of the variables and functions is given also here. However, the essence of the operation is documented in the corresponding place in the *Reference Guide*, where the principles of the given solution are described in detail.

The available platforms

hanks to the used multi-platform package, the program can be (in principle) available on different computers (IBM-PC, a range of UNIX workstations, McIntosh, VAX, etc.) under different operating systems (MS Win32, Linux, UNIX, VMS, Mac-OS) and is being supported by different graphic packages (like GTK and Motif). The program on different platforms has a native "look and feel" and because of this, it behaves a little bit differently (where it is important, the platform-specific difference is explicitly marked in the text). Of course, the basic commands must work in any combination, but it would not be wise not to use out the extra possibilities provided only on one of the platforms. These extra possibilities are explicitly marked in the text.

The main developer platforms are MS Windows 2000 and Linux Debian 3.0, later some other platforms might be added. (The availability of the program on other platforms depends on the availability of development capacity for the author on that platforms.) The screen shots are taken from Windows 2000 and Linux KDE screens.

Acknowledgements

The distributed Win32 and Linux binaries are compressed using UPX [OM-96]. The Win32 installer packeg is prepared with Inno Setup [RJ-96]. The Linux binary package uses the standard GNU tools. This document is prepared with MikTeX [CS-01] and edited with TeXnicCenter [TC-99]. The Programmer's Guide is made with Doxygen [HD-97].

Typographic notations

W hen looking at the format of this document, you'll occasionally notice things in different fonts:

- *Emphasized Style* is used for general emphasis, book and paper titles, names of sections of other manuals, object data types, notes from the author, etc.
- Typewriter is used for program and file names, rules, code fragments, etc.
- NOUN STYLE is used for command and rule names.
- **boldface** is used for object member names, function members, etc.

Part I

User's Guide

Chapter 1

Base terms

 \prod he task addressed by this chapter is to explain the terms related to spectrum processing and expert systems, at least as they are used in the present project. The way as they are realized, is the subject of the next chapters.

1.1 Spectrum acquisistion and evaluation

valuating spectra (here) means to interpret measured data, to extract qualitative and quantitative characteristic of the studied process.

1.1.1 Acquisition

S pectrum acquisition means to measure some electron density function using some electron spectrometer, point by point, and to produce the dataset (aka "electron spectrum"), the subject of the evaluation. Usually, some quantitative information shall be extracted from the data, so the measuring conditions shall be selected carefully according to the goals to be reached. Some good advices can be extracted from [HH-92]. Part of the suggested procedures is implemented for the future expert system, as detailed in section 4 on page 31.

1.1.2 Modeling

The modeling approach is necessary here because some rules refer to objects like spectrum, peak, post-peak slope, etc. When evaluating spectra, the user has to define such terms. It can be done in such a way that the data evaluation program allows to create peaks, or creates them automatically, or simply the user marks some regions of the spectrum where he locates the peak. In the demo application, these objects are represented by some panels (and behind them, some peak object), where the peak can be located via giving their energy and some other characteristics. Otherwise, the modeling approach is not (yet) used; in later phases (especially when cooperating with data evaluation software) it will be much more meaningful.

1.2 XPS base terms

S pectra of electrons excited by X-rays, have several particular features, the special handling of which will be detailed in this section.

1.2.1 Binding/kinetic energy

 $\prod_{k=1}^{nergy} \text{ scale type is of central importance in evaluating XPS spectra. As it is well known, in XPS/AES the energy values can be given either on binding or kinetic energy scale. The two energy values are related by¹$

$$E_b = E_X - E_k \tag{1.1}$$

where E_X is the energy of the excitation source, E_k is the kinetic energy, E_b is the binding energy. Since usually some numeric values of the component parameters are based on the actual energy values, care shall be taken when changing the scale type or the excitation energy.

1.2.2 Energy scale types

I n some rule sets, see for example section 6.7 on page 41, both the kinetic and binding energy values are requested. Because of this, the spectrum objects have to be able to calculate their energies on both scales. To do so, the spectrum objects have to own their energy type and excitation energy value. Upon creating them, these characteristics are inherited from the parent spectrum, later they can be set independently.

The parent spectrum has the primary energy type and values; the spectrum components derive their own energy type and values from it.

1.2.3 Charging shift

D ue to the bombarding with X-rays, electrons escape from the sample so (depending on the conductivity of the sample) the sample might have on a non-zero electrostatic potential. Several instrumental corrections can be (and usually are) applied to compensate for this (apparent) energy shift, originating from this effect. Despite this, the data evaluation procedures must be prepared for the fact that the electrons appear in the spectrum at an energy, which does not exatly match their nominal energy. The charging shift (here) in handled as follows.

1.3 Objects

or such a complex task like implementing an expert system, using object oriented programming seems to be a necessity. For non-programmers, the term 'object'

¹charging, etc. effects are not accounted for here

sounds rather mystic. Probably the best way to understand the term is via the "Disneymetaphor", as shown in Figure 1.1 on page 14. According to this, the 'cleaning objects' -which usually comprise only data, like volume, length, weight, etc.- are personalized and can also perform actions like fill, wipe, clean, etc. In this way how the action is carried out remains hidden; the "wizard" asks for some action (*what*) and the "cleaning objects" carry it out. The unnecessary details (*how*) are *encapsulated* into the objects, allowing for a better overview of the task. New objects can be derived from an object: the new object *inherits* all features and data from its *ancestor* and makes something more (what) or the same thing in a different way (how). For a more detailed (and professionally correct) discussion please refer to the numerous textbooks on object oriented programming, for example [EB-99].



Objects are empowered with intelligent and appropriate behavior

Figure 1.1: The Disney metaphor for objects

1.4 Expert systems

E xpert systems – as any other specialized fields of the science – have their own terminology. Most terms in this project are used in accordance with the generally accepted meaning. The "embedded expert system" means here a limited input/output ability (no native language input) package, which can be built into data acquisition and evaluation software. The "reasoning inference engine" here means a software, which can make decisions and as by-product, produces also a string, which explains why this particular decision was made.



Figure 1.2: Access levels to the "embedded expert system"

1.5 The 3-valued logic

I n the real life the replies of an expert person to a question can not only be a definite "yes" or "no", but even "maybe yes" or "probably no". It might also happen that some information is not available or uncertain or unknown, as well the expert person might say "in lack of ... I cannot decide" or "I am not sure" or even "I do not know". These replies can also be used as input information when formulating another question to an expert person. Obviously, a "maybe yes" is not identical with "yes". One of the most critical points of an expert system is the type of the output it can give and the type of the input it can receive.

For operating the inference engine properly, one has to extend the generally used (two valued) Boolean logic to multiple-valued logic. The idea is not strange at all: see fuzzy logic [SRS98] and its applications. Obviously, it would be useless to extend the "yes/no world" with all the mentioned reply types; rather introducing a third state ("unknown/not

xps4xps V0.09

set/do not know") would suffice. Even this idea is not new: see [FA-66]. Also note that even the ANSII standard for SQL database-handling language (ANSI SQL 89) applies a triple-valued logic [DCJ89]. Introducing this extension enables the expert system to simulate an expert person who is able to deal with incomplete/not fully reliable input data and is able to give a reply other than a definite "yes" or "no". The extended boolean logic is introduced on page 34 in section 5.1.

1.6 The inference engine

O ne of the most mystic parts of the expert system is the 'inference engine'. For the users of an expert system, it is a "black box", providing an extended boolean value, which depends in a pre-defined way on some external conditions. In addition, it is able to explain why a particular decision was made. In most of the known "expert system shells" a special language is constructed to describe the rules. In the present system only one special method is used which returns a rule, and, which is transparent for both the users and the programmers. Otherwise, the language's standard elements ("for" cycles, "if then else" structures, etc.) are used to combine the rules, to calculate, etc.

There are many expert system, based on different "shells", in use. The one which we need to build in the data handling software, shall be somewhat simplified, but still be able to simulate a real domain expert. Since the field of the intended application needs some well defined, community verified rules, it is an acceptable compromise not to provide native language input and only use the inferencing ability of the system. In additition to this, the system shall be able to explain its decisions, because (especially in the data evaluation applications) the users might need it.

1.7 The multi-platform feature

E xpert systems can exist on different software/hardware platforms, and also the developers can use different development tools. At this point one can make good use of the excellent support provided by the multi-platform package wxWindows [WX-92], because

- Once, the package needs some objects, like strings, lists, etc. This package provides the possibility to install the software at very different hardware and software platforms.
- Second, the development (including rule verifying phase) can be carried out on very different platforms, too.
- Third, the demo applications can be run on different platforms, allowing all uses to study and improve the rules.

A later goal of the project might be to "live without" that package, i.e. at least the base + rules shall be compilable without that package, i.e. some simple such structures (lists, strings, etc.) shall be implemented independently and added to the package.

Chapter 2

Installing

I nstallation is a very much platform-dependent and -typically- simple task. In this case it is complicated by the fact that the released packets comprise components of different nature. This chapter deals with the subject how to install the components, as well as what to install and why to install the very different components.

2.1 How to install

I n this early phase of development, only the 32-bit Windows and Linux operating systems are supported. On these platforms, the installation follows the standards on that platform, see below. The provided material is grouped in different *packages* and as the development proceeds, different *releases* are issued.

2.2 Packages

C ackages here means that some (related) material is put together and made available for the users in some grouping. Presently, the following packages are in use.

2.2.1 Executable for 32-bit Windows

W indows versions with 32-bit word length (Win9x, NT 4, 2000, XP) share one single distribution packet. The released packet is assembled by the Inno Setup installer [RJ-96]. The installation procedure tends to be conventional: you can download the actual release package from the download site [XPS02], in form of a .zip file. The file name contains also the version number. Just unzip it to some temporary place and start the resulting "setup.exe" program. Press button "NEXT" on the popup window if you want really install it. Select the "home directory" you prefer (some subdirectories will be created in that directory during the installation); even the directory name can be changed.

2.2.2 Sources for 32-bit Windows

A lthough (thanks to the multi-platform feature of the package) all platforms share the same *source* text, some peculiarities of Windows (special compiler project files, icons, carriage control, install software, registry, etc) support to make a separate source package.

Like the executable, the sources come in an installer package, too. Howevere, here the selection needs more attention and knowledge: it offers several components to install. Because of the multi-purpose nature of the distributed package, all these components can be selected independently, there are no mandatory components and no dependence checking. It is hoped that the descriptive text on the side of the installable components will guide you through the install process.

The first checkbox controls the documentation files (change log, readme, licence, etc). This is the only "mandatory" part of the package, so the corresponding checkbox is disabled (grayed out). The next few checkboxes control installing source files (for developers/coders/experts). The next selection offers to install more source files, necessary to build the demo program. For this goal, some makefiles/project control files are provided for some selected SW platforms. Select what you like to install and press "next" again. Also, select the start menu folder for the release and finish selecting with checking the checkboxes if you want to create desktop icon and quick launch icon. Now a summary page comes up, where you can make the final checking of your settings. This is the last chance to abandon installing. You can decide to go "BACK" or to "INSTALL". Choosing the latter case a progress dialog comes up and the install process finishes with viewing the package's "readme.txt" file. You might also be interested in the contents of the "changes.txt" file, where the changes between releases are listed. Now, the distribution is installed.

Note: The setup programs come with complete uninstall facility. Shortcut, directories, registry items, etc. are completely and safely removed. However, the "foreign" files (for example, results of compiling the contained sources) are to be removed by hand, preferably before "uninstall". If removing after "uninstall", the containing directories shall also be removed by hand.

2.2.3 Static binary for Linux

L inux users usually *make* their own binary on their own particular system. However, this demo program needs further libraries, which are usually not present "out of the box" in any distribution and their installation needs root permissions. Because of this, a static binary is available here (which, however, still needs the correct version of the *glibc* library). In case you wish to build your own binary, please use the *tarball* file, see below.

2.2.4 Sources for Linux

Sources for generating your own binary are provided in form of *tarball* file and your system manager can install it for you. These application sources depend on the already mentioned base wxWindows [WX-92], so you have to install that package first. After installing wxGTK (i.e. wxWindows), you can install the expert system demo as

```
./configure
make
```

Optionally, also (as root)

make install ldconfig

To remove, also (as root)

make uninstall ldconfig

2.2.5 Portable documentation

D ocumentation is provided in form of plaform-independent PDF (Portable Document Format) files. They are provided in different package but with the same release number. In oder to avoid platform-dependency, the document files are not packed. Just download them and store them together with the other material you use.

2.3 What to install

A ll what you are interested in, in general. What you are interested in, it depends on your goal with the package. Below you find a short explanation what files are necessary if you are a member of the targeted groups. Anyhow, the documentation files are the only mandatory part of the package: (and not only to install, but also to carefully study it :-)). It is strongly hoped that the installation package "component names" give enough hints to choose; see also the next section.

2.3.1 Developer

D eveloper -here- means a software expert, who wants to implement new rules (following the advices of an XPS expert) or wants to implement the rules existing in the package in his software product. For this goal at least the sources of the base system and the selected package are necessary, maybe you can make a good use of the project files for one of the popular development environments. The best choice is to make a full install, but you may leave out the demo executable.

2.3.2 Expert

E xpert -here- means a person who provides his experiences in spectroscopy for formulating domain-specific rules. Also a "knowledge engineer" is involved in this process. These users definitely need the sources of the rule sets. They all use existing functions and also create new ones. To develop and verify rules, they might make good use of the demo project files, demo sources.

2.3.3 User

U sers -here- means all spectrocopists, users of the end-product, produced by the different manufacturers. They shall know the documentation, at least the reference manual, and it is a very good experience to play with the demo. They usually do not need the sources, project files, etc.

2.4 The installed files

B elow it is assumed that you made a "Full install", i.e. all the files described here will only be present in that case, otherwise you will see a subset only. The subdirectories and their contents can be SW platform-dependent. The common description is given here; the platform specific deviations/supplements are given in the subsections.

In the main directory (.)

- unins000.dat : uninstall database
- unins000.exe : the uninstaller excutable

The "./bin" subdirectory is for the executable file (demonstrating the usage of the package)

• xps4c1s.exe : demo application for the C 1s rule set (for Win32)

The "./bin/lang" subdirectory is for the language localization files In the "./bin/lang/hu" subdirectory is for the Hungarian language localization files

- xps4xps.po : demo-specific message translations, source
- xps4xps.mo : demo-specific message translation, compiled
- wxstd.po : base package message translations, source
- wxstd.mo : base package message translations, compiled

In the "./cpp" subdirectory

- $\bullet\ xp4c1sVC.dsp$: project file for building the C1s demo using MS Visual C 6
- xps4c1sVC.dsw : compiler settings for building the C1s demo using MS Visual C 6

xps4xps V0.09

- $\bullet\,$ makefile.vc : a command-line make file for building the C1s demo with MS Visual C $_{6}$
- The "./cpp/bitmaps" subdirectory contains
- icon4xps.xpm : the icon for building the xps4xps demo application
- wiz4xps.xpm : the "Carbon wizard" icon
- The "./cpp/icon" subdirectory contains
- uninsxps.ico : the icon for uninstalling the application
- The "./cpp/include" subdirectory contains the corresponding headers & co
- cwizard.h : header for the 'Carbon wizard'
- mainpage.h : just comments, for documenting with Doxygen
- menu_ids.h : menu item and other constants
- panelc1s.h : header for simulator pages for the C1s rule demo
- papers.h : just comments, for documenting with Doxygen
- setup.h : stuff for setting up conditions for the application
- truthtbl.html : a file for indluding by Doxygen
- utils.h : some utility routines
- version.h : contains demo (and release) version, text and numeric
- xbool.h : header & implementation of the 3-valued logic
- xprule.h : header for the generic rule implementation
- xps4c1s.h : header for main program of the C1s rule demo
- xps4c1s.rc : resource file for the main program of the C1s rule demo
- xps4xps.h : header for the general XPS-related objects
- xrcarbon.h : header for the carbon-contamination related rules
- xrxps.h : header for the general XPS-related rules

The "./cpp/src" subdirectory is for the C++ source files.

- cwizard.cpp : stuff for the 'Carbon wizard'
- panelc1s.cpp : simulator pages for the C1s rule demo

- utils.cpp : some utility routines
- xprule.cpp : the generic rule implementation
- xps4c1s.cpp : main program of the C1s rule demo
- xps4xps.cpp : the general XPS-related objects are implemented here
- xrcarbon.cpp : the carbon-contamination related rules
- xrxps.cpp : the general XPS-related rules

The "./docs" subdirectory is for the documentation

- changes.txt : changes between releases
- copying.lib : the licence conditions of the package
- readme.txt : post-deadline & other infos

Chapter 3

Using rules

O oding rules is a complex activity, involving experts, knowledge engineers and software developers (as well as end users in the testing phase). Since no natural language parser is involved, the correspondance between the expert-provided rules and the programmer-written code must be assured during coding the rules. Thanks to the underlying functionality (the inference engine), coding rules is quite close to the natural language.

A general principle is that the expert system can provide answers to questions of type 'Is ...', 'Has ...'. Correspondingly, the name of the rules is of form IsXXX or HasXXX, and the class name of the implemented rule is xrIsXXX or xrHasXXX, where xr stands for 'eXpert Rule'.

3.1 Coding a rule

 $A\,$ s an example, consider the rule resulting status if the binding energy is available. In everyday language:

```
Binding energy is available

if

energy data are on binding energy scale

or

energy data are on kinetic energy scale

and

excitation energy is known
```

To describe the rule in a more formal way, a meta-language format is used. The name of the rule will be 'IsEnergyAvailBE' (see section 6.7 on page 41), the implemented class will be 'xrIsEnergyAvailBE'. The method "**Fire()**" of the implemented class will contain the rule and its value will be calculated by the method "**CalculateValue()**".

The general behaviour of a rule is independent of how many rules or external information are involved: the only requirement is that the **Fire()** method shall result in an expression, using which the **CalculateValue()** method can calculate the value of the rule. Presently the base system contains only pre-defined combination of rules (i.e. is hard-wired), which corresponds to the expert-proven rules sets. However, for verifying the rules, a run-time assembled rule handling could be easily constructed.

As you see in the code file "xrxps.h", the coding you have to do is to derive a new rule from the generic XPS rule, give it a name (a logical one as class name and a string one for using in reasoning), and to write the "Fire" method (see below). I.e. you have to write the native language expression using the syntax of the used program language. In case of C++ it looks like

As it is seen, it is quite straightforward to translate the everyday terminology to the logical expression: just formulate the expression according to the syntax rules of the used language. I. e. change the natural language words "AND" and "OR" to the corresponding operators "&&" and "III" (if you like), respectively, and put the resulting expression in the body of the member function "**Fire()**". In describing the rules, 'AND', 'OR' and 'NOT' notation are used for the corresponding logical operations. Note that in the C++ implementation below, these notations can also be used for the corresponding logical operations, just to make reading the generated code easier.

The brackets in this particular case are not really necessary (the precedence of the operators would result in the same execution order without brackets, too; they just emphasize how exactly the expression is meant. The resulting rule is completely specified: now (thanks to the underlying object's functionality) the '**CalculateValue()**' member function can calculate the resulting value (whether one can use the binding energy) and the method '**GetReasoning()**' can tell the reason to the user. Depending on the conditions you set, you might receive results like

```
"Binding energy available" is TRUE because
"Energy-Data are BE" is TRUE
or
"Binding energy available" is TRUE because
("Energy-Data are KE" is TRUE
AND
"Exciting energy known" is TRUE)
```

3.2 Reasoning

I n the examples above, you cannot see any sign of reasoning in the implementation of the method "**Fire()**". It is because a so called "default reasoning" is built in the *operators* of the inference engine. During rule!evaluation, the inference engine combines the name of the rule, its value, furthermore the name and value of the evaluated rules, together with the connecting operations. This mechanism is called as "default reasoning".

Note that these reasonings are resulted in the default 'shortcut' evaluation mode. In the 'complete' evaluation mode (if the menu item "OPTIONS | SETTINGS | SHORTCUT MODE" is false), the first example sounds:

```
"Binding energy available" is TRUE because
("Energy-Data are BE" is TRUE
OR
("Energy-Data are KE" is FALSE
AND
"Exciting energy known" is UNKNOWN))
```

As it can be seen easily, the information content is identical but the output is verbose. In shortcut mode it is simpler to understand the reasoning of the rule.

Of course, if you are not satisfied with the "default reasoning" method (otherwise: this is used throughout the examples), you might write your own reasoning. In some cases, it is even a must. For example, some rules (the so called wrapper rules, see below) the rule involves only one single rule and no operation. Since the reasoning is built into the operations, in such a case the custom reasoning is a must. It only means to give the proper value to the member variable *HistoryString*. For example, the **Fire()** member function of the rule "IsEnergyBE" looks like

```
{ xpSpectrumBase *xpS = (xpSpectrumBase *) InfoSource;
    if(xpS) value = xpS->GetEnergyBE();
    else value = Unknown;
    if(xboolean::True == value)
        HistoryString << "Energy data are on binding scale";
    else if (xboolean::False == value)
        HistoryString << "Energy data are NOT on binding scale";
    else HistoryString << "No spectrum known";
    return *this;
}
```

First the returned value is calculated, then -depending on that value- the reasoning text is collected.

xps4xps V0.09

3.3 Wrapper rules

special case of using rules is when no operation is involved in calculating the value of the rule. In this particular case an extra requirement is to make custom reasoning, because the default reasoning is built into the operators, as described in section 3.2 on 26.

These wrapper rules play an important role when simulating data evaluation and acquisition software. In case of simulation, the wrapper rules usually return the value of some internal flag. In case of real-life software, the returned value is calculated by some algorithm, found in some database, signalized by an external device, etc. For the rule, which uses the value returned by the wrapper rule, the real responder is not visible at all, and changing the agent will be transparent for all rules using only wrapper rules. A further advantage: a wrapper rule can be extended without any effect on the rules involving that rule.

This special kind of rule is also ideal for using alternative information sources. For example, the excitation energy is present is some spectrum data input formats, while in some other cases the user has to be asked. Anyhow, the rule will return the correct reply, and the calling rule shall not deal with the details.

The wrapper rules are also ideal to provide an alternative form for some logical functions: the responder returns a 3-valued boolean value, without reasoning. Putting this logical function into a wrapper rule, it will be possible to get the same reply, with the corresponding resoning. Choosing the adequate alternative, one can make "invisible decision" or "extended reasoning".

3.4 Combined rules

ogical operations are interpreted between rules, so the general purpose logical operators can be applied and logical expressions of any complexity can be constructed. Since the operators between rules also result in a default reasoning, in most cases nothing has to be done to get a usable reasoning. However, the possibility is open: as described in section 3.2 on page 26, you can write your own reasoning also for combined rules.

Although the rules involve expressions, resulting in a logical value, in the expressions any kind of operation can be used. In this way different kinds of combined rules can be constructed. Some special kinds of rules deserve mentioning here.

3.4.1 Subrange-type rules

I n case of subrange-type rule it is possible to provide alternative rules for the same goal, which rules will be valid under different conditions. The range of the individual comprised rules is limited, but the resulting rule covers the full range, thanks to the appropriate combination.

```
xrIsInRange(ThresholdLow, ThresholdHigh) =
{ xrIsInSubRangel(ThresholdLow,Vall)
        OR
        ....
        IsInSubRangeN(Val2,ThresholdHigh)
}
```

I.e. the rules can have a range of validity, and they "fire" only in case the conditions lie in their particular range of validity, otherwise reply with "I do not know", thus allowing other methods to decide. The rule then can be a simple "OR" of these functions.

3.4.2 Multiple condition rule

A nother example is the 'Multiple condition' rule, used by Castle [CB-99]. Here the different conditions contribute different amounts of certainty to decide the value of the rule. Finally the contributions are summed up. The rule results in False if the sum is below the lower threshold value, True if above the upper threshold value, and Unknown between them. Any other combination of rules can be easily assembled.

```
xrMultipleCondition(ThresholdLow, ThresholdHigh) =
{ Percent = 0; //
    if(Condition1 is True) Percent = Percent +15;
    .....
    if(ConditionN is True) Percent = Percent +12;
    if(Percent<ThresholdLow)
      return False;
    else if(Percent>ThresholdHigh)
      return True;
    else return Unknown;
}
```

Note: "if True" must be here checked: "if not True" does not mean any more false!

3.5 Verifying rules

V erifying rules is of crucial importance for the whole expert system. In the case of real-life acquisition and evaluation programs, the replies are based on some calculation or other measurement-data related operation. The behaviour of the expert system (or more precisely, the incorrect value of a rule) can be the consequence of either the incorrectly assembled rule, or providing incorrect status flags for the otherwise correct rules. Because of this, it is of utmost importance to separate these two error sources and simulating the replies from the real-life programs provides an excellent possibility to do so.

In the demo program, the external conditions and quantities are provided via Graphic User Interface elements, completely controlled by the user. In this way the net effect of the rule assembly can be verified: the user can set all possible combinations and systematically vary all influencing factors, until the rule is completely tested. After this phase, the simulated replies can be replaced with real-life ones and in this phase the incorrect operating of the rules can only come due to the improper algorithm results, data handling failures, etc. Some information can be coded into configuration files, database values, etc., which can cause similar effects. This kind of errors can be successfully eliminated via simulating their effect first. The demo application is a good example of programming such a verifying setup.

3.6 Rules vs wizards

W izards are also available for making logical conclusions and reasoning with the present system. From the user's point of view, wizards are the safe way to reach some goal, without the need to know much details. From the programmer's point of view, the wizards are specialized and directed dialogs. As they are defined in the multi-platform package wxWindows [WX-92]:

These dialogs are mostly familiar to Windows users and are nothing else but a sequence of 'pages' each of them displayed inside a dialog which has the buttons to pass to the next (and previous) pages. The wizards are typically used to decompose a complex dialog into several simple steps and are mainly useful to the novice users, hence it is important to keep them as simple as possible.

In the present demo, the wizards direct the user to establish the necessary conditions, ask for the requested information pieces, and even force some way of usage via disabling temporarily some operations, making some routes one-way only, etc. Some control elements (used in the non-wizard mode) are duplicated on the wizard pages and their operation also results in handling the other control element (on the notebook, i.e. their 'alterego').

The main difference between using wizards and using the rules directly is that the wizard "knows" the necessary conditions as well as the ways as they can be established and directs/forces the user to follow those steps, while using the rules directly the user has to figure out from the reasoning he receives from the system, what is still missing; i.e., using wizards is a 'hard-wired' way of receiving a reply from the inference engine. With wizard either all phases are passed (during which the needed replies/info pieces are delivered and the logical parameters of the rules established) resulting in some conclusion or the wizard is cancelled, delivering no reply at all. Without wizard, the user has to know

(or can conclude from the reasoning of the conclusion of the unsuccessful trial) which conditions are necessary for the rule to deliver the reply he wants. Anyhow, the logic engine behind the scenes remains the same. Although it is probably correct that it is easier for the very beginner to use a wizard, with growing familiarity with the rules and conditions, the user will find much slower to reach the goal with wizards than using the rules directly.

Also note some important differences. When using wizards, the user is the medium who gives the replies needed by the expert system engine. Because of this, wizards have some important disadvantages:

- user's intelligence has to be involved in the decision process
- the process cannot be automated
- the user will be the only information source
- the result will contain some subjective elements (replies)
- allows 'improvisation'

On the other hand, it has also some advantages:

- no direct integration with the acquisition/evaluation software is necessary
- any kind of input information can be used
- allows 'improvisation'

The present demo provides both ways, saying that the more important point is the rule and the logical interrelations between some parameters, and the way as they are communicated to the inference engine is of secondary importance. The user might reach its goal safely using the wizards (and even might learn the 'how's and 'why's), or might use directly the different rules, using the success/failure method.

Chapter 4

Data acquisition

A cquiring the measured data is an important, determining part of the analyst's job. The final goal is usually to extract some compositional information, so determining the time to achieve the requested precision or to distribute the available measurement time optimally is of vital importance.

Harrison and Hazell have published some good advices and hints[HH-92] to achieve this goal. Although the paper contains some misprints, the conclusions are good and helpful, so the formulas are implemented as they are published. As shown in Fig. 8.4 on page 53, the peaks now have a parameter block called "Quantification". This block comprises quantification-related items. They are used as described in [HH-92]. (Presently, the concentration uncertainty calculation is not yet implemented).

You can use it as a specialized spreadsheet: just change any of the parameters and the rest of the parameters will be updated correspondingly. (presently, no element identification takes place, so you have to set all parameters manually, since no database connection is yet implemented.) If you add several peaks, the concentration of the individual elements (represented by the individual peaks) will be updated correspondingly. Part II

Reference Guide

Chapter 5

The base system

The base system is provided in form of a general base package, corresponding to the lowest two levels of the expert system, see Figure 1.2 on page 15. This is a general purpose package, with extensions to the spectroscopy, as described in section 5.4 on page 36. The package is made XPS/AES domain specific as described in chapter 6 on page 39.

5.1 Extended boolean logic

The new logic can be constructed as an extension to the Boolean logic: operators (defined by their "truth table", see the tables below) can be defined and implemented. As shown in tables, the two-valued Boolean logic is included as a subset of the newly introduced three-valued logic: this subset of the truth tables is exactly equivalent with the truth tables for the 2-valued logic. The "promotion" operation (mixing 3-valued and 2-valued logical variables) is also possible: operations between 2-valued and 3-valued Boolean values can be interpreted; their result is as shown in tables 5.1-5.4. As you see, in the tables the logical operators are used also in a form, corresponding to the syntax of the C++ language.

Table 5.1: Truth table for the extended boolean NOT operation

(NOT)		Α	
!A	U	F	Т
	U	Т	F

5.2 Generic rule

The 'rules' can be represented in a convenient way as objects. The 'inference engine' functionality is hidden in the operator functions, which take logical variables or

(AND)			Α	
A	&& B	U	F	Т
	U	U	F	U
В	F	F	F	F
	Т	U	F	Τ

 Table 5.2: Truth table for the extended boolean AND operation

Table 5.3: Truth table for the extended boolean OR operation

(OR)			Α	
Α	B	U	F	Т
	U	U	U	Т
B	F	U	F	Т
	Т	Τ	Т	Т

rules as arguments and they provide generic rules as result. The data of the objects are an extended boolean value and text strings, furthermore the methods are functions (comprising operators, logical functions and other computational methods). These rules have 'their own' value, which is calculated from some other rules and/or from some (properly communicated) external values. Since the resulting rules are logical functions, they are usable in rules, too, as described on page 24 in section 3.

In addition to performing the logical operation, these operator functions make 'history notes' about the actual state of the used arguments. When asking for reasoning the concluded value, this history string (preceded by the name and the actual value of the rule) is returned. Although the built-in 'default reasoning' is appropriate for most purposes, also 'custom reasoning' is possible.

Note that two different evaluation modes can be set for the rules. In the 'complete' mode all logical variables and functions comprising the rules are evaluated, independently from the actual values the rules deliver. In 'shortcut' mode only those, from which the result of the rule can be doubtlessly evaluated. For example if the rule is '**A** and **B**', and the value of **A** is false, the result of the logical expression will be false, independently of the value of **B**. Because of this, in shortcut mode **B** will not be evaluated at all. Since the variables and functions, which are not used actually for calculating the value of the rule, are redundant; leaving them away makes the evaluation shorter and quicker as well as the reasoning more clear. Because of this, the 'shortcut' mode is selected as default.

5.3 Agents

 \neg xchanging information with its environment is vital for any expert system. This explanation is done through *agents*, a frequently used term in the artificial intelligence.

xps4xps V0.09

(E	QUALS)		Α	
A	A == B	U	F	Т
	U	U	U	U
B	F	U	T	F
	Т	U	F	Т

Table 5.4: Truth table for the extended boolean EQUALS operation

The information source can be of very different nature: the spectrum data, some spectrum component, a database, the user, etc. The only common feature of the agents is that they can understand the received questions and can reply to them. The base system introduces a generic 'Agent' object, from which several specific agents are derived.

5.3.1 Database

D atabases are a standard part in some expert systems, on any field of application. The rules read the parameters from the database and decide if to "FIRE". Not yet implemented.

5.3.2 User

U sers, in some expert systems, are the only possible external information source, so to handle users, one needs special user-handling agents. The user gives questions and replies to the questions concerning database contents, evaluation-related information, etc. Not yet implemented.

5.4 Extensions for spectroscopy

I n case of real-life using an expert system in relation with data acquisition and evaluation software, one has to have some utility objects, like spectrum, peaks, background, etc. In this project, these softwares are simulated, so also the utility objects serve for this goal: they provide simulated actions, but in case of real-life usage the real objects shall provide the same functions.

5.4.1 Spectrum

In all spectroscopies, the spectrum (the measured data) is of central importance. It is the source of all measurement-related information. As it is shown in the object interdependence diagram (see Figure 5.1 on page 37), in questions concerning data evaluation, the computed spectrum (i.e. the envelope resulted by the mathematical model to the measured data) is the most important object. It has knowledge about the measured data and related information, like sample and measurement device and is built



Figure 5.1: The spectrum objects interdependence

up from components of different kind. The objects shown carry some evaluation related information, like peak position, tail height, background slope, etc.

5.4.2 Sample

M easured spectra, unfortunately, do not contain all the information, needed for assembling expert replies. One of such areas is information about the sample, which is typically rather purely covered by the experimental spectrum file, so typically additional information (configuration files, user, etc) have to be used. The sample object provides such information and is actively used in simulating sample-related information.

5.4.3 Background

 $B_{\rm object.}^{\rm ackground has no special role (yet) in the rules, so it is implemented only formally.$

5.4.4 Peak

sually, the peaks (and some peak-related objects) are the main source of evaluation information. They deliver energy levels, intensity ratios, background slopes near to

xps4xps V0.09

the peak, and so on.

Chapter 6

General XPS rules

In general, it is a good praxis to derive a 'domain-specific generic rule' from the generic rule for the specific field and to derive all rules from it. In the case of XPS, the rule xpRuleXPS serves for this goal.

6.1 Generic XPS rule

A common, domain specific rule (derived from the general generic rule) can be the common anchestor for all rules for this specific field.

Note that for non-programmers the shorthand notations 'AND', 'OR' and 'NOT' have been introduced for the C++ operations '&&', '||' and '!', respectively, and in this part of the manual the former notations are used for the easier readability.

Also note that a lot of rules exist in two versions. Similar rules are valid in case of binding energy scale and in case of kinetic energy scale. For completeness, both versions of the rules are shown here, although there is no functional difference between them. The rules of form xxxBE refer to the binding energy scale, while the rules of form xxKE refer to the kinetic energy scale.

6.2 Global variables

S ome global variables have important effect on most of the rules. They are pre-set (in the xpsetup.h source file), but your application can set it again any time.

6.2.1 E4Background

I loating. This variable is defined as the minimum extra length of spectrum on both sides of energy, used to calculate the background under the peak, see section 6.9 on page 42. Its default value is 1 eV.

6.2.2 EnergyTolerance

D ouble. This value is used when comparing energy values. If the compares energies differ less than this tolerance value, they are considered to be equal. It can be set as described in section 8.3.2.1.2 on page 55.

6.2.3 xpShortcutMode

 $B_{8.3.2.1.4}$ ool. This variable determines if the evaluation of the rules is to be done in complete or shortcut mode, see section 3.2 on page 26. It can be set as described in section 8.3.2.1.4 on page 56.

6.3 HasPeakInRangeBE

I t is a frequently used component in the rules whether the spectrum has peak at all in the region given on binding energy scale. This rule calculates the value of the rule according to the expression

```
HasPeakInRangeBE(Spectrum,BELow,BEHigh) =
{
  found = false;
  for ((Peak in Spectrum.Peaks) or found)
   {
    found = found
        OR
        IsPeakInRangeBE(Peak,BELow,BEHigh)
  }
}
```

6.4 HasPeakInRangeKE

I t is a frequently used component in the rules whether the spectrum has peak at all in the region given on kinetic energy scale. This rule calculates the value of the rule according to the expression

6.5 IsEnergyBE

This simple 'rule' is a wrapper rule and returns the state of an internal status flag. It answers the question whether the energy data are given on binding energy scale. In the demo program, the actual reply is given correspondingly to the setting of the 'Energy type' radiobox on the spectrum panel. In real evaluation/acquisition software, the spectrum contains this information but some other method (like asking the user) might also be applied.

```
IsEnergyKE(Spectrum) =
{
   Spectrum.EnergyBE == True;
}
```

6.6 IsEnergyKE

This simple 'rule' is a wrapper rule and returns the state of an internal status flag. It answers the question whether the energy data are given on kinetic energy scale. In the demo program, the actual reply is given correspondingly to the setting of the 'Energy type' radiobox on the spectrum panel. In real evaluation/acquisition software, the spectrum contains this information but some other method (like asking the user) might also be applied.

```
IsEnergyKE(Spectrum) =
{
   Spectrum.EnergyBE == False;
}
```

Note: Neither of the two rules above fires in case the energy type is 'Unknown'.

6.7 IsEnergyAvailBE

I t is important to know when working with some algorithms whether the energy data are available on binding energy scale. This rule determines to calculate the value of the rule according to the expression

```
IsEnergyAvailBE(Spectrum) =
{
    IsEnergyBE(Spectrum)
    OR
        IsEnergyKE(Spectrum)
        AND
        IsXEnergyKnown(Spectrum)
}
```

Since only (wrapped) rules are involved, no difference between demo and real-life behaviour.

6.8 IsEnergyAvailKE

I t is important to know when working with some algorithms whether the energy data are available on kinetic energy scale. This rule determines to calculate the value of the rule according to the expression

```
IsEnergyAvailKE(Spectrum) =
{
    IsEnergyKE(Spectrum)
        OR
        IsEnergyBE(Spectrum)
        AND
        IsXEnergyKnown(Spectrum)
}
```

Since only (wrapped) rules are involved, no difference between demo and real-life behaviour.

6.9 IsPeakInRangeBE

A typically internally used rule. Here the information source is a peak object. The returned value is true if the peak lies in the given energy range, false otherwise. The rule internally verifies if the available data allow for the secure background determination on both sides of the peak. If not, the result will be False.

```
IsPeakInRangeBE(Peak,BLow,BHigh) =
{
   // Transform Peak.Energy if on kinetic scale
   IsRegionMeasuredBE(Spectrum,BLow-E4Background,BHigh+E4Background)
        AND
   BLow <= Peak.Energy <= BHigh
}</pre>
```

6.10 IsPeakInRangeKE

A typically internally used command. Here the information source is a peak object. The returned value is true if the peak lies in the given energy range, false otherwise. The rule internally verifies if the available data allow for the secure background determination on both sides of the peak. If not, the result will be False.

xps4xps V0.09

```
IsPeakInRangeKE(Peak,KLow,KHigh) =
{
    // Transform Peak.Energy if on binding scale
    IsRegionMeasuredKE(Spectrum,KLow-E4Background,KHigh+E4Background)
        AND
        KLow <= Peak.Energy <= KHigh
}</pre>
```

6.11 IsRegionMeasuredBE

S ometimes it is important to know whether the binding energy region in question is measured at all. It is 'True' if the given region is completely included in the measured energy region. This rule determines to calculate the value of the rule according to the expression

```
IsRegionMeasuredBE(Spectrum,BLow,BHigh) =
{
   // Transform Spectrum.MinEnergy and Spectrum.MaxEnergy,
   // if given on kinetic scale
   Spectrum.MinEnergy <= BLow <= Spectrum.MaxEnergy
   AND
   Spectrum.MinEnergy <= BHigh <= Spectrum.MaxEnergy
}</pre>
```

As you see, the energy limits of the measured spectrum are calculated to binding energy scale, if the limits of the measured energy region are given on kinetic energy scale.

6.12 IsRegionMeasuredKE

S ometimes it is important to know whether the kinetic energy region in question is measured at all. It is 'True' if the given region is completely included in the measured energy region. This rule determines to calculate the value of the rule according to the expression

```
IsRegionMeasuredKE(Spectrum,KLow,KHigh) =
{
   // Transform Spectrum.MinEnergy and Spectrum.MaxEnergy,
   // if given on binding scale
   Spectrum.MinEnergy <= KLow <= Spectrum.MaxEnergy
   AND
   Spectrum.MinEnergy <= KHigh <= Spectrum.MaxEnergy
}</pre>
```

As you see, the energy limits of the measured spectrum are calculated to kinetic energy scale, if the limits of the measured energy region are given on binding energy scale.

6.13 IsXEnergyKnown

t is a frequently used wrapper rule whether the excitation energy is known at all. The reply is returned based on some internal flag.

```
IsXEnergyKnown =
{
   Spectrum.XEnergyKnown == True;
}
```

In the demo program, the internal flag is set correspondingly to the setting of the 'XEnergyKnown' radiobox on the spectrum panel. In real evaluation/acquisition software, the flag is set based on the spectrum contained information but some other method (like asking the user) might also be applied.

Chapter 7

Carbon contamination rules

C arbon is an important element in the experimenter's praxis. It might be present in the spectrum as one of its constituent, or might build-up on the sample surface due to environmental effects.

7.1 DoMarkCarbon1sPeak

A nactive rule, for finding the Carbon 1s peak in the spectrum and setting its corresponding flag. If the flag of some peak in the spectrum is already marked the rule accepts that peak as "Carbon 1s peak" and returns without setting any flag. If more than one peaks having that flag set are present in the spectrum, the rule does nothing, but returns with an error message, naming the first two peaks found as having the flag set.

7.2 DoesSampleContainCarbon

t is a wrapper rule used in the carbon-contamination rules, whether the sample itself contains carbon. The reply is returned based on some internal flag.

```
DoesSampleContainCarbon =
{
   Sample.ContainsCarbon == True;
}
```

In its present form, the rule returns the status of "ContainsCarbon" of the "sample" object. In the demo, that status is set from the "Sample info" notebook page, in real-life case it might be contained in the spectrum-attached file or the user might be asked.

7.3 HasCarbon1sPeak

f the carbon is present in the spectrum, can be checked by this rule as proposed in [CB-99]. It is used in this project as

```
HasCarbonlsPeak (Spectrum) =
{
    IsCarbonXPresent(Spectrum)
    AND
    IsCarbonAugerPresent(Spectrum)
    AND
    NOT IsRutheniumPresent(Spectrum)
}
```

7.4 IsCarbon1sPeak

he rule set proposed in [CB-99] does not define this rule. One possible way could be to 'fire' if the rule 'HasCarbon1sPeak' fires, but is might be a problem if more than one peaks in the corresponding region exist. Presently, the rule "fires" if the flag Peak.Carbon1s_flag is set.

```
IsCarbon1sPeak =
{
    Peak.Carbon1sPeak == True;
}
```

7.5 IsCarbonAngleRatioBiggest

F rom the whole spectrum ratioing method follows that the ptotopeaks from elements located at different depts in the sample result a different intensity ratio for the photopeak measured at different angles. If this ratio for carbon is higher than that for all other elements, it increases the certainty that carbon is present as contamination.

7.6 IsCarbonAugerPresent

his is a wrapper rule used in rule HasCarbon1sPeak. It uses internally the rule HasPeakInrangeKE. It might be necessary to add some more rules later.

```
IsCarbonAugerPresent (Spectrum)
{
    HasPeakInrangeKE(Spectrum, 269, 275) //KLL Auger present
}
```

7.7 IsCarbonContaminationConsensus

f the sample is carbon contaminated, cannot be decided from one single condition. Castle (maybe arbitrarily) suggested to use a special rule involving five subsets, where

xps4xps V0.09

the consensus of the comprised rules decides about the carbon contamination. These used subrules are

- IsCarbonEnergySeparationOK (see section 7.10)
- DoesSampleContainCarbon (see section 7.2)
- IsCarbonShirleyTailHigh (see section 7.10)
- IsCarbonPostPeakSlopeBiggest (see section 7.9)
- IsCarbonAngleRatioBiggest (see section 7.5)

These rules have a credit and the rule results in true if the percentage of the weighted "True" replies from the subrules is above some upper threshold value (by default 70%), "False" if it is below some lower threshold value (by default 25%), "Unknown" between them. The resulting reasoning contains the reasonings of the subrules under "Details".

7.8 IsCarbonEnergySeparationOK

In nergy separation between the two major excursions in the first derivative of the KLL spectrum of carbon is indicative of the hybridization state of the carbon compound. The energy separation value should be ca. 17 eV, if it is between 15 eV and 20 eV, it increases the certainty that carbon is present as contamination.

7.9 IsCarbonPostPeakSlopeBiggest

I t is expected for carbon as contamination that the slope of the tail on the low energy side of the peak is higher than the slope of the peaks from other elements. The rule finds all peaks and compares this parameter to that of the C1s peak. If the condition that the slope is biggest for the carbon peak is fulfilled, it increases the certainty that carbon is present as contamination.

7.10 IsCarbonShirleyTailHigh

S hirley scattering parameter (the tail height to the peak height ratio) is also one of the signs used in identifying the carbon as contamination. This rule expects a Carbon 1s line present in the spectrum. The tail height is compared against the Shirley height threshold value (by default 0.1). A value above that threshold increases the certainty that carbon is present as contamination.

7.11 IsCarbonXPresent

his is a wrapper rule used in rule HasCarbon1sPeak. It uses internally the rule HasPeakInRangeBE. It might be necessary to add some more rules later.

```
IsCarbonXPresent (Spectrum)
{
    HasPeakInRangeBE(Spectrum, 282, 288) //Carbon line(s) present
}
```

7.12 IsRutheniumPresent

his is a wrapper rule used in rule HasCarbon1sPeak. It might be necessary to add some more rules later. It uses internally the rule HasPeakInRangeBE.

```
IsRutheniumPresent (Spectrum)
{
    HasPeakInRangeBE(Spectrum, 458, 462) //ruthenium present
}
```

Chapter 8

The demo program

In order to illustrate the abilities of the library and also to provide a way to verify the created rules under simulated external conditions, a demo program is also provided with the package. The application is based on the multi-platform macro package wxWindows [WX-92]. It provides a **G**raphic **U**ser Interface for making the necessary settings/changes for the rules, the used settings, the reasoning and so on.

The demo program is provided to simulate the data acquisition and evaluation software and to allow the "embedded expert system" to interact with them. Here the mentioned components are simulated (via GUI screen elements settings) and the expert system rules are really acting on the simulated replies. The demo applications on the different software platforms have a very similar functionality, except that they have a "native", platform-specific appearance and behaviour. This difference might also involve minor functional differences.

Note that this program is only for illustration, i.e. it is not equipped with a lot of comfort utilities and also it is not bullet-proof. Please use it setting "reasonable" conditions only; it was not a point of design to protect it from mishandling. Playing with the demo might be of interest for all end-users, as well as it is the proper way of acquiring initial impressions about the functionality implemented in the different rule sets. Note that similar applications can effectively assist the experts to study the rule sets in their experimental phase.

The executable demo file (*.exe in case of Win32 platform) can be found in the "/bin" subdirectory. It is ready to operate after installation: just start it as any other application on the given platform.

Note: The new value written in the text input boxes becomes effective only if you terminate the input with the "enter" key. In that case the typed string is interpreted, converted to the internal representation, and displayed in a normalized form.

lease choose la	nguage:
(System default	
English (U.S.) Maguar	
magyar	

Figure 8.1: The spectrum page of the demo application

8.1 The main window

A fter starting up the executable xps4c1s, first a language selection window (see Figure 8.1 on page 50) pops up, and you can select your preferred language. Note that the default language of the application is English.

After selecting the language, a main frame (see Figure 8.2 on page 51), containing a "notebook" comes up (i.e. several notebook pages of different function can be selected with the "tabs" at the top of the internal frame). Right now, the notebook is only equipped with one page (labeled 'REASONS') which contains the welcome text. Later (as you will see it in a moment) it will contain the reasoning of the actual rule. It is cleared before selecting another rule to 'Fire!', so it contains always the reasoning of the result of one single rule.

Above the notebook and below the heading line, you find the "menubar", the menu items of which give you a way to try out different rules, to change some settings, to simulate the actions of the data acquisition/simulation software, etc.

In its initial state, the demo application simulates a started-up data evaluation program, with no experimental data loaded. In order to do any action, you need to use the menu system as you used at any other application on your platform: click or use schortcut or whatever you prefer to handle the menu items. Just note: you can exit the demo program via executing command "SPECTRUM | EXIT". Here and below, the bar character "|" is used to separate the different level menu items, so in this way "white space" characters in the menu item names are allowed.

8.2 Verifying rules

n order to simulate a data evaluation program, select the "SPECTRUM | LOAD" command. You migh see that the menu commands are "context sensitive": some of the

X→ xps4xps V0.06 demo application	• 🗆 🗙
<u>S</u> pectrum <u>R</u> ule <u>O</u> ptions <u>H</u> elp	
Reasons	
Welcome to the XPS C1s demo * * xps4xps V0.06 for Linux 2.4.19 i686 * Started at 13:01:14, on 2002-10-25 *	

Figure 8.2: The main page of the demo application

menu items are disabled (grayed out) in certain states. Having loaded measured experimental data, some of the formerly disabled items get enabled, so you have more possibilities in the "evaluation". Namely, after reading in some experimental data, you have one more notebook page (tagged with "Spectrum"), simulating the read-in experimental spectrum.

To make a rule "Fire", just select the corresponding rule (in some case the "OPTIONS | SETTINGS" menu items shall also be used, as discussed at the individual rules). Right now, select "RULE | XPS | ISENERGYAVAILBE". Doing so, the notebook page changes to the 'REASONS' page, and the reasoning (shown in the notebook page) tells you in the first line the result of calculating the value of the rule. The line contains between quotes the name of the rule and in capitals its value, and in the forthcoming line(s) the reasoning, i.e. it shows what are the conditions under which the inference engine concludes such a reply. Click on tab "SPECTRUM" now, change the settings of the radioboxes and select rule "RULE | XPS | ISENERGYAVAILBE" again. The inference engine will make reasoning as

xps4xps V0.09



Figure 8.3: The spectrum page of the demo application

described in chapter 3 on page 24.

You may also check the effect of the "shortcut" evaluation mode, see section 5.2 on page 34. Select command "OPTIONS | SETTINGS | SHORTCUT MODE" (see below) and select the rule "RULE | XPS | ISENERGYAVAILBE" again. Now the reasoning is the same, but more verbose. To try out some more rules, see the notebook pages and menu items in the sections below.

8.2.1 The spectrum page

Spectrum notebook page (see Figure 8.3 on page 52), created with command "SPECTRUM I LOAD", contains some characteristics of the "measured spectrum": the demo sets the low and high energy of the measured data range, as well as the value of the X-ray excitation energy. Please notice that as shown by the radioboxes next to the corresponding energy values, the energy type is assumed to be of type binding: i.e. the demo program assumes that the measured spectrum data file does contain this information (say the measured counts are given in VAMAS form). Similarly, the status if the excitation en-

ergy is known, is set to 'True', and for convenience, the well-known $Al K_{\alpha}$ default value is pre-set. The page also contains the value of the charging correction and the energy separation of the carbon peaks.

Note that the demo can only handle one "spectrum" at a time; consequently, the command generating the notebook page can only be used once. Of course, all other pages can only be created when the spectrum page exists already.

📲 xps4xps ¥0.08 demo application
Spectrum <u>R</u> ule <u>W</u> izard <u>O</u> ptions <u>H</u> elp
Reasoning Spectrum Peak 1
Identification Energy type Peak energy (eV) 285.00 XEnergyKnown ① Unknown X energy (eV) 1486.60 ① Unknown ① Unknown ② Kinetic ③ Ealse ③ Irue ③ Irue Carbon contamination Is C1s peak Post Peak Slope 0.321 ③ Unknown ④ False ④ ① ④ Ealse ④ 0.121 ④
Quantification Sensitivity (rel) 1.00 Peak rate (c/s) 200.00 Intensity (counts) 200.00 Concentration (%) 100.00 Bgnd rate (c/s) 100.00 Bgnd (counts) 100.00 d Conc (%) 0.00 Rel prec (%) 17.32 Meas time(sec) 1.00

Figure 8.4: The peak page of the demo application

8.2.2 The peak page

P eak notebook page (see Figure 8.4 on page 53), created with command "Spectrum | ADD PEAK", contains the expert-system related characteristics of a peak, in three groups.

The very basic parameter block serves the "Identification" of the peak. The very basic parameter of the peak is its energy, which is not only a value but also an energy type, as shown by the radiobox next to the value. The energy type is inherited from the parent spectrum (the actual value is used when creating the peak), but after creating it can be set independently. Similarly, the value of the exciting energy (used to change between kinetic and binding energy scale) and its type are inherited from the parent spectrum, but after creating the peak they can be set independently.

The parameters in the next block supports the "Carbon contamination" rule set. The radiobox and the two values are used when dealing with "carbon contamination" rules; for details see [CB-99] and references therein.

The parameters "Quantification" box are used for calculating the optimal measurement time or precision. For details see section 4 on page 31.

Note that this page can only be created when the "spectrum page" created already.

👯 xps4xps ¥0.05 demo application	<u> </u>
Spectrum Rule Options Help	
Reasons Spectrum Sample info	
Contains carbon ⊙ ∐nknown ⊙ Ealse ⊙ Irue	

Figure 8.5: The sample page of the demo application

8.2.3 The sample page

 $S_{\rm ADD \ SAMPLE \ INFO"} \ \text{(see Figure 8.5 on page 54), created with command "Spectrum I ADD \ SAMPLE \ INFO"}, \ \text{will \ contain \ sample-related \ information, \ usually \ found \ in \ the spectrum \ file. For the present rule set, only the carbon \ content-related \ info \ is \ important.}$

Note that this page can only be created when the "spectrum page" created already.

8.3 The menu system

The demo application can be controlled via commands, which are reachable through the menu items of the application. Between the title line and the notebook page, a menubar is shown. The items (menus, submenus) can be selected as usual on your platform.

8.3.1 Help menu

elp menu items usually contain on-line help facilities, some description, etc. In this early phase, only one menu item provided.

8.3.1.1 About

B usiness card of the demo application will be shown if you select this command. It contains the author's address, version number of its own as well as that of the wxWindows package it is based upon.

8.3.2 Options menu

nder this menu group you will find some settings, options, etc., which commands do not fit in the other menu groups.

8.3.2.1 Settings

ome setting affect seriously the operation of the demo program. Those settings are described in this command group.

8.3.2.1.1 Background length The peak evaluation procedure need some extra spectrum region, to determine the background. This commands allows to set this value (by default 1eV). The rules of type "Is peak in region" use this value to determine is the peak can be safely evaluated from the data measured on that region.

8.3.2.1.2 Energy tolerance When comparing energy values of float type, the program uses the "within this tolerance interval" criterium rather than exact match. This command allows to change the value of this interval (0.01 eV by default).

8.3.2.1.3 Use Local Numeric mode Data format of floating numbers, allowed in the expert system data files, as well as those in the results, depend on the actual language (more precisely: on the *corresponding local settings*) selected, because some languages use "comma" as decimal point, while other languages use "comma" for separating data fields in the input/output files. If your computer is set up to use these local settings, especially "local numerics", then you might wish to use either your local number formatting or the

conventional ("C"-like) formatting when reading and writing floating numbers. Depending on the actual value of this flag, xps4xps sets the locale numeric settings to those used in the language "C" (this is the default) if the flag is disabled or uses the numeric settings corresponding to the selected language if it is enabled. In the latter case numbers will be formatted in the output files and are expected to be formatted in the input files according to the settings used in that language. The setting will be valid until you change it again.

8.3.2.1.4 Shortcut mode As described in section 3.2 on page 26, the rule evaluation can be carried out in two different evaluation modes. This menu item is checkable, i.e. in case the shortcut mode is selected, a checkmark can be seen before the menu item name. Checking the menu item repeatedly, the variable *xpShortcutMode* changes from true to false or oppositely. Its default value is 'true'.

8.3.2.1.5 Trial energy max Some commands use an energy range, given by their lower and upper bound. This command allows to set the upper limit of the trial energy region.

8.3.2.1.6 Trial energy min Some commands use an energy range, given by their lower and upper bound. This command allows to set the lower limit of the trial energy region.

8.3.3 Rule menu

R ules are the most important component of the present expert system. Selecting one of the menu items in this group, you can try out the rules in this library. Note that all rules are operating on simulated external conditions, and the simulation is done by GUI elements, as described at the individual rules. The name of the menu items is identical with those used in the "Reference manual".

8.3.3.1 Carbon

 γ arbon-related rules are included in this group, like rules for checking the presence γ of carbon in general or carbon as contamination.

8.3.3.1.1 DoMarkCarbon1sPeak Finds the carbon 1s peak as discussed in section 7.3 on page 45 and marks it (sets the flag "Is C1s peak"). Also checks if more than one peaks are in the spectrum with this flag set.

8.3.3.1.2 HasCarbon1sPeak Returns value of the rule, if carbon 1s peak is present in the spectrum, as discussed in section 7.3 on page 45.

8.3.3.1.3 IsCarbonAugerPresent Returns the value of the rule if the Carbon Auger group is present, as discussed in section 7.6 on page 46.

8.3.3.1.4 IsCarbonContaminationConsensus This menu item invokes the rule Is-CarbonContaminationConsensus as discussed in section 7.7 on page 46.

8.3.3.1.5 IsCarbonXPresent Returns the value of the rule if the C1s peak is present, as discussed in section 7.11 on page 48.

8.3.3.1.6 IsRutheniumPresent Returns the value of the rule if the Ruthenium is present, as discussed in section 7.12 on page 48.

8.3.3.2 Sample

S ample-related rules are included in this group. Typically, they are only used to verify sub-rules, used in the more "physical" rules.

8.3.3.2.1 Contains Carbon Returns value of the rule, if carbon present in the sample by its composition, as discussed in section 7.2 on page 45.

8.3.3.3 Truth table

ruth table of the 3-valued logic will be printed in the log window. The values are set as outlined in section 5.1 on page 34 and in tables 5.1-5.4.

8.3.3.4 XPS

eneral XPS-related rules belong to this group. Although the rules are very basic ones, it is worth to study and understand them, because they are frequently used in the more advanced rules.

8.3.3.4.1 HasPeakInRangeBE Returns the value of the rule if the given binding energy region contains a peak, as discussed in section 6.3 on page 40.

8.3.3.4.2 HasPeakInRangeKE Returns the value of the rule if the given kinetic energy region contains a peak, as discussed in section 6.4 on page 40.

8.3.3.4.3 IsEnergyAvailBE Returns the value of the rule if the binding energy values are available in the generic spectrum object, as discussed in section 6.7 on page 41.

8.3.3.4.4 IsEnergyAvailKE Returns the value of the rule if the kinetic energy values are available in the generic spectrum object, as discussed in section 6.8 on page 42.

8.3.3.4.5 IsRegionMeasuredBE Returns the value of the rule if the trial energy region is within the measured energy region. It is assumed that the trial energy region limits are given on binding energy scale.

8.3.3.4.6 IsRegionMeasuredKE Returns the value of the rule if the trial energy region is within the measured energy region. It is assumed that the trial energy region limits are given on kinetic energy scale.

8.3.4 Spectrum menu

S pectrum operations are only simulated in the demo application. However, the functional equivalents shall be available on the real-time data acquisition and evaluation systems.

8.3.4.1 Add peak

P eak component can be added via using this command. As a result, a new panel appears in the 'notebook' and some peak parameters can be set via the controls on that panel. This command is only available if "spectrum is loaded" already.

8.3.4.2 Add sample

S ample information can be added via using this command. As a result, a new panel appears in the 'notebook' and some sample parameters can be set via the controls on that panel. This command is only available if "spectrum is loaded" already.

8.3.4.3 Exit

he EXIT command is by convention placed under the main command group, in this case here, although it has not much to do with the 'Spectrum' group's functionality.

8.3.4.4 Load

oad-ing a spectrum is absolutely necessary to deal with some spectrum in the siml ulated data evaluation software. No other "spectrum evaluation operations" are possible before it. After 'Load'ing, a new spectrum panel appears and some characteristics of the simulated spectrum can be set using the controls on this panel.

8.3.5 Wizard menu

W izards are the easy and safe way for the beginners to use the expert system (see section 3.6 on page 29). Just select the wizard you need and follow the on-screen instructions (and set the different control elements on the different wizard pages) until it finishes and read the conclusion from the screen. Because the wizards use the same rules and reasoning, you will surely arrive at the same conclusion. However, some (but not all) conditions are built-in the wizard, so it might be a safer way to reach your goal.

Note: The inference method and the source of information for the wizards are the same as for the rules. Because of this, in this demo program the information source panels in the main notebook are used (or created if not created before starting the wizard). While the wizard is running, you can only handle the 'copy' you have on the wizard page, but upon a wizard page change, the 'original' is also updated.

The "IsCarbonPresent" wizard				×
	Setup spectrum attrib You will need bot and binding ener Please set the ch below according	utes th the kinetic, gies in your spe leckboxes and v ly.	ctrum. alues	
	Energy type ○ <u>U</u> nknown ○ <u>K</u> inetic ⊙ <u>B</u> inding	E low (eV) E high (eV) Charging (eV)	100.00 1400.00 1.50	
	- XEnergyKnown - O <u>U</u> nknown O False	X energy (eV)	1486.60	
		< <u>B</u> ack	<u>N</u> ext >	Cancel

Figure 8.6: The spectrum setup page of the carbon wizard

All wizard pages (for an example see Figure 8.6 on page 59) contain a title line, a main panel and a bottom control line. In the control line some buttons allow to go to the previous and the next wizard page, or to cancel executing the wizard. The "Previus" and "Next" direction buttons might get disabled on some conditions temporarily. Although there is a pre-defined sequence for displaying the wizard pages, some of them (depending on the actual state of the control elements and the user replies) might be skipped.

The left side of the main panel contains a bitmap (a real XPS wizard). The right side does the real job. The text at the top of the panel explains the goal of the current step. The bottom part contains the control elements (the source of the information). Setting the control elements properly is a precondition to advance to the next page. Also sometimes parts of the intruction text (at the top of the panel) as well as some controls (at the bottom of the panel) are greyed out, what means that that instruction/control is not actual/effective any more. During stepping through the wizard, all necessary conditions are established and all control elements are set, so the wizard ends up with telling the conclusion and its reasoning. The final conclusion is told when the wizard finishes. Before this, it tells a temporary conclusion and its reasoning; at this point you might go back and correct some settings. After the wizard finishes, you need to run the wizard again if you wish to see how the conclusion changes with some setting. If such repeated running occurs, the previously established conditions are used (i.e. you do not start from scratch the second time). You might also use these conditions if you use the rules directly.

8.3.5.1 Is Carbon present

r or determining if the carbon is present in your spectrum, several steps are necessary. This "carbon wizard" will guide you through the steps you have to do and also helps to establish the necessary conditions.

As all wizards, it begins with a greeting page. In the first two function pages you might define the spectrum you are dealing with (i.e. it allows to "load" a spectrum) and allows to set the spectrum-related parameters. If the spectrum is already present, the first page will be skipped.

The next two wizard pages allow to define and parametrize the Carbon 1s peak (the peak you guess to be that). If no peaks are available (the available peaks are which are defined for the spectrum and are not yet assigned to any of the peaks needed for the Carbon rule), it allows to create a new peak. The new peak inherits the energy type and the other settings from the parent spectrum, but you are allowed to change any of them.

According to the "Carbon rule", the presence of the Carbon Auger line is also necessary to confirm the presence of the C1s line. The next two pages (in complete analogy with the previous two pages) allow to define and parametrize these Auger peaks.

A further requirement might be the absence of the Ruthenium. The third pair of wizard pages defines and parametrizes the Ruthenium peak (if any).

Of course, it might happen that there is no such peak in the sample. In such a case, just *uncheck* the checkbox at the very bottom of the panel and you will see the conclusion immediately.

The wizard pages allow you some freedom. They tell what is expected at that position, but you are free to set improper energy type or value, for example. The rule will evaluate this situation correctly, so in this way you cannot change the conclusion. An alternative would be requiring to define a peak with parameters which meet the expectation. Part III

Appendix

Bibliography

- [CB-99] J. E. Castle, M. A. Baker, The feasibility of an XPS expert system demonstrated by a rule set for carbon contamination. J. Electron Spectroscopy and Related Phenomena 105(1999)245-256
- [CJ-02] J. E. Castle, *A wizard source of expertize in XPS.* Surface and Interface Analysis 33(2002)196-202
- [DCJ89] C. J. Date, *Be Careful with SQL EXISTS!* Database Programming and Design 2(1989)50-52 or http://www.firstsql.com/inulls.htm
- [DHS88] W. A. Dench, L. B. Hazell, M. P. Seah, VAMAS Surface Chemical Analysis Standard Data Transfer Format with Skeleton Decoding Programs. NPL Report DMA(A)164, July 1988; Surface and Interface Analysis 13(1988)63-122.
- [EB-99] B. Eckel, http://www.mindview.net/Books
- [FA-66] M. Fisch, T. Atwell, *Peirce's Triadic Logic.* Transactions of the Charles S. Peirce Society 11(1966)71-85 or http://plato.stanford.edu/entries/peirce-logic/
- [HD-97] Dimitri van Heesch, *doxygen, a documentation system for C++, C, Java, IDL* http://www.doxygen.org
- [HH-92] K. Harrison, L. B. Hazell, The Determination of Uncertainties in Quantitative XPS/AES and its Impact on Data Acquisition Strategy. Surface and Interface Analysis 18(1992)368-376.
- [OM-96] M. F. X. J. Oberhumer, L. Molnár, the Ultimate Packer for eXecutables http://upx.sf.net/
- [OS-01] The Open Source Initiative, *The Open Source development* http://www.opensource.org/
- [RJ-96] J. Russel, Inno Setup self-installer package http://www.jrsoftware.org/isinfo.php

xps4xps V0.09

[SC-01]	Christian Schenck, MiKTeX, typesetting beautiful documents
	http://www.miktex.org/

- [SF-02] Open Source Development Network, Inc, *Breaking Down the Barriers to Open* Source Development http://www.sourceforge.net/
- [SRS98] Seatle Robotics Society, *Fuzzy Logic Tutorial* http://www.seattlerobotics.org/encoder/mar98/fuz/flindex.html
- [TC-99] TeXnicCenter, An integrated development environment (IDE) for developing LaTeX-documents on Microsoft Windows http://www.toolscenter.org/products/texniccenter/
- [VJ-02] J. Végh, XPS4XPS: an 'embedded' eXPert System for XPS J. Electron Spectroscopy and Related Phenomena 133(2003)87. to be published in Computer Physics Communications
- [VJ-88] J. Végh, *The analytical form of the Shirley-type background.*J. Electron Spectroscopy and Related Phenomena 46(1988)411.
- [WX-92] J. Smart, et al. *The open source, cross-platform GUI framework* http://wxwindows.sourceforge.net/
- [XPS02] J. Végh, An "embedded" expert system for XPS http://xps4xps.sourceforge.net/

Index

[CB-99], 7, 28, 45, 46, 54 [CS-01], 9 [DCJ89], 16 [EB-99], 14 [FA-66], 16 [HD-97], 8, 9 [HH-92], 7, 12, 31 [OM-96], 9 [OS-01], 7 [RJ-96], 9, 18 [SF-02], 7 [SRS98], 15 [TC-99], 9 [VJ-02], 7 [WX-92], 6, 8, 16, 20, 29 [XPS02], 18 39 &&, 25, 39 **OPTIONS** SETTINGS, 51 SHORTCUT MODE, 26, 52 RULE XPS ISENERGYAVAILBE, 51, 52 **S**PECTRUM ADD PEAK, 53 ADD SAMPLE INFO, 54 EXIT, 50 LOAD, 50, 52 3-valued boolean, 27 AND, 25, 35, 39 background length, 39, 55 boolean

xps4xps V0.09

extended, 16 C++, 8, 25 C-like number formatting, 56 carbon wizard, 60 changes.txt, 19 charging shift, 13 class name, 25 comma, 55 complete mode, 26 copyright, 6 custom reasoning, 26, 27, 35 decimal point, 55 default reasoning, 26, 27, 35 DoesSampleContainCarbon, 45 domain-specific rule, 21 DoMarkCarbon1sPeak, 45 energy shift, 13 tolerance, 40, 55 trial maximum, 56 minimum, 56 energy scale binding, 13 kinetic, 13 EQU, 36 evaluation mode, 40 complete, 26 shortcut, 26 expert knowledge, 7 expert system, 7, 14, 24 shell. 16

extended logic, 15 Free Software Foundation, 6 General Public Licence, 6 generic rule, 35 GNU. 6 HasCarbon1sPeak. 45 HasPeakInRangeBE, 40, 57 HasPeakInRangeKE, 57 HasPeakInRangKE, 40 inference engine, 7, 14, 16, 24, 26 Inno Setup installer, 18 IsCarbon1sPeak, 46 IsCarbonAngleRatioBiggest, 46 IsCarbonAugerPresent, 46 IsCarbonContaminationConsensus, 46 IsCarbonEnergySeparationOK, 47 IsCarbonPostPeakSlopeBiggest, 47 IsCarbonShirleyTailHigh, 47 IsCarbonXPresent, 48 IsEnergyAvailBE, 41, 57 IsEnergyAvailKE, 42, 57 IsEnergyBE, 41, 42 IsEnergyKE, 41, 42 IsPeakInRangeBE, 40, 42 IsPeakInRangeKE, 40, 42 IsRegionMeasuredBE, 42, 43, 57 IsRegionMeasuredKE, 42, 43, 58 IsRutheniumPresent. 45, 48 IsXEnergyKnown, 41, 42, 44 language, 50 local number formatting, 55 logic 3-valued, 16 extended, 15 logical operator, 27 menubar, 50

NOT, 34, 39 notebook, 50 number formatting, 55 numeric mode local, 55 object, 14 oriented programming, 14 OR, 25, 35, 39 radiobox ContainsCarbon, 45 energy type, 41 XEnergyKnown, 44 readme.txt, 19 reasoning, 16, 26, 27, 50 custom, 26, 27 default, 26, 27 rule coding, 24 domain specific, 21 evaluation, 26 generic, 35 wrapper, 26, 27 shortcut mode, 26, 35, 52, 56 system locale, 55 wizard. 29. 58 carbon, 60 wrapper rule, 26, 27 wxWindows, 6, 16 X-ray, 13 xrDoesSampleContainCarbon, 45 xrDoMarkCarbon1sPeak. 45 xrHasCarbon1sPeak, 45 xrHasPeakInRangeBE, 40, 48 xrHasPeakInRangeKE, 40 xrHasPeakInrangeKE, 46 xrIsCarbon1sPeak, 46 xrIsCarbonAngleRatioBiggest, 46 xrIsCarbonAugerPresent, 45, 46

xps4xps V0.09

xrIsCarbonEnergySeparationOK, 47 xrIsCarbonPostPeakSlopeBiggest, 47 xrIsCarbonShirleyTailHigh, 47 xrIsCarbonXPresent, 45, 48 xrIsEnergyAvailBE, 41 xrIsEnergyAvailKE, 42 xrIsEnergyBE, 41 xrIsEnergyKE, 41 xrIsPeakInRangeBE, 42 xrIsPeakInRangeKE, 42 xrIsRegionMeasuredBE, 43 xrIsRegionMeasuredKE, 43 xrIsRutheniumPresent, 48 xrIsXEnergyKnown, 44