S_2 State Photodissociation of Diphenylcyclopropenone, Vibrational Energy Transfer along Aliphatic Chains, and Energy Calculations of Noble Gas–Halide Clusters

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"

der Georg-August-Universität Göttingen
im Promotionsprogramm Chemie
der Georg-August University School of Science (GAUSS)

vorgelegt von

Hendrik Vennekate

aus Haan

Göttingen, 2014

Betreuungsausschuss

Prof. Dr. Dirk Schwarzer, Forschungsgruppe Reaktionsdynamik,
Max-Planck-Institut für biophysikalische Chemie

Prof. Dr. Jürgen Troe, Institut für Physikalische Chemie
Georg-August-Universität Göttingen

Prof. Dr. Jörg Schroeder, Institut für Physikalische Chemie
Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Dirk Schwarzer, Forschungsgruppe Reaktionsdynamik,
Max-Planck-Institut für biophysikalische Chemie

Korreferent: Prof. Dr. Jürgen Troe, Institut für Physikalische Chemie
Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Jörg Schroeder, Institut für Physikalische Chemie
Georg-August-Universität Göttingen

Prof. Dr. Markus Münzenberg, I. Physikalisches Institut
Georg-August-Universität Göttingen

Prof. Dr. Claus Ropers, IV. Physikalisches Institut
Georg-August-Universität Göttingen

Prof. Dr. Carsten Damm, Institut für Informatik
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 26. Mai 2014

Table of Contents

List of Figures vii

Abbreviations xi

Abstract xiii

Introduction xv

1 Experimental **1**

1.1 IR difference spectra – general considerations 1

1.2 Optical setup 3

1.2.1 Chirped pulse amplification 3

1.2.2 Optical parametric amplifier (two stages) 8

1.2.3 Difference IR spectrometer 9

1.2.4 Third harmonic generation 11

1.2.5 Optical parametric amplifier (three stages) 12

1.2.6 Frequency doubling and quadrupling 14

1.2.7 Electronic synchronization 14

1.2.8 Sample cell 15

1.3 Data processing 16

2 S_2 state photochemistry of diphenylcyclopropenone **21**

2.1 Previous studies 22

2.2 Experimental results 24

2.2.1 UV spectra 24

2.2.2 Transient IR spectra of diphenylacetylene 25
2.2.3 Transient IR spectra of diphenylcyclopropenone 27
2.3 Discussion 32

3 Intramolecular vibrational energy transport 35
3.1 Literature review 35
3.1.1 Depositing vibrational energy in molecules 38
3.1.2 Monitoring the progress of energy transport 42
3.1.3 Propagation of energy along molecular structures 44
3.2 Stationary IR spectra 46
3.2.1 *N*-(Azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides 46
3.2.2 *N*-(Oxo-alkyl)-2-(1-azulenyl)-acetamides 48
3.3 Investigation via UV pump IR probe spectroscopy 51
3.3.1 Effect of vibrational excitation on IR spectra 51
3.3.2 The azulene ring distortion mode 55
3.3.3 The amide I mode 61
3.3.4 The azide asymmetric stretching mode and the carbonyl mode . 65
3.4 A simple model of energy transfer 74
3.4.1 Application 76
3.4.2 Discussion 80
3.5 Towards a mode-resolved picture 86
3.5.1 IR spectra at elevated temperatures 88
3.5.2 Canonical and microcanonical ensembles of oscillators 93
3.5.3 Computational procedure 96
3.5.4 Mode delocalization 97
3.5.5 Anharmonic constants 98
3.5.6 Simulated spectra 100
3.6 Conclusions and Outlook 103

4 Interactions in weakly bound noble gas–halide clusters **107**

4.1 Model of the potential energy 107

4.1.1 Potential energy of neutral clusters 109

4.1.2 Potential energy of anionic clusters 114

4.2 Comparison of model and experimental results 119

4.3 Conclusion 126

Appendices **149**

A Material and equipment used **151**

B Geometry used for the computation of anharmonicity constants **153**

C Potential parameters for noble gas–halide clusters **155**

D Computer Programs **161**

D.1 Data processing 161

D.2 Computation of anharmonicity constants 473

D.3 Halide–noble gas cluster energies 481

List of Figures

1.1 Illustration of difference IR spectroscopy. 2

1.2 Schematic of the optical setup. 4

1.3 Schematic of the chirped pulse amplification unit 6

1.4 Schematic of the two-stage optical parametric amplifier 8

1.5 Schematic of the difference IR spectrometer 10

1.6 Schematic of the third harmonic generation unit 11

1.7 Schematic of the three-stage optical parametric amplifier 13

1.8 Schematic view of the sample cell. 15

1.9 Data processing exemplified. 16

1.10 Three-dimensional representation of spectral data. 18

2.1 Suggested reaction pathways for the decarbonylation of diphenylcyclopropenone. 21

2.2 UV spectra of diphenylcyclopropenone and diphenylacetylene in acetonitrile. 25

2.3 Transient IR spectra of diphenylacetylene with small and large delay, respectively, after excitation at 267 nm and stationary IR spectrum, all measured in CD_3CN. 26

2.4 Transient IR spectra of diphenylcyclopropenone with small and large delay, respectively, after excitation at 267 nm and stationary IR spectrum, all measured in CD_3CN. 28

2.5 Transient IR spectra of the 1498 cm^{-1} absorption of diphenylacetylene. . 30

2.6 Transient spectra of diphenylcyclopropenone and diphenylacetylene after excitation at various different wavelengths. 31

2.7 Transient IR spectra of the carbonyl absorption of diphenylcyclopropenone. 32

3.1 Idealized comparison between diffusive and ballistic vibrational energy transport. 36

3.2 Selected systems used by the Rubtsov group. 38

3.3 Systems used for experimental studies by the Schwarzer group. 39

3.4 Peptide labeling with an azulene group for excitation and an azide-bearing amino acid as performed by the Bredenbeck group. 40

3.5 Systems used by the Dlott group. 40

3.6 An example system used by the Hamm group. 41

3.7 Stationary IR spectra of *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, normalized to the azulene absorption band at about 1580 cm^{-1}. 47

3.8 Stationary IR spectra of *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides, normalized to the azulene absorption band at about 1580 cm^{-1}. 49

3.9 Visualization of the modes observed using the geometry used for calculating anharmonicity constants. 52

3.10 Transient difference spectra of the azulene ring distortion absorption. . . 56

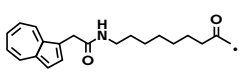
3.11 Kinetic traces of the azulene ring distortion mode. 58

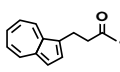
3.12 Comparison of the $t_{1/2}$ value of the azulene distortion mode. 60

3.13 Transient difference spectra of the amide I absorption. 62

3.14 Kinetic traces of the amide I mode. 64

3.15 Transient difference spectra of the azide absorption. 66

3.16 Transient difference spectra of  68

3.17 Transient difference spectra of  69

3.18 Kinetic traces of the carbonyl and asymmetric azide mode. 71

3.19 Comparison of the t_{max} of the azide asymmetric stretching – and carbonyl modes. 72

3.20 Maximum sensor mode signals relative to the maximum amide I mode signal. 74

3.21 Application of the Hamm group's[1] diffusive master equation model. . 78

3.22 Dependence of the arrival time t_{max} on chain length obtained with the fit parameters of the master equation model. 82

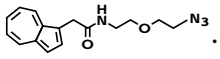
3.23 Dependence of the signal intensity on chain length obtained with the fit parameters of the master equation model. 84

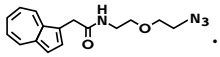
3.24 Temperature-dependent FTIR spectra recorded from potassium bromide pellets. 87

3.25 Original[2] and recalculated spectra and difference spectra of benzene at selected temperatures. 93

3.26 Distribution of total internal energy E in azulene for thermal ensembles and after laser excitation. 94

3.27 Energy distribution in a single oscillator of the azulene molecule. 95

3.28 Participation ratios p_k of the normal modes of  98

3.29 Anharmonic constants x_{ij} for selected modes j of  99

3.30 Difference spectra of key absorption bands calculated based on anharmonic constants x_{ij}. 103

4.1 Schematic representation of the energy levels of a halide–noble gas system, adapted from [3]. 108

4.2 Lengths and angles for the triple dipole potential. 110

4.3 Comparison of experimental and model results for the difference in solvation energy ΔSE for various halide–noble gas systems. 121

4.4 Ratio of induction nonadditivity to the sum of binary potential contributions between the halide anion and all noble gas atoms as a function of cluster size. 122

4.5 Halide–noble gas distances obtained from the model calculations for the respective anionic clusters. 124

4.6 Contributions to the total cluster solvation energy (SE) for the Xe_nBr system (data for the neutral system is for the X state; the unmodified potential was used for the anion). 125

D.1 Screenshot of the data processing program. 162

Abbreviations

2DIR	two-dimensional infrared spectroscopy
BBO	beta barium borate
CPA	chirped pulse amplification
DFM	difference frequency mixing
EA	electron affinity
IR	infrared
IVR	intramolecular vibrational energy redistribution
MCT	mercury cadmium telluride
MD	molecular dynamics
OPA	optical parametric amplifier
PEG	polyethylene glycol
PTFE	polytetrafluoroethylene
THG	third harmonic generation
Ti:Sa	titanium sapphire
VDE	vertical detachment energy
VET	vibrational energy transfer
YLF	yttrium lithium fluoride

Abstract

In this work, three major topics are investigated. The decarbonylation of diphenylcyclopropenone from its second excited electronic state forms the first part. This reaction was investigated using pump laser pulses of several different wavelengths (267 nm and 295–340 nm). Earlier reports[4] that excited state diphenylacetylene is generated as a product are dismissed on three grounds. First, the intensity of the S_1 state absorption of diphenylacetylene at 1553 cm^{-1} after decarbonylation of diphenylcyclopropenone was found to be much too weak. Second, ground state diphenylacetylene could be observed almost immediately within few picoseconds after the reaction had been triggered. Third, the pump wavelength limit of the appearance of the S_1 state absorption of diphenylacetylene was very similar in both the direct excitation of diphenylacetylene and photo-decarbonylation of diphenylcyclopropenone. The alternative hypothesis of internal conversion followed by a hot ground state reaction[5] could not be substantiated. Hence it is concluded that this reaction proceeds non-adiabatically to the electronic ground state of the product. These findings have been corroborated by model calculations and observations of the visible to near-UV transient spectra reported elsewhere[6].

In the main part, the investigation of IVR in several azulenyl-acetamides with aliphatic side chains of different lengths is reported. Three marker bands were monitored to assess the progress of intramolecular vibrational energy transport (IVR) after excitation of the azulene moiety to its S_1 state at 610 nm and the well-known subsequent internal conversion: an azulene ring distortion mode, the amide I mode of the acetamide, and a characteristic mode of a group installed at the opposite end of the chain, i.e. either an asymmetric azide stretching mode or a carbonyl mode. The side chains themselves consisted of methylene groups or ethylene glycol oligoethers. The velocity of energy loss from the azulene group – in agreement with previous research[7] – could be confirmed to saturate with increasing chain length. Energy transport was found to occur fast, with transport times approximately proportional to chain length,

as reported earlier for similar systems[8], and hence concluded to be ballistic in nature. Transport efficiency, on the other hand, was found to decay greatly with chain length. Finally, the amide group presented a suitable reporter for intermediate steps of IVR, exhibiting a distinct spectral response during the progress of energy redistribution. Efforts to shed light on the underlying cause of this phenomenon as well as on the spectral signatures of the other observed marker bands through constants of anharmonicity did not yield conclusive results.

The setup used for both of these experimental works consisted of a transient difference IR spectrometer using laser pulses of roughly 100 fs width, capable of monitoring the range from 1250 to 2400 cm^{-1}.

The third part is devoted to weakly bound halide–noble gas clusters and motivated by an earlier experimental work[9] with a special focus on contributions to the potential energy which are not additive in a pairwise fashion. Two improvements are proposed to a description of those systems put forth by Yourshaw and coworkers[3]: A linear algebraic calculation of non-additive effects in electrostatic induction and a concise analytic solution to the $|jm_j\rangle$-Hamiltonian governing the interaction of a ^2P atom (halide) with a number of closed shell atoms (noble gas). Subsequently, calculations of the electron affinities of the species investigated experimentally[9] are presented, covering a much greater number of systems than previously discussed[3,10,11]. While the theoretical values are qualitatively in agreement with most of the experimental data, quantitative agreement appears to be hampered by the imprecision of the binary potentials used. In particular, inaccuracies in equilibrium distances appear to be amplified by non-additive contributions to the potential energy. For fluorine-containing clusters, except those with argon, the description is dissatisfying even at a qualitative level.

Introduction

Chemical reactions as well as many biophysical processes are driven and determined by the redistribution of vibrational energy. This is particularly true, for instance, in the case of dissociation reactions, where vibrational energy has to be localized in or near the bond to be ruptured. A detailed understanding of the processes involved in this redistribution is therefore essential for a precise description of the dynamics of chemical reactions.

Conceptually, two aspects of energy transfer can be discerned: intramolecular vibrational energy redistribution (IVR) within a molecule and vibrational energy transfer (VET) between different molecules, e.g. particularly between a solute molecule and the surrounding solvent. VET is important for chemical reactions as it can both undermine them by reducing an internal energy of a potential reactant via energy release to the solvent and promote them by introducing more energy into the molecule. Generally, VET is connected most closely to the excitation of a reactant by heating and the description usually rests on collision theory of some kind.

Here, the focus is on intramolecular processes. These are of particular interest in phenomena involving optical excitation, since energy is deposited directly into the molecule of interest. Applications range from energy transport in proteins[12] to the control of chemical reactions[13]. In larger molecules, such as those used for this work, a state-resolved treatment of IVR is infeasible, and the energy redistribution process is generally more statistical in nature. To some extent, this allows for the application of classical concepts of heat conduction. Consequently, much attention has been devoted – and will be devoted here – to the measurement of the speed of energy transfer over a certain distance. Naturally, this is also the most interesting property for the description of any phenomena in which IVR plays an essential role, such as chemical reactions.

Transient IR spectroscopy is an appropriate experimental tool for such investigations. The high time resolution possible due to the advent of ultra short pulsed laser light sources is imperative for fast processes, such as IVR, especially in solution. IR

spectroscopy, on the other hand, offers the most direct reporting on vibrational energy distribution as it probes the oscillators themselves directly.

This work is organized as follows: The first chapter serves as description of the experimental apparatus and procedures used for the investigations presented in the following chapters. Due to its complexity, the optical setup is most prominently featured at this point.

As a demonstration of the capabilities of transient IR spectroscopy as a tool for identifying substances and electronic states, the second chapter is devoted to recent research on the photochemistry of diphenylcyclopropenone. Several pathways have been suggested for its dissociation from its second excited electronic state – most notably an adiabatic process generating the product, diphenylacetylene, in its second excited electronic state[4,14]. Due to the overlapping electronic spectra of both substances, this latter finding is somewhat questionable. The research conducted covers both the dynamics of the excited electronic states of diphenylacetylene as well as the reaction dynamics of diphenylcyclopropenone over a wide range of excitation energies and thus aims at resolving the question whether or not photo-decarbonylation of diphenylcyclopropenone from its second excited electronic state is an adiabatic process.

The third chapter is then entirely devoted to the phenomenon of IVR in azulene derivatives. After a review of previous efforts, especially during the last one and a half decades, a detailed analysis follows of the spectral evolution of several marker bands in a number of azulene derivatives. The chemical groups associated with these marker bands were carefully selected due to their spectroscopic properties and inserted into the investigated molecules in such a fashion as to elucidate the energy transport along chains of methylene groups or ethylene glycol ethers of various lengths. The general characteristics of the energy transport, such as velocity and efficiency, are then compared with other works[7,8], including a diffusion-like model proposed by the Hamm group[1]. Finally, an attempt will be undertaken to understand the nature of the spectroscopic observations and to unravel the underlying energy distribution at the level of anharmonicity constants.

The fourth chapter deals with an effort of describing the interactions in weakly-bound clusters of halide – and noble gas atoms. It is motivated by and complementary to an earlier experimental work[9] and rests on a model proposed by Neumark and Yourshaw[3]. For two of those interactions, improvements to their computational procedures will be suggested, including the analytic solution of the $|jm_j\rangle$-Hamiltonian of the ^2P state. Following these theoretical considerations, results for a large number of

clusters will be discussed with respect to necessary improvements in the model and the interaction potentials.

Chapter 1

Experimental

This chapter serves for describing the experimental apparatus and procedures used. After a brief description of difference (IR) spectroscopy, the optical setup including its electronic synchronization and sample handling will be laid out. Then the general routine of processing raw data will be explained.

1.1 IR difference spectra – general considerations

Difference spectra, i.e. the difference between a spectrum recorded before and after introducing a certain perturbation – the pump pulse in the experiments described here –, are a useful tool for highlighting changes caused by said perturbation. A very simplified illustration is given in Figure 1.1. With the conventions used here (see below), wherever original constituents of the sample (solvent and solute) and those created as a consequence of the perturbation (e.g. fragments of a photolysis reaction or merely vibrationally excited species) differ in optical density, a negative value denotes a stronger absorption in the spectrum of the original sample, whereas a positive value indicates that the perturbed sample causes a more intense absorption. Stimulated emission of vibronically excited species produced by the perturbation can also lead to a negative signal, which often diminishes absorption of the same species.

Note that at nominally negative time delays of small magnitude – i.e. when the pump pulse trails the proble pulse – the signal is caused by perturbed free induction decay.[15] That is, the field emitted as a consequence of the decay of vibrational excitation induced by the probe pulse is perturbed by the pump pulse.

It should be noted that the depicted spectra are not the actual source of the differ-

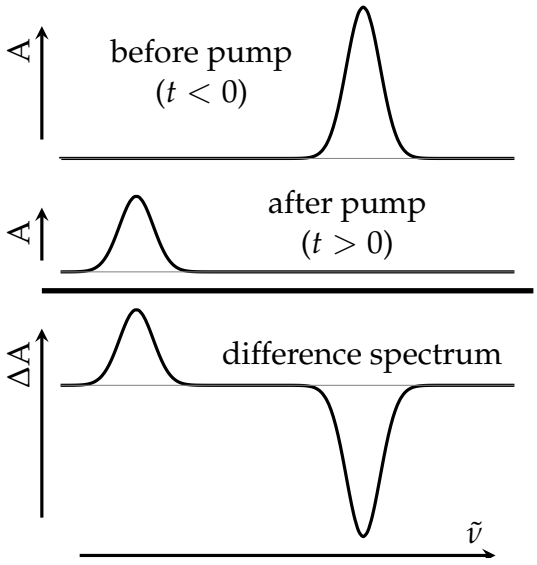


Figure 1.1: Illustration of difference IR spectroscopy.

ence spectrum as recorded by the spectrometer described in Figure 1.2.3. Rather, both the intensity of a reference pulse I_0, passing through the sample before the pump pulse, and the intensity actual probe pulse I were recorded. Since the reference and probe beams are not identical due to imperfections in the beam splitter and different optical paths traveled by either one, pairs of probe and reference pulses were also recorded while blocking the pump beam, as described in Figure 1.2.4. Denoting the intensities of these pulses, where no pump pulse perturbed the sample, by the superscript $\circ$ and using the Beer-Lambert law, the difference in absorbance ΔA – or synonymously the difference in optical density ΔOD is

$$\Delta\mathrm{A} \equiv \Delta\mathrm{OD} = \lg \frac{I_0}{I} - \lg \frac{I_0^\circ}{I^\circ} \tag{1.1}$$

where lg is the decadic logarithm. Here, "mOD" is used to indicate that the value has been multiplied by 1000; i.e. 1 mOD means $\Delta\mathrm{A} = 0.001$.

The principal causes of spectral changes due to the perturbing pump pulse with which this work is concerned are electronic excitation, chemical reaction and vibrational excitation. Other causes include, for instance, changes in the interaction with the solvent. Electronic excitation would mostly be expected to weaken chemical bonds and thus decrease vibrational frequencies. However, in many cases, as in the case of diphenylacetylene discussed here, it leads to a fundamentally different spectrum as it also changes transition intensities and even the structural parameters (bond lengths and angles) of the sample molecule, which in turn leads to completely different nor-

mal modes. In short, it is comparable to chemical reaction as a cause of spectral change, where the spectra of reactant(s) and product(s) often differ greatly. Vibrational excitation, finally, leads to much more gradual changes of the spectrum, where – in solution – individual vibrational states are discernible only for very small molecules.

1.2 Optical setup

The optical setup used for the experiments discussed in this work has been described, although sometimes in slightly modified versions, in many other works[16–20]. Figure 1.2 shows the complete setup. To ease the discussion, the numbering of optical elements roughly follows the beam propagation. Suppliers of all major components are listed in Appendix A.

The apparatus comprised a unit for chirped pulse amplification (CPA), which amplified 800 nm pulses provided by a commercial Coherent Vitesse system, Ti:Sa laser head pumped by a Nd:YVO_4 laser Part of the amplified pulses' energy was fed into a two-staged optical parametric amplifier (OPA), whose output was in turn used to operate a diffractive difference IR spectrometer. The remaining energy of the pulses from the CPA was passed on to either a third harmonic generation (THG) unit to obtain 267 nm pulses to trigger photochemical reactions, or to a second, three-staged OPA. This second OPA was used to produce either 610 nm pulses for azulene S_1 state excitation or tunable pulses in the 295–340 nm range for electronic excitation of diphenylacetylene and diphenylcyclopropenone, by either doubling or quadrupling, respectively, the signal output of the OPA.

Individual units of the optical setup were mounted on aluminium bread boards, indicated as rectangles in Figure 1.2. Additionally, these bread boards, except in the case of the three-stage OPA, were equipped with perimeter walls of anodized aluminum sheets and clear plastic covers to prevent the accumulation of dust. Compartments through which IR radiation was guided were additionally purged with dry nitrogen (as indicated in Figure 1.2) to reduce absorption by water vapor and carbon dioxide from the ambient air.

1.2.1 Chirped pulse amplification

An enlarged sketch of the chirped pulse amplification unit is given in Figure 1.3. Seed pulses for the optical system were obtained by redirecting 50% of the output power

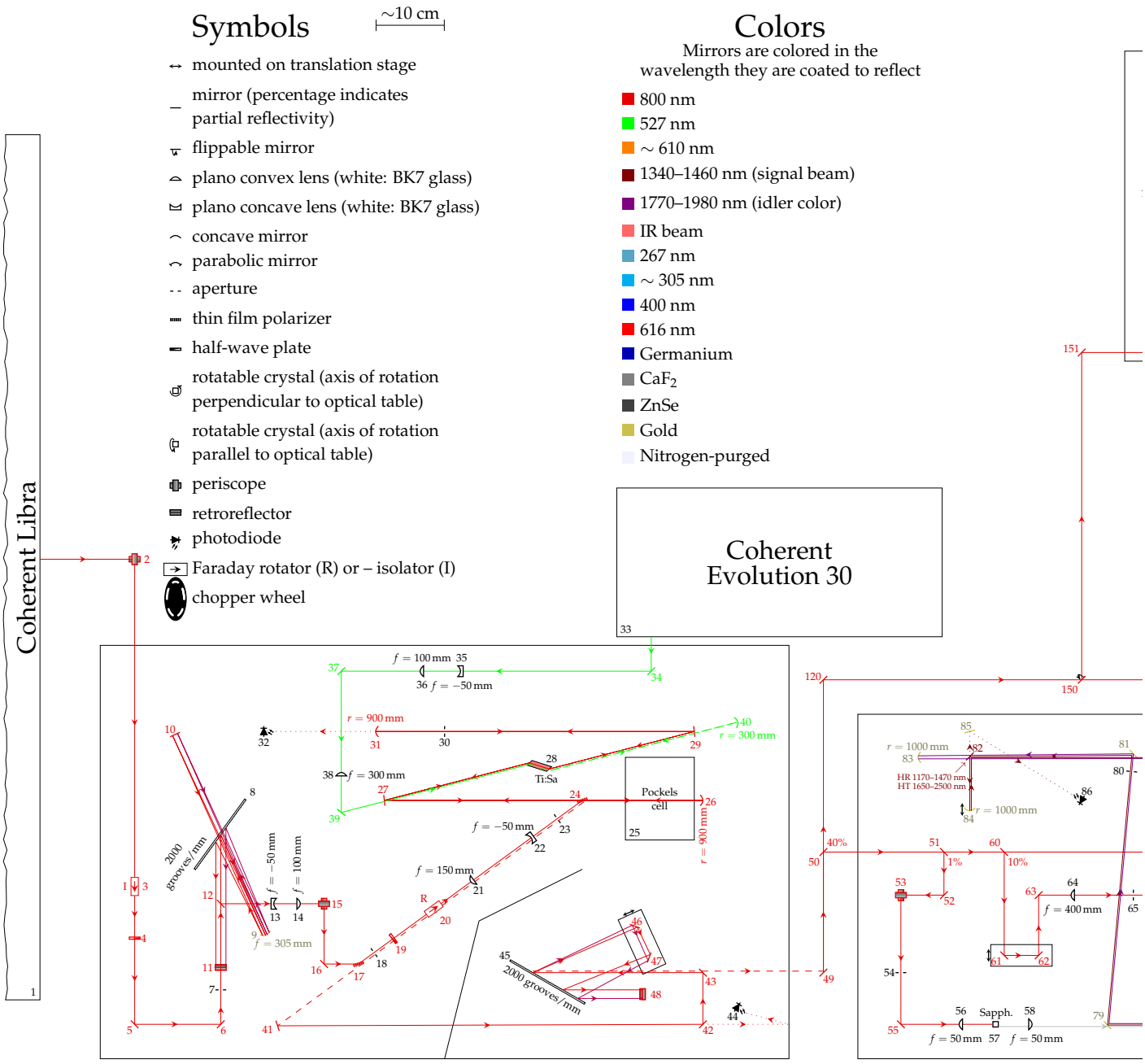
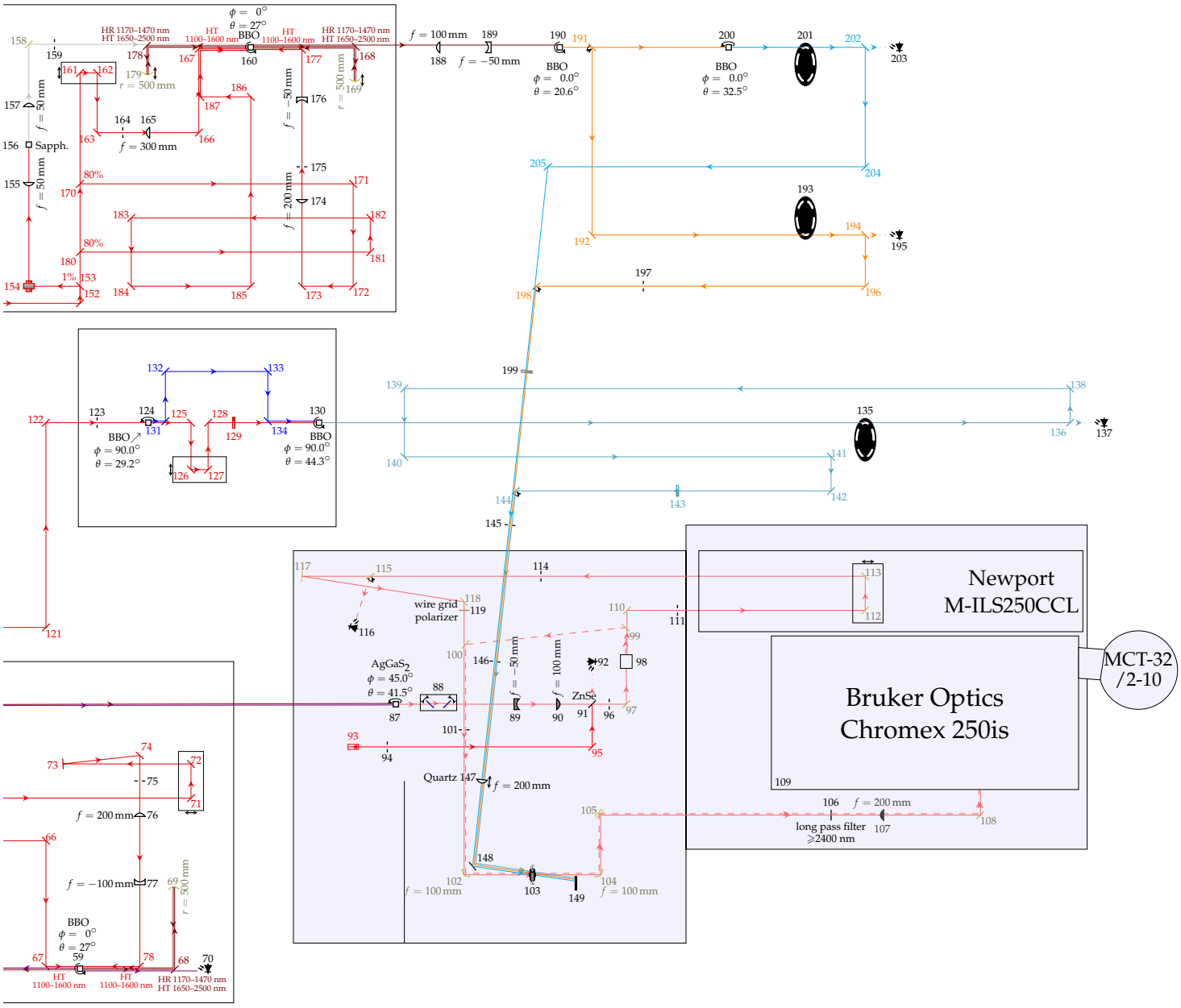


Figure 1.2: Schematic of the optical setup.


Newport
M-ILS250CCL
MCT-32
/2-10
Bruker Optics
Chromex 250is
wire grid
polarizer
AgGaS2
ϕ = 45.0°
θ = 41.5°
ZnSe
Quartz 147
f = 200 mm
long pass filter
⩾2400 nm
BBO
ϕ = 0°
θ = 27°
BBO
ϕ = 90.0°
θ = 29.2°
BBO
ϕ = 90.0°
θ = 44.3°
BBO
ϕ = 0.0°
θ = 20.6°
BBO
ϕ = 0.0°
θ = 32.5°
Sapph.
HR 1170–1470 nm
HT 1650–2500 nm
HT
1100–1600 nm
f = 100 mm
f = −50 mm
f = 50 mm
f = −50 mm
f = 300 mm
f = 200 mm
f = 100 mm
f = −100 mm
r = 500 mm
80%
1%

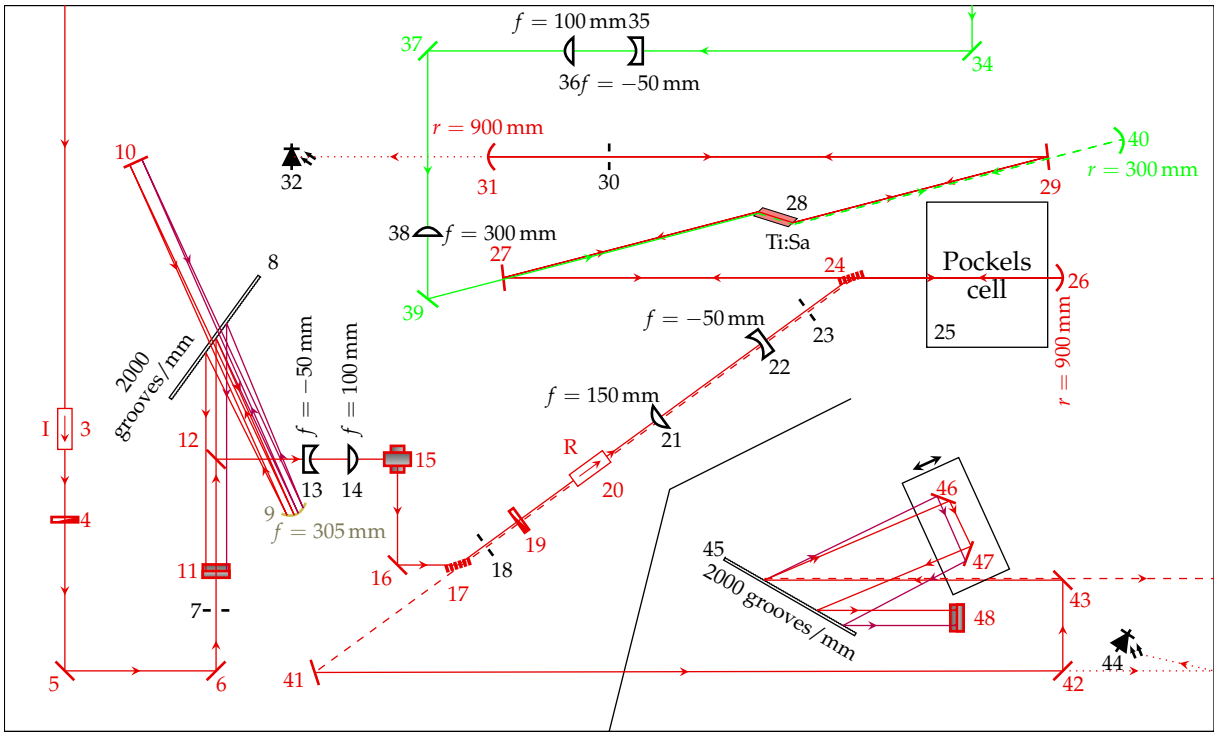


Figure 1.3: Schematic of the chirped pulse amplification unit

of a Coherent Vitesse system (originally 300 mW). The Vitesse system was part of a Coherent Libra system, for which the remaining power was employed, and generated pulses of an energy of 3.75 nJ with a central wavelength of 800 nm, a width of 100 fs, and a spectral width of 175 cm^{-1} at a repetition rate of 80 MHz. These were passed via periscope 2 for beam height adjustment and Faraday isolator 3 to protect the Vitesse system. Subsequently, p-polarization was restored by half-wave plate 4, before mirrors 5 and 6 guided the beam through alignment aperture 7 into the stretcher unit.

In the stretcher unit, the beam was diffracted by grating 8, reflected back onto the grating via concave mirror 9, then flat mirror 10, and again 9 to create a path difference and thus a temporal stretch (positive chirp), between the red and blue edges of the pulses. Concave mirror 9 acts as the reflective equivalent of multiple convex lenses that would be required in an unfolded setup of refractive elements. Retroreflector 11 – a set of two mirrors mounted one upon the other and at a relative angle of ninety degrees – was used to offset the height of the beams as it returned on the same path via grating 8 and mirrors 9 and 10 to arrive at mirror 12 slightly lower than the incoming beam. In Figure 1.2, only one direction is shown for clarity and the red and blue edges of the beam are indicated.

The beam was subsequently widened by a Galilean telescope (lenses 13 and 14)

and its polarization changed to s-polarization by periscope 15. This allowed, after reflection from mirror 16, for thin film polarizers 17 and 24 to guide the beam into the amplifier cavity. Half-wave plate 19 was set to compensate the polarization tilt of 45° induced by Faraday rotator 20 on the way into the cavity. The telescope consisting of lenses 21 and 22 was used to change the width of the beam and divergence to match the configuration of the cavity, where it was focused on the amplifying medium.

The amplifier cavity itself was comprised of Pockels cell 25, terminal concave mirrors 26 and 31, folding mirrors 27 and 29, as well as the amplifying medium Ti:Sa crystal 28, the latter cut to Brewster's angle and cooled to 19 °C. Photodiode 32 was used to monitor the amplification build-up by minuscule light transmitted through terminal mirror 31. Energy for the amplification was provided by Coherent Evolution laser 33, an Nd:YLF laser head operating at a repetition rate of 1 kHz, a pulse width of 300–400 ns, and a central wavelength of 527 nm, whose beam was collimated by lenses 35 and 36 and then focused by lens 38. In order to more completely utilize the pump energy, unabsorbed pump radiation was reflected and focused back into the Ti:Sa crystal by concave mirror 40.

Pockels cell 25 was configured such that it acted as a quarter-wave plate, producing circularly polarized light upon a single passage of the previously s-polarized light and p-polarized light after a second passage. After this flip in polarization, a pulse would then not be reflected by thin film polarizer 24. Upon application of a voltage of approximately 3.1 kV, the characteristics of Pockels cell 25 could be changed into those of an effective half-wave plate, thus leaving the polarization of the p-polarized pulse in the cavity unchanged The pulse would subsequently remain in the cavity for about thirty cycles. Application of a second, opposing voltage restored the original properties of Pockels cell 25, to allow the pulse, s-polarized after two passages, to exit the cavity via thin film polarizer 24.

Faraday rotator 20 and half-wave plate 19, passed in opposite direction as before, then led to a rotation of the polarization of the exiting beam to p-polarization, thus allowing it to pass through thin film polarizer 17 and continue on into the stretcher unit via mirrors 41 through 43. At this point, amplification to a pulse energy of about 1 mJ had been achieved, while the repetition rate had been lowered to the 1 kHz rate provided by the Evolution laser head. Photodiode 44 was used to verify the characteristics of the pulse upon leaving the cavity by using light scattered off from the casing of the RGA unit.

After refraction by grating 45, which was of identical characteristics as grating 8,

the positive chirp was removed by passing the beam via the perpendicularly aligned mirrors 46 and 47 back to grating 45 and returning it on the same path, but at slightly greater height, after reflection by retroreflector 48. Mirrors 46 and 47 were mounted on a translation stage to tune the performance of the compressor unit (see below). The compressed pulse of about 700–750 µJ then exited the compressor unit slightly above mirror 43.

1.2.2 Optical parametric amplifier (two stages)

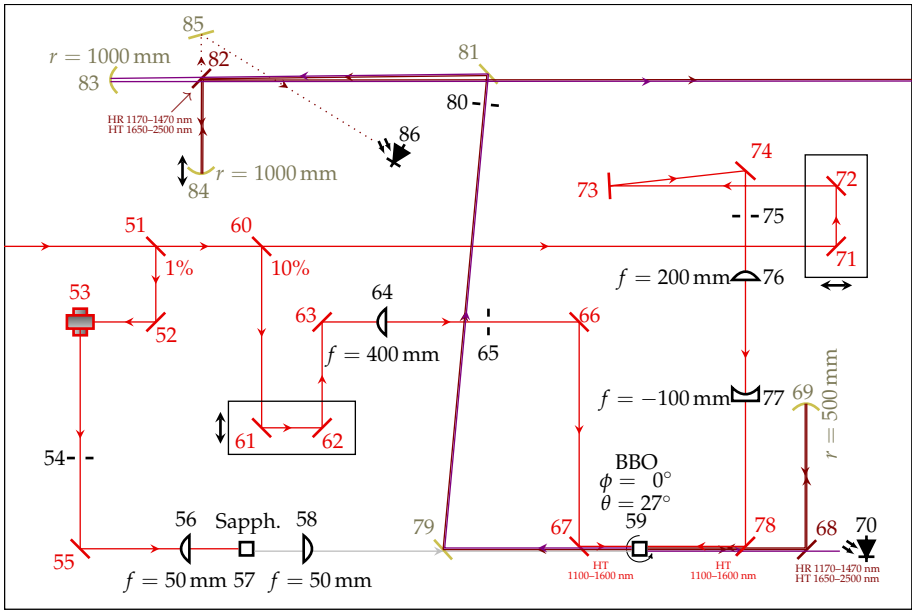


Figure 1.4: Schematic of the two-stage optical parametric amplifier

Split off by partially reflective mirror 50, two fifths of the amplified pulse were employed to generate mid-IR radiation for detection purposes. The optical parametric amplifier (OPA, see Figure 1.4) used to this end followed a design by Peter Hamm[21,22] and it was also used to first generate two near-IR photons from one 800 nm photon and then to achieve the production of light of even longer wavelength by difference frequency mixing (DFM).

Initially, a small amount of mid-IR radiation was generated from a seeding white light continuum. The latter was created by using a very small portion of about a single percent (mirror 51) of the incoming laser power and focusing it into sapphire disc 57 with lens 56 after having flipped its polarization to s-polarization by periscope 53. Aperture 54 could be used to attenuate the beam up to the point where white light

generation ceased. At this point, the compressor unit could be tuned by adjusting the translational position of mirrors 46 and 47 to restore continuum generation (see above). Lens 58 was set to focus the emerging continuum beam slightly behind beta barium borate (BBO) crystal 59 after it had passed above mirror 79. Amplification was attained by focusing about ten percent of the beam power (mirror 60) into BBO crystal 59 using lens 64. Mirrors 61 and 62 were mounted on a translation stage to adjust the temporal overlap of continuum and pump pulse. Further, aperture 65 was used to carefully attenuate the pump beam.

Dichroic mirrors 67 and 78 were coated to allow for passage of the near-IR radiation, while dichroic mirror 68 was coated to reflect the signal beam, whose wavelength varied usually from 1330 to 1460 nm, while transmitting the idler beam of a complementary wavelength of 2000 to 1770 nm. The signal pulse was then refocused into BBO crystal 59 using concave gold mirror 69 and the idler pulse monitored by photodiode 70.

For the second stage of amplification, the remaining power of the 800 nm beam was used. After retardation by mirrors 71–74, of which mirrors 71 and 72 were again mounted on a translation stage to ease the adjustment of temporal overlap, it was reduced in width by the telescope formed by lenses 76 and 77 to better match the profile of the signal beam. Finally, the same BBO crystal 59 was used for the second amplification, thus attaining pulse energies of about 50 μJ (signal and idler beam combined).

For generation of mid-IR light, both signal and idler beams of the second amplification were passed on by gold mirrors 79 and 81 into a Michelson-like interferometer formed by splitting mirror 82 and concave gold mirrors 83 and 84. Mirror 82 was coated in the same fashion as mirror 68, to split signal and idler pulses. To allow for the fine adjustment of the temporal overlap of signal and idler pulses, concave mirror 84 was situated on a translation stage driven by a piezo electrical actuator. Photodiode 86 was used to guide this adjustment. The final frequency mixing occurred in $AgGaS_2$ crystal 87 to yield a p-polarized beam of IR light.

1.2.3 Difference IR spectrometer

An enlarged sketch of the spectrometer described in this section can be found in Figure 1.5. The emergent mid-IR pulses were attenuated by attenuator 88, which consisted of two germanium discs mounted on synchronized rotation stages to compensate their respective beam displacement. Further, the telescope consisting of CaF_2 lenses 89

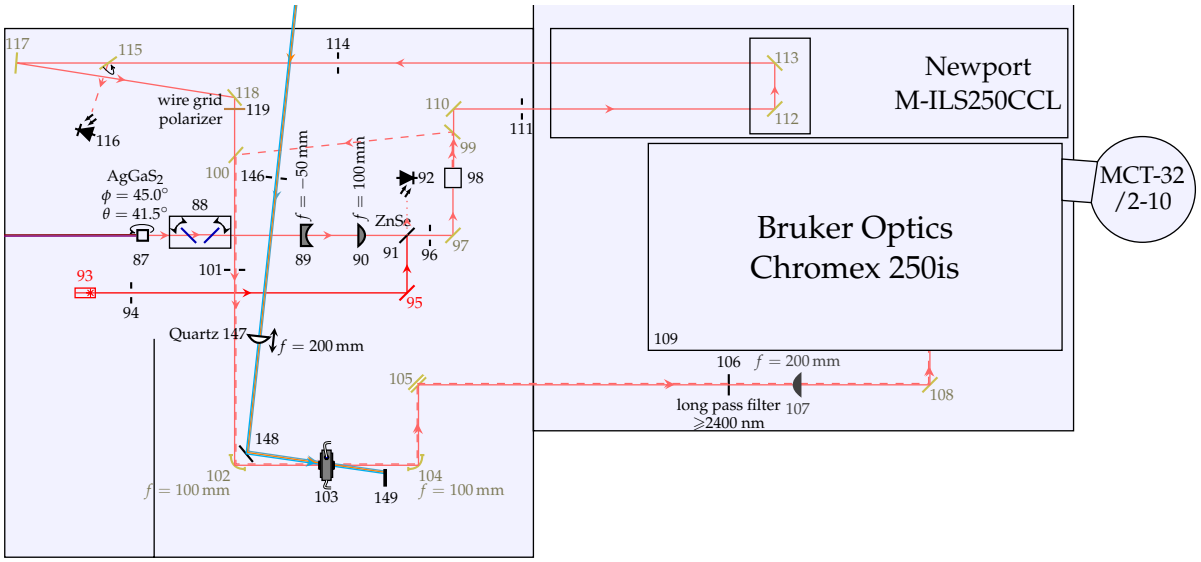


Figure 1.5: Schematic of the difference IR spectrometer

and 90 widened the IR beam and ZnSe disc 91 was used to eliminate remaining signal and idler components as well as traces of visible light (harmonic frequencies of signal and/or idler beam). Photodiode 92 was used to guide the adjustment of the beam. To aid adjustment of the invisible beam after this point, the beam of laser diode 93 (630–690 nm) was superimposed onto the IR beam using ZnSe disc 91 as a mirror and apertures 96 and 111 for alignment.

In order to record a difference IR spectrum, the IR pulse was split using a wedge-shaped BaF_2 disc and a gold mirror in beamsplitter 98, the details of which have been described elsewhere[16]. Different from earlier applications, the beam splitter was used to produce two vertically separated IR beams here. The lower of these beams was directly passed on via mirrors 99 and 100, attenuated by aperture 101 as necessary, and focused into the sample cell 103 by parabolic gold mirror 102. After collimation by parabolic gold mirror 104, the upper mirror of mirror pair 105 was used to guide it into the grating polychromator 109 after undesired radiation had been blocked by long pass filter 106 and the beam had been focused by ZnSe lens 107. This first IR pulse arrived at the sample cell prior to the pump pulse and served as a reference beam, whose intensity shall later be referred to as I_0.

The upper IR beam from beam splitter 98 (henceforth designated “probe beam”) had to travel a considerably longer distance in order to arrive simultaneously with or after the pump pulse. Mirrors 112 and 113 were mounted on a motorized translation stage in order to precisely and automatically adjust its temporal offset versus the pump

pulse. Adjustment of the beam passing over this motorized stage was supported by apertures 111 and 114 as well as photodiode 116. Additionally, the polarization of the probe beam was purified by wire grid polarizer 119. Passing above aperture 101, the probe beam then traveled parallel to the reference beam to parabolic mirror 102, eventually passing through the sample cell in approximately the same spot to arrive at the lower mirror of mirror pair 105 and then continue to the spectrometer. The intensity of this beam shall later be referred to as I.

The nitrogen-cooled mercury cadmium telluride (MCT) detector MCT-32/2-10 used consisted of two rows of 32 detection elements each for simultaneously recording the reference and the probe beam. Each detection element covered about 11.4 nm of the spectrum. The central wavelength of each detection element was calibrated regularly using known line positions of water vapor and carbon dioxide as recorded by an FTIR spectrometer. For all measurements a grating of 150 grooves/mm blazed at 6 µm was used. Integration of the electronic signal was accomplished by an IR-6416 signal integrator.

Parts of the apparatus through which IR beams propagated were set up in compartments flushed with nitrogen, as indicated in Figure 1.2, to reduce the amount of water vapor and carbon dioxide.

1.2.4 Third harmonic generation

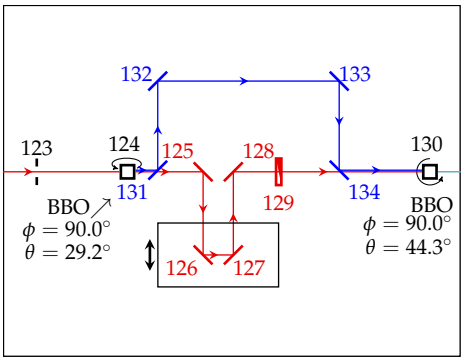


Figure 1.6: Schematic of the third harmonic generation unit

To pump samples at 267 nm, third harmonic generation (see Figure 1.6) was employed using the remaining output of the RGA. First, 400 nm radiation was obtained by type I frequency doubling in BBO crystal 124. The remaining intensity of the fundamental wavelength was then adjusted to s-polarization by half-wave plate 129 to un-

dergo type I frequency mixing with the second harmonic beam (400 nm) in BBO crystal 130. In between, the fundamental and frequency-doubled pulses were separated and mirrors 126 and 127, mounted on a translation stage, used to adjust their temporal overlap. A mismatch in temporal overlap was generally used to attenuate the intensity of the pump beam as the attainable pulse energy of 30 μJ was excessive for the desired use.

Synchronization with the probe pulse was established using the retarding mirrors 136 and 138 through 142 and 144. By means of a chopper wheel 135, only two successive pulses were used to pump the sample, while the next pair of pump pulses were blocked to record a reference spectrum for correction of the imperfections in sample and reference IR beam. Photodiode 137 was used to determine which probe pulses were accompanied by a pump pulse and which were to be used for referencing. Additionally, the polarization of the pump pulse was set to "magic angle" (54.7°) configuration with respect to the probe beam by half-wave plate 143, in order to eliminate the influence of the rotation of the sample molecules on the signal[18]. Quartz lens 147 was used to achieve a focus slightly behind sample cell 103. The pump beam then crossed the probe beam in the sample cell and was finally blocked by beam stopper 149.

1.2.5 Optical parametric amplifier (three stages)

For other pump wavelengths and tunable pump wavelengths, a second OPA (see Figure 1.7) was used, which has been described elsewhere[18], in a slightly modified form. Again one percent of the arriving pulse was split off by mirror 153, elevated and rotated in polarization by ninety degrees by periscope 154, and then focused into sapphire disc 156 by lens 155. The emerging white light continuum was then focused into BBO crystal 160 by lens 157. In the BBO it met about four percent of the original pulse as a first stage of amplification. The latter was synchronized with the white light pulse by mirrors 161 and 162, mounted on a translation stage, and subsequently focused by lens 165. Mirror 178 was arranged such that the white light continuum could pass above it, as was mirror 187 to allow for passage of the first pump beam overhead.

The generated signal beam, selected by the appropriately coated mirror 168, was refocused into the same BBO crystal 160 and superimposed with a second pump beam by concave gold mirror 169. The latter was mounted on a translation stage to facilitate synchronization with the second pump pulse. The second pump beam itself – roughly fifteen percent of the original beam – was split off by beam splitter 170, directed using mirrors 171 through 173, narrowed by the telescope consisting of lenses 174 and 176,

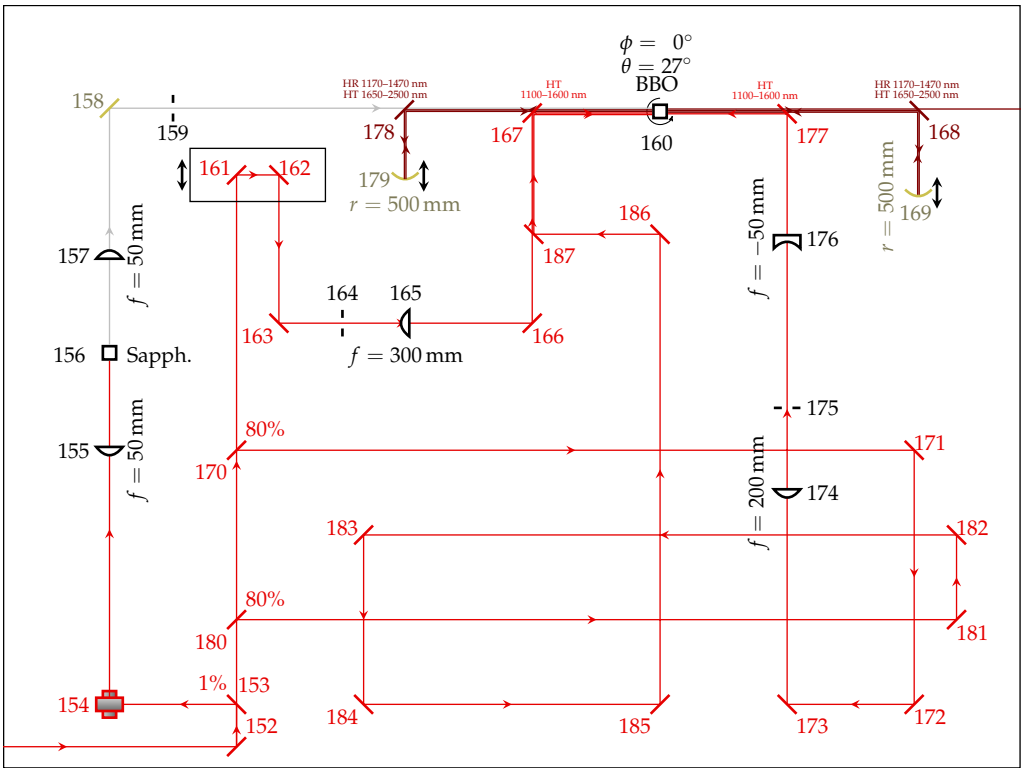


Figure 1.7: Schematic of the three-stage optical parametric amplifier

and finally joined with the signal beam by mirror 177.

Contrary to the two-stage OPA described above, signal and idler beam were not used after the second amplification, but rather treated in the same fashion as in the case of the first stage of amplification, i.e. the signal beam was separated from the idler beam by mirror 178 and refocused by concave gold mirror 179. A third amplification beam of about eighty percent of the original laser power was split off by beam splitter 180 and directed into BBO crystal 160 by mirrors 181 through 187 and 167. As with the second stage of amplification, the concave gold mirror, here 179, was installed on a translation stage for synchronization with the third pump beam.

The signal and idler pulses resulting from the third stage of amplification, of a combined energy of 50–100 µJ – depending on the selected wavelength – were then used for pump pulse generation, after leaving the OPA by passing above mirror 168. It should be mentioned, that mirrors 168, 178, and 187 had indeed the shape of a semicircle in order to allow other beams to pass above them. A detailed description of this arrangement and the adjustment procedure can be found elsewhere[18].

1.2.6 Frequency doubling and quadrupling

Emerging from the three-stage OPA, the beam was narrowed and collimated using lenses 188 and 189 and frequency-doubled using BBO crystal 190. For a desired pump wavelength of around 610 nm, the frequency-doubled pulse of up to 8 μJ of energy could be passed on via mirrors 191, 192, 194, 196, and 198 to then be adjusted to "magic angle" polarization with respect to the probe beam by half-wave plate 199. Again, the blocking of two consecutive pump pulses – out of a group of four – was achieved by means of chopper wheel 193 and recorded by photodiode 195, as described above for third harmonic pump radiation. The width of the pulse obtained before passing through the half-wave plate was confirmed as about 100 fs by means of autocorrelation.

Alternatively, for shorter wavelengths in the range of 295–340 nm, the frequency of the laser pulses could be doubled again by BBO crystal 200 to produce pulses of an energy of up to 2.5 μJ, which were subsequently synchronized with the probe pulses and purified by mirrors 202, 204 and 205. For this option, chopper wheel 201 and photodiode 203 were used to produce reference spectra. Again, half-wave plate 199 established proper "magic angle" polarization relative to the probe pulse.

Naturally, half-wave plate 199 had to be exchanged when changing the pump wavelength. Also, the final mirror 148 required exchanging in such cases and lens 147 was adjusted to compensate for the wavelength dependence of its focal length. Apertures 145 and 146 were used to define the final approach of the pump beam to the sample.

Finally, it should be noted that yet other wavelengths could be produced easily by switching the positions of BBO crystals 190 and 200 and using the idler beam instead of the signal beam.

1.2.7 Electronic synchronization

The clock rate of 1 kHz of the experiment was provided by the SDG Elite control module of the Coherent Libra system, defining the starting point of the electronic synchronization t_0. Further delays were then generated by a delay generator DG535. Amplification was initiated by a suitable control pulse 3872 ns later and terminated by another pulse after another about 250–350 ns, where the latter point was dependent on the performance and settings of the pump laser. Signal integration by the MCT detector lasted for 2614 ns, starting 4044 ns after the initial clock at t_0. Finally, the chopper wheel was

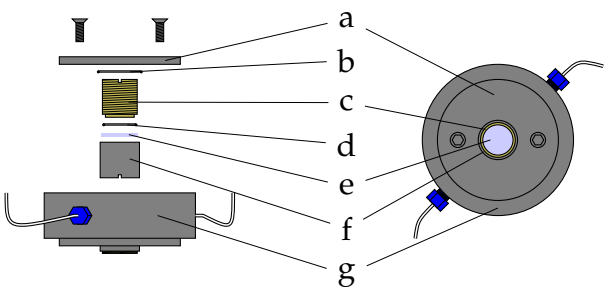


Figure 1.8: Schematic view of the sample cell. Left: top view, right: front view. Not to scale. Colors of materials are: yellow – brass, gray – stainless steel, light blue – calcium fluoride, dark gray – Viton®.

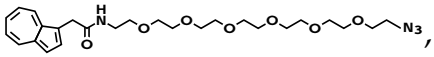
synchronized by a pulse 33 µs after t_0.

The translation stage of mirrors 112 and 113, as well as the spectrometer 109, and the signal integrator were controlled by a home-built software developed in LabView[23].

1.2.8 Sample cell

The sample cell used is depicted schematically in Figure 1.8. Calcium fluoride windows (e) of 1 mm thickness on both sides of the sample chamber were each fixed in a stainless steel cylinder (f) using a threaded brass cylinder (c) and a Viton® O-ring (d) for sealing. This assembly itself was held in place by a stainless steel ring (a) and another Viton® O-ring (b), of which the former was attached to the stainless steel cell body element (g) by two hexagonal socket screws. The optical path length could be varied to some degree. However, it was generally set to the minimum possible value of 0.3 mm as permitted by the shape of the stainless steel cylinders holding the calcium fluoride windows.

Sample solutions were introduced into the cell from the bottom inlet and flowed upwards diagonally. Omnifit Labware PTFE (polytetrafluoroethylene) tubing with an inner diameter of 0.8 mm were used in conjunction with the appropriate Omni-Lok connectors to transport the sample solution between the sample cell and a Micropump GAX21. P8FS.B pump. To remove debris from the sample solution, a Whatman GF 6 filter in an appropriately designed casing filtered the solution as it returned from the cell to the pump.

Sample solutions generally had a volume of about 10 ml and generally contained a few milligrams of the substance to be investigated. In rare cases, such as  , several tens of milligrams were used. Thus, the concentration of the sample solution was generally between 1 and 10 mM.

1.3 Data processing

The data recorded by the experimental setup described in section 1.2 was processed using a home-built software described in section D.1. While details on the processing and evaluation of the data varied, the general and primary protocol of data processing shall be outlined in this section. The steps of this general protocol are illustrated in

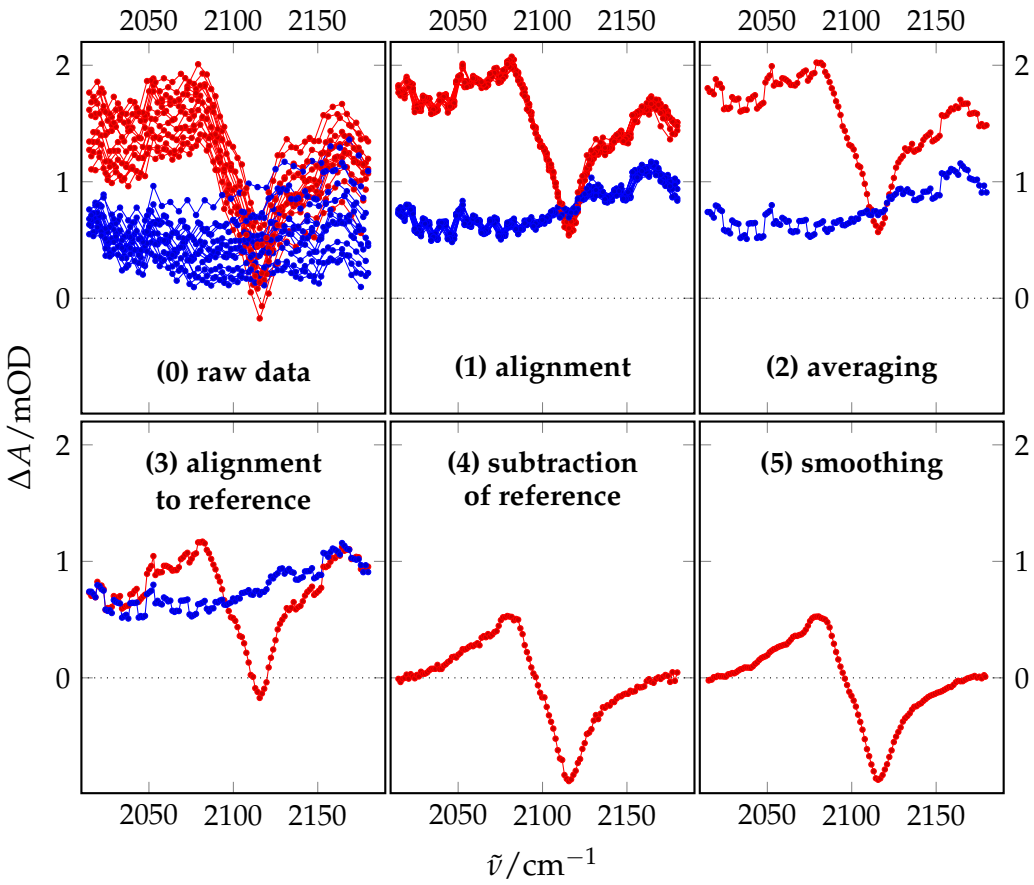


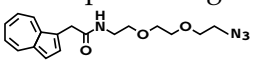
Figure 1.9: Data processing exemplified using a data set of the azido mode of  in CH_2Cl_2 after excitation at 610 nm. —●— is reference data obtained 20 ps before the arrival of the peak of the pump pulse, whereas —●— represents data recorded 8.5 ps after its arrival.

Figure 1.9.

Even after averaging – in this case – 750 individual laser shots and equally many reference shots (see below), a substantial amount of noise, originating mainly from fluctuating laser power, remained, as is visible from the "raw data" (panel 0 in Figure 1.9). Hence, spectra of the same time delay are first brought to mutual alignment (panel 1) and then averaged (panel 2). For automatic alignment, the first spectrum in a series is always used as a base. Alignment itself corresponds to a first-order baseline correction, i.e. a linear function was least-squares fitted to the difference between the spectrum to be aligned and this base and was subsequently subtracted from the spectrum to be aligned.

At this point, the averaged data still lacks an alignment with an actual baseline and exhibits a number of stepwise jumps, which in the given example are most pronounced at the left or "red" edge of the spectrum. These steps are clearly caused by subtle differences in sensitivity or alignment of the individual detection elements as they occur at an interval of four data points. Those four data points originate from the same detection element as the central wavelength of the polychromator was shifted by small ($\approx$3 nm) amounts during the recording and returned to its original setting after three of those steps. This procedure was introduced to improve the spectral resolution of the recorded data. Both of these aspects – the missing baseline and the detector imperfection – can be compensated by using a spectrum recorded well before the arrival of the pump pulse, which – in an ideal experiment – ought to exhibit a zero change in optical density at all wavelengths. Thus, the exemplary spectrum at a delay time of 8.5 ps is aligned to the one recorded at -20 ps delay time (panel 3) and the latter is then subtracted from the former (panel 4).

Optionally, smoothing of the resulting spectrum may be performed (panel 5). In the example given in Figure 1.9, a running average of three adjacent points was computed.

Measuring at various appropriate time delays between pump and probe pulse, it was finally possible to assemble a set of data describing the temporal evolution of the absorption spectrum of a substance after excitation. A full three-dimensional representation of such a set of data is shown in Figure 1.10. Generally, for a number of delay and wavelength settings, the wavelengths were iterated over at one given delay before changing the delay setting – this constituted the "inner loop" of a measurement cycle. At each step of the inner loop, data from between 1500 and 2000 laser shots were collected and averaged, half of which included a pump pulse, while for the other

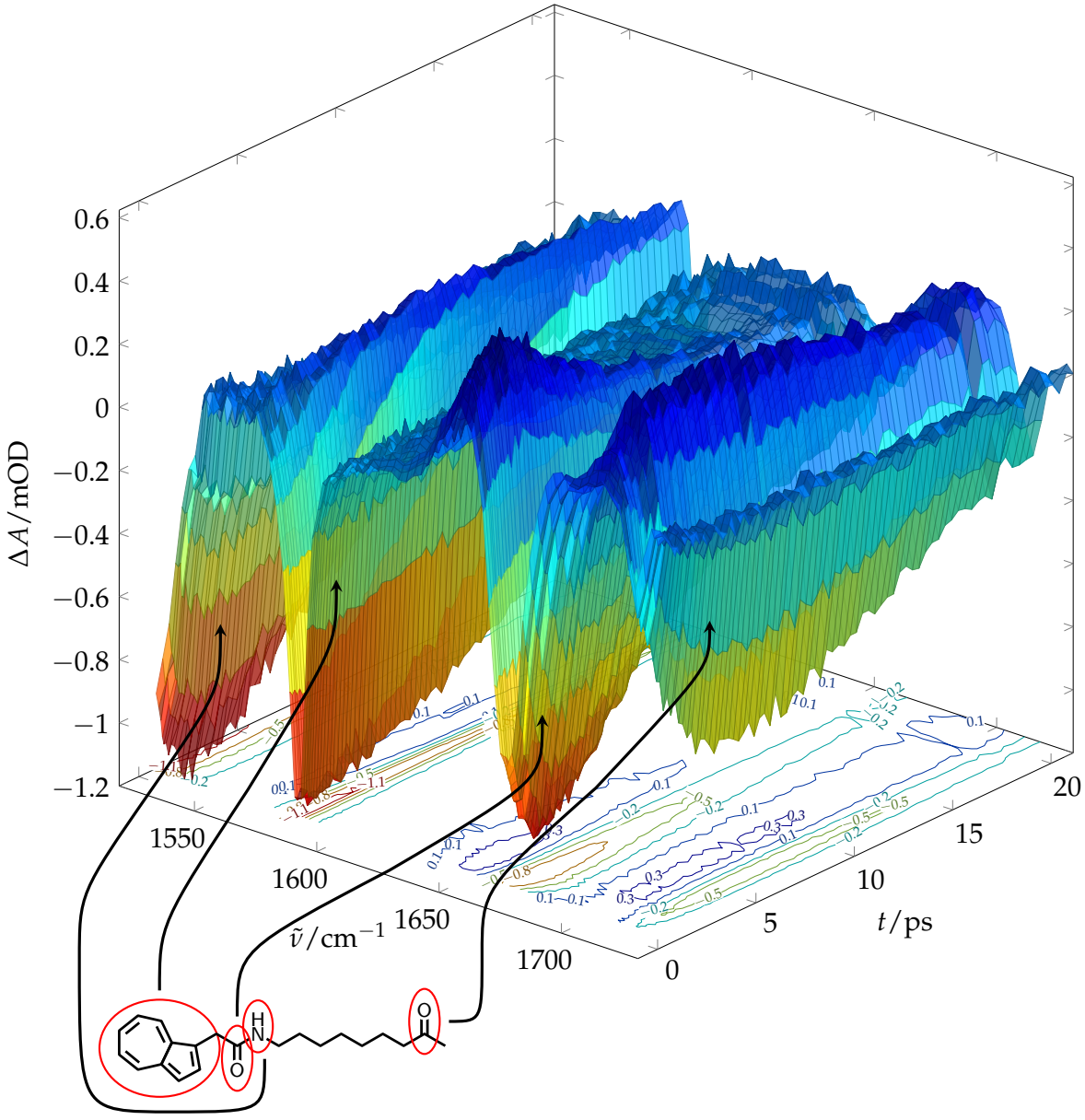
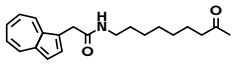


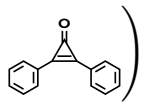
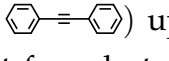
Figure 1.10: Three-dimensional representation of spectral data obtained for  in CH_2Cl_2 after irradiation at 610 nm. Clearly discernible are the carbonyl absorption (centered at 1703 cm^{-1} in the stationary IR spectrum), the amide I mode (1660 cm^{-1}), and a ring distortion mode of the azulene moiety (1576 cm^{-1}). The spectrum of the amide II absorption (N-H wagging, 1539 cm^{-1}) is partially included.

half the pump beam was blocked, as described in Figure 1.2.4. Delay settings, on the other hand were iterated over bidirectionally, i.e. from smallest to largest and back to the smallest delay value, constituting the "outer loop". The outer loop was repeated a number of times, usually four.

Determining the precise delay at which pump and probe pulses "simultaneously" arrive at the sample, i.e. the geometric configuration of the optical apparatus at which their overlap upon reaching the sample is maximized, constitutes a substantial challenge. In this work, coherence artifacts due to the optical Kerr effect[24] were used to determine this zero point of time. It should be noted that the error in that determination is approximately 0.3 ps. This error ultimately has to be considered in any time measured with respect to time-zero (e.g. $t_{1/2}$ in section 3.3.2 or $t_{\max}$ in section 3.3.3 and section 3.3.4). Accordingly, it was added to the respective statistical error (generally comprising a 90% confidence interval) in all data reported. Data points shown in a paler color were excluded from fitting.

Chapter 2

S_2 state photochemistry of diphenylcyclopropenone

Diphenylcyclopropenone  is one of the more intensely investigated cyclopropenones. Besides being of medical interest for the possible treatment of warts[25] and hair loss[26], its photo-decarbonylation is noteworthy as it was found to generate electronically excited diphenylacetylene () upon excitation to its S_2 state[4]. The relative energetic situation of the first few electronic states of both diphenylcy-

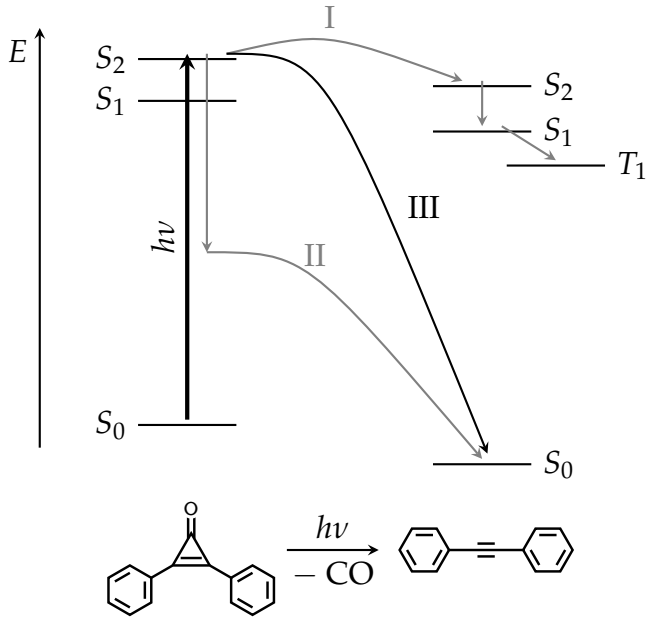


Figure 2.1: Suggested reaction pathways for the decarbonylation of diphenylcyclopropenone.

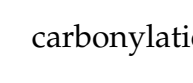
clopropenone and diphenylacetylene is depicted in Figure 2.1 with approximate energies obtained from the respective UV absorption spectra and a reaction enthalpy of $-3500 \pm 1000\,\mathrm{cm}^{-1}$[27]. While photo-decarbonylation generally proceeds with unit quantum yield even from the S_1 state (337 nm in benzene)[28], thermal decarbonylation occurs only after heating to 130–140 °C[29].

Initially, the chance to monitor the relaxation of vibrationally excited carbon monoxide sparked interest in the photodecarbonylation reaction of diphenylcyclopropenone. Being a single oscillator of fairly high frequency, carbon monoxide can reasonably be expected to equilibrate with the solvent bath on a timescale of several nanoseconds (cf. the asymmetric stretching vibration of CO_2 discussed in [30]). However, the IR intensity of this vibration of $65.5 \pm 0.8\,\mathrm{km\,mol}^{-1}$[31] is rather small and it possesses a tendency to exhibit a large linewidth in solution due to almost unhindered rotation[32,33]. In addition, the excess vibrational energy borne by the molecules emerging from the dissociation reaction certainly causes further broadening. The slow relaxation of the vibrational excitation then makes detection even more difficult. Combined together, these effects prevented this endeavor from succeeding. Important insights – not on the vibrational relaxation of carbon monoxide in solution, but on the mechanism of the photodecarbonylation of diphenylcyclopropenone – were gleaned nevertheless.

2.1 Previous studies

Initial research of the photochemistry was conducted by Hirata *et al.*[4,34] and employed a transient absorption setup with a temporal resolution of approximately 20 ps. Excitation was carried out at several wavelengths between 280 and 320 nm while monitoring the spectral range from 390 to 1000 nm in *n*-hexane. Based on the similarity of the spectra obtained for diphenylacetylene after direct excitation and as a product generated by decarbonylation of diphenylcyclopropenone, respectively, and the S_2 state lifetimes of the reaction product at different wavelengths, it was concluded that decarbonylation from the S_2 state of cyclopropenone proceeds adiabatically to the S_2 state of the product (pathway I in Figure 2.1). Furthermore, it was stated that the S_2 state of the product is formed after photo-decarbonylation at excitation wavelengths up to of 320–325 nm, but to an increasingly smaller amount as the excitation wavelength increases. This finding was later supported by Takeuchi and Tahara[35], who used 267 nm excitation and cyclohexane as a solvent while monitoring 390 to 750 nm with a resolution of 650 fs, and separately monitored at 480 nm with 70 fs resolution, although

spectral differences between S_2 diphenylacetylene generated by photo-excitation and as a reaction product were noted.

In a different work, quantum chemical calculations were used to study the decarbonylation of cyclopropenone  , cyclopropenethione  , and their respective mono-fluorinated derivatives in the electronic ground state.[5] It was concluded that the reaction proceeds as an asymmetric detachment via two transition states and one intermediate. The large excitation energies of all these species were interpreted to suggest that the photodecarbonylation probably proceeds on the ground state surface as well (i.e. pathway II in Figure 2.1).

This latter conjecture ought to be viewed critically: First, the argument that large excitation energies prevent the reaction from occurring on the surface of the excited state rests on plausibility. At the same time, from the values given it appears that all energies of transition states on the excited state surface would be less than the calculated excitation energy of the product (diphenylacetylene), which is known to be similar to the excitation energy of the reactant (diphenylcyclopropenone). Second, it is by no means evident that the path of the reaction on the ground state potential energy surface should translate equivalently to the excited state, i.e. that the same transition state geometries are involved if the reaction takes place on the energy surface of the excited state. Third, the authors did not elaborate on how fast or at what geometry a required internal conversion to the ground state would occur. To the contrary, the rate constant obtained from statistical theories was later found to be $(20\,\mathrm{ns})^{-1}$ and thus much slower than all observed values.[6] Finally, this would be in direct opposition to the predissociative character of the reaction asserted earlier[4].

Following up on the aforementioned quantum chemical results, Poloukhtine and Popik found a zwitterionic intermediate, whose lifetime could be extended well into the tens of picoseconds by substitution of the cyclopropenone and the choice of suitable solvents.[14] For bis-*p*-anisylcyclopropenone, it was further concluded that the decarbonylation leads to the product ground state (pathway III in Figure 2.1), while for diphenylcyclopropenone Poloukhtine and Popik state that decarbonylation leads to the S_1 and eventually T_1 states of the product, without explaining their finding in detail.[14]

For the decarbonylation product, diphenylacetylene, photo-excitation does not lead to the adiabatic S_1 state (^1A_u), but populates the S_2 state ($^1B_{1u}$), which is considered to be lower in energy at the ground state equilibrium geometry.[36,37] The lifetime of the S_2 state was reported to be about 8–9 ps at 296 K[34]. An activated transition

with a barrier height of 14.0 kJ/mol (in the condensed phase) then leads to the S_1 state, alongside with radiative transition and non-radiative transitions to the ground state and non-radiative transition to the first triplet state T_1.[36] From the S_1 state, whose equilibrium geometry is believed to be lower in energy than that of the S_2 state, but different from the equilibrium geometry of both the S_2 state and the ground state[36,37], intersystem crossing to the T_1 state ensues with a time constant of about 200 ps, with a variation of few tens of picoseconds depending on the solvent.[34,36]

The ground state of diphenylacetylene is assumed to be of $\mathcal{D}_{2h}$ symmetry, i.e. planar with a linear acetylenic bond[38,39]. On the other hand, its S_1 state was found to have a substantially weakened central C-C bond of double bond order with a vibrational frequency of less than 1600 cm^{-1} and was thus concluded to be of $\mathcal{C}_{2h}$ symmetry, i.e. *trans*-bent.[36,40–42] The S_2 state was then again reported to be linear with a somewhat weakened central C-C bond, but still in the triple bond range[41]. It should be noted, that two other excited singlet states have been predicted theoretically, transition to which is either parity (A_g) or orbital (B_{2u}) forbidden from the ground state.[43,44]

2.2 Experimental results

The apparatus described in section 1.2 was employed to gain a better understanding and to test previous findings on the precise nature of the photodecarbonylation of diphenylcyclopropenone from its S_2 state, with a particular emphasis on the discussion of the three suggested pathways (adiabatic, hot ground state, and diabatic; see Figure 2.1). Additional results, including transient electronic spectra revealing the very initial stages of the reaction, have been published elsewhere.[6]

2.2.1 UV spectra

The UV spectra displayed in Figure 2.2 are of vital importance to understand the difficulties in an unambiguous determination the course of the decarbonylation reaction of diphenylcyclopropenone. The S_1 state absorption of diphenylcyclopropenone begins at approximately 360 nm and remains leveled from 340 nm on. This absorption has been associated with an $n\pi^*$ transition of the carbonyl group.[45]

The S_2 states of diphenylcyclopropenone and diphenylacetylene roughly stretch from 320 to 260 nm and 300 to 240 nm, respectively. Consequently, this poses the challenge of discerning these species when exciting at the popular 266 nm[14,35] or 295 nm[4]

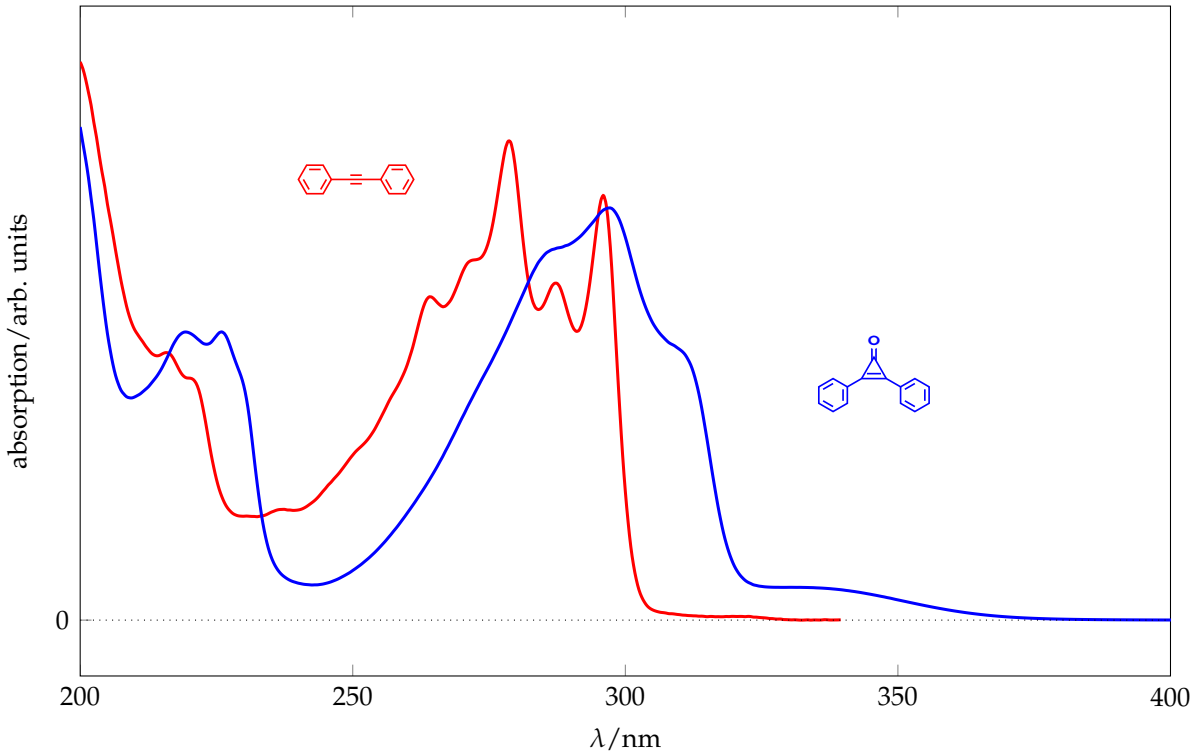


Figure 2.2: UV spectra of diphenylcyclopropenone and diphenylacetylene in acetonitrile.

as the product absorbs about three times stronger at the former and similarly strong at the latter wavelength. The transition of diphenylcyclopropenone has been attributed $\pi\pi^*$ character[45].

2.2.2 Transient IR spectra of diphenylacetylene

In order to establish the spectral characteristics of the excited states of diphenylacetylene, transient spectra of diphenylacetylene were recorded in CD_3CN using an excitation wavelength of 267 nm. Along with the stationary absorption spectrum, two of these are shown in Figure 2.3: At 25 ps after an initial excitation using a 267 nm light pulse, the S_1 state is the prevalent excited state, whereas at 1500 ps, there is only little electronically excited population in states other than the T_1. Thus, neglecting stimulated emission, any positive signal in either transient spectrum can be attributed to those respective states, while negative signals originate from depopulation of the electronic ground state.

The stationary spectrum of diphenylacetylene features four absorptions, at 1444, 1498, 1574, and 1602 cm^{-1}, respectively, which have been identified as linear combi-

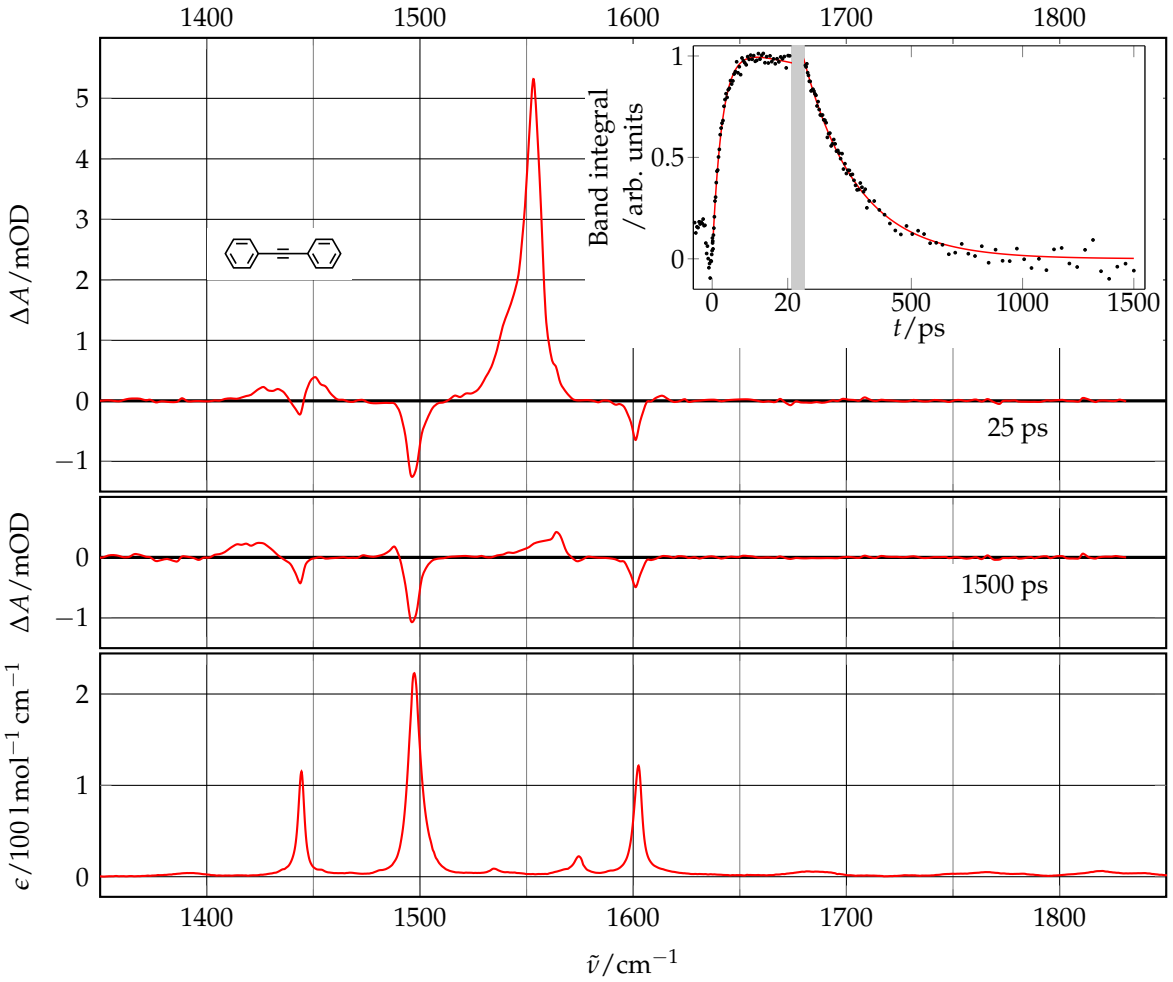


Figure 2.3: From top to bottom: transient IR spectra of diphenylacetylene with small (top panel) and large (middle panel) delays after excitation at 267 nm and stationary IR spectrum (bottom panel), all measured in CD_3CN. Inset: Temporal evolution of the integral of the 1553 cm^{-1} band, where —— is a biexponential fit: $(1.055 \pm 0.007)\exp[-t/(240 \pm 4)\,\mathrm{ps}] - (1.05 \pm 0.02)\exp[-t/(2.59 \pm 0.10)\,\mathrm{ps}]$ (errors are solely based on the asymptotic standard error of the fit and do not include any estimation of systematic – or other further errors). Note the change of scale at the gray gap.

nations of the 19b, 19a, 8b, and 8a modes (in Wilson's notation[46]), respectively, of the two phenyl rings[39,42,47]. The T_1 state exhibits three notable absorptions at 1425, 1490, and 1560 cm^{-1}, for which so far no assignment exists, except for a Raman band at 1565 cm^{-1}, which has been attributed to the 8a mode[47,48], but should not be observable in IR absorption experiments as the T_1 state was found to be of $\mathcal{D}_{2h}$ symmetry[36]. The excited singlet states have been studied extensively,[40–42] and the intense band of the S_1 state at 1553 cm^{-1} has been attributed to the 8a mode, while the broader band covered by a ground state bleach at 1445 cm^{-1} has been reported to originate from the 19a mode with a frequency of 1453 cm^{-1}.[42]

The temporal evolution of the strongest S_1 band at 1553 cm^{-1} is also shown in Figure 2.3 and a biexponential fit yields a relatively short S_2 lifetime of 2.5 $\pm$ 0.2 ps, while the value of 220 $\pm$ 20 ps for the S_1 state is slightly larger than reported previously[34]. The short S_2 lifetime, however, has been confirmed by measurements of the transient electronic spectra[6], and it has to be assumed that the older values by Hirata *et al.*[34] were a result of insufficient temporal resolution.

By comparing with the stationary IR spectrum and the bleached S_0 state, it is further possible to determine the integrated absorption coefficient of the strongest absorption of the S_1 state at 1553 cm^{-1}. With an intensity of 18.8 $km\,mol^{-1}$ for the band of the S_0 state at 1498 cm^{-1}, the absorption of the S_1 state at 1553 cm^{-1} has an intensity of 130 $\pm$ 20 $km\,mol^{-1}$. It should be noted that the different resolutions of the FTIR and transient apparatuses make it necessary to use integrated rather than peak intensity absorption coefficients.

2.2.3 Transient IR spectra of diphenylcyclopropenone

Spectra of diphenylcyclopropenone corresponding to the ones discussed earlier for diphenylacetylene are shown in Figure 2.4. In the stationary spectrum, the most prominent features are strong bands at 1632 and 1854 cm^{-1} assignable to a combination of the carbonyl mode and the central C-C double bond as well as the carbonyl mode and the cyclopropene ring distortion mode (assignment from a quantum chemical calculation using the GAMESS[49] package with the B3LYP functional[50] and 6-311G basis set[51]). Smaller bands at 1340, 1448, and 1498 cm^{-1} originate from ring vibrations of the phenyl moieties and the cyclopropene ring.

In the stationary spectrum, the absorption at 1632 cm^{-1} has an integrated absorption coefficient of 180 $km\,mol^{-1}$. For the spectrum recorded 25 ps after excitation, one

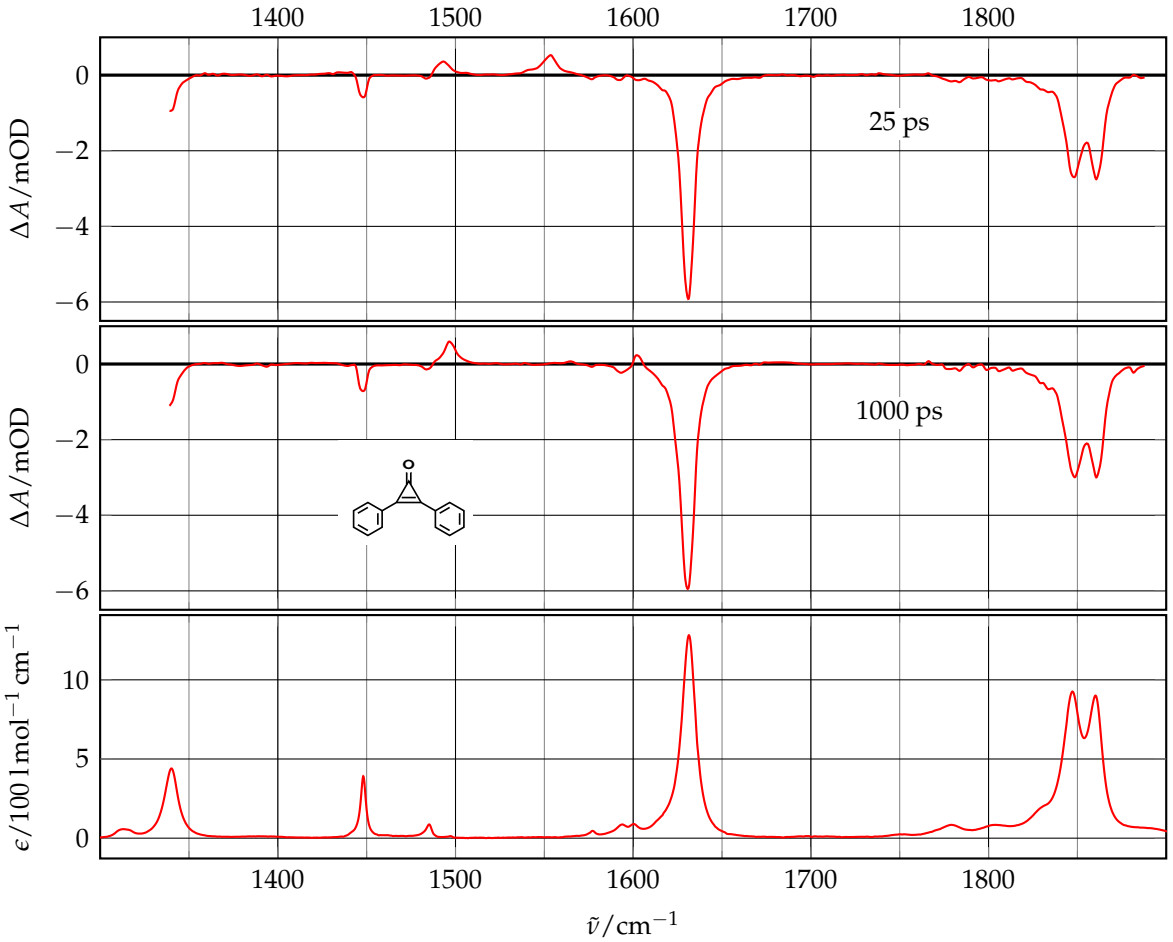


Figure 2.4: From top to bottom: transient IR spectra of diphenylcyclopropenone with small (top panel) and large (middle panel) delays after excitation at 267 nm and stationary IR spectrum (bottom panel), all measured in CD_3CN.

would expect a strong absorption of the S_1 state of diphenylacetylene, which is obviously not the case. Rather, the observed absorption corresponds to a few percent of diphenylacetylene in its S_1 state relative to the bleached spectrum of diphenylcyclopropenone. It should be noted that it is imperative to only employ fresh solutions of diphenylcyclopropenone, as has been pointed out by others as well[4,14,35], due to the absorption characteristics of the product, as discussed in subsection 2.2.1.

It is obvious, however, that all the characteristic absorptions of ground state diphenylacetylene are also present, albeit covered more or less strongly by the bleached spectrum of diphenylcyclopropenone. In the 1000 ps spectrum, the absorptions at 1602 cm^{-1} and – even stronger – at 1498 cm^{-1} are clearly visible, and the intensity of the 1498 cm^{-1} band is in good agreement with the amount of diphenylcyclopro-

penone consumed in the reaction as determined from its 1632 cm^{-1} band. The remaining two absorptions at 1444 and 1574 cm^{-1} are apparent at closer inspection, but covered strongly by bleached bands of diphenylcyclopropenone. The 25 ps spectrum, on the other hand, indicates substantial vibrational excitation of the ground state diphenylacetylene, as the 1602 and 1498 cm^{-1} bands are somewhat red-shifted, while the remaining diphenylacetylene bands are not immediately visible.

To take a closer look at the origin of electronic ground state diphenylacetylene, transient spectra of its 1498 cm^{-1} absorption are depicted in Figure 2.5. In the earliest spectrum, only two bleached absorptions are visible, at 1485 and 1498 cm^{-1}, which – by comparison with the static spectra – can be attributed to diphenylcyclopropenone and an impurity of diphenylacetylene, respectively. Subtracting this earliest spectrum from the other transient spectra of Figure 2.5 generates a more precise picture of the evolution of the 1498 cm^{-1} band and thus of the fate of diphenylacetylene generated in its ground electronic state during the decarbonylation of diphenylcyclopropenone. The obvious red-shift in the earlier spectra reflects the vibrational excitation of the reaction product, while its decline in the succeeding spectra is caused by vibrational cooling, i.e. transfer of excess vibrational energy to the solvent bath. The integral of this band saturates with a time constant of $\tau = (16.6 \pm 0.4)$ ps and thus much slower than the S_2 state lifetime of diphenylcyclopropenone determined previously as only a few hundred femtoseconds[6,35]. This can be interpreted as a decrease in oscillator strength of the 1498 cm^{-1} absorption upon vibrational excitation of the molecule (see subsection 3.3.1) and explains the smaller appearance of bands in the 25 ps spectrum in Figure 2.4.

With the apparatus used, it was further possible to cover all previously employed wavelengths of excitation. The results of exciting either diphenylacetylene or diphenylcyclopropenone at 267 nm or 295 nm and various longer wavelengths are shown in Figure 2.6. Two aspects are particularly noteworthy: First, the 1553 cm^{-1} absorption of diphenylacetylene ceases to be observable in both scenarios, excitation of the pure substance and photo-decarbonylation of diphenylcyclopropenone, at approximately the same excitation wavelength of 307.5 to 310 nm. This is in sharp contrast to the earlier finding of a wavelength limit of approximately 320 nm[4]. Second, the red-shift of diphenylacetylene S_0 state absorption, hinting at vibrational excitation of the reaction product, appears to reflect the rising amount of excitation energy at shorter wavelengths.

Finally, an experimental test of pathway II of Figure 2.1 – internal conversion from the electronically excited state followed by dissociation from a vibrationally hot elec-

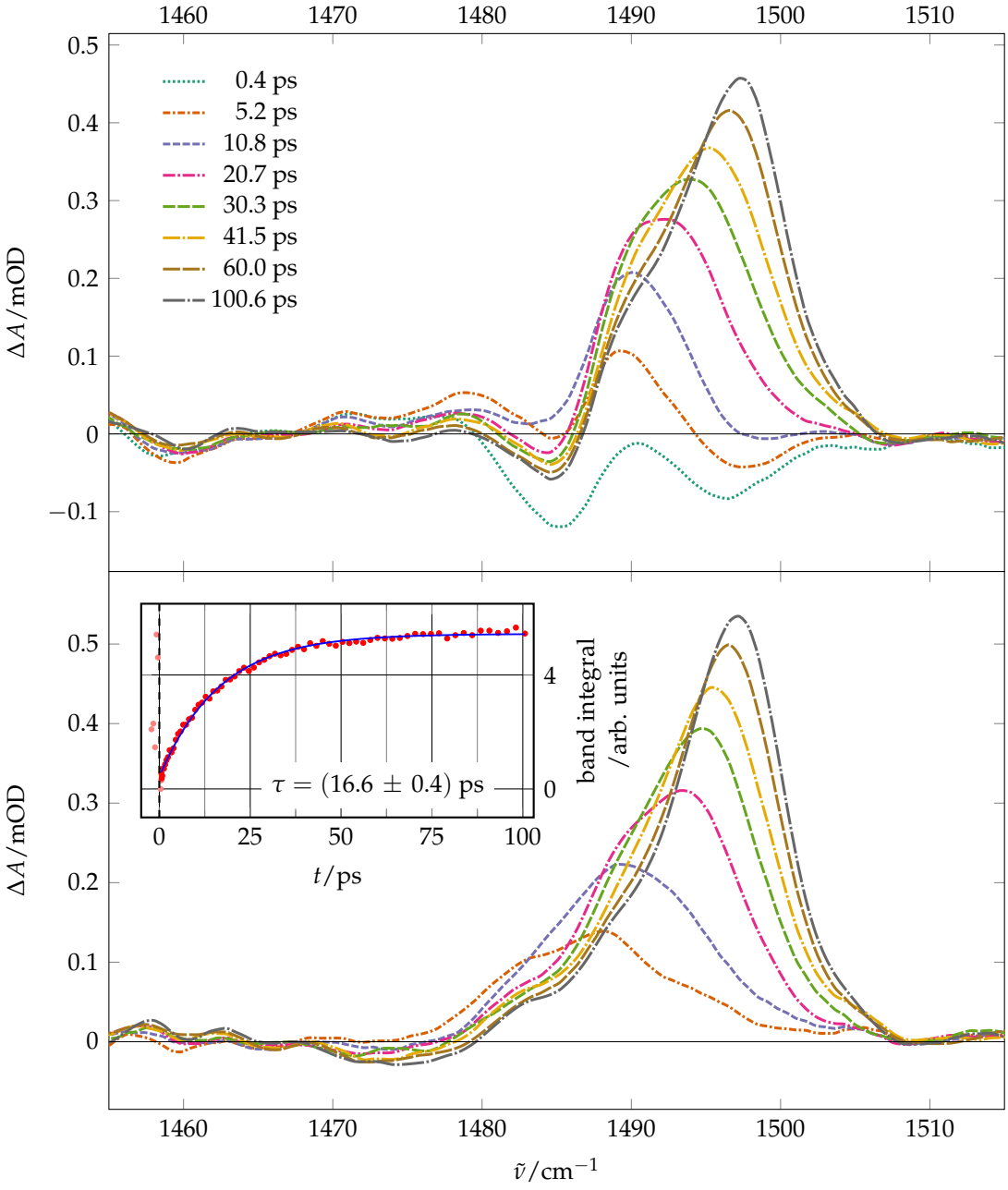


Figure 2.5: Transient IR spectra of the 1498 cm^{-1} absorption of diphenylacetylene. Spectra were recorded in CD_3CN after excitation of diphenylcyclopropenone at 267 nm. The bottom panel shows the spectra from the top panel with the earliest spectrum subtracted. The inset in the bottom panel shows the band integral of the spectra in the bottom panel (including spectra not shown) and an exponential fit.

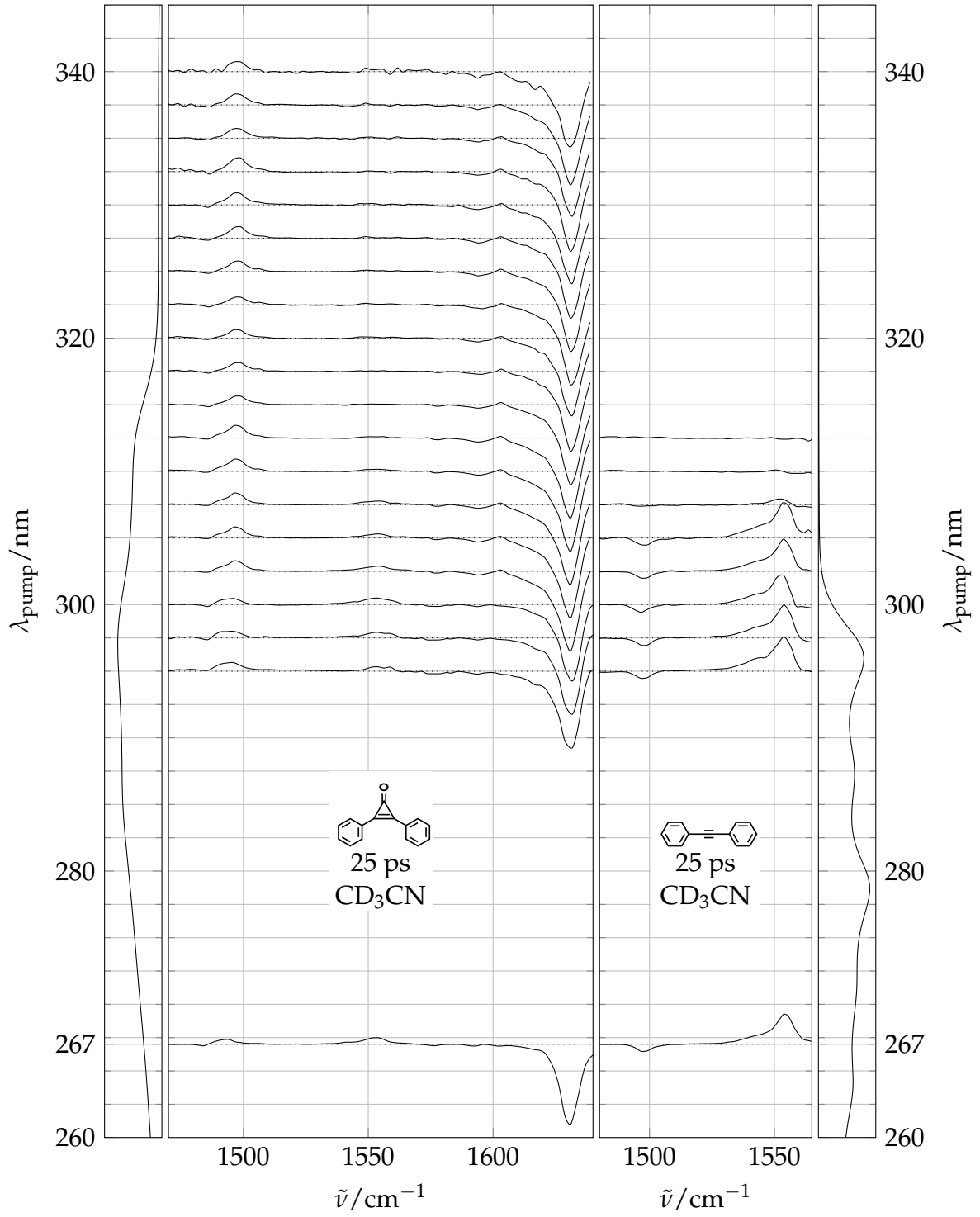


Figure 2.6: Transient spectra of diphenylcyclopropenone and diphenylacetylene after excitation at various different wavelengths. Spectra were normalized with respect to the bleach of the 1632 cm^{-1} absorption of diphenylcyclopropenone or the S_1 state absorption of diphenylacetylene at 1553 cm^{-1}, respectively, and the respective excitation wavelengths added to the change in optical density. Corresponding UV absorption spectra are shown in the margins.

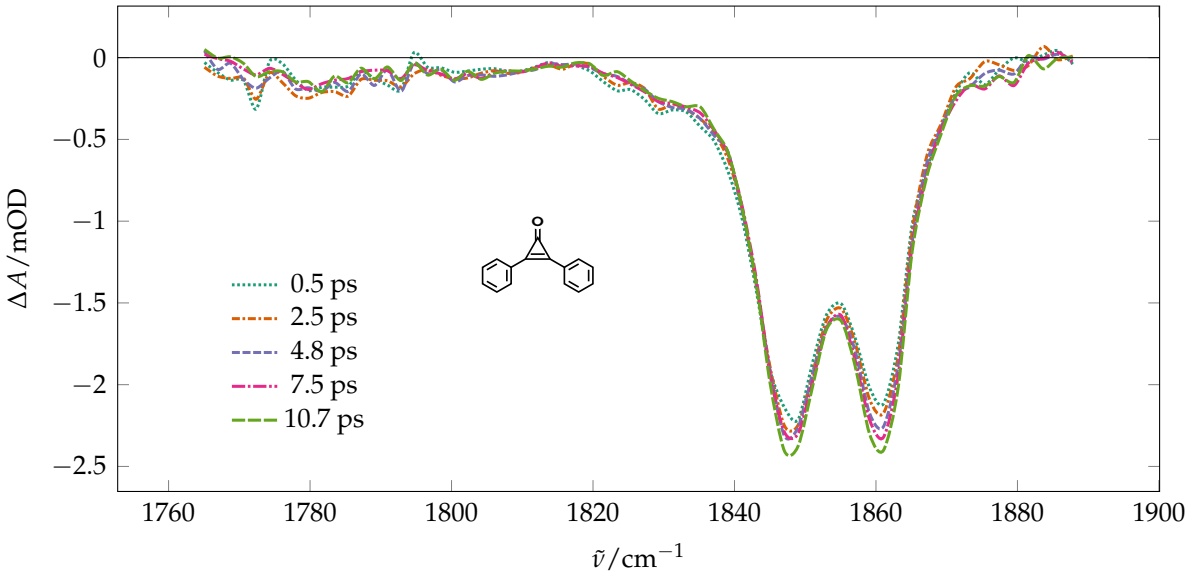


Figure 2.7: Transient IR spectra of the carbonyl absorption of diphenylcyclopropenone. Recorded after excitation at 267 nm in CD_3CN to monitor the possible emergence of the vibrationally hot ground state of the reactant.

tronic ground state – would be to observe vibrationally excited ground state diphenylcyclopropenone, i.e. the situation before it undergoes reaction in this postulated scenario. The carbonyl stretching absorption at 1854 cm^{-1} is most appropriate for this test, as obviously no absorptions of any state of diphenylacetylene obscure it. But as is clearly obvious from the spectra shown in Figure 2.7 covering the first eleven picoseconds after excitation, such vibrationally excited ground state diphenylcyclopropenone could not be observed.

2.3 Discussion

The results of the transient IR experiments rule out both pathways I (adiabatic dissociation) and II (hot ground state dissociation) from Figure 2.1, while supporting pathway III (non-adiabatic dissociation). Vibrationally hot diphenylcyclopropenone, as required for pathway II, is lacking, and this supports the argument of the hypothetical statistical reaction on the ground state energy surface being much slower than the observed product formation (see section 2.1).

As for pathway I, the amount of S_1 diphenylacetylene observed after decarbonylation of diphenylcyclopropenone is much less than expected and more readily explained as an impurity or by secondary excitation (see below). Additionally, the depen-

dence on pump wavelength of its appearance supports this finding, as it does not show any significant difference between the direct excitation of the pure diphenylacetylene and its generation as a reaction product. To the contrary, S_0 state diphenylacetylene is clearly evident at all excitation wavelengths and at smaller delay times than expected. At longer delays, the ground state absorption matches with the consumption of the reactant, while its attenuated intensity at shorter delay times is probably caused by a decreased absorption coefficient as a result of vibrational excitation. It should be noted that the non-radiative decay proposed on the basis of fluorescence experiments[36] is much too slow to explain the observation of such large amounts of ground state reaction product.

More recent experiments have indicated chain-reaction mechanisms in the decarbonylation of solid diphenylcyclopropenone, where a quantum yield of 3.3 was determined[52], and, based also on experiments in solution[52–54], this finding was rationalized resorting to pathway I. It remains to be explained, why these observations deviate from the transient IR results presented. First and foremost, it has to be noted that none of the aforementioned experiments in the visible and near UV spectral range was ever capable of monitoring ground state diphenylacetylene. That is, in all of these experiments wavelengths shorter than 390 nm were not monitored. Additionally, substantial differences between the spectra obtained from direct excitation of diphenylacetylene and from photodecarbonylation of diphenylcyclopropenone usually had to be conceded[4,35].

Most of the previous studies employed light with a wavelength of about 267 nm for reaction initiation[35,54], which inevitably increases the risk of exciting impurities of diphenylacetylene, whose absorption is much stronger at this wavelength than that of the reactant (cf. subsection 2.2.1). While all authors mention this problem and their precautions against it (usually purification of the reactant and high flow rates of the sample solution, as was implemented for this study), a more immediate danger is hardly ever considered: the excitation of nascent diphenylacetylene by the trailing edge of the pump pulse. This problem is particularly grave for experiments with picosecond excitation pulses[4] or higher excitation intensities[35].

Based on pathway III, a model has been proposed, which rests on the following kinetic sequence:

$$\mathrm{DPCP}(S_0) \xrightarrow{k_1(t)} \mathrm{DPCP}(S_2) \xrightarrow[-\mathrm{CO}]{k_2} \mathrm{DPA}(S_0) \xrightarrow{k_3(t)} \mathrm{DPA}(S_2)$$

where

$$k_1(t) = \epsilon_{\mathrm{DPCP}} \cdot I(t) \cdot \ln 10$$
$$k_3(t) = \epsilon_{\mathrm{DPA}} \cdot \ I(t) \cdot \ln 10$$

and $I(t)$ is the intensity of the pump pulse, while k_2 is the rate of dissociation of the S_2 state of diphenylcyclopropenone. This model neglects effects due to stimulated emission and changes in absorption coefficient due to vibrational excitation of the product, as they are expected to be very small. For the experiments discussed here, with 10 mM sample solutions, an (idealized) cylindrical pump volume of 0.6 mm length and 0.2 mm diameter, a Gaussian-shaped pump pulse of 150 fs duration and 1 µJ energy, decadic absorption coefficients of $\epsilon_{\mathrm{DPCP}} = 10^4\,\mathrm{l\,mol^{-1}\,cm^{-1}}$ and $\epsilon_{\mathrm{DPA}} = 2 \times 10^4\,\mathrm{l\,mol^{-1}\,cm^{-1}}$ – corresponding to the situation at 267 nm – and a decarbonylation rate constant of $k_2 = (0.2\,\mathrm{ps})^{-1}$, almost 5% of diphenylacetylene generated as a result of the decarbonylation of diphenylcyclopropenone are excited to the S_2 state in this model, as determined by numerical integration.[6]

For other experiments, some of the required parameters are usually not clearly stated, but it can be assumed that especially for the earliest experiments by Hirata[4], although performed at 295 nm, but with pulses of 8 ps fwhm and 600 µJ, this effect should lead to a degree of excitation of reaction-generated diphenylacetylene indiscernible from excitation of the substance itself. It has to be mentioned, however, that newer experiments with detection in the visible spectrum and short pump pulses (35 fs) should not suffer this defect, yet on the other hand they do not monitor the ground state of the product.[54]

Nevertheless, the results and considerations presented here, alongside with accompanying measurements of the transient visible and near-UV spectra[6], strongly point toward pathway III, the non-adiabatic dissociation process, as the actual reaction mechanism.

Chapter 3

Intramolecular vibrational energy transport

In this chapter, investigations of the transport of vibrational energy along molecular chains shall be presented. To this end, azulene derivatives were used as the internal conversion of azulene after electronic excitation to its S_1 state offered a simple means of depositing energy in a molecule. The derivatives contained different side chains, some of which were simple aliphatic chains, others made up of either polyethylene glycol units. Monitoring of the energy flow was accomplished by installing suitable sensor groups with marked IR absorptions.

This chapter is organized as follows: After an introductory review of the research conducted in this field during the last one and a half decades, results obtained for the present study shall be discussed. The discussions will follow a pattern of first focusing on spectral information, to then present the temporal evolution of various signals, and finally compare the different substances used. Subsequently, a rate-equation model of energy transport will be applied to the experimental data and discussed. Finally, an attempt will be undertaken to model the spectral data obtained experimentally.

3.1 Literature review

Since the distribution of vibrational energy within a molecule is essential for its access to different regions of the potential energy surface, i.e. for determining which reactions it can or cannot undergo, knowledge about the transport of vibrational energy is vital for fine-grained control of reactions.[55] Practical applications are numerous and

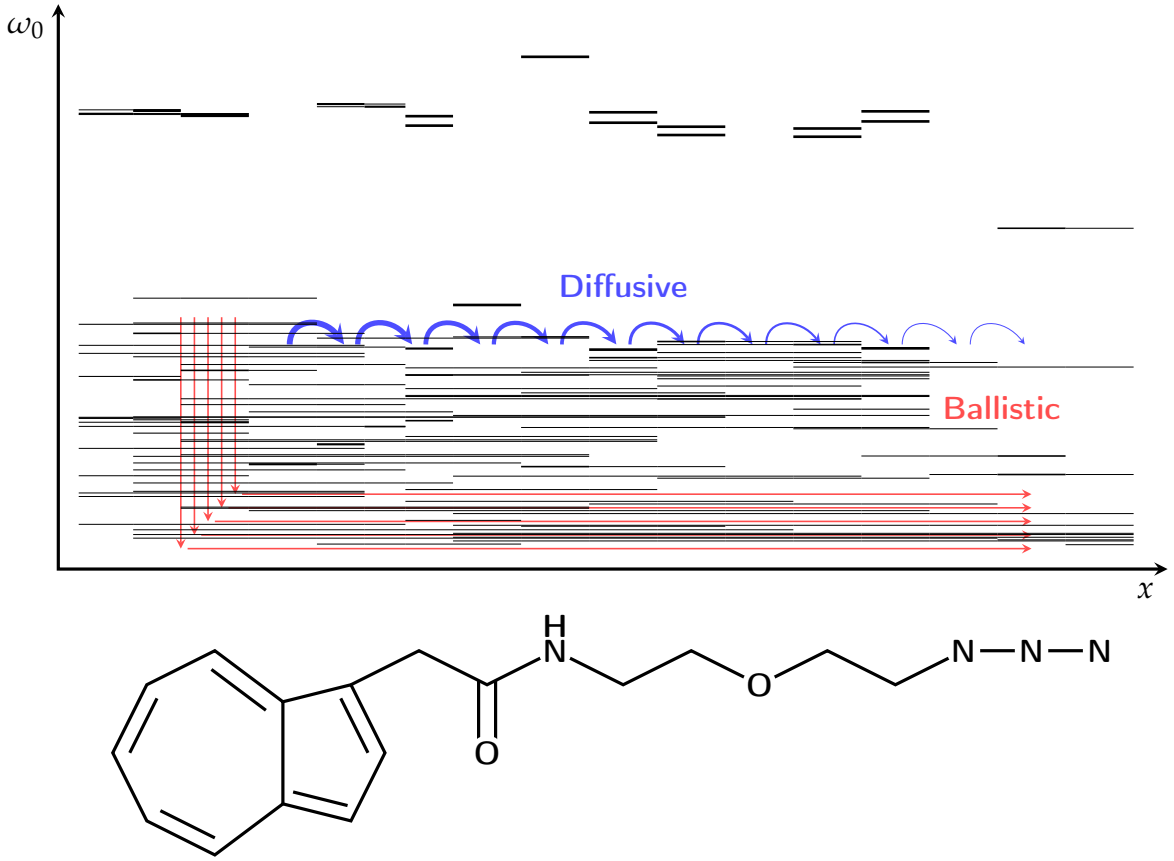


Figure 3.1: Idealized comparison between diffusive and ballistic vibrational energy transport. The lines represent normal modes, with the widths indicating the relative magnitude of their coefficients at the corresponding sites.

include heat conduction in nano-sized electronic devices[56–59], the stability and mechanics of biological molecules – especially proteins[1,12,60], the use of nanoparticles in cancer therapy[61], as well as the general control of chemical reactions[8,62,63].

With the advent of ultra-fast, pulsed lasers, direct observation of intramolecular vibrational energy redistribution (IVR) largely replaced earlier approaches, which relied mainly on frequency-resolved techniques[7]. The general microscopic design of these experiments usually comprises – in metaphorical terms – a heat source, a heat conductor, and a thermometer, where "heat" is randomized vibrational energy. This section shall serve as a survey of the components used in these roles as well as the insights obtained by previous research.

Two limiting cases of energy transport are commonly identified: ballistic and diffusive energy transport (see Figure 3.1). Originating from solid state physics, ballistic heat transport is characterized by few phonon scattering events and, accordingly, a

long mean free path of phonons.[64] In molecular systems, this corresponds to transport by few highly delocalized normal modes in the form of a wave packet.[7,8,63,65,66] This mode of transfer is fast with regard to its "signal speed", that is usually of the same order of magnitude as the speed of sound,[8,63,67] although occurences of slower speeds have been interpreted as an indication against the involvement of acoustic phonons[68]. Energy flux, and consequently thermal conductivity are – due to the low number and large spatial extent of the transporting modes – independent of chain length[57,69], or – in purely one- or two-dimensional simulations – even divergent with system size[64]. To attain a situation in which an appreciable number of modes is greatly delocalized, ordered chains[8,67], low temperatures[60,70], and harmonic or near-harmonic systems[57] are advantageous. Since under these conditions (especially at low temperatures and with little anharmonicity) true energy redistribution among modes is limited and energy may become "trapped" in localized, high-frequency modes, however, switching to the diffusive transport regime by an increase of temperature was found to improve transport efficiency.[60,65,70]

Conversely, in diffusive energy transport, phonon scattering events are frequent and mean free paths short.[71] In molecular systems it involves transfer of energy to many localized modes and possibly into side-chains, thus slowing down the arrival of energy at its destination, but accelerating the overall redistribution of energy among the modes of a molecule.[70,72] The quantitative importance of anharmonic coupling of localized modes, which is vital to diffusive transport, has repeatedly been emphasized in theoretical works.[65,73]

The simplest formal description of diffusive heat conduction is Fourier's law[74]

$$\dot{Q} = -\kappa \nabla T \tag{3.1}$$

which establishes a direct proportionality between the amount of heat transferred per unit time $\dot{Q}$ and the temperature gradient ∇T via the thermal (heat) conductivity κ. Analogous to conservation of mass leading from Fick's first – to Fick's second law of diffusion, energy conservation yields the heat equation

$$\dot{T} = \alpha \nabla^2 T = \frac{\kappa}{c_p} \nabla^2 T$$

where α is the heat diffusivity and c_p the heat capacity at constant pressure.[75,76] If phonons are the only carriers of heat, heat diffusion equals phonon diffusion in the

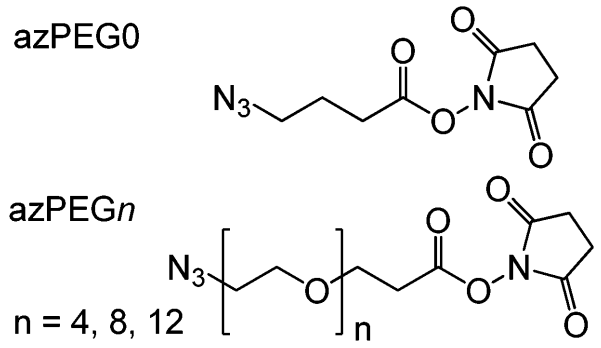


Figure 3.2: Selected systems used by the Rubtsov group. Reproduced from *Physical Chemistry Chemical Physics*, 2012, 14, 10445–10454 with permission of the PCCP Owner Societies. © 2012 the PCCP Owner Societies.

microscopic picture[76,77]

$$\alpha = \frac{1}{3} v \ell c_v$$

and can thus be related to their velocity v and mean free path ℓ. c_v is the lattice heat capacity per unit volume. In the ballistic regime Equation 3.1 applies to the temperature difference between the origin and destination of heat transfer rather than to the temperature gradient in between.[7]

3.1.1 Depositing vibrational energy in molecules

Three main ways of creating vibrational excitation in a molecule are commonly used. The most direct way is to directly excite a functional group exhibiting strong infrared absorption. Additionally, the absorption needs to be distinct from those of other functional groups present in the molecule and the surrounding solvent. Rubtsov and coworkers used the azide group and its asymmetric stretching vibration absorbing at around 2106 cm^{-1} in their investigation of energy transfer in polyethylene glycol chains (see Figure 3.2).[8,63] Also, the Hamm group, in an effort to elucidate the energy flow in helical peptides, used both the C=O stretching vibration of a carbamate[78] and the C-D stretching vibrations of a perdeuterated *iso*-butyl moiety of leucine[1].

A second way is the electronic excitation of a functional group known to return to its electronic ground state in a fast, radiationless process, i.e. internal conversion. As there is only little time for transfer of energy to the surrounding solvent, a comparatively large amount of electronic energy can be deposited in the molecule and

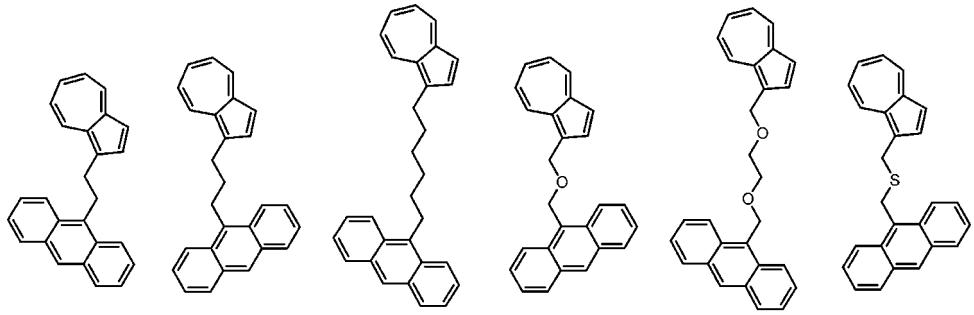


Figure 3.3: Systems used for experimental studies by the Schwarzer group. Reprinted in part with permission from *Journal of Chemical Physics*, 2004, 121, 1754–1764. © 2004 AIP Publishing LLC.

subsequently converted into vibrational energy. In the Schwarzer group – as also in this work – this approach was used very productively in a number of earlier studies[7,69,75,79–82] employing azulene as a source of vibrational energy (see Figure 3.3). Azulene is well known to undergo fast internal conversion on a timescale of about one picosecond[83–85] and is thus very suitable for converting roughly 16500 cm^{-1} of electronic excitation into vibrational energy.

The Hamm group used a similar approach by linking azobenzene to a protein helix and depositing about 23500 cm^{-1} of initial electronic energy.[1,70,72,78] Here, internal conversion via *cis-trans* isomerization was found to predominantly occur with an even faster time constant of 170 fs starting from the *cis*-isomer and 320 fs if the *trans*-isomer was employed, but to also include a slower, minor component of about 2 ps.[86] As pointed out by Müller-Werkmeister and Bredenbeck[87], large changes of structure – as those entailed by isomerization – or other properties of a moiety used for excitation are particularly unfavorable in biological macromolecules. This led them to favor the azulene group for excitation in their molecule (see Figure 3.4).

Recently, excitation originating from surfaces and nanoparticles has also come into use. Hamm and coworkers made an effort in this field by attaching their previously developed molecular system to gold nanoparticles.[61] In these experiments the gold nanoparticles were heated using ultra-fast laser pulses to excite both off and on the plasmonic resonance of the particles. The Dlott group, on the other hand, employed gold layers of 50 nm width and heated them with sub-picosecond 800 nm pulses impacting on the rear side of the layer.[56,59,62,68,88] After this generation of hot electrons and their following diffusion through the gold layer, electron-phonon coupling mediates ultra-fast thermalization of the gold surface within about 1 ps and subsequent transfer occurs to a molecule bound to the front side of the layer (see Figure 3.5).[68]

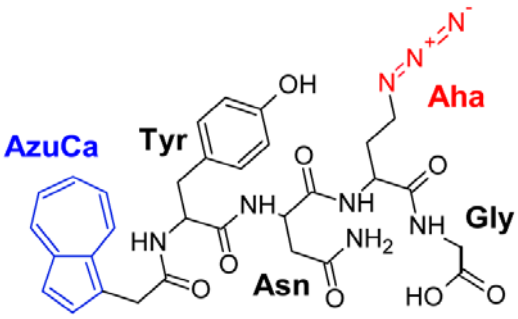


Figure 3.4: Peptide labeling with an azulene group for excitation and an azide-bearing amino acid as performed by the Bredenbeck group. Reproduced from *Physical Chemistry Chemical Physics*, 2013 (accepted manuscript) with permission of the PCCP Owner Societies. © 2013 the PCCP Owner Societies.

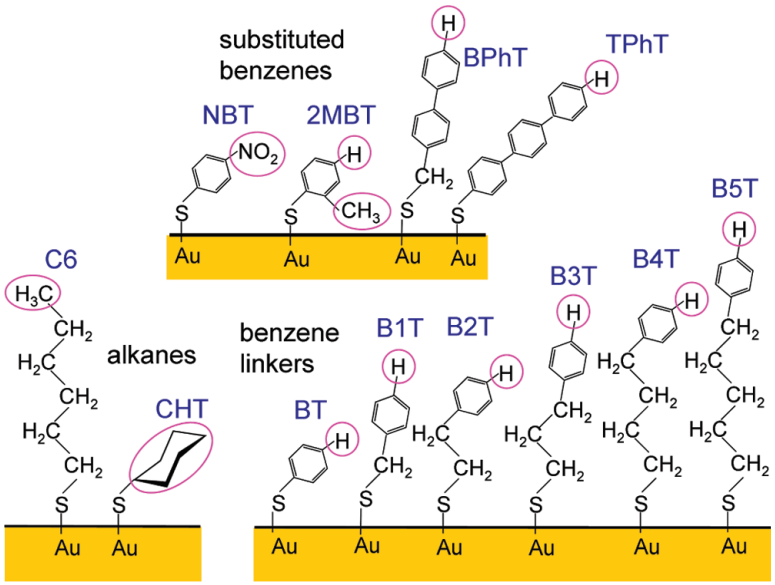


Figure 3.5: Systems used by the Dlott group. Reprinted with permission from *The Journal of Physical Chemistry A*, 2009, 113, 12105–12114. © 2009 American Chemical Society.

These approaches feature the benefit of providing a thermally randomized source of energy for the experiment.

Few studies have thus far been conducted to compare these three techniques of excitation and their influence on the ensuing energy transfer. It appears fairly certain, though, that while infrared excitation provides the most well-defined initial state, the usage of nanoparticles and surfaces, on the other hand, accomplishes complete randomization prior to passing the energy on. As mentioned above, in an attempt to compare the impact of the technique of excitation, the Hamm group conducted experi-

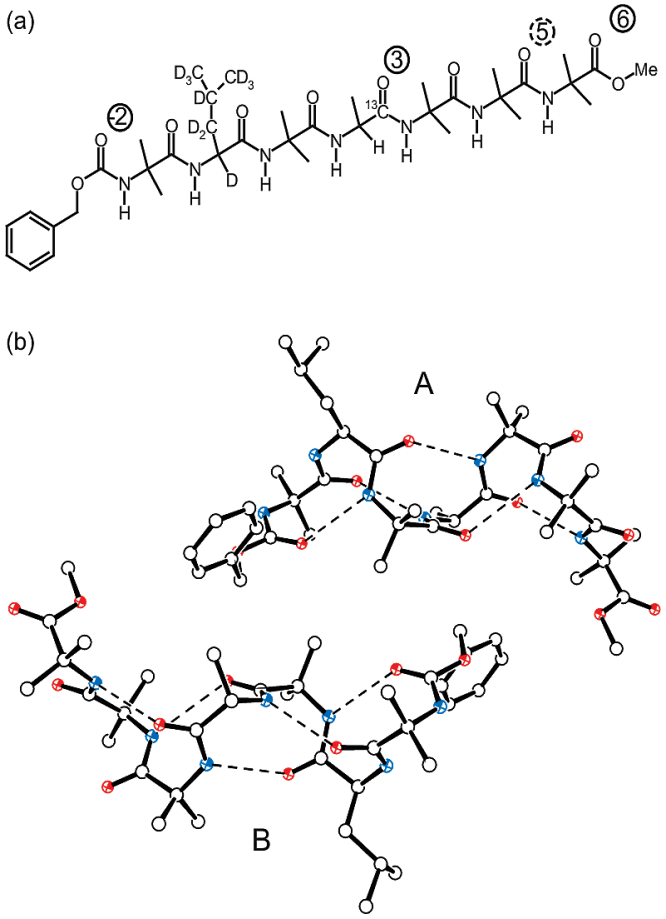


Figure 3.6: An example system used by the Hamm group. (a) Chemical Structure. (b) Structure of two independent molecules in the asymmetric unit. Reprinted with permission from *The Journal of Physical Chemistry B*, 2009, 113, 13393–13397. © 2009 American Chemical Society.

ments using either infrared excitation via the carbonyl absorption of a carbamate or the photo-induced *cis-trans* isomerization of azobenzene[78]. Initially, the high-energy excitation seemed to result in slower energy transport, which was attributed to the greater involvement of high-frequency modes with a greater tendency to slow, diffusive transport. It was later found, however, that choosing a carbonyl absorption in a protein helix as the initially excited vibration enabled resonant energy transport among the amide carbonyl vibrations and in addition obscured the early steps of energy transport due to spectral overlap among the carbonyl absorptions.[1,65] Using instead C-D stretching vibrations of the perdeuterated alkyl rest of leucine for excitation (see Figure 3.6) lead to essentially the same transport velocities as in the high-energy case.[1] This finding was later also corroborated theoretically by molecular dynamics calculations.[89]

Even earlier, the Dlott group also highlighted the impact of decay cascades on transport velocity. Exciting different modes of surfactant molecules in reverse micelles, they found transport from O-H stretching vibration of the water cores of the micelles to the solvent (CCl_4) surrounding the micelles to be faster than from the C-H stretching vibrations of the lipophilic tails. This counter-intuitive result was explained by the longer lifetimes of the C-C stretching and C-H deformation modes, which formed the primary decay channel of the C-H stretching excitation, as compared to the modes of the head groups of the surfactant molecules, which deliver the energy to the solvent faster and are the primary successors to the O-H stretching excitation.[90]

3.1.2 Monitoring the progress of energy transport

In order to measure the velocity of energy transport, two basic strategies exist: one may either attempt to equip a molecule with multiple "sensors" at different distances from the source of excitation or use a number of similar molecules where the same single "sensor" is at different distances from the source of excitation. The first approach was used by the Hamm group numerous times with minor variations in peptide helices. With their amide groups these peptide helices provide a straightforward way to monitor heat flux using the characteristic amide infrared absorptions, particularly the C=O stretching vibration. However, spectra are usually congested, and monitoring particular amide groups is challenging and requires isotopic labeling to achieve some level of distinction.[72]

The second approach of using similar systems in which the heat conducting structure was varied in length has been employed by the Schwarzer, Dlott, and Rubtsov groups. The Schwarzer group relied mainly on changes in the visible and UV spectra of their azulene-anthracene compounds. It was thus possible to assess both the energy content of the initially excited azulene unit via its S_2 state absorption as well as the arrival of energy in the anthracene unit at the opposing end of an aliphatic chain.[7,75,80].

The technique most similar to the approach chosen for this work is the Rubtsov group's use of two-dimensional IR (2DIR) spectroscopy.[8,63,66,91–94] Particularly, a succinimide rest linked to an azide group via a polyethylene glycol chain was used in their most recent works (see Figure 3.2).[8,63,66] This combination provided a total of three IR absorptions – the asymmetric stretching vibration of the succinimide carbonyls and two bands attributed to a combination of their symmetric stretching vibration and the ester carbonyl vibration – that were employed as reporters of IVR.[8] More recently, the Bredenbeck group also employed IR labels for detection of IVR. As for four of the sub-

stances discussed here, they favored the azide group (see Figure 3.4), since its highest frequency mode absorbs in a spectral window not rendered opaque by typical absorptions of peptides or solvents commonly used for peptides.[87]

In studying vibrational excitation of various types of self-assembled monolayers of either aryl or alkyl chains tied to a gold surface via a thiolate bridge, Dlott and co-workers, on the other hand, used sum-frequency generation to follow the heating process.[56,62,68,88,95,96] This method is sensitive to changes in both population of vibrational levels and order of the monolayers, and the signal originates mostly from vibrations of the terminal groups, which in their case included CH_3, NO_2, and CH in *para*-position to the surface link in phenyls.[56] Unfortunately, the sensitivity of the sum-frequency signal to temperature is somewhat restricted, i.e. for a methyl group the applicable range is between 300 and 600 K.[62,68]

It is generally important to be certain of the underlying causes of an observed signal and to use caution in its interpretation, as recent results on carbonyl modes in 3_{10} helices indicate.[89] The earlier conjecture, that these results accurately and immediately report on the "local temperature" or residue energy[1,61,70,72,78], proved to be inaccurate. Instead, the amount of energy localized in the carbonyl modes did not match the total amino acid residue energy in MD simulations – a finding that ultimately helped to explain the difference between the heat diffusion constants found in earlier simulations (10 $Å^2\ ps^{-1}$[72]) and experiments (2 $Å^2\ ps^{-1}$[1,72,78], see below).[89,97]

Early-time signals pose an additional challenge: While they are indubitably caused by anharmonic coupling in the case of pure IR excitation (see subsection 3.3.1)[93,94,98], at higher excitation energies (UV or visible) this is not quite as obvious. In so far as early-time signals could be resolved in those cases, they were found to exhibit a time-dependence and thus have to be related to a dynamical process. The Schwarzer group attributed this to "harmonic energy flow" (see below), i.e. the process of dephasing of initially excited normal modes; an interpretation largely founded on accompanying MD calculations.[7,75] On the other hand, the Hamm group offered a slightly different interpretation of the phenomenon as ballistic energy transfer by delocalized modes, that had received initial excitation.[65,72] While the former description stems from a rather classical mechanical interpretation and a focus on the kinetic energies of individual atoms, the latter is somewhat more "quantum mechanical" and has a stronger focus on vibrational modes.

3.1.3 Propagation of energy along molecular structures

In the studies conducted in recent years, mainly three systems were researched with respect to their energy transport properties: chains of methylene (–CH_2–) groups, polyethylene glycol (–CH_2CH_2O–) chains, and peptide helices. Peptide 3_{10}-helices were chiefly used by the Hamm group[1,60,61,70,72,78] as to elucidate their suspected[99] role in energy transport in proteins.[72] In these experiments, an overall heat diffusion constant of 2 $Å^2$ ps^{-1} was found and energy transport in general characterized as "diffusive" in the sense that it could be modeled using a simple linear kinetic master equation involving only energy transfer to neighboring amino acids and the solvent (see also section 3.4).[1,72] This diffusive transfer was preceded, however, by a seemingly instantaneous signal rise attributed to "ballistic" energy transfer in the sense that it was inferred from theoretical studies to occur at the speed of sound (see above).[72] A lowered temperature decreased the diffusive contribution to energy transport, but also lowered its efficiency as fewer high-frequency modes participate in the transport under those circumstances (see above).[60] Additionally, as mentioned above, it was discovered that amide groups themselves provide efficient pathways of resonant energy transport with a heat diffusion constant of 8 $Å^2$ ps^{-1}, which obfuscated the transport along the protein backbone.[1,65,78]

Chains of methylene groups and polyethylene glycol chains, on the other hand, are also widely used as building blocks of molecular structures in IVR experiments. With methylene groups, the length of the heat conducting chain can be reduced even to very short distances and adjusted to within a single chemical bond, while ethylene glycol is an important precursor to polymers, of which many of the shorter oligomers are commercially available with various flavors of terminal groups.

Previously, the Schwarzer group predominantly employed short chains of zero to three, six, and eight methylene groups (see Figure 3.3). In these studies, time constants of IVR between 1 and 5 ps were found and it was concluded that for chains of more than four chemical bonds, i.e. three methylene groups between heat source and heat sensing group, energy transport from the heat source is "ballistic" in the sense that its time constant for IVR from the azulene moiety is independent of chain length. Also, an ultra-fast first stage of energy transport termed "harmonic energy flow" was identified and found to occur on a timescale of about 300 fs: As normal modes of the molecule excited by the initial internal conversion of the azulene heat source include motion of atoms of the chain and anthracene heat sensor, 15 to 20% of the vibrational energy is redistributed due to the dephasing of these modes. This effect was interpreted to also

indicate randomization of energy within the azulene moiety.[7,75,80,82]

A more recent theoretical work indicated that strong Kapitza effects occur between the chain of methylene groups and both the azulene (energy source) and anthracene moieties (energy sink). Heat resistance was found to be vastly greater at the boundaries, i.e. up to five atoms into the chain, to the two aromatic moieties than the internal heat resistance of the chain. The smaller Kapitza effect at the boundary to the anthracene moiety became apparent only in steady state simulations.[69]

Some perturbations were also introduced in these compounds by using oxygen, sulfur, and an oxy-ethylene glycol unit between two methylene groups, again with azulene as an energy source and anthracene as a receiving heat reservoir at the opposite end of the chain. For the single ether, a delay of 2 ps and for the ethylene glycol unit of 1 ps was found in IVR time constants compared to corresponding chains of the same number of bonds made up solely of methylene groups.[7,75]

The Rubtsov group, on the other hand, vastly extended the length of heat conducting molecular chains by using 0, 4, 8, and 12 ethylene glycol units, thus extending the distance of heat transport up to 60 Å. In these experiments, 2DIR cross peaks were observed and the different points in time when those reached their maximum value in various substances found to indicate a constant energy transport velocity of about 450 m/s and thus a ballistic transport regime, while the peak intensity decayed exponentially with a characteristic distance of 15.7 Å. Further, the initial cross peak intensity, corresponding to direct coupling between excited and monitored mode, was found to decay as $\sim L^{-1.4}$ with L being the chain distance.[8,63]

Another series of studies[92,93,98,100,101], involving mostly aryls substituted with nitrile and carbonyl groups confirmed the approximate proportionality of distance and energy transport speed. Moreover, transport speeds were even found to be roughly additive[93], but also depended somewhat on the substitution pattern of the aryl[92]. Altogether, these findings form the basis of the Rubtsov group's "relaxation-assisted two-dimensional infrared (RA 2DIR) spectroscopy". This method seeks to enhance the range of 2DIR spectroscopy by exploiting the amplification of cross peaks between distant groups as a consequence of IVR[98]: While the initial cross peak between two distant localized modes – i.e. the change in the absorption spectrum of one induced by the excitation of the other – is small, due to small direct anharmonic coupling (see subsection 3.3.1), this signal increases as IVR progresses and energy is transferred to modes exhibiting larger coupling to the queried mode.

The Dlott group also concluded transport to be ballistic in alkyl chains at speeds

of up to 1 km/s.[62,68] In the process of transferring heat from the gold surface into the self-assembled monolayers, three stages were identified. Initially, hot surface electrons contribute a small amount of energy very selectively transferred to a few vibrational degrees of freedom, followed by an burst of energy originating from phonons of the gold lattice and traveling ballistically, which was attributed to the onset of the signal. Finally, the heating of the chain and increasing loss of order in the self-assembled monolayer is determined by the heat capacity of the chain, indicating a thermal barrier between the chain and the interface, similar to the above-mentioned Kapitza effect. For both processes, it was noted that the initial step in energy transfer (both phononic and electronic) extends about four to five carbon atoms into the chain (cf. the Schwarzer group's results above).[56,62,68,88,95]

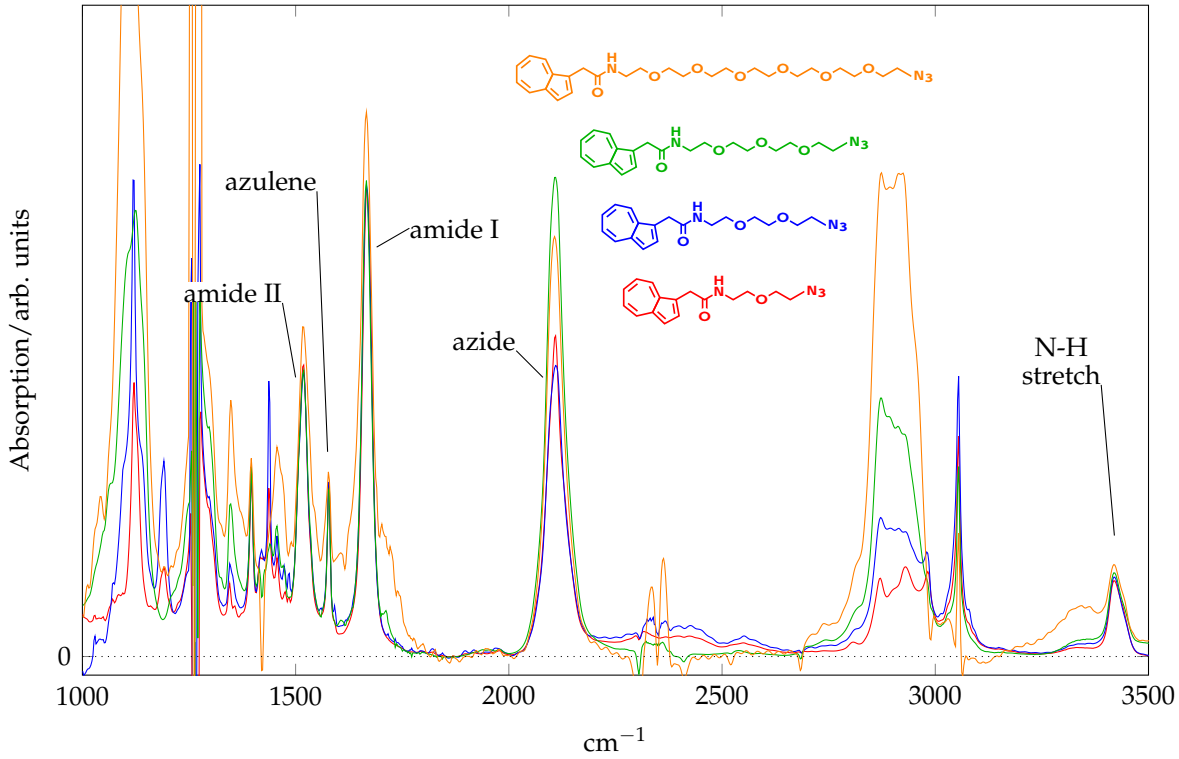
3.2 Stationary IR spectra

Before investigating the dynamic properties of the molecules used in this study with respect to energy transport, it is imperative to analyze their stationary IR spectra. From these spectra important information may be gleaned with respect to the structure and situation of the molecules, as well as the coupling of vibrational modes.

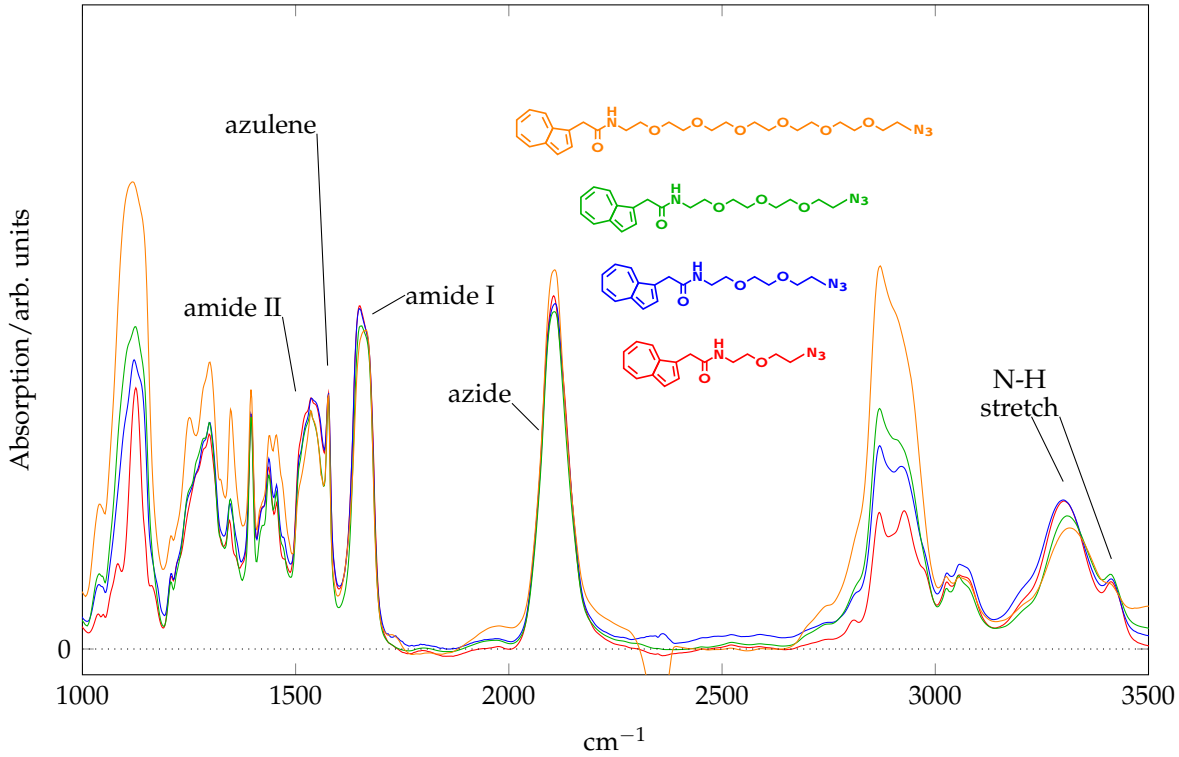
3.2.1 *N*-(Azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides

The IR spectra of the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides in CH_2Cl_2 shown in Figure 3.7(a) exhibit marker bands of all functional groups, none of which are covered up by solvent absorptions: amide II (N-H wagging) at 1518 cm^{-1}, azulene ring distortion at 1578 cm^{-1}, amide I (C-O stretching) at around 1666 cm^{-1}, and an azide mode at approximately 2110 cm^{-1}. These modes were hence chosen for the investigation of IVR dynamics.

Further, an N-H stretching vibration (amide A mode) is apparent at 3420 cm^{-1}. The symmetric azide stretching vibration, which is to be expected somewhere between 1250 cm^{-1} and 1300 cm^{-1}[102,103], is largely covered by absorptions of the solvent. But even in the spectra of the pure substances in Figure 3.7(b) this absorption can hardly be identified unambiguously. Marked differences in relative intensities are naturally to be found in the absorption ranges associated with modes of C-H wagging (1050–1170 cm^{-1}) and stretching (2750–3000 cm^{-1}) in ethylene glycol ethers.[104] The absorption at 3054 cm^{-1} is a remainder of the absorption of the solvent, which



(a) In CH_2Cl_2



(b) As pure substances

Figure 3.7: Stationary IR spectra of *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, normalized to the azulene absorption band at about 1580 cm^{-1}.

could not be corrected satisfactorily.

The positions of the absorptions – especially those of the azide and amide modes – are independent of the length of the respective polyethylene glycol (PEG) chain. It may hence safely be concluded that there is no significant coupling between these modes. This fact is important to note, since coupling may of course lead to different characteristics of IVR along the PEG chain (see subsection 3.1.3)[1,65,78].

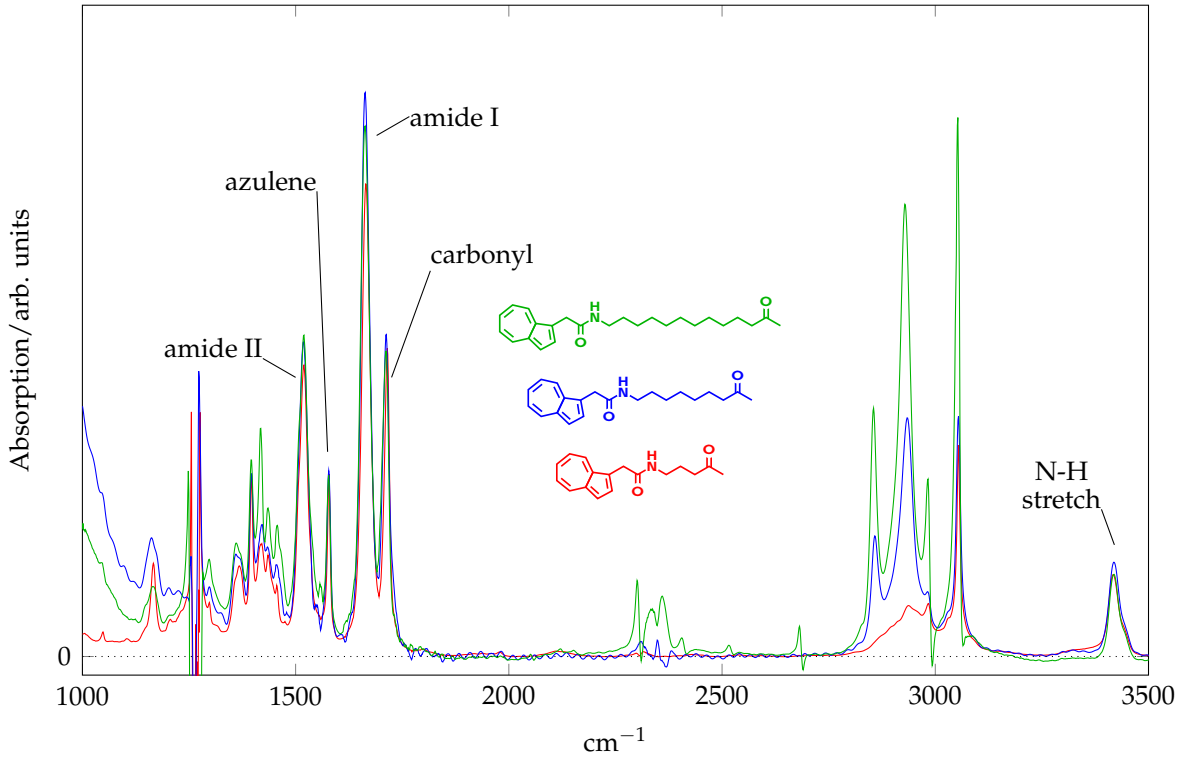
Between 2300 and 2400 cm^{-1}, all stationary IR spectra – including those discussed subsequently – show some noise and apparent absorptions, which do not belong to the substances or solvent themselves. Rather, these are artifacts caused by varying amounts of carbon dioxide in the spectrometer.

The spectra of the pure compounds depicted in Figure 3.7(b) differ remarkably from those of the CH_2Cl_2 solutions. While the positions of absorptions of the azulene and amide modes remain unchanged, those of the C-O and N-H stretching modes of the amide groups predominantly shift to the red (to 1650 cm^{-1} and 3300 cm^{-1}, respectively), whereas the N-H wagging modes shift to the blue (to about 1535 cm^{-1}). In the case of the amide-related modes, some absorption remains at the positions found in CH_2Cl_2 with the ratio of both intensities not varying very much among all given molecules. These shifts are clearly indicative of the formation of hydrogen bonds.[105,106]

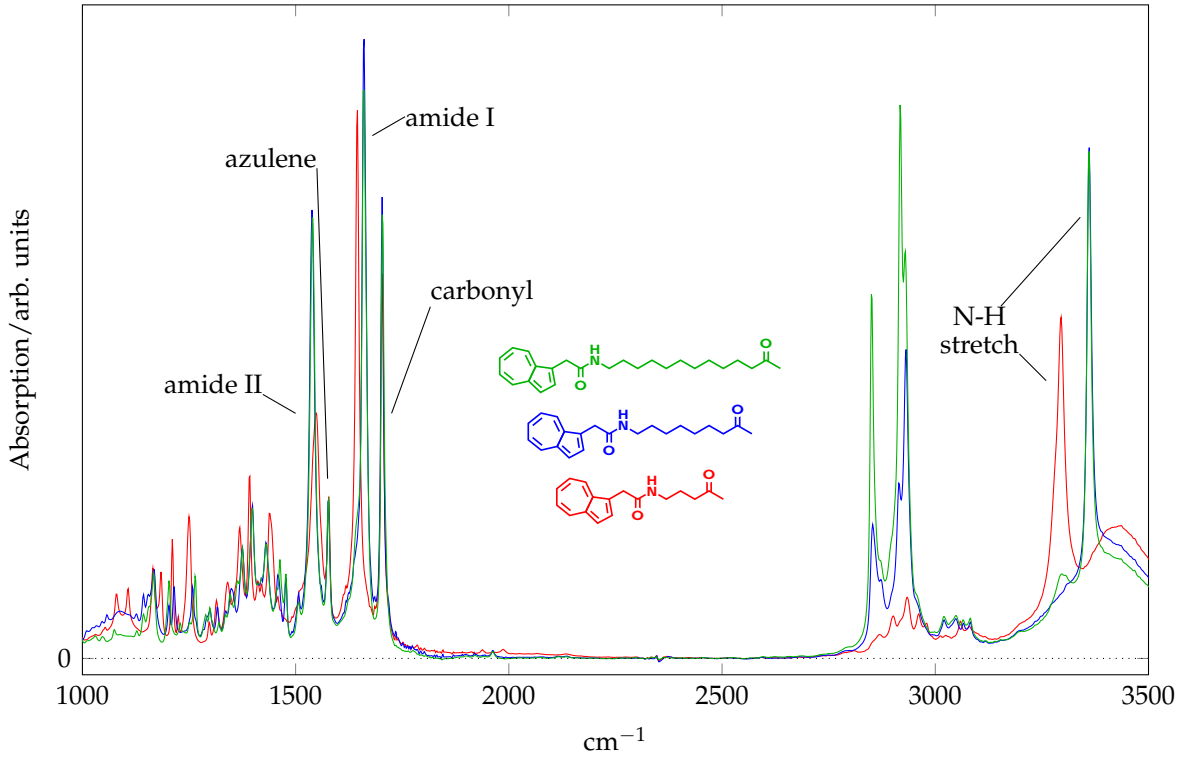
As both the N-H modes and the C-O modes are affected and since each of the given molecules contains only one amide group and no other potential hydrogen bond donors or acceptors, it must be concluded that the pure substances form networks of hydrogen bonds. The amount of absorption recorded at the positions of the isolated molecules may hint at the number of C=O and N-H groups not engaged in hydrogen bonds. It should be noted, however, that the substances are all oils at room temperature and that hence the potential networks of hydrogen bonds are probably subject to dynamic exchange processes. Also, no information on distribution of the sizes of these networks or their structural nature (e.g. chain lengths, frequency of possible bifurcations) can be obtained from these spectra.

3.2.2 *N*-(Oxo-alkyl)-2-(1-azulenyl)-acetamides

In the IR spectra of the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides in CH_2Cl_2 displayed in Figure 3.8(a), the functional groups shared in common with the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides lead to absorption bands at practically iden-



(a) In CH_2Cl_2

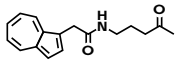


(b) As KBr pellets

Figure 3.8: Stationary IR spectra of *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides, normalized to the azulene absorption band at about 1580 cm^{-1}.

tical positions, namely: the amide II mode at 1519 cm^{-1}, the azulene ring distortion mode at 1578 cm^{-1}, the amide I mode at 1663–1665 cm^{-1}, and the amide A mode at 3417–3419 cm^{-1}. In addition, a ketone absorption can be found at 1712–1715 cm^{-1}, while – of course – the asymmetric stretching absorption of the azide group is missing.

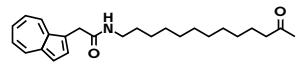
As in the case of the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, relative changes in intensity between different substances are apparent in the region of C-H stretching vibrations of 2800–3000 cm^{-1}. However, differences in the C-H bending region – to be expected at higher frequencies of about 1100–1400 cm^{-1}[105] – are by far not as pronounced. Once more, a peak occurring at 3054 cm^{-1} is due to the solvent CH_2Cl_2.

Although minor shifts can be observed for  of about 2 cm^{-1} for the amide and ketone absorptions to the blue and of the N-H stretching absorption to the red, there seems to be no significant influence of the chain length on the IR spectrum. Therefore, it may again be inferred that there is no coupling of the groups at the respective ends of the alkyl chain.

In the spectra of the corresponding KBr pellets, the signature of hydrogen bonds involving the amide group is nearly absent except for the molecule with the shortest chain of methylene groups. Here, the bands are shifted by about $+8$ cm^{-1} in the case of the amide II, by -16 cm^{-1} in the case of the amide I, and by -64 cm^{-1} in the case of the N-H stretching absorption. It should be noted, however, that the molecule containing the longest linker chain exhibits very small traces of the same signature. This is in sharp contrast to the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, where all pure substances exhibited marked shifts in the bands of the amide group regardless of the length of the PEG linker chain.

Clearly, these shifts can again be attributed to the formation of hydrogen bonds. Given that there is no significant shift of the ketone absorptions of the respective molecules, it seems very unlikely that the ketone group would engage in hydrogen bonding as an acceptor. However, the details of hydrogen bonding in the solid *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides obviously differ substantially from the situation in the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides. Only the molecule bearing the shortest linker chain shows strong indications of hydrogen bonding, but, at the same time, its spectrum utterly lacks absorptions at the positions indicative of isolated amide groups.

Since all *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides are solids, one likely explanation is that dynamic effects do not play a role with regard to the presence or absence of

hydrogen bonded networks. Rather, the structure of the substance determines their viability. The small absorptions indicative of hydrogen bonding in  may be interpreted as some inhomogeneity among the microcrystalline particles in the KBr pellets.

3.3 Investigation via UV pump IR probe spectroscopy

From the discussion of the stationary IR spectra, it is clear that no hydrogen bonding or other coupling affects the marker groups among themselves. This validates their use as monitors of vibrational energy transport and excludes potential alternative intramolecular energy transport pathways, such as in coupled amide carbonyl groups[1,65].

In this section, the pump-probe experiments conducted with the apparatus described in section 1.2 shall be discussed. These investigations were focused on three modes of prominence in the IR spectra: the amide modes I and II, the azulene ring deformation mode, and the azide or carbonyl mode. For illustrative purposes, those modes are depicted in Figure 3.9 as they have been obtained from the quantum chemical calculations described later (section 3.5).

3.3.1 Effect of vibrational excitation on IR spectra

For the experiments presented in the second part of this work, the "signal", as described in section 1.1, is the absorption of an IR laser beam and the predominant cause for spectral change is vibrational excitation. The spectral change observed is then owed to mechanical as well as electrical anharmonicity of the oscillators observed. Mechanical anharmonicity, i.e. the deviation of the restoring force F from strict proportionality to the displacement Δx of the oscillator from its equilibrium position x_0, influences the spectral position of the absorption of an oscillator. It also alters the intensity of the transition as it changes the wave functions of the oscillator involved in the transition. Electrical anharmonicity, i.e. the deviation of the dipole moment μ from proportionality to the displacement of the oscillator, on the other hand, is the cause of possibly changing intensity in absorption of an oscillator, as the intensity depends on the transition dipole moment.[107–110]

The usual theoretical treatment[111,112] starts by diagonalizing the potential energy V of second order in the displacements Δr_k^α of the αth Cartesian coordinate of

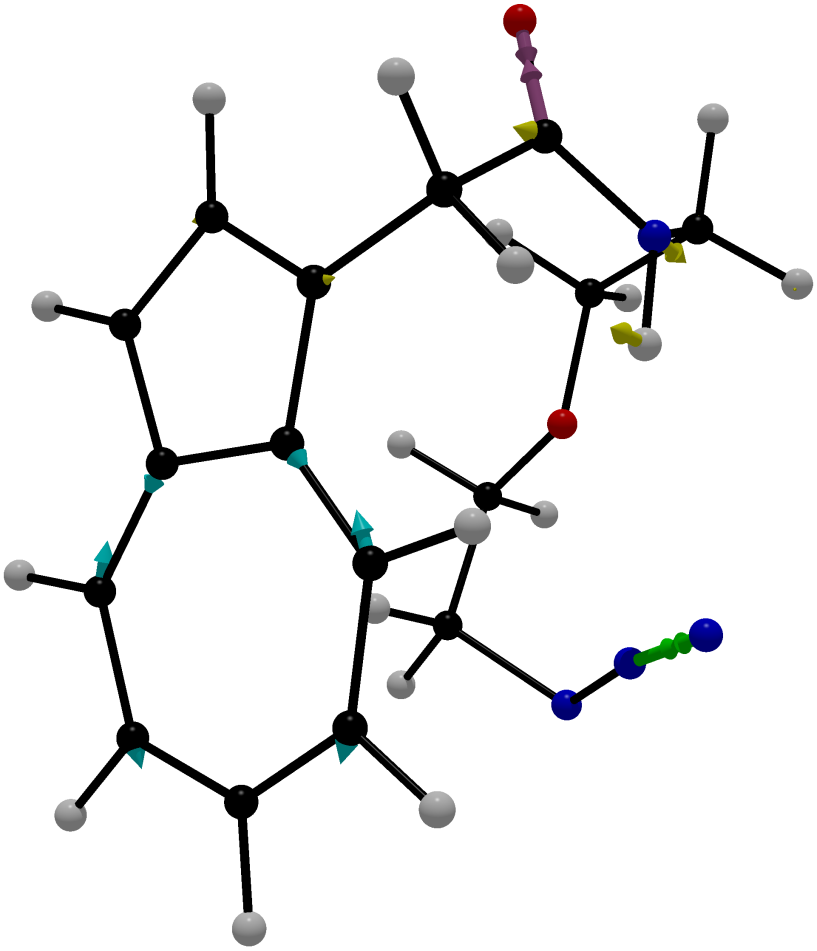


Figure 3.9: Visualization of the modes observed using the geometry used for calculating anharmonicity constants. Atoms are colored using the common convention (carbon – black, hydrogen – white, nitrogen – blue, oxygen – red), while displacement arrows are colored as follows: cyan – azulene ring distortion mode, yellow – amide II mode, purple – amide I mode, green – azide asymmetric stretching mode.

the kth atom from the equilibrium geometry of the molecule

$$V \approx \frac{1}{2} \sum_{k,l,\alpha,\beta} f_{kl}^{\alpha\beta} \, \Delta r_k^{\alpha} \, \Delta r_l^{\beta} \qquad \text{with} \quad \alpha, \beta = x, y, z$$

to obtain dimensionless normal coordinates q_i leading to

$$V \approx \frac{1}{2} \sum_i \omega_i \, q_i$$

where ω_i is the normal frequency of the ith normal mode. Adding higher terms to the

potential

$$V = \frac{1}{2}\sum_i \omega_i\, q_i + \frac{1}{3!}\sum_{ijk} \phi_{ijk}\, q_i\, q_j\, q_k + \frac{1}{4!}\sum_{ijkl} \phi_{ijkl}\, q_i\, q_j\, q_k\, q_l + ...$$

then introduces the above-mentioned mechanical anharmonicity. Similarly, electrical anharmonicity[108,112] originates from the higher order terms of the expansion of the dipole moment μ

$$\mu = \underbrace{\mu_0}_{\text{static}} + \underbrace{\sum_i \mu_i\, q_i}_{\text{harmonic}} + \frac{1}{2!}\sum_{ij} \mu_{ij}\, q_i\, q_j + \frac{1}{3!}\sum_{ijk} \mu_{ijk}\, q_i\, q_j\, q_k + ...$$

in normal coordinates q_i.

From the experimental perspective, transition energies have usually received greater attention than intensities. The empirical formula[107,109–111,113,114] of the energy E of a vibrational state $(v_1, ...)$, reminiscent of the vibrational Dunham expansion[115],

$$E(v_i, ...) = \sum_i \omega_i\ (v_i + 1/2) + \sum_{i \leqslant k} x_{ik}\ (v_i + 1/2)\ (v_k + 1/2) + ... \tag{3.2}$$

where v_i is the vibrational quantum number of the ith oscillator relative to the minimum energy of the electronic state, leads to transition energies[2]

$$\tilde{\nu}_i(v_1, ...) = \tilde{\nu}_i^0 + 2x_{ii}\, v_i + \sum_{k \neq i} x_{ik}\, v_k \quad \text{with} \quad \tilde{\nu}_i^0 = \omega_i + 2x_{ii} + \frac{1}{2}\sum_{k \neq i} x_{ik} \tag{3.3}$$

that are shifted depending on the quantum numbers of the observed mode (x_{ii}, *diagonal* anharmonicity) as well as the other modes of the molecule (x_{ik} with $i \neq k$, *off-diagonal* anharmonicity). $\tilde{\nu}_i^0 = \omega_i + 2x_{ii} + \frac{1}{2}\sum_{k \neq i} x_{ik}$ is the fundamental frequency, i.e. the frequency of the transition of the ith mode from its ground – to its first excited state with all other modes being in their ground states. It should be noted that additional coupling constants may be necessary for the satisfactory treatment of degenerate modes.[113] Using a perturbative treatment, it is then possible to derive the constants of anharmonicity x_{ij} from the third and fourth order potential constants ϕ_{ijk} and ϕ_{ijkl}

from above.[109–111,114] To second order, the corresponding expressions are[111,116,117]:

$$\begin{aligned} x_{ii} &= \frac{1}{16}\left(\phi_{iiii} - \sum_m \frac{\phi_{iim}^2}{2}\,\Omega_{im}\right) \\ x_{ik} &= \frac{1}{4}\left(\phi_{iikk} - \sum_m \frac{\phi_{iim}\,\phi_{kkm}}{\omega_m} - \sum_m \frac{\phi_{ikm}^2}{2}\,\Omega_{ikm}\right) \\ &\quad + \sum_{\alpha=x,\,y,\,z} B_\alpha^{\mathrm{eq}}\,(\zeta_{ik}^\alpha)^2\left(\frac{\omega_i}{\omega_k} + \frac{\omega_k}{\omega_i}\right) \end{aligned} \tag{3.4}$$

with

$$\begin{aligned} \Omega_{im} &= \frac{1}{2\omega_i + \omega_m} + \frac{4}{\omega_m} - \frac{1}{2\omega_i - \omega_m} \\ \Omega_{ikm} &= \frac{1}{\omega_i + \omega_k + \omega_m} + \frac{1}{-\omega_i + \omega_k + \omega_m} + \frac{1}{\omega_i - \omega_k + \omega_m} \\ &\quad - \frac{1}{\omega_i + \omega_k - \omega_m} \\ B_\alpha^{\mathrm{eq}} &= \frac{\hbar}{4\pi\, c_0\, I_\alpha^{\mathrm{eq}}} \\ \zeta_{ik}^\alpha &= \sum_m L_{im}^\beta\, L_{km}^\gamma - L_{km}^\beta\, L_{im}^\gamma \\ &\quad \alpha,\,\beta,\,\gamma = x,\,y,\,z \end{aligned}$$

where B_α^{eq} are the rotational constants and I_α^{eq} the moments of inertia taken at the equilibrium geometry of the molecule. c_0 is the speed of light in vacuum, and ζ_{ik}^α are the Coriolis constants to be obtained from L_{km}^α, the elements of the matrix $\mathbf{L}$. The latter is the matrix transforming Cartesian displacement coordinates Δr_m^α of the α coordinate of the mth atom to normal coordinates Q_k as follows:

$$\Delta r_m^\alpha = \sum_k L_{km}^\alpha\, Q_k \qquad \text{with} \qquad Q_k = q_k \sqrt{\frac{\hbar}{2\pi\, c_0\, \omega_k}}. \tag{3.5}$$

In the case of a Fermi resonance, divergent terms need to be dropped from Ω_{im} and Ω_{ikm} and energies of the resonant states may be obtained by explicitly diagonalizing the appropriate Hamiltonian.[116,117] Note that the weighting factors Ω_{im} and Ω_{ikm} defined here do not follow the notation used in Ref. [116].

If changes in intensity upon vibrational excitation as well as higher-order shifts are negligible, the mere knowledge of the constants of anharmonicity x_{ij} is sufficient to

simulate a difference spectrum and thus help explain the underlying energy distribution within an ensemble of molecules (see subsection 3.5.1).[2,30,118,119] In the general case of a medium-sized organic molecule, individual vibrational states will mostly not be discernible in the IR difference spectrum and hence it is not directly evident which modes are excited and contribute to anharmonic shifting. For the case of statistical intramolecular equilibrium, a nearly thermal distribution can be assumed (see subsection 3.5.2), and it has been shown that fitting a thermal spectrum to the experimental data can be successful[2,118].

For the experiments considered later in this work, however, the starting condition is obviously a molecule in which the vibrational energy is fairly well localized in an azulene moiety and IVR will only gradually lead to a situation of randomized energy distribution among all modes. Since a population of excited states of the "sensor" modes themselves seems unlikely due to their relatively high frequencies (carbonyl – and azide modes), it has been asserted that the spectral shift of their absorptions is caused mainly by those modes which "spatially overlap" with the sensor mode.[8,98] Spatial overlap is to be understood as modes having their greatest amplitudes of displacement located at the same atoms, increasing the coupling constants of those modes with the sensor mode.[98] In a sense, the observed modes can thus be thought of as local heat sensors.[72] It must be emphasized that albeit in most cases researchers have in the past resolved to interpret reduced optical density at the band origin itself, there is no per se or a priori direct proportionality of this quantity to energy or temperature (if that concept is to be considered adequate at all) in the vicinity of a functional group.

3.3.2 The azulene ring distortion mode

Spectral evolution

Figure 3.10 shows a number of transient spectra of the azulene ring distortion mode obtained in CH_2Cl_2. It should be noted that while a good baseline correction could be performed for the higher frequency edge of the spectra, the lower frequency detection range does not provide any points or intervals that would allow for the application of the customarily used procedures described in section 1.3. This is due to overlapping of the bleach of the amide II mode starting around 1540 cm^{-1} with the absorption of the vibrationally hot azulene mode. Accordingly, spectral information in said region has to be considered somewhat less reliable than in the higher frequency region.

The earliest spectrum indicates complete change in absorption characteristics of

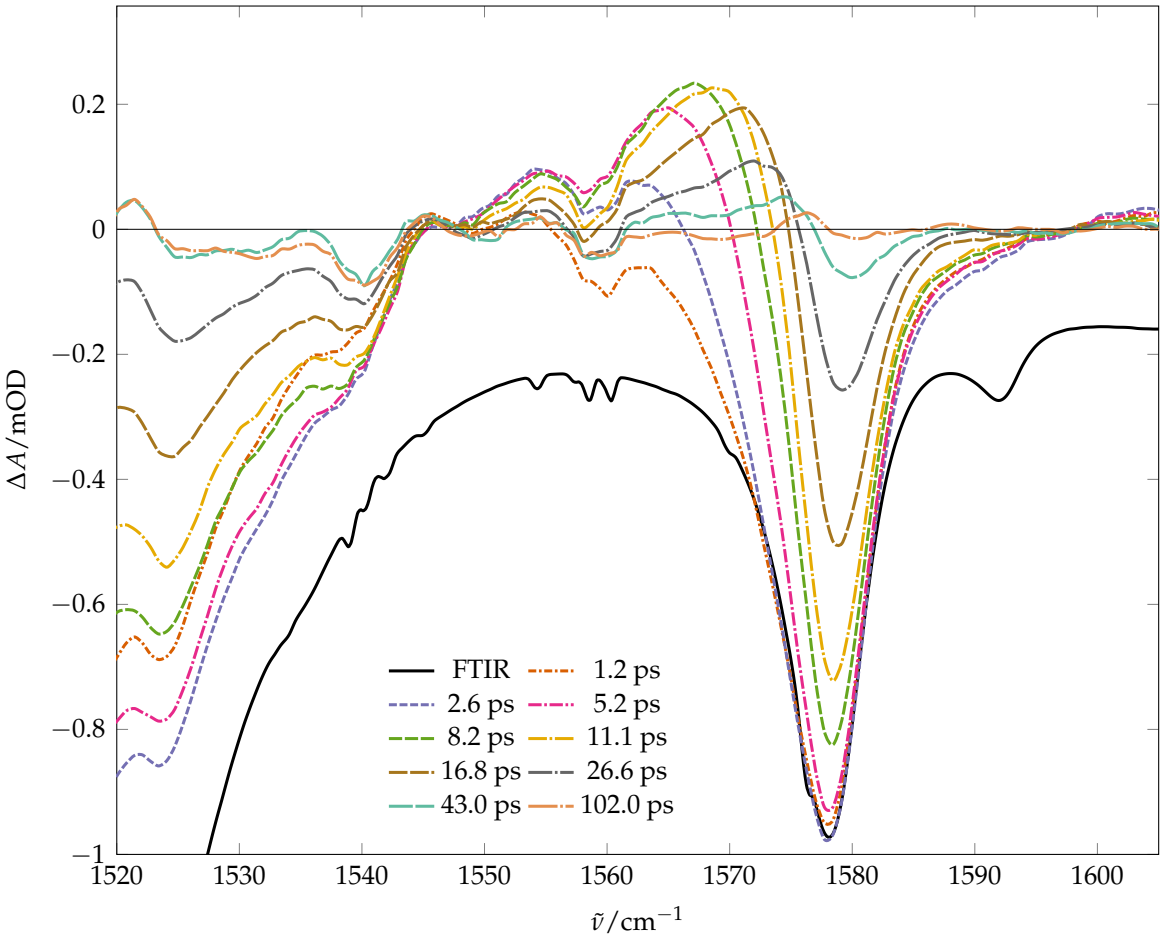
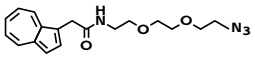


Figure 3.10: Transient difference spectra of the azulene ring distortion absorption of  in CH_2Cl_2 after excitation at 610 nm.

the excited species previously causing the original absorption of the ring distortion mode, as is obvious by its near-identity with the stationary spectrum. Assuming a great frequency shift upon electronic excitation, a partial explanation may be due to the ongoing relaxation from the S_1 state[83–85], which was reported to be bi-exponential with the slower component having a time constant of about one picosecond for pure azulene in cyclohexane[85]. Much of the population which was initially excited electronically, however, appears to be highly excited vibrationally after its return to the electronic ground state, as indicated by a tremendous red-shift ranging far into the absorption of the amide II mode. The room temperature absorption of the latter is itself bleached due to harmonic energy flow (see subsection 3.1.3) and anharmonic coupling to excited modes (see subsection 3.3.1). It is not quite clear, if the small shoulder at 1590 cm^{-1} is due to impurities. In either case, there is practically no blue-shifted absorption of the azulene ring distortion mode, hinting at all anharmonic constants being

less than zero for this mode.

As energy transfer beyond harmonic energy flow within the molecule itself and to a lesser degree also to the solvent progresses, the bleach of the amide II mode appears to increase. In part, however, this is certainly owed to the cooling of the azulene unit and the consequent return of the absorption of the ring distortion mode to its original position, rather than energy transfer to the amide unit. In the last spectrum, nearly full recovery of the room temperature vibrational population seems to be achieved. The very small residual signal reflects an overall heating of the sample solution by the release of energy from the excited solute to the solvent. This has been shown to amount to less than a Kelvin under similar conditions. The magnitude of the residual signal compared to the direct excitation is owed to the greater number of participating molecules: all within the focus of the probe laser versus only a small fraction of those in the pump lasers focus in the case of direct excitation.[1,72,78]

Temporal evolution

Kinetic traces were obtained by picking the minimum signal value (within a suitable spectral window containing the azulene ring distortion mode) at each delay. In order to compare the different velocities of vibrational cooling, an asymmetric sigmoidal "growth and saturation" function first described by Gompertz[120] of the form

$$f(t) = A \ \exp\Big(-\exp\big(\gamma(t_0 - t)\big)\Big) + c \tag{3.6}$$

was used. This function provides sufficiently many parameters to fit the data reasonably well in an empirical manner. It is, of course, not intended to be understood to have any theoretical justification by the actual microscopic processes taking place in IVR and VER, but rather serves as a phenomenological description.

Naturally, picking the minimum value will result in negative values due to noise even after the actual signal has vanished. Accordingly, the offset c was subtracted from both the data and the fit result. While this also eliminates the temperature rise, as discussed in section 3.3.2, the latter is fairly small and nearly of the same magnitude as the noise of the signal. This makes a better treatment, i.e. the separation of the temperature rise from the IVR signal, infeasible. Furthermore, both fit and experimental trace were normalized using the amplitude of the fit A. The result of applying this protocol to all the data obtained is shown in Figure 3.11.

Note should be made that there is no immediate quantitative relationship between

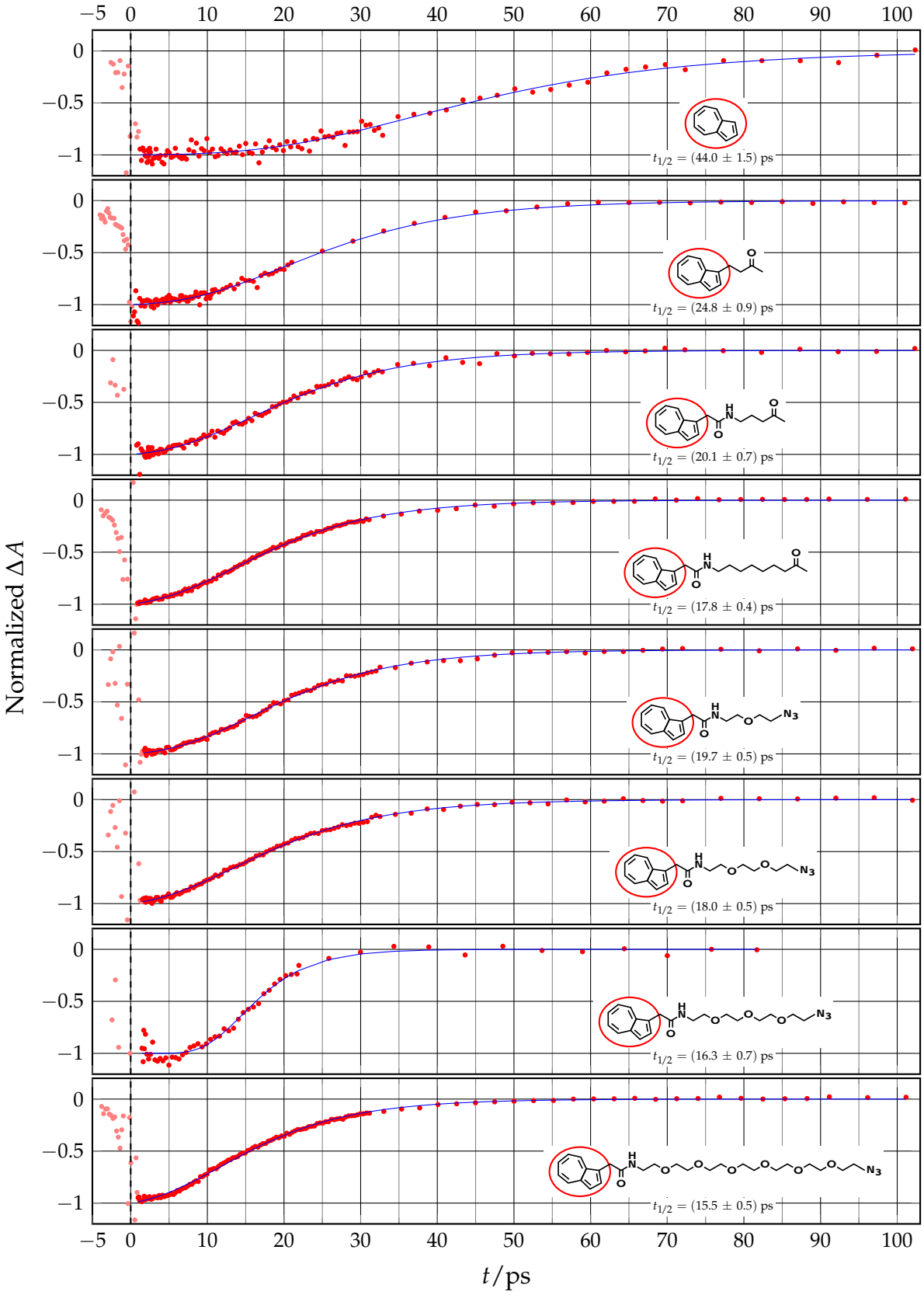


Figure 3.11: Kinetic traces of the azulene ring distortion mode. Measured in CH_2Cl_2 using an excitation wavelength of 610 nm.

the minimum signal value on the one hand, and the average energy content or temperature or any other physically relevant quantification of the energy content of the observed molecule on the other (cf. subsection 3.3.1). However, as the initial state is well-defined by the preparation and identical for all substances – namely the excitation and subsequent internal conversion of the azulene moiety – any fixed fraction of the magnitude of the respective initial signal is suited for quantifying the progress of vibrational cooling. Due to the same original value, i.e. the value at $t = 0$, this fraction will also represent the same energy content in all the species, provided that the observed mode is not coupled strongly to any modes not common to all substances (i.e. localized outside the azulene moiety) and that IVR leads to randomization on a sufficiently fast timescale.

With an exponential rate law for the loss of energy to the solvent[121,122], this fraction value is directly proportional to the time constant of VET τ_{VET}. The chosen function Equation 3.6 makes it fairly simple to extract the half-step value $t_{1/2}$ defined by the following equation:

$$f(t_{1/2}) = \frac{A}{2} + c$$

or simply by reading off the abscissa value corresponding to an ordinate value of one half in the normalized and offset-corrected graph.

Comparison of substances

The value of $t_{1/2} = (44.0 \pm 1.5)$ ps for azulene is substantially larger than those reported for the cooling constant of azulene in comparable solvents (e.g. 14.6 ps in acetonitrile) when pumped between 580 and 600 nm[121]. The value given by Schultz *et al.* of (19 ± 3) ps for azulene in CH_2Cl_2[122], on the other hand, was not calibrated to reflect energy loss, but rather decay of spectral signal at 740 nm after pumping at 590 nm. Certainly, however, the $t_{1/2}$ values are larger than the characteristic decay time, meaning that they indicate a point at which the energy content of the azulene unit has arrived at substantially less than $1/e$ of its initial value.

As vibrational ground state recovery is measured here, the explanation of the wavelength dependence of relaxation times[122,123] holds here as well: a decrease in temperature affects the occupation of high energy vibrational states more strongly than that of states lower in energy. Here this does of course include states coupling to the observed mode. The spectra presented in Figure 3.10 illustrate this effect, alongside

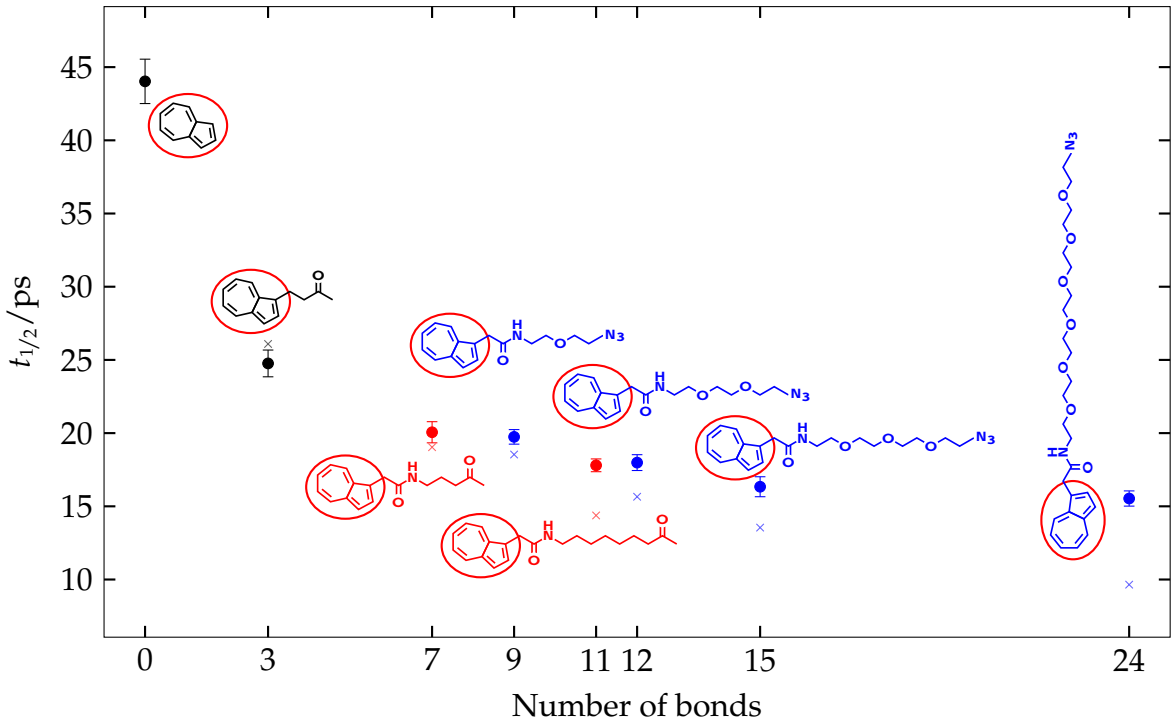


Figure 3.12: Comparison of the $t_{1/2}$ value of the azulene distortion mode for all substances investigated. All measurements were conducted in CH_2Cl_2 using a pump-wavelength of 610 nm. × represent values extrapolated using the value of azulene and assuming proportionality of $t_{1/2}^{-1}$ to the number of vibrational degrees of freedom n_{vib}.

with the initial lag time of the kinetic traces of Figure 3.11.

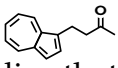
In Figure 3.12, the $t_{1/2}$ values for all the substances investigated are compared. As was already evident from Figure 3.11, longer aliphatic residues tend to speed up the loss of energy to the solvent. In a very naive picture, neglecting possible resonances of solvent and solute vibrations as well as the nature of the energy distribution among the vibrational degrees of freedom of the solute, one might assume this energy transfer to be more or less proportional to the number of vibrational degrees of freedom n_{vib}. More sophisticated descriptions might include the size of the "surface" of the molecule exposed to the solvent and thereby the probability of collisions with the solvent. This, however, would either require additional assumptions on the specific conformation of the molecule – or rather distribution of conformations in many cases – while not promising substantially improved insight at the same time.[1]

Assuming that energy transfer to the solvent is described best by the number of

[1] A very simple-minded approach assuming that a cylindrical shape of the molecule and thus $t_{1/2} \propto (a+\ell)^{-1}$, where ℓ is the approximate length and a is the approximate diameter of that cylinder (assuming the molecule is not coiled or folded), leads to rather poor results.

vibrational degrees of freedom of the solute rather than the number of collisions with the solvent, and further asserting that $t_{1/2}$ is a good quantification of that process as laid out in section 3.3.2, the value of $t_{1/2}$ of azulene can be used to predict those of the other substances. In Figure 3.12, the results are depicted by ×. While the agreement with the first three experimental data points is remarkable, it is simultaneously obvious that, while the asymptote of the prediction is zero for $n_{\mathrm{vib}} \to \infty$, the real asymptote appears to be around 14 ps.

This divergence may well be interpreted to reflect – beyond the shortcomings of the naive assumptions – the inability of a number of oscillators to transfer energy to the solvent as they are initially not excited. Ultimately, this is the signature of the initial energy distribution following the internal conversion of the azulene moiety as well as of the ensuing competition between IVR and VET.

It has been found by the Schwarzer group[7,69,75,81,82] that energy transfer from a vibrationally excited azulene moiety to an anthracene unit via an aliphatic chain incurs congestion along the first four atoms of the chain. The present results confirm these findings, as for longer chains than  the values of $t_{1/2}$ are almost at the apparent value of the asymptote. This implies that IVR up to about four or five bonds from the azulene moiety is very fast – and energy can be transferred to the solvent by those atoms practically at the same rate as by the azulene moiety itself – but then slows and is finally in competition with VET.

3.3.3 The amide I mode

Spectral evolution

A sample of the spectral evolution of the amide I mode is displayed in Figure 3.13. Even at rather short delays after the pump pulse, a substantial bleach of the original absorption and shift of optical density to lower frequencies is visible. This is a manifestation of what has been dubbed "harmonic energy flow"[7,75,81], i.e. the excitation of normal modes involving non-azulene atoms as a consequence of the internal conversion of the azulene moiety and the coupling of the amide I mode to them. Contrary to the azulene mode discussed in section 3.3.2, the signal of the amide I mode exhibits a further rise to reach its maximum at approximately 2.15 $\pm$ 0.05 ps (see section 3.3.3). The agreement of the higher frequency flank of the bleach with the stationary is quite good (notice that the shoulders in the FTIR spectrum do not belong to the amide I absorption), indicating practically no anharmonic constants greater than zero.

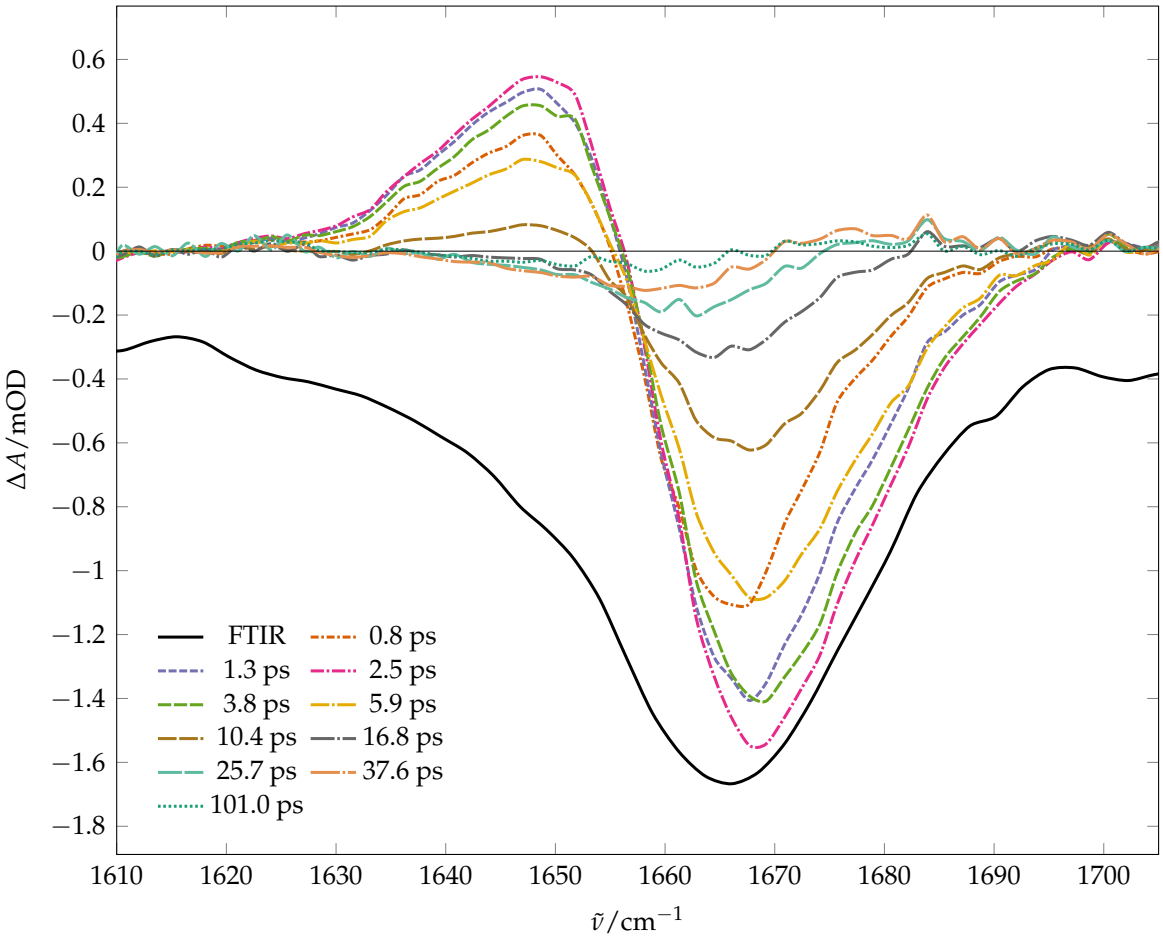
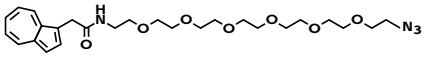


Figure 3.13: Transient difference spectra of the amide I absorption of  in CH_2Cl_2 after excitation at 610 nm.

An evident characteristic of the signal of the amide I mode is that its integral is non-vanishing, i.e. the absorption of the newly populated vibrational states upon excitation (positive signal) is less intense than that of the in turn depopulated states (negative signal, "bleach"). Accordingly, the harmonic approximation of the transition dipole moment is not valid in this case and a computation of spectra according to subsection 3.3.1 cannot be supposed to agree with the experimental data without a better-than-harmonic description of the transition dipole moment.

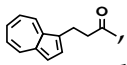
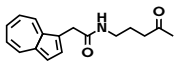
Close inspection of spectra of similar bleach intensity before and after reaching maximum signal intensity – two pairs of which are given in Figure 3.13 – yields another insight. Most prominently, the maximum of the bleach – but to a lesser degree also other features – is shifted to higher frequencies by a few wavenumbers in spectra recorded at delays after the maximum signal as compared to those recorded before that point. As delays increase, the maximum of the bleach gradually shifts to smaller

frequencies, even beyond the position it exhibited in the spectra of very short delays. Presumably, the "later" spectra depict a situation in which IVR is largely complete; the average intramolecular energy distribution is nearly "thermal" (see subsection 3.5.2); and energy release to the solvent is prevalent. Consequently, the red-shift of the bleach in "earlier" spectra is a signature of the non-equilibrium distributions occurring during progressing IVR and thus an indication of its ballistic nature. It can tentatively be regarded as the signature – although a weak one – of a traveling wave packet as has been suggested based on the conical intersection between the S_1 and S_0 states of azulene[124].

Finally, the residual signal due to the rise of the temperature of the sample solution is not quite as marked as in the case of the azulene ring distortion mode (see section 3.3.2), thus indicating a somewhat smaller sensitivity of the amide I band to temperature.

Temporal evolution and comparison of substances

The evaluation used for the ring distortion vibration of the azulene moiety as described in section 3.3.2 was carried out analogously for the amide I vibration, i.e. the CO stretching vibration of the amide group. Here, however, a bi-exponential function was used to fit the data. Subtraction of an offset – the asymptotic value – and normalization to the maximum value were carried out as before. On the same grounds as stated for the azulene absorption in section 3.3.2, this can be regarded an acceptable neglect of the rising temperature of the solution. The result is displayed in Figure 3.14. As a characteristic figure, the maximum (negative) signal value $t_{\max}$ was chosen, since this is the least ambiguous parameter and has been used by others before[8,63].

Within error limits, all values of $t_{\max}$ agree and even the kinetic traces are all fairly similar, which may at first seem contradicting the results obtained for the values of $t_{1/2}$ for the azulene moieties in section 3.3.2. First, however, the differences in the signals of the azulene moieties are largest in the two molecules not containing any amide group, i.e. azulene and  , which obviously cannot be compared with the amide I signals here. Second, among the remaining substances, differences in $t_{1/2}$ are minute and occur only at fairly large delay values ($\gtrsim$ 10 ps), when IVR is practically complete (cf. section 3.3.4). Before that point the kinetic traces obtained for the azulene ring distortion mode are at their initial plateau value. Finally, the value of $t_{\max}$ obtained for  being somewhat larger than the remaining values might be

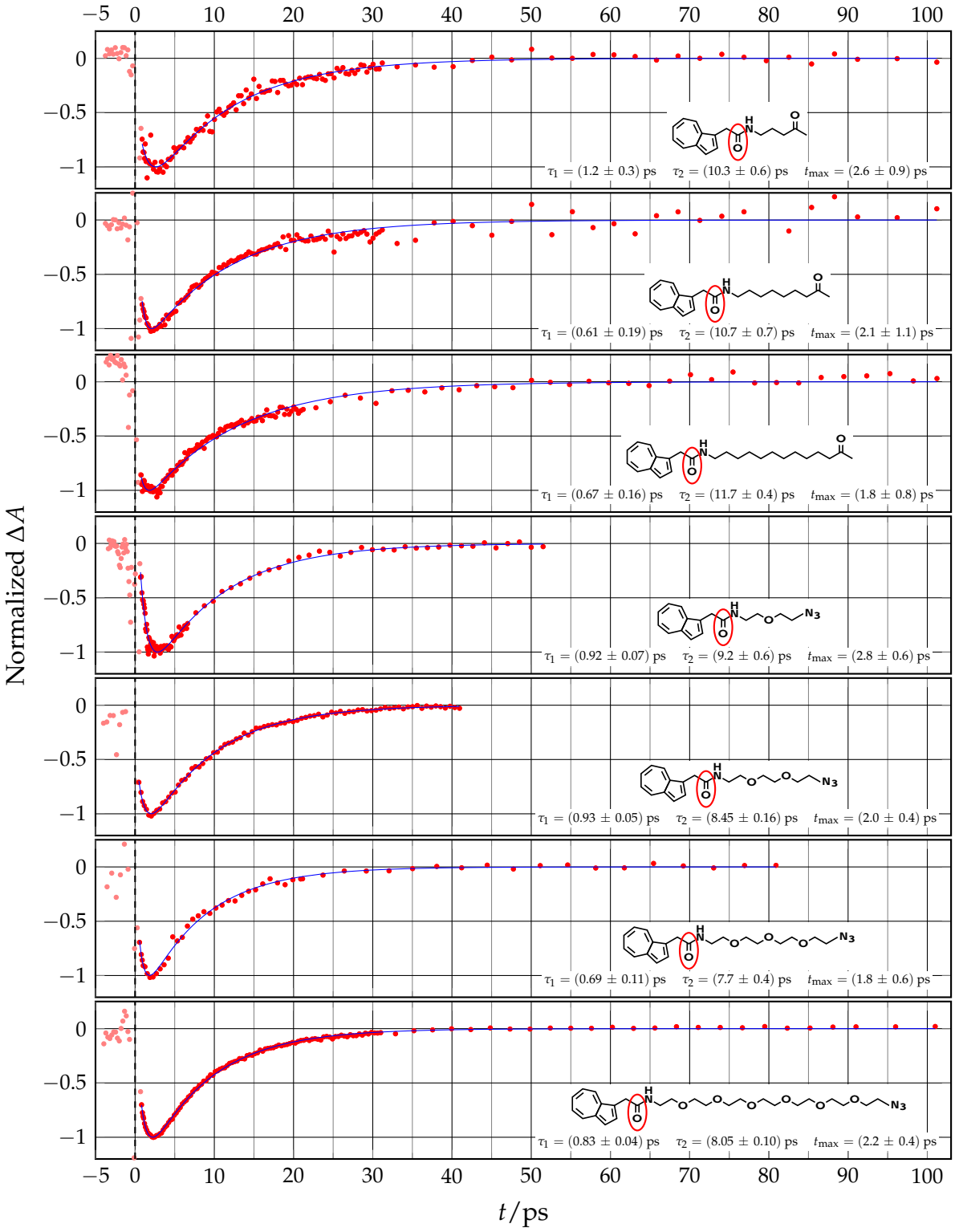


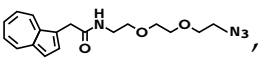
Figure 3.14: Kinetic traces of the amide I mode. Measured in CH_2Cl_2 using an excitation wavelength of 610 nm. τ_1 and τ_2 are the time constants of the bi-exponential fit function.

reminiscent of that effect, although it has to be conceded that its error limits contain the overall average of 2.15 $\pm$ 0.05 ps. Generally speaking, while the amide I mode signal depicts IVR and, more precisely, the similarity of IVR in the substances investigated, the signal of the azulene group discussed in section 3.3.2 is a better sensor of the succeeding VET processes.

3.3.4 The azide asymmetric stretching mode and the carbonyl mode

Spectral evolution of the azide asymmetric stretching mode

In four substances used for this study, an azide group was employed as a sensor to monitor the progress of IVR along a polyethylene glycol type chain. With regard to the frequencies of the three main azide modes, symmetric stretching (ν_1), bending (ν_2), and asymmetric stretching (ν_3), the vibrational spectrum of aliphatic azides generally resembles more that of the azide anion than that of the corresponding radical[103,125,126]. The absorptions of the symmetric stretching and the bending vibration lie in the fingerprint range[103,126]. This fact makes them unsuitable for IVR monitoring using difference IR spectroscopy, because this region is opaque as a result of solvent absorptions and congested by various other absorptions in the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides. Further, the intensity of the of the ν_2 mode is predicted to be very small[103] and would fall outside the spectrally available range of the apparatus used.

With the solvent chosen for this study, CH_2Cl_2, the asymmetric stretching vibration remains observable. Accordingly, it presents a natural choice for IVR monitoring. The data obtained for all of the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides differ mainly in their temporal and much less in their spectral characteristics. Therefore, only one compound, namely  , shall be discussed here as an example.

Difference spectra recorded at various times are shown in Figure 3.15. Obviously, these spectra depict the superposition of red-shifted absorption (positive signal) and a bleached room temperature spectrum (negative signal), the latter with a peak around 2108 cm^{-1} (cf. subsection 3.2.1). Starting with a non-zero signal, the maximum of both positive and negative signals is reached after a little more than seven picoseconds for this compound, followed by a much slower decay, leaving only little residual heat after one hundred picoseconds, which reflects a general rise in temperature of the sample solution (see section 3.3.2).

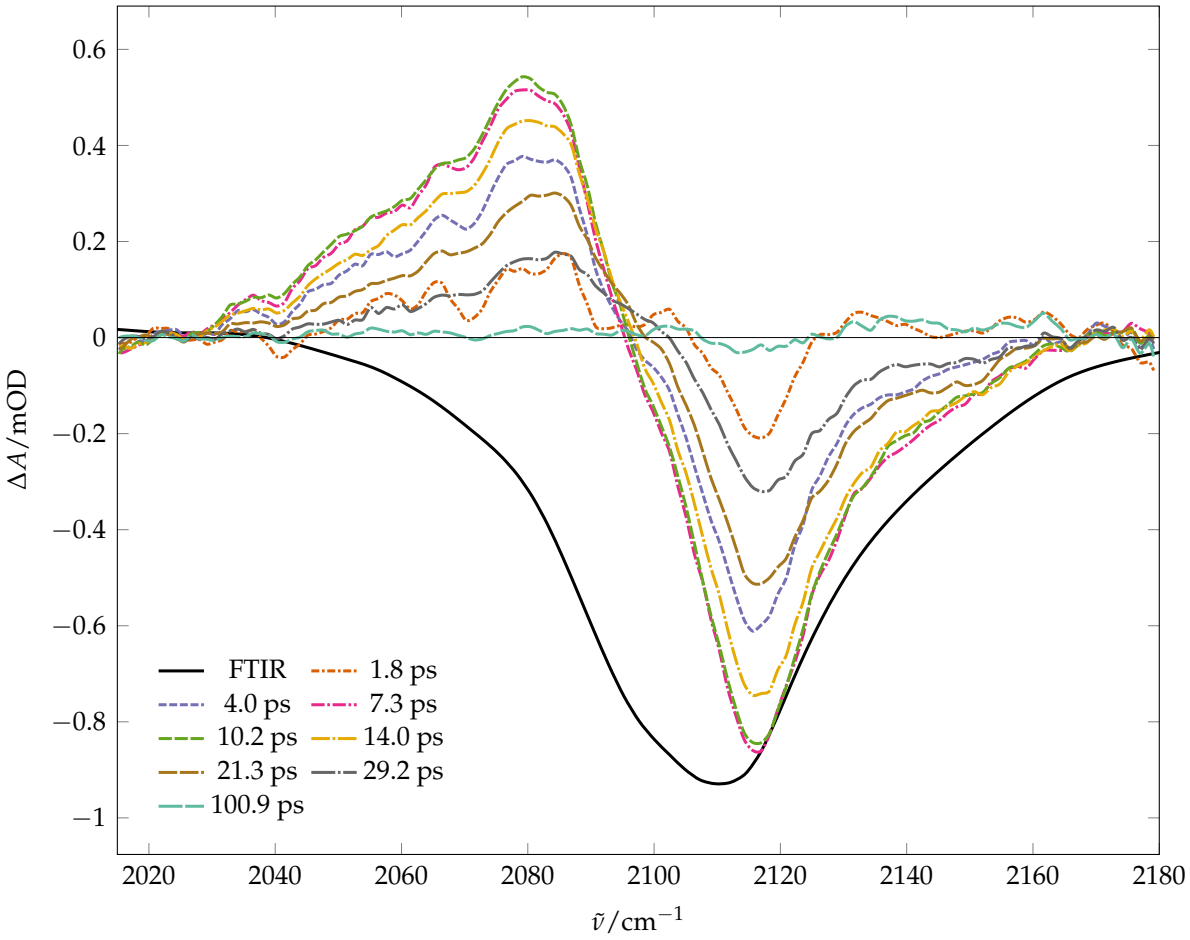
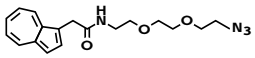


Figure 3.15: Transient difference spectra of the azide absorption of  in CH_2Cl_2 after excitation at 610 nm.

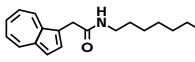
As in the case of the amide I mode (section 3.3.3), the overall integral of the signal is less than zero, indicating some decline in oscillator strength upon excitation of anharmonically coupled modes. Contrary to the amide I and azulene modes, however, the transient spectra of the asymmetric azide absorption deviate considerably from the stationary spectrum, which could be explained by an appreciable number of anharmonic coupling constants bearing positive sign. Also, there is no significant shift of the bleach as the signal evolves and thus no marked difference of the signatures of IVR and post-IVR energy distributions as in the case of the amide I mode. Finally, the signal at short delay times due to harmonic energy flow is, as expected, much smaller than in the case of the amide I mode.

It is noteworthy that the peak positions of both the bleach and the red-shifted positive signal do not change significantly throughout the IVR and ensuing VER processes. This differs substantially from what was observed for the azulene ring distortion mode

and from what one would normally expect in a medium-sized molecule with a certain distribution of anharmonicity constants to occur (see, e.g. Figure 2.5): the gradual shift of the absorption as the internal energy of the molecule increases or decreases. This can tentatively be interpreted as a limited number of specific modes (or even individual vibrational states) coupling to the asymmetric azide mode at their respective anharmonicity constants. The signal of the azide mode itself would then reflect the population of these modes (states). As no shift of its spectral signal occurs, it would have to be inferred that energy is released to the solvent directly from these modes (or states) rather than redistributed within the molecule.

From the calculated anharmonic constants of the azide asymmetric stretching mode (subsection 3.5.5) it appears that it reflects only upon a very limited part of the molecule, i.e. essentially almost exclusively the azide group itself. Thus, it would not be a good reporter for IVR taking place throughout the remainder of the molecule, and the latter should not be ruled out based on the spectrum of the azide mode alone.

Spectral evolution of azulene ring distortion, amide I, and carbonyl mode of *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides

The characteristic absorptions evaluated for the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides used in this work largely resemble those of the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, as exemplified in Figure 3.16 for  . Much of what has been said regarding the asymmetric azide mode (section 3.3.4) applies to the carbonyl mode as well: The initial signal due to harmonic energy flow is much smaller than for the amide I mode and there is no appreciable shift of the maximum negative signal that would allow to discern different stages of progress of IVR. Interpretation of such features on the low frequency edge of the absorption is questionable due to the overlapping bleach of the amide I absorption. As with the asymmetric azide absorption, some of the anharmonic constants seem to be greater than zero, since the high frequency portion of the bleach does not match the stationary spectrum. With regard to the integral of the difference signal, it seems that of all the absorptions considered, the electrically harmonic approximation is best fulfilled for the carbonyl absorption, considering that parts of the positive signal reach over into – and are consequently masked by – the negative signal of the amide I mode.

The features previously discussed of the azulene ring distortion mode and the amide modes are consistently found in *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides as

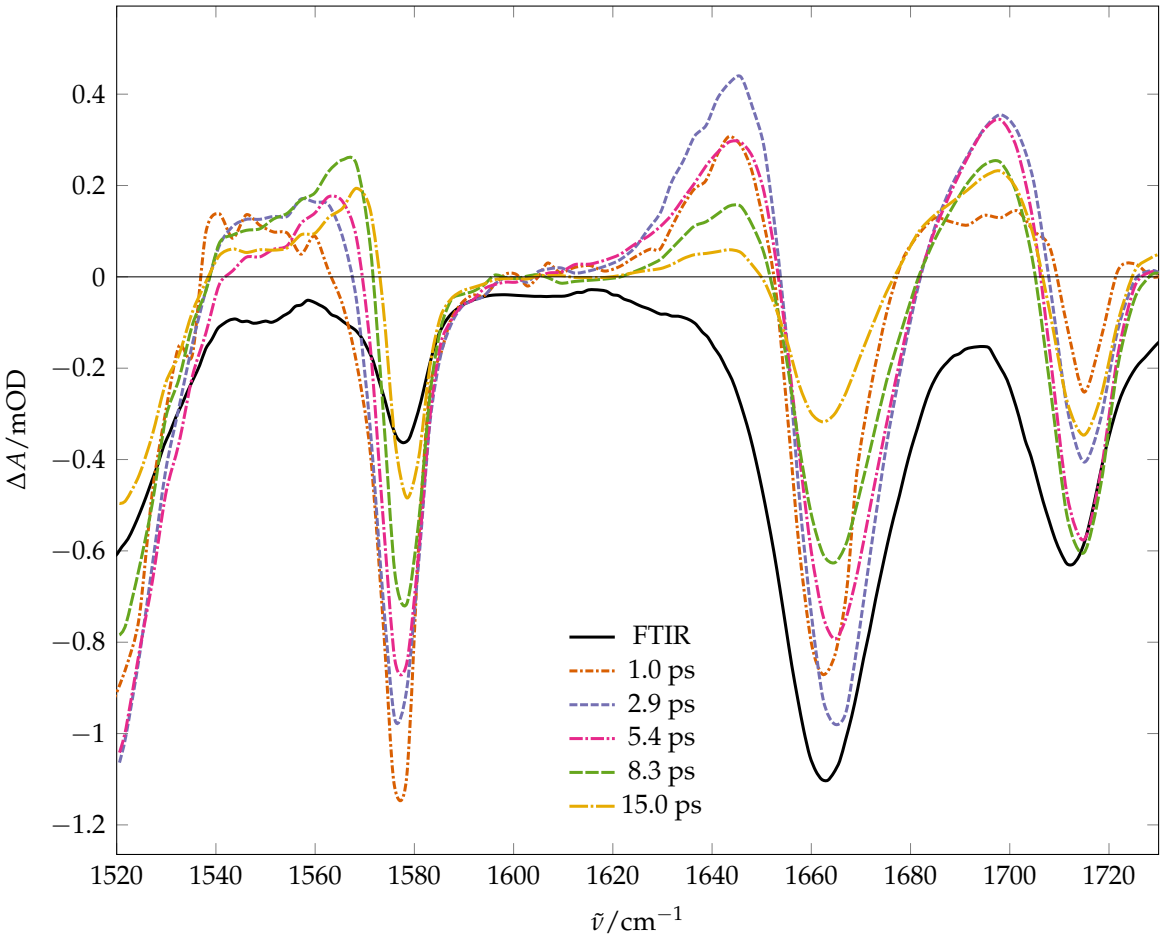
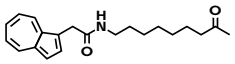


Figure 3.16: Transient difference spectra of  in CH_2Cl_2 after excitation at 610 nm. The low frequency edge is somewhat unreliable as alignment of individual spectra was hampered by overlapping absorptions.

well. It is evident that the bleaches of the azulene ring distortion mode and also of the amide II mode are much stronger than those of the amide I and carbonyl modes, when comparing them to their respective stationary absorptions. To some extent this may be explained by "spatial overlap"[8,98] with the site of excitation, i.e. the azulene moiety. However, electrical and mechanical anharmonicities would probably also lead to differing signal intensities in a thermalized ensemble, which would have to be considered as well. Finally, as hinted at above, the mismatch of the high frequency edge of the bleach of the amide I absorption can mostly be attributed to the redshift of the carbonyl absorptions, thus upholding the earlier assessment that most anharmonic constants are less than zero for the amide I mode. Considering this overlap of both signals, it is not obvious whether the absorption of the mode shifts continuously upon excitation, as the azulene ring distortion mode, or remains fixed, as the azide

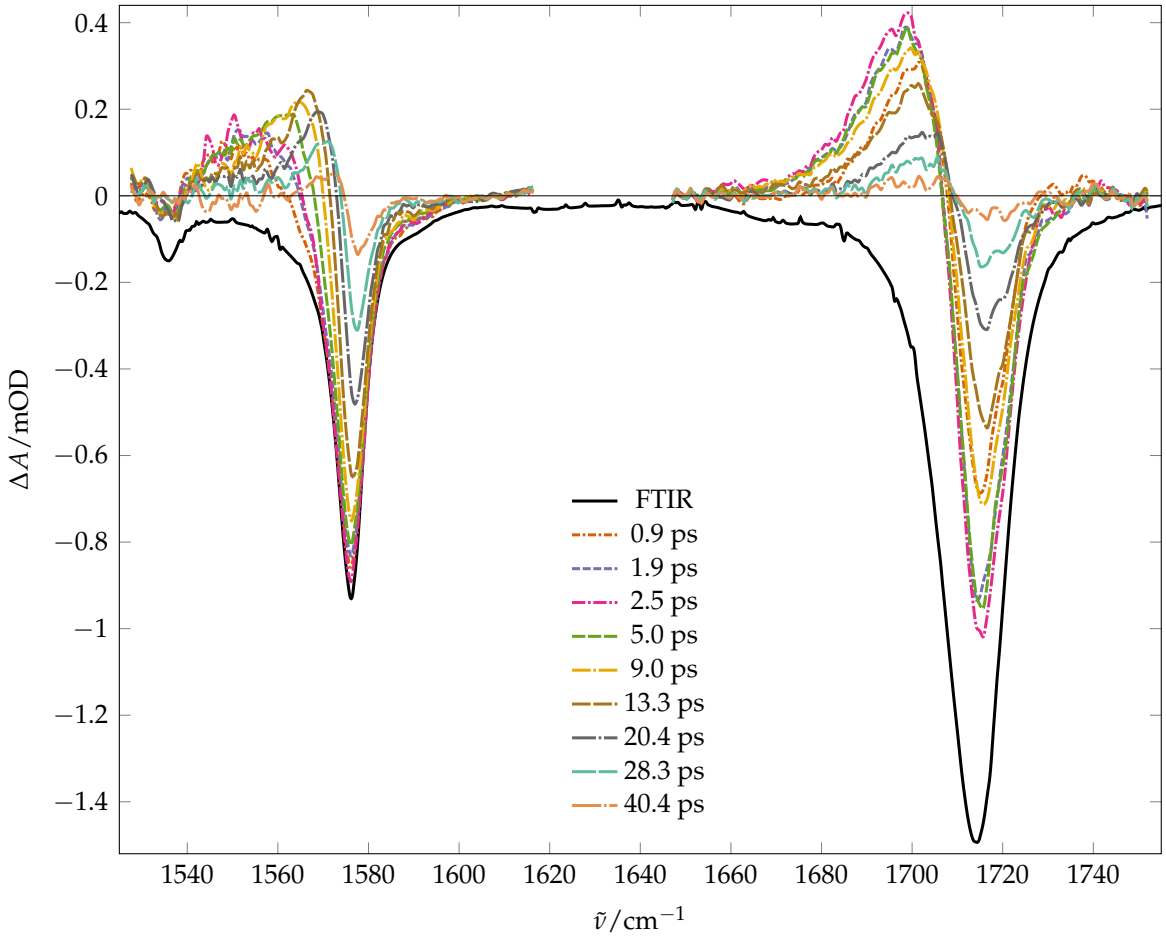
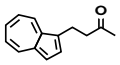
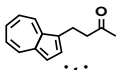


Figure 3.17: Transient difference spectra of  in CH_2Cl_2 after excitation at 610 nm.

asymmetric stretching mode.

The transient spectra of  shown in Figure 3.17 are not as ambiguous, on the other hand, and a shift in the positive peak – and thus of the "hot" absorption spectrum – of the carbonyl mode at 1714 cm^{-1} (FTIR) as well as in the bleach is clearly visible. For this substance, however, the carbonyl group is separated from the azulene moiety by far fewer bonds than in all *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides. Accordingly, the tentative explanation for the lack of a gradual shift put forth in section 3.3.4 for the azide asymmetric stretching vibration – a transfer of certain low-frequency modes directly to the solvent rather than to other intramolecular modes – might not hold here, as the short chain here possesses far fewer modes which might couple to the solvent. Additionally, the close proximity of the carbonyl group to the azulene moiety will lead to some appreciable coupling between the carbonyl mode and modes of the azulene moiety. The carbonyl mode would then reflect the gradual loss of energy from the

azulene moiety rather than the population of a privileged channel of energy loss to the solvent.

Temporal evolution

In processing the data of the sensor mode in the respective molecule, i.e. the carbonyl – or asymmetric azide mode, the protocol described for the amide I mode in section 3.3.3 was adhered to analogously. The results are shown in Figure 3.18.

The positions of the respective maximum signals correlate well with the chain lengths of the molecules. As already established[7,75,81], IVR is slower in chains containing heteroatoms than in those of equal length containing only methylene groups. Additionally, in shorter chains an (extrapolated) non-vanishing value shortly after $t = 0$ is an indication of harmonic energy flow[7,75,81]. The resolution of the data does not allow to resolve harmonic energy flow and possibly separate it from direct anharmonic coupling. For molecules with longer chains this effect subsides and is eventually replaced by a delay interval which precedes the rise of the signal.

The decaying flank of the bi-exponential described by τ_2 is fairly similar in all substances, thus indicating similar velocities of VET to the solvent. Comparing this to the results obtained for the azulene ring distortion mode in section 3.3.2, it must be concluded that the signal of the azulene mode reflects energy release not only to the solvent, but also to the attached substituent. Thus the increase in energy loss from the azulene moiety with increasing chain length would be due to a larger heat capacity of the attached chain.

Comparison of substances

Values of $t_{\max}$ for all substances are shown in Figure 3.19 alongside a range of fitted linear functions reported by the Rubtsov group[63]. Both *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides and *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides agree well with the slope of Rubtsov's values, which were obtained for polyethylene glycol oligomers in CCl_4[8] and also with a value of 0.4 ps per methylene group reported by Wang and coworkers[127]. For *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, the ordinate intersect seems to be larger than Rubtsov's value, however, which is most likely an indicator of slower IVR from the azulene moiety to the chain. The Rubtsov group used IR excitation of the asymmetric stretching vibration of an azide group to deposit energy in their system and reported a mere 1.2 ps for the lifetime of

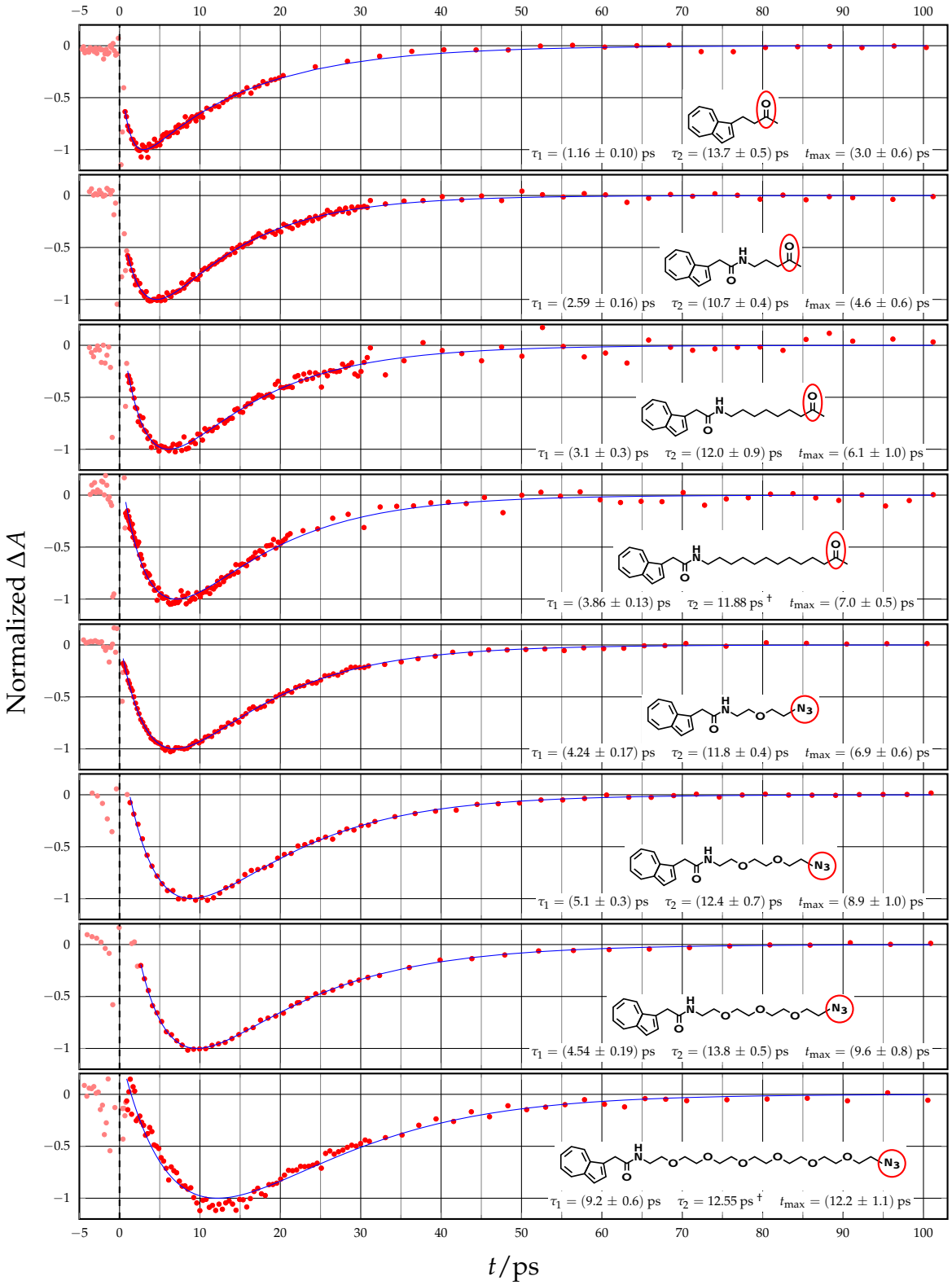


Figure 3.18: Kinetic traces of the carbonyl and asymmetric azide mode. Measured in CH_2Cl_2 using an excitation wavelength of 610 nm. τ_1 and τ_2 are the time constants of the bi-exponential fit function.
† Variable fixed to average value obtained from other substances in order to better localize least squares minimum.

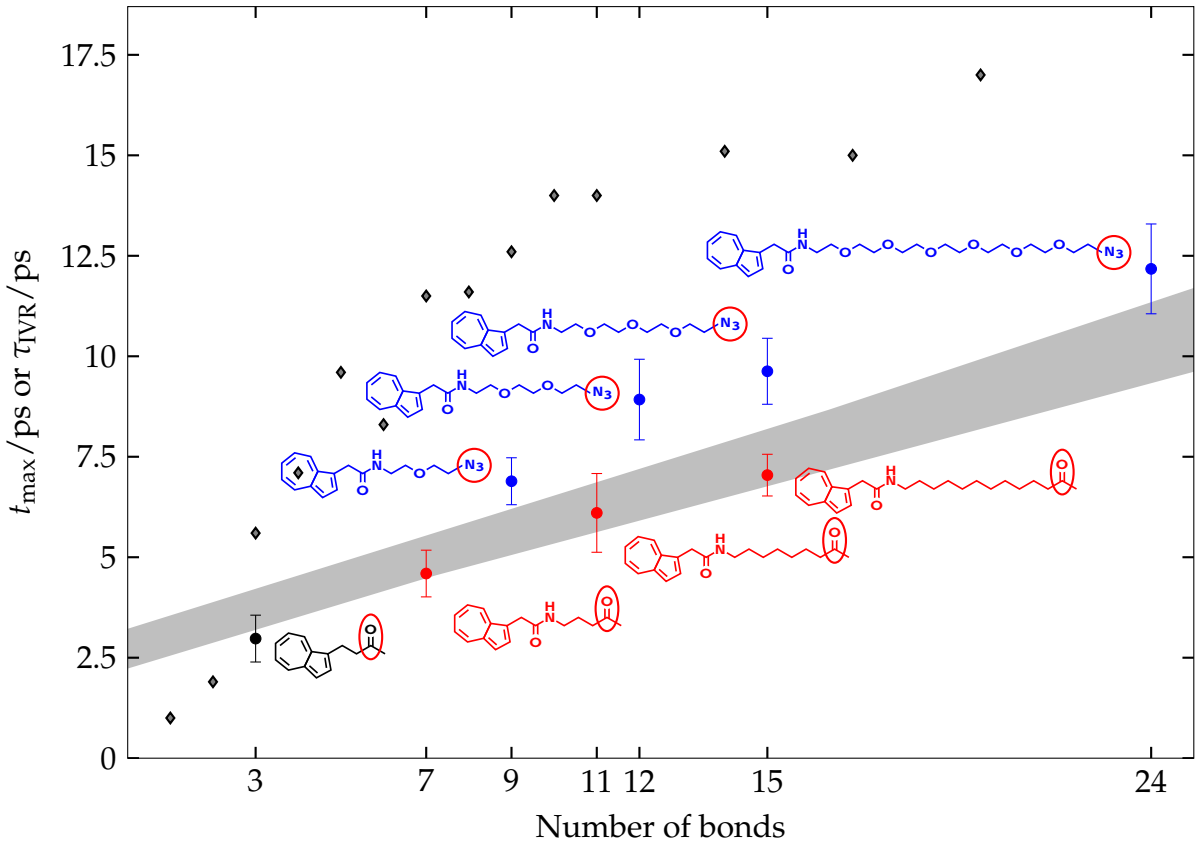


Figure 3.19: Comparison of the t_{max} of the azide asymmetric stretching – and carbonyl modes for all substances investigated. All measurements were conducted in CH_2Cl_2 using a pump-wavelength of 610 nm. The gray, shaded area represents all fits reported by Lin *et al.*[8] for polyethylene glycol oligomers in CCl_4, including the respective error bars, assuming a bond length of 1.5 Å. ⧫ are the theoretical values of τ_{IVR} for anthracene taken from [69].

the excitation[8].

Slower IVR in chains containing heteroatoms than in those composed solely of methylene groups has been observed before[7,75,81] and attributed mainly to changes in the potential energy surface rather than the greater mass of the heteroatoms as compared to carbon[75,81,128]. From Figure 3.19 it appears that this effect is predominantly a constant difference due to the nature of the system and increases only slightly as chain length and number of hetero atoms are increased (cf. section 3.4.2).

The results do not support the assessment that IVR times are independent of chain length in larger chains[7,75,81]. Rather – and in accord with the Rubtsov group's findings – IVR times appear to be proportional to chain length.[8,63]

In the light of the Schwarzer group's earlier results for even shorter chains[7,75], it seems reconcilable with the present data, that a fall-off behavior exists for the depen-

dence of IVR time on chain length for very short chains. The effect was also supported by the Schwarzer group's later theoretical work[69], from which some data are also shown in Figure 3.19, and, while yielding IVR values roughly twice as large as the experimental ones [sic], it shows both the fall-off behavior and an approximately linear trend for longer chains. A comparison with the present data is difficult, since IVR times were mostly obtained as the fast component of the bi-exponential of the energy content of the azulene moiety, rather than arrival times or $t_{\max}$ values of a sensor group in the theoretical work. Further, the chains used here were not sufficiently short to show a marked fall-off. With the results of section 3.3.2, the use of the azulene moiety as a reporter for IVR seems dangerous as IVR and VET may become difficult to distinguish for longer chains. This does, however, not refute the IVR time constants of earlier works[7,75] as they were well separable from VET.

In Figure 3.20, maximum signals of the respective sensor absorptions are compared. It should be noted that while this data is normalized with respect to the maximum amide signal amplitude of the respective substance – as the amide band is fairly insensitive to the remaining chain – it was not accounted for differing absorption coefficients of azide and carbonyl marker bands (see section 3.2). There is no immediate quantitative relationship between the bleach of an absorption and the intensity of that same absorption in a stationary spectrum, if it overlaps with its red-shifted absorption, as in this case. First, this is particularly true comparing the azide and carbonyl modes, as these will probably differ in their sensitivity to temperature[8]. Second, even within a group of substances normalization to the stationary spectrum is not sound, as the vibrational energy and virtual temperature of a molecule at the point in time when the bleach of the sensor reaches its maximum depends on chain length: the longer the chain, the later this point will be reached, the more energy will have been lost to the solvent, apart from that the heat capacity of the molecule will be greater. Hence the maximum signal corresponds to entirely different energetic situations (total amount of vibrational energy as well as its distribution over more or fewer degrees of freedom) in the various molecules.

Lin *et al.*[8] used a single exponential to describe the decay of the maximum of the signal of the sensor with chain length. Based on the dependence on chain length determined by them and the first data point in each series of substances, fairly similar predictions for both groups of substances are shown in Figure 3.20. Although the number of data points is somewhat scarce, two systematic differences can be inferred: The experimental points suggest a greater rate of decline than the exponential and they seem to reach a non-zero asymptote. The most probable explanation for both aspects

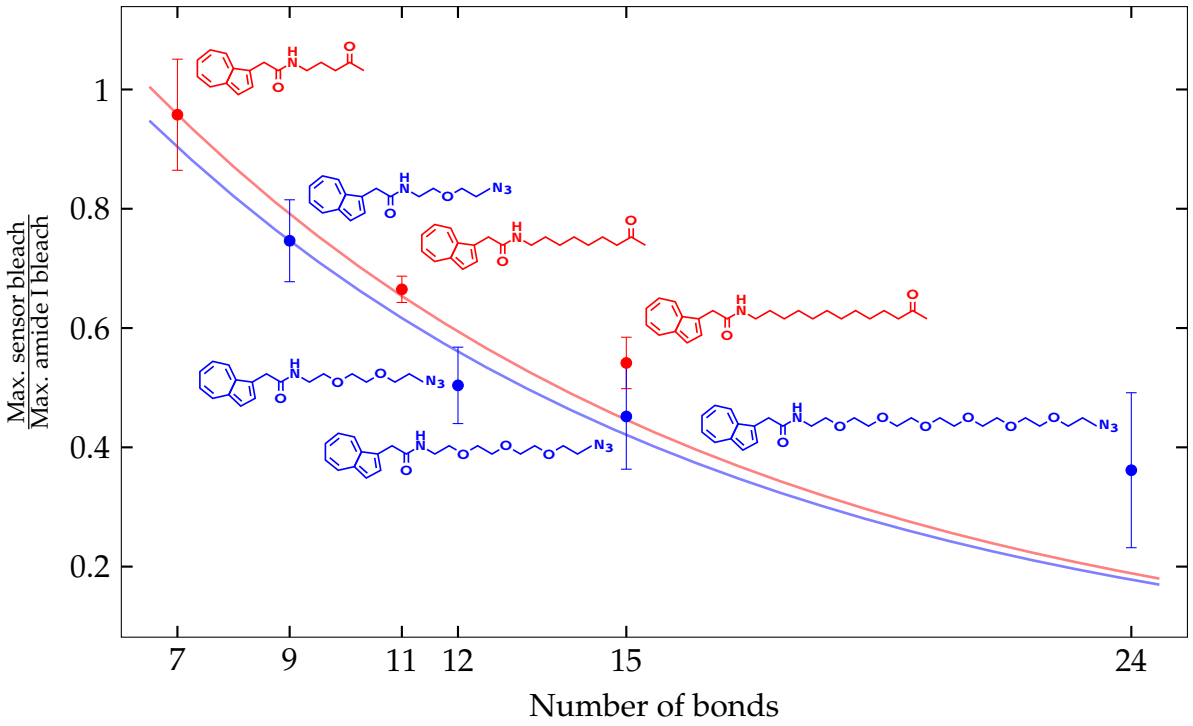


Figure 3.20: Maximum sensor mode signals relative to the maximum amide I mode signal. No normalization was performed to account for the differing intensities of the sensor absorptions (see text). Lines represent exponential decays according to Lin *et al.*[8], using a characteristic decay length of 15.7 Å, assuming bond lengths of 1.5 Å, and scaled to the value of the smallest molecule in each set.

is the difference in excitation methods. While in Lin *et al.*'s case IR excitation was used and hence the initial bleach of the sensor band reflects directly its coupling to the excited mode, here excitation of an electronic state and successive internal conversion were employed, which invariably leads to the excitation of a multitude of modes[124]. This multitude will include both strongly localized normal modes and modes delocalized over the entire molecule. While the anharmonic coupling of the former to the sensor mode ought to decay quickly with distance, the latter may be varying little in their coupling to the sensor mode[129]. Accordingly, the delocalized modes probably cause the apparent plateau or a second, much slower decline.

3.4 A simple model of energy transfer

The Hamm group[1,78] proposed a simple rate equation-based scheme as a model for the IVR in peptide helices; a refined version of the same group's earlier approach[72]. In this model, the individual amino acids were considered to be reservoirs of energy

which exchange energy with only their immediate neighbors and release energy to the solvent. The bleach intensities of the IR absorptions of the amide I modes were assumed to be directly proportional to the energy content of the corresponding amino acid. In this section, those assumptions shall be used to implement a model describing IVR in the molecules researched in this work.

For the energy E_i of the ith unit of a chain-shaped molecule, the rate equation of the model is

$$\frac{\mathrm{d}E_i}{\mathrm{d}t} = -k_s E_i + \sum_{k}^{\text{neighbors}} k\ (E_k - E_i) \tag{3.7}$$

where k is the rate constant of energy transfer among neighboring units (IVR) and k_s the rate constant of transfer to the solvent (VET). In the original model[1], amino acids were an obvious choice to represent individual units. Here, non-hydrogen atoms along the chain shall be considered as smallest units.

However, it is to be understood that the model is by no means to be interpreted literally: Obviously, mode delocalization plays an important role in IVR and is not represented properly by this model. It is probably better to consider the normal modes of the molecule to be the constituting units of the system. In the absence of much more detailed information about their mutual couplings (i.e. the concept of "neighboring" groups or modes above), this is the only feasible approach.

One further difficulty arises with regard to the azulene moiety: while it would appear straightforward to consider each of its carbon atoms a unit, this approach would raise great arbitrariness in the initial distribution of energy and the couplings among those units. Instead, it seems more appropriate to consider the azulene moiety as a single unit with a greater heat capacity than the other units. Equation 3.7 is essentially Fourier's Law of heat conduction if one assumes heat capacity to be independent of temperature and asserts that all units possess the same heat capacity. While there is – again – no obvious way to easily remedy the former assumption, the latter can quickly be disentangled in order to account for the obviously much greater heat capacity of azulene c_a. For the exchange of heat between the azulene moiety and its direct neighbor C1, whose heat capacity shall be denoted c, the rate equation would then be

$$\begin{aligned} \frac{\mathrm{d}E_a}{\mathrm{d}t} &= -k\,c\Delta T = -k\,c\left(\frac{E_a}{c_a} - \frac{E_{\mathrm{C1}}}{c}\right) = -k\,\frac{c}{c_a}\,E_a + k\,E_{\mathrm{C1}} \\ &= -k_a\,E_a + k\,E_{\mathrm{C1}} \end{aligned} \tag{3.8}$$

and simply require the rate constant for transfer from the azulene moiety to the chain to be smaller than those for the remainder of the chain. The same reasoning has to be applied to the transfer of energy from azulene to the bath, resulting in an analogous scaling of the rate constant of energy transfer from the azulene unit to the bath. It should be pointed out that Equation 3.8 differs from the Hamm group's[1] way of coupling an alien moiety to the otherwise homogeneous system. While their approach primarily aims at allowing for a faster coupling of the first unit of the chain (same rate constant for energy to and from the alien unit), the above treatment accounts for the different heat capacities (different rate constants for transport to and from the alien unit, respectively).

3.4.1 Application

In describing the given molecules in terms of the proposed model, it seems appropriate to expect different parameters for the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides and the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides as their constituents (i.e. chains of methylene groups and glycol units, respectively) differ substantially. Beyond the experimental evidence obtained in this study (see section 3.3.4 and section 3.3.4), this has also been found by other experimental[7,75] and theoretical studies[7,81].

Allowing for different parameters for each substance, on the other hand, does not appear advantageous. While this procedure would result in a much better agreement with the experimental data, the reasons for this improved agreement are fairly obvious and valuable insights about the fitfulness of the model's descriptive capabilities are discarded simultaneously: As it stands, the model's master equation Equation 3.7 is linear and will hence have a solution that is essentially a superposition of exponentials. Applied to a single set of data, this would of course just be an augmented version of the bi-exponential fits shown in Figure 3.18 and thus would probably not help much in exposing possible weaknesses or strengths of the model with regard to its capability of accurately describing IVR processes. Moreover, any predictive power would be obliterated, unless one were to introduce some scheme or theory to explain the change of the fitting parameters with the size of the system. Such an extension is not obvious, however. To the contrary, allowing for different sets of parameters for the two classes of substances, makes it possible to verify some of the assumptions made in the derivation of the model.

The model itself was implemented numerically with time steps of 10 fs and an

	k/ps^{-1}	k_a/ps^{-1}	k_s/ps^{-1}	α
N-(oxo-alkyl)-2-(1-azulenyl)-acetamides	13 ± 2	0.7 ± 0.4	0.22 ± 0.08	1.3 ± 0.3
N-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides	10.0 ± 1.7	0.45 ± 0.16	0.21 ± 0.06	3.0 ± 1.4

Table 3.1: Fit parameters obtained for the master equation model.

initial distribution of the energy of Gaussian shape

$$E(i) = \sqrt{\alpha}\, \exp\left(-\alpha\, i^2\right)$$

was assumed, where i is the index of the unit, starting at zero. This was done to account for harmonic energy flux[7,75,81] (see also section 3.3.4).

Using a standard implementation[130] of the Levenberg-Marquardt algorithm[131], the parameters of the modified model were fitted to the normalized experimental data from Figure 3.18. The model data were scaled to match up with the experimental data before each fitting step. The results are summarized in Table 3.1 and Figure 3.21.

The fitted parameters capture the general trend of slower transport along chains containing heteroatoms, although the experimental data suggested this to be an offset rather than a difference in transport speed. As expected, transport from the azulene moiety (k_a) and into the solvent (k_s) are both slower than the "intra-chain" transport (k). Moreover, it is in qualitative accord with the assumptions leading to Equation 3.8 that the transfer from the azulene moiety is slower in *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides, i.e. $k_a \propto k$. The ratio of $k : k_a$ of about 19 and 22, respectively, seems somewhat large, however, given the fact that azulene has only ten times as many non-hydrogen atoms as the "ordinary" units of the molecular backbone chains. Finally, the determination of the parameter of the initial distribution α is rather imprecise, especially with the large error in the value obtained for the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides. This is understandable given the rather long chains in that set of molecules, which lead to large distances between sensor group and azulene moiety and ultimately to a small initial signal in the sensor's absorption.

Comparing fits and experimental data, some fundamental properties of the model

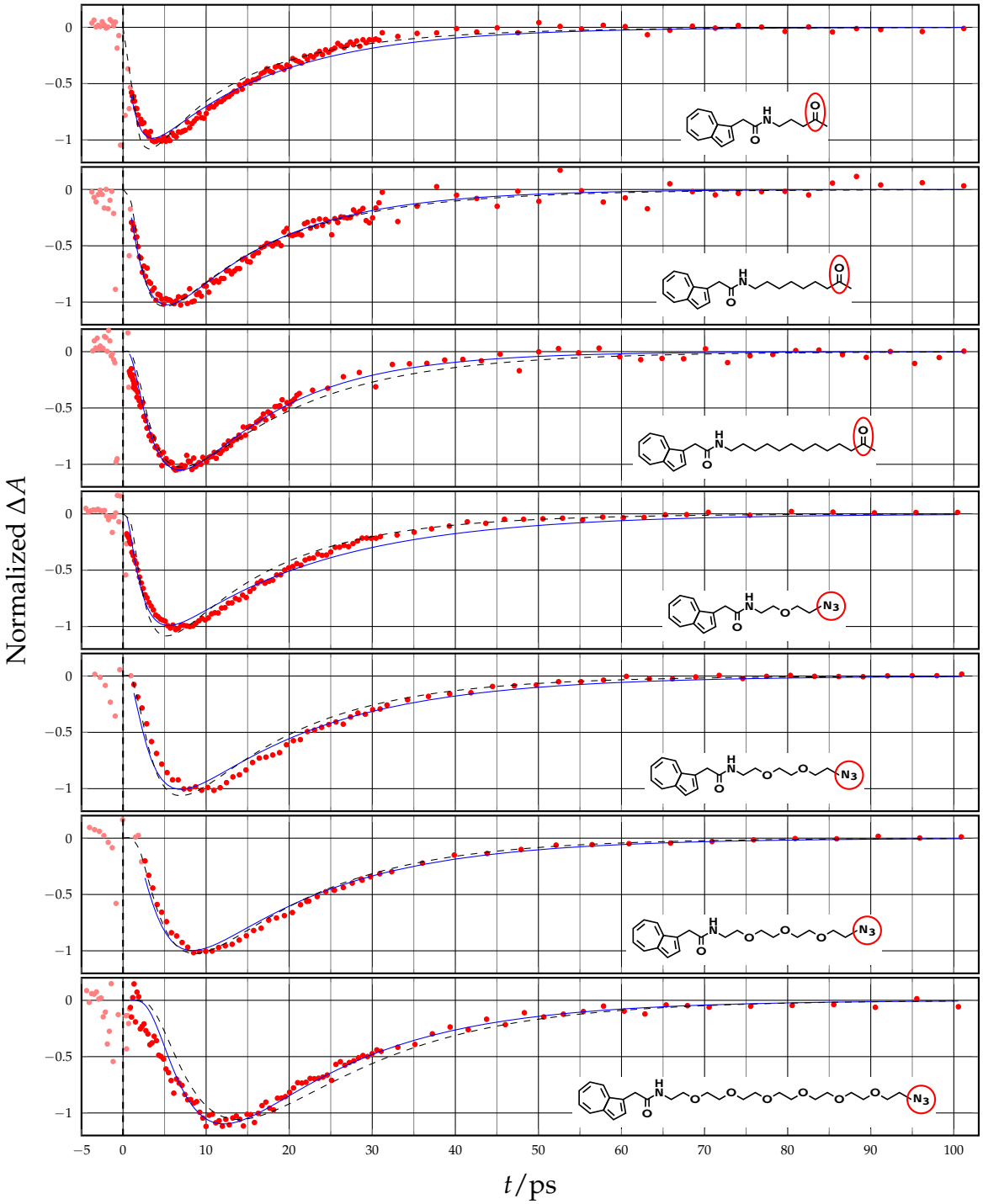
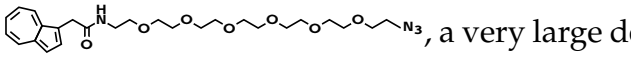
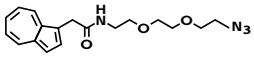
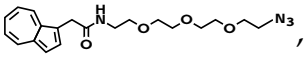


Figure 3.21: Application of the Hamm group's[1] diffusive master equation model. Note: different parameters were used for the set of *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides and the set of *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides. Also, all model results were scaled to match the experimental data before handing the results over to the Levenberg-Marquardt algorithm.[130] The black dashed lines are fits using one-dimensional diffusion with decay (see text).

	x_0	τ_s / ps
N-(oxo-alkyl)-2-(1-azulenyl)-acetamides	8.23 ± 0.19	17.7 ± 0.7
N-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides	8.9 ± 0.3	16.7 ± 0.7

Table 3.2: Fit parameters obtained using Equation 3.9.

seem to be not quite in accord with the reality of IVR in the investigated molecules. While it succeeds – at least for the longest chains in either group – at describing the position of the signal's maximum and the release of energy to the solvent, it appears to be incapable to mimic the initial phase, as already indicated by the Hamm group[72]. Particularly in the case of  , a very large delay in the sensor's response is clearly visible in the model, where it is much smaller in the experimental data. Generally, in shorter chains the model seems to predict faster IVR than the experiment would suggest, while in longer chains the inverse is true. Likewise, with regard to the maximum signal positions, the fits result in smaller values of $t_{\max}$ than the experiments in all but the longest chains of each set. Both findings hint at a tendency to exaggerate the dependence of IVR speed on chain length. This suggests that the values obtained for k are somewhat too large, as those are of course the speed of IVR in the chain and thus also most directly impact the change of $t_{\max}$ with chain length. In the case of  and  , however, it seems likely that most of the deviation results from an imprecisely or incorrectly determined $t = 0$ rather than from shortcomings of the model, as the coherence artifact is particularly broad in these cases.

An additional fit is shown as dashed black lines in Figure 3.21. For this fit, the one-dimensional diffusion function with decay as given by Tesar *et al.*[101]

$$P(x,t) = (2\pi Dt)^{-\frac{1}{2}} \exp\left(-\frac{t}{\tau_s} - \frac{x^2}{2Dt}\right) \tag{3.9}$$

was used with D being the diffusion constant as obtained earlier (22.5 Å^2 ps^{-1} for the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides and 29.25 Å^2 ps^{-1} for the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides) and τ_s is the time constant of energy loss to the solvent. Instead of the square root for normalization, an individual scaling factor was employed for each data set. To compensate for the azulene unit, a length offset x_0 was added to the number of non-carbon between the amide group and the sensor group n, such that the distance variable was $x = x_0 + n$. The fit results are listed in

Table 3.2.

Especially for the longest chains in each group, the latter fit struggles to describe the VET part of the signal properly. For the shortest chains, this is somewhat compensated by the scaling factor, leading to the curve's peak being too large. Further, the rising edge of the signal is predicted much too steep for the shortest chains. Presumably, this vindicates the additional effort undertaken in describing the role of the azulene group with respect to both the solvent, leading to a better description of VET, and the chain, providing for a better description of short-range IVR.

3.4.2 Discussion

Comparison to the original model

There is no straightforward way to compare the values of k obtained here to the Hamm group's value of $(2.0 \pm 0.5)^{-1}$ ps^{-1}[1] for one amino acid (see Figure 3.6), as the physical meaning of k differs fundamentally. This is an inherent weakness of the model, as there is no obvious way of defining the model's smallest unit – a dilemma best visualized by symbolic transfer matrices. The elements of such a transfer matrix $\underline{\underline{M}}$ can be defined as the linearized version of Equation 3.7:

$$E_i(t + \Delta t) = \sum_k M_{ik}\, E_k(t)$$

and they advance the system by one time step. By limiting transfer to neighbors, $M_{ik} = 0$ if $|i - k| > 1$, while for a homogeneous chain $M_{ik} = k$ if $|i - k| = 1$, and due to energy conservation $M_{ii} \equiv \tilde{k} = 1 - k_s - \sum_{k \neq i} k$. Thus, the transfer matrix becomes

$$\underline{\underline{M}} = \begin{pmatrix} \tilde{k} & k & 0 \\ k & \tilde{k} & k \\ 0 & k & \tilde{k} \end{pmatrix}$$

while for a system of units larger by a factor of 1.5, applying the model's rules, it would be

$$\underline{\underline{M}}' = \begin{pmatrix} \tilde{k}' & 0 & k' \\ 0 & 0 & 0 \\ k' & 0 & \tilde{k}' \end{pmatrix}$$

Clearly, $\underline{\underline{M}}$ and $\underline{\underline{M}}'$ are not equivalent. In a physical sense, by combining two units, the retarding effect of IVR among those two units is eliminated and replaced by medium range transfer that effectively hops over the central unit in $\underline{\underline{M}}$. In essence, this is the

situation when comparing the model of this work to the Hamm group's implementation, except that it is three backbone atoms (model unit of this work) combined into a single unit[1,72].

Heat diffusion constants, on the other hand, are certainly of more robust physical meaning. Assuming a bond distance of about 1.5 Å, the heat diffusion constants $D = k \cdot \Delta x^2$[1] would be in the range between 20 and 30 $\text{Å}^2\ \text{ps}^{-1}$, which is somewhat larger than expected (10–20 $\text{Å}^2\ \text{ps}^{-1}$[1]), and could thus be interpreted as an indication of very fast ballistic transport. For helical peptides, this value was determined to be much slower ($2.0 \pm 0.5\ \text{Å}^2\ \text{ps}^{-1}$).[1]

Transfer to the solvent, on the other hand, was assigned a rate constant of $(7.6\ \text{ps})^{-1}$ by the Hamm group[1]. In this case, the situation is simpler and a generally similar value is to expected since this rate constant reflects a cooling proportional to the temperature difference between solute and solvent, irrespective of the size of the system. To show this, one can separate $\underline{\underline{M}} = \underline{\underline{M}}_{\text{IVR}} + \underline{\underline{M}}_{\text{VET}}$ where $\underline{\underline{M}}_{\text{VET}} = -k_s \cdot \underline{\underline{1}}$ and $\underline{\underline{1}}$ is the unit matrix. Since $\text{tr}\left(\underline{\underline{M}}_{\text{VET}}\right) = -n\,k_s$ where n is the number of units in the system, the system's total energy decays exponentially with a time constant of $n\ k_s/n = k_s$, i.e. independent of the size of the system. Thus, it can be viewed as fairly good agreement, that the values found here are only a little more than 1.5 times as large. Note should be made, however, that k_s for the azulene moiety was set to be lower than for the other chain members by a factor of k_a/k. Of course, some deviation due to differing systems and solvents (CH_2Cl_2 here and chloroform in [1]) and their respective mechanical characteristics is to be expected.

While this agreement might at first sight appear reassuring, it truly reveals a severe deficiency of the rate equation model, as the rate of energy transfer to the solvent should depend on system size (cf. 3.3.2).

Predictive capabilities

In their recent study[8,63], Lin *et al.* came to the conclusion that the effective arrival time t_{max}, i.e. the time until the sensor's signal reaches its maximum, depends linearly on the length of a molecular chain between an initially excited group and the sensor group. This relationship was found to hold approximately even for very different groups and structures.[93,98] The maximum value of the signal itself, on the other hand, was found to decrease exponentially with increasing chain length. The former finding is in contrast to earlier conclusions[7,75,81], which proposed IVR times to have

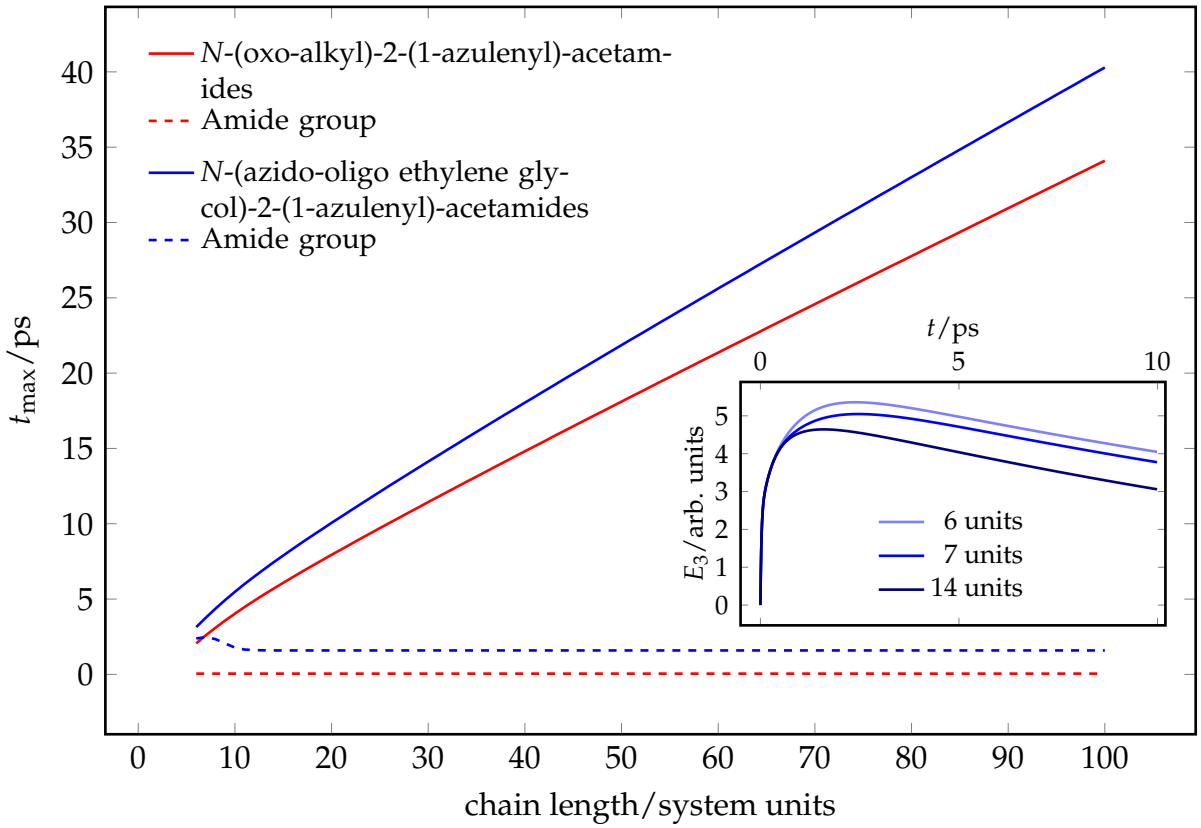


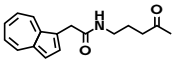
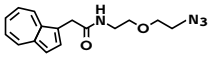
Figure 3.22: Dependence of the arrival time t_{max} on chain length obtained with the fit parameters of the master equation model. The amide group is assumed to be the third chain member, while the sensor group is the last, if it is an azide, or the last but one in the case of a carbonyl. Inset: Simulated energy content for the amide unit in systems of different chain lengths, using the set of parameters obtained for *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides.

an asymptotic upper limit as a result of ballistic transport. Given the greater chain lengths investigated by Lin *et al.* and the results of this work (see section 3.3.4), a linear dependence seems more likely with a deviation in very short chains, as has recently been also found theoretically for azulene-anthracene compounds[69].

The model obviously would by construction not predict an upper limit. As is evident from Figure 3.22, a nearly linear dependence is predicted instead. The slopes correspond to 0.22 ps Å^{-1} (*N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides) and 0.25 ps Å^{-1} (*N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides), respectively, in reasonable agreement with the values determined by Lin and coworkers for polyethylene glycol chains (0.198 $\pm$ 0.008 ps Å^{-1}, 0.225 $\pm$ 0.011 ps Å^{-1}, and 0.219 $\pm$ 0.002 ps Å^{-1})[8]. In converting the abscissa values, an average bond length of 1.5 Å was assumed. Errors depend somewhat on the precise range of chain lengths considered, but the given results are consistent over a broad range. Note that this is in contrast to the Hamm

group's rather poor fit of a r^2 dependence[72], whose functional form is reminiscent of the result for one-dimensional random walk diffusion in the limiting case of long solvent relaxation times τ_s[101].

A small fall-off region exists for short chains but neither the change in slope nor the magnitude are nearly as pronounced as those found previously[7,75]. This is another indicator that the description of very early and very short range IVR is not quite accurately represented within this model. As most of the molecules investigated in this work are of chain lengths in or near what might be considered the fall-off region, this might explain the difficulties in fitting the model to the early time signals mentioned above.

The results for the amide group are consistent with the findings of this work in that no dependence on chain length is predicted for the longer ($\geqslant$ 14 units) *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides systems. For shorter chains, $t_{\max}$ of the amide mode is slightly larger, which is owed to a similar effect as the faster energy loss of the azulene unit in systems with greater chain lengths described in section 3.3.2. As the number of succeeding chain units increases, the loss of energy to those units grows, first slightly increasing $t_{\max}$ (from six to seven units, see inset in Figure 3.22) and then decreasing it to a value of 1.59 ps for chains of lengths $\geqslant$ 14. While the numeric value is not in accord with the experimental findings of an overall average of 2.15 $\pm$ 0.05 ps, one might concede that the phenomenon of a delayed maximum in shorter chains might be present as the values for  and  are substantially larger than for all the other molecules, although 2.1 ps is within the error limits of the former and barely outside those of the latter(cf. section 3.3.3). For the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides, predictions are of little use as the Gaussian parameter α is too small and leads to a situation in which the maximum energy value in the amide unit is always at $t = 0$. This is a clear indication that the distribution used is not a realistic representation of the actual situation.

With regard to the magnitude of the signal, the model's prediction of exponential decay is in very good qualitative agreement with Lin *et al.*'s findings. Again assuming bond lengths of 1.5 Å, one arrives at characteristic $1/e$-decay lengths of 10.9 to 11.0 Å (*N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides) and 10.0 to 10.1 Å (*N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides) in fair agreement with Lin *et al.*'s values of 13.2 $\pm$ 0.5 Å, 15.6 $\pm$ 0.7 Å, and 16.2 $\pm$ 1.6 Å[8]. As with the arrival time, this is again contrary to the Hamm group's rather poor fit of an r^{-1} dependence[72].

With the parameters obtained from the *N*-(azido-oligo ethylene glycol)-2-(1-azu-

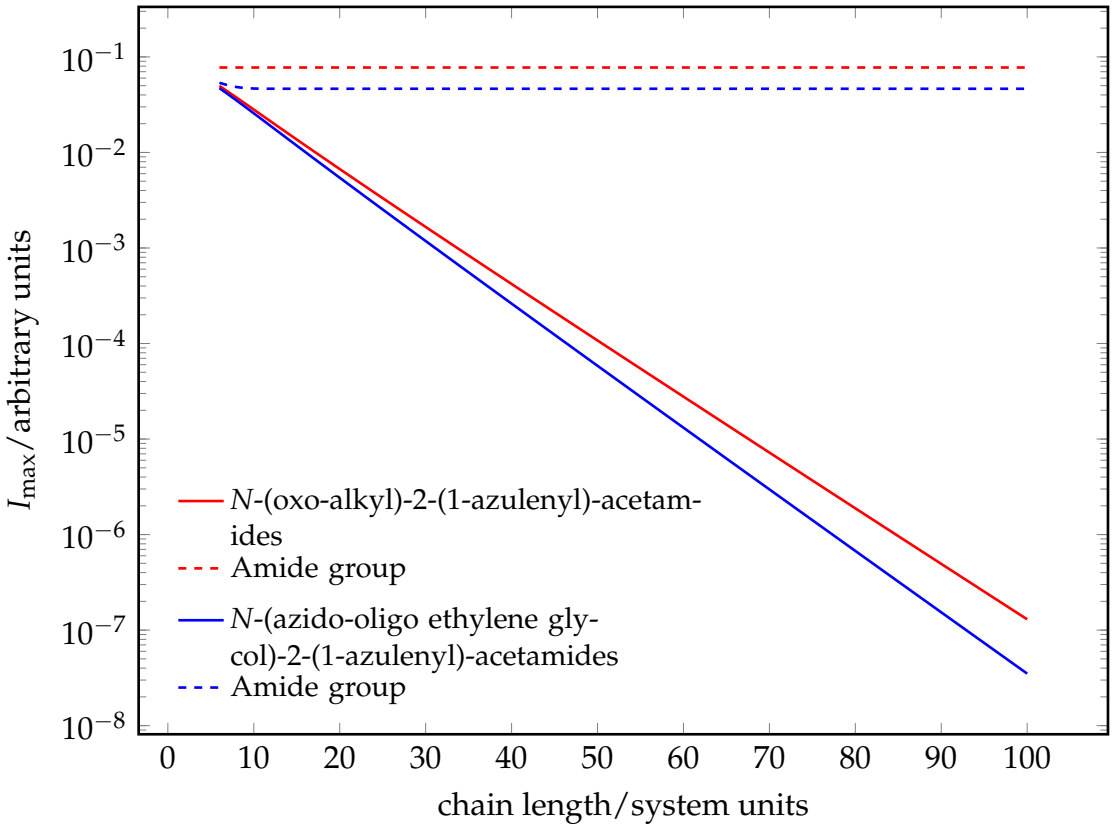


Figure 3.23: Dependence of the signal intensity on chain length obtained with the fit parameters of the master equation model. The amide group is assumed to be the third chain member, while the sensor group is the last, if it is an azide, or the last but one in the case of a carbonyl.

lenyl)-acetamides fit, maximum values for the energy of the amide group, on the other hand, are largely independent of chain length, with the exception of short chains due to effects described above. As also stated above, the values obtained with the *N*-(oxo-alkyl)-2-(1-azulenyl)-acetamides parameters provide no substantial insight other than that the initial distribution is likely flawed.

Summary of model results

The many *ad hoc* assumptions and corrections made in the previous sections make evident that there are many aspects of the proposed model which are rather vague. First and foremost, while the idea of treating the system as a chain of thermal reservoirs exchanging heat only with their immediate neighbors is intuitive, this concept does not represent some of the most important characteristics of IVR that are well known and established. The harmonic energy flux[7,75,81] is merely accounted for in the par-

ticular implementation used here by introducing an initial distribution of energy that extends beyond the first member of the chain. Whether this distribution actually is of Gaussian shape as assumed here is of course utterly unclear, but at the same time the experimental data available does not hint at an obvious choice.

Even if one were to derive such an initial distribution from the approaches pursued by some of the theoretical works in the field[69,70,72,78,81], one would still have to define the precise correspondent of the "model units": Do they represent groups of atoms or of normal modes? In either case the model would still not cover the role of highly delocalized transporting modes as it was established by the Hamm group in a theoretical work[65]. A possible remedy, including interactions reaching further than the immediate neighbors, would – without further assumptions – lead to an unreasonably large number of variables. This second shortcoming might also explain the unsatisfactory description of experimental data of the early, IVR-dominated phase.

Furthermore, the treatment of units of different heat capacities is not quite clear. For the azulene moiety it was assumed in section 3.4, that its greater heat capacity would delay transfer of energy to its neighbors as well as to the solvent. That is, a greater heat capacity was thought to delay the transfer of energy in the sense of Fourier's Law of heat conduction. Ultimately, the transfer to the solvent is much more delicate and greatly depends on overlap of the respective normal modes of solvent and solute.[17,30,119] The behavior of a homogeneous chain with equal values of k_s for all units is, as mentioned before, not to be reconciled with these intricate features of solvent-solute interaction and fails even to properly account for scaling with system size, as was elaborated above.

Overall, the model is of little quantitative predictive power, since many uncertainties remain regarding the *a priori* determination of its parameters. At the same time, approximate quantitative agreement between the fits of this work and of the Hamm group for VET was found, as well as agreement with Lin *et al.*'s findings about the general characteristics of IVR . While the model's description is intrinsically diffusive, the transport in systems used here and by Rubtsov is generally considered to be ballistic in nature. It may hence be concluded that ballistic and diffusive transport can obviously not be discerned on the basis of these general characteristics, but on the early stages of IVR, which the model failed to describe appropriately, and the speed at which a heat signal is propagated, which is substantially greater in ballistic transport.

3.5 Towards a mode-resolved picture

While many studies have emerged in recent years on IVR along molecular chains, as presented in section 3.1, only recently has more effort been spent on resolving or interpreting spectral information in greater detail.[66,97,100,101] As already alluded to in subsection 3.1.2, the Hamm group recently began to gain more detailed insight into the properties of carbonyl absorptions, which had previously been regarded as "local thermometers"[70,72,78]. Indeed, the modes by which transient changes in the absorption of those carbonyl modes are induced appear to become populated predominantly as a result of intra-site IVR, which may be substantially delayed with respect to IVR along the molecular backbone.[89,97] An investigation of the underlying anharmonic constants x_{ij} was not conducted, as it is "still very challenging [sic]"[97]. Instead, the interpretation of the measured signals rests on generalized results found in one-dimensional glasses by Leitner[73], since a more detailed analysis proved inconclusive[132].

Rubtsov, on the other hand, describes "spatially overlap modes" as predominantly responsible for anharmonically shifted absorption[8,93], which rests on theoretical studies of a one-dimensional glass[73]. Some efforts have been put forth to determine anharmonicity constants experimentally[66], and they have been used to describe IVR in terms of the Marcus electron transfer theory.[100,101]

Finally, Schwarzer *et al.* undertook the probably most diligent effort to relate their observed signal to the "kinematics" of the systems studied by calibrating against temperature-dependent changes in absorption.[7,75,80] Of course, this approach assumes an energy distribution very similar to a canonical one, which is – as shall be demonstrated in subsection 3.5.2 – an acceptable assumption for larger molecules. An attempt to obtain similar reference data by measuring potassium bromide pellets at elevated temperatures failed. The sample apparently underwent irreversible changes at temperatures of more than 70 °C (see Figure 3.24).

Besides this practical obstacle, it seems more promising to resolve mode-specific anharmonicity theoretically and then use those results to explain the experimentally observed spectra, because a conceptual separation into units of large heat capacity as in the case of the azulene-anthracene compounds is not possible for the molecules investigated here. Based on the foundation laid out in subsection 3.3.1, this shall be the purpose of this section.

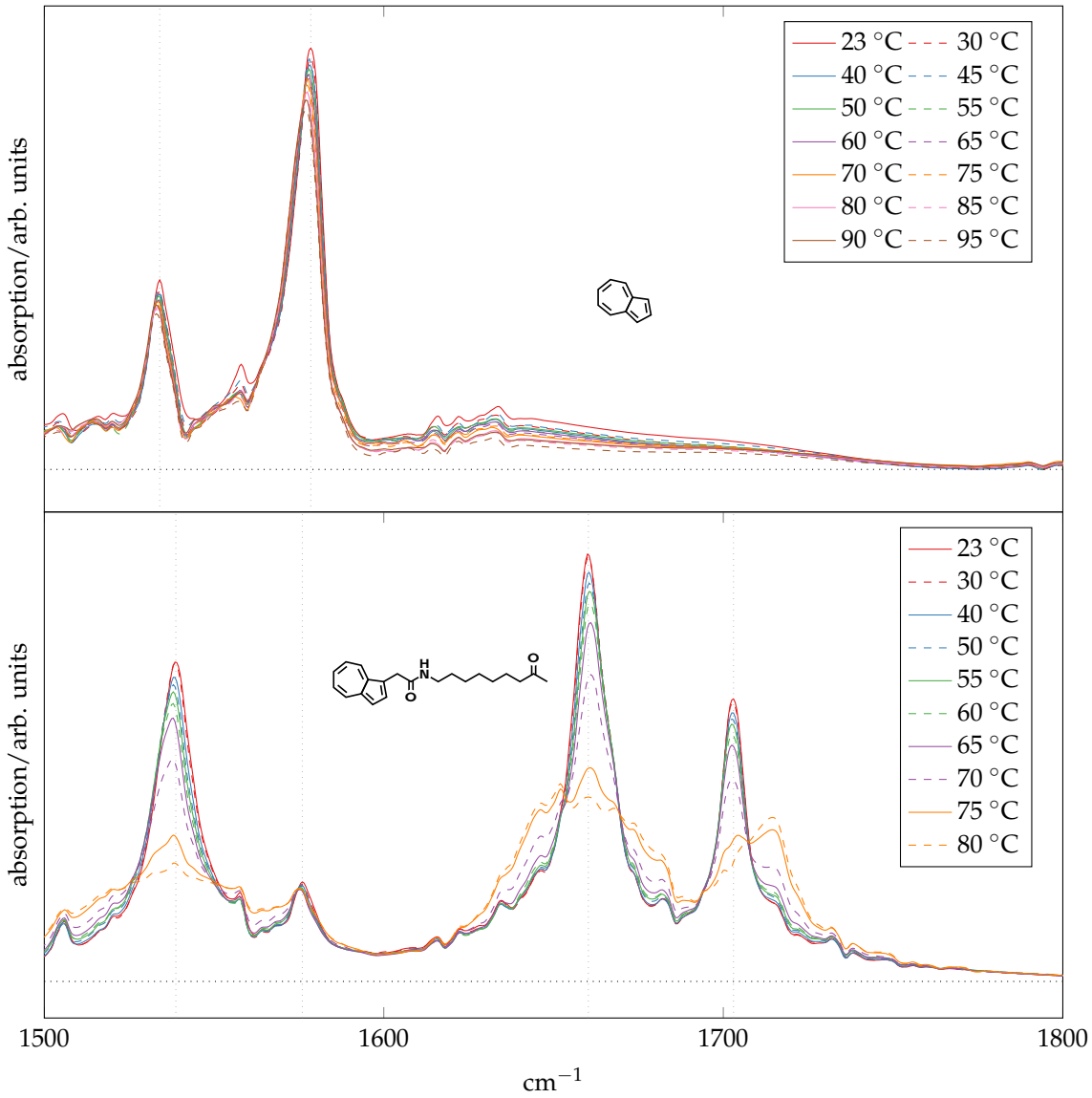
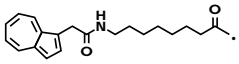
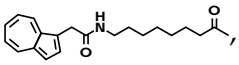


Figure 3.24: Temperature-dependent FTIR spectra recorded from potassium bromide pellets. Top: pure azulene, bottom:  . Dotted lines indicate peak positions at room temperature. Spectra were recorded by heating an aluminum block holding the sample, a heating capsule, and a resistivity thermometer. Spectra were generally recorded and averaged for both heating and cooling to a set temperature, i.e. approaching it "from below" and "from above". For  , the spectrum recorded at the highest temperature was persistent after cooling to room temperature.

3.5.1 IR spectra at elevated temperatures

As laid out in subsection 3.3.1, it is generally possible to model difference IR spectra with just the knowledge of the constants of anharmonicity. Here, we shall follow the scheme proposed by Hamm and coworkers[2], which models the absorbance of the kth mode according to

$$A_k(\omega) \propto \sum_{\substack{\text{states} \\ (v_1, v_2, \ldots, v_n)}} \left(\prod_i P_i(v_i) \right) \cdot L \left(\omega - \tilde{\nu}_k^0 - 2x_{kk}\, v_k - \sum_i x_{ik}\, v_i \right)$$

where v_i are quantum numbers of the modes indexed by i,

$$P_i(v_i) = \begin{cases} P(v_i\, \omega_i) & \text{for } i \neq k \\ \left[P(v_i\, \omega_i) - P((v_i + 1)\, \omega_i) \right] (v_i + 1) & \text{for } i = k \end{cases}$$

is a probability function including absorption and stimulated emission for the observed mode k, and $L(\omega)$ is a line shape function. $P(v_i\, \omega_i)$ is a population distribution function, e.g. a Boltzmann distribution function. The fundamental frequency $\tilde{\nu}_i^0$ may, of course, be determined experimentally, rather than by means of the expression given in subsection 3.3.1.

Three approximations have been included up to this point:

- No higher than the first anharmonic constants x_{ik} are used for the transition frequencies.
- Transition intensities are treated in the harmonic picture, i.e. proportionality to $(v_k + 1)$ and no dependence on any modes other than the observed one.
- The probability function depends only on the quantum number of the individual oscillators separately.

As a consequence of the last approximation, it is then necessary to also derive populations for the probability function only from harmonic frequencies.

In principle, it would be possible to include dependence of the transition dipole moment in $P_i(v_i)$, but the terms arising from second order perturbation theory[133] would couple all quantum numbers of a state and thus destroy the purely multiplicative character of this probability function. Generally, the effect of the level of excitation

of various modes on intensities of fundamental transitions is considered to be small, however.[133]

Algorithm for simulating IR spectra of vibrationally hot species

All of these approximations can be justified by the rather statistical nature of such spectra for larger molecules and serve the purpose of enabling particularly simple algorithms for calculating the resulting spectrum. To adopt this technique, it is furthermore necessary to discretize the spectral resolution by rounding all anharmonic constants to a smallest grain (e.g. 0.1 cm^{-1}).

```
A = array of zeros of size n                                          } absorption
A_{ṽ_k} = 1 (complete initial population at ṽ_k)                      } probability
for i in modes                                                        } distribution
        A^new = array of zeros of size n                              }
        for v_i in quantum numbers of i                               }
                shift = v_i · x_ik                                    }
                for m = max(1, 1−shift), ..., min(n, n−shift)         }
                        A^new_{m+shift} += A_m · P_i(v_i)             }
        A = A^new                                                     }

S = array of zeros of size n                                          } line shape
for i = 1, ..., n                                                     } convolution
        for m = 1, ..., n                                             }
                S_i = A_m · L(i − m)                                  }
```

Listing 3.1: Pseudocode of the Hamm group's algorithm for computing the absorption spectrum of the kth mode.[134]

For clarity, the algorithm is shown in Listing 3.1. First, an array of n elements, representing absorption probabilities at discrete and equidistant spectral positions, is prepared. The first stage of the algorithm redistributes the absorption probability initially found only in the element of the array associated with the fundamental absorption of the kth mode. In a second stage, the obtained distribution of absorption probabilities is then convoluted with a line shape function.

Owed mainly to the discretization of spectrum and the decoupling of the modes, this algorithm scales only linearly in the number of modes, but quadratic in the num-

ber of spectral positions n. The first part is essentially analogous to the well-known Beyer-Swinehart algorithm[135].

Vibrational angular momentum

A particular difficulty arises in the treatment of doubly degenerate vibrational modes, as they occur in molecules of highly symmetric equilibrium geometry, such as the benzene molecule initially treated by Hamm[2]. The coupling among those modes, as well as of those modes to the molecules angular momentum dictates the choice of eigenfunctions of the angular momentum operator for these modes.[109,110,113] In the harmonic approximation, the state of the tth doubly degenerate mode is thus described by the vibrational quantum number v_t and the vibrational angular momentum quantum number l_t, which is an integer obeying $-v_t \leqslant l_t \leqslant v_t$ and of the same parity as v_t.[136] For these modes, equation Equation 3.2 has to be adapted by replacing all $(v_t + 1/2)$ by $(v_t + 1)$ to account for their double degeneracy and by adding $\sum_{u \leqslant t} g_{tu}\, l_t\, l_u$ to account for the coupling of their vibrational angular momenta, such that

$$\begin{aligned} E(v_i,...) = & \underbrace{\sum_i \omega_i\ (v_i + 1/2)}_{\text{I}} + \underbrace{\sum_t \omega_t\ (v_t + 1)}_{\text{II}} + \underbrace{\sum_{i \leqslant k} x_{ik}\ (v_i + 1/2)\ (v_k + 1/2)}_{\text{III}} \\ & + \underbrace{\sum_{i,\,t} x_{it}\ (v_i + 1/2)\ (v_t + 1)}_{\text{IV}} + \underbrace{\sum_{t \leqslant u} x_{tu}\ (v_t + 1)\ (v_u + 1)}_{\text{V}} \\ & + \underbrace{\sum_{t \leqslant u} g_{tu}\, l_t\, l_u}_{\text{VI}} + ... \end{aligned} \tag{3.10}$$

Any coupling that requires to keep state-specific, rather than mode-specific information in memory will necessarily break the linear scaling behavior found to be very advantageous in the algorithm discussed in section 3.5.1. It is, however, possible to maintain these advantages by describing the state of a doubly degenerate mode t by two quantum numbers v_t^+ and v_t^-, chosen to satisfy

$$v_t = v_t^+ + v_t^- \qquad \text{and} \qquad l_t = v_t^+ - v_t^-$$

for which the regular form of equation Equation 3.2 applies. Parts I through IV are then obviously included simply by using the same harmonic frequencies ω_t and anharmonic coupling constants shared with non-degenerate modes x_{it} for both v_t^+ and

v_t^-, as all terms containing them will then add up to yield the correct result for doubly degenerate modes:

$$(v_t^+ + 1/2) + (v_t^- + 1/2) = (v + 1)$$

Parts V and VI require changing the coupling constants among doubly degenerate modes x_{tu} to include the coupling of their vibrational angular momenta g_{tu}. Since

$$\begin{aligned} x_{tu}\,(v_t+1)(v_u+1) + g_{tu}\,l_t\,l_u = \quad & (v_t^+ + 1/2)(v_u^+ + 1/2)(x_{tu} + g_{tu}) \\ & + (v_t^+ + 1/2)(v_u^- + 1/2)(x_{tu} - g_{tu}) \\ & + (v_t^- + 1/2)(v_u^+ + 1/2)(x_{tu} - g_{tu}) \\ & + (v_t^- + 1/2)(v_u^- + 1/2)(x_{tu} + g_{tu}) \end{aligned}$$

choosing new constants of anharmonicity

$$x_{tu}^{++} = x_{tu}^{--} = x_{tu} + g_{tu} \qquad \text{and} \qquad x_{tu}^{+-} = x_{tu}^{-+} = (x_{tu} - g_{tu})(1 + \delta_{tu})$$

serves that purpose. Here, δ_{tu} is the Kronecker delta that serves to remedy the failure of the summation condition ($i \leqslant j$) to invoke both x_{tt}^{+-} and x_{tt}^{-+}.

The selection rules for doubly degenerate modes are $\Delta v_t = \pm 1$ and $\Delta l_t = \pm 1$ and the square of the absolute value of the matrix elements relevant for transition intensities are proportional to[110,137,138]

$$\begin{gathered} |\langle v_t,\, l_t | q_{t,\sigma_t} | v_t + 1,\, l_t \pm 1 \rangle|^2 \propto v_t \pm l_t + 2 \\ \text{and} \\ |\langle v_t,\, l_t | q_{t,\sigma_t} | v_t - 1,\, l_t \pm 1 \rangle|^2 \propto v_t \pm l_t \end{gathered}$$

which is the same proportionality as can be obtained for $v_t^\pm \rightarrow v_t^\pm \pm 1$ with the ordinary harmonic oscillator treatment[107].

To finally achieve complete equivalence of the proposed new quantum numbers in the framework of the Hamm group's scheme for predicting spectra of vibrationally hot molecules[2], the population function $P(v_i \cdot \omega_i)$ need to be shown to be equivalent. This requires that

$$P(v_t\,\omega_t) = P(v_t^+\,\omega_t) \cdot P(v_t^-\,\omega_t)$$

which is obviously fulfilled for a Boltzmann distribution, but not in the case of e.g. a

microcanonical distribution.

Note should be made that this treatment does not provide for including vibrational l-doubling[139]. As the required constants r_{tu} were not given in the literature[113], this is an acceptable shortcoming.

Example calculation for the ν_{19} mode of benzene

In the original work[2], the model was applied to benzene, whose anharmonic constants had been determined with good precision earlier[113]. It should be mentioned, however, that this treatment included all of the ten doubly degenerate modes only once[134], thus effectively treating a molecule with a total of twenty normal modes. Restoring twofold degeneracy for the appropriate modes, spectra shown in Figure 3.25 were obtained. Since the vibrational angular momentum quantum number may arguably not be appropriate in solution, spectra were also computed without including vibrational angular momentum coupling.

Predictably and most notably, the increased number of states leads to a substantially greater red shift as the number of vibrational states contributing some shift from the ground state absorption is much greater with thirty than with twenty modes and the anharmonic constants mostly bear negative signs. Moreover, the bands appear much more featureless, as in the case of the coupling to the ν_{14} mode that leads to the small peak at $-7\ \mathrm{cm}^{-1}$ in the 400 K spectrum of the original work and to a shoulder at 600 K, which is rather a shoulder in the recalculated spectra and indiscernible at higher temperatures. Likewise the very small features in the original 800 K spectrum at shifts between $-2\ \mathrm{cm}^{-1}$ and $-7\ \mathrm{cm}^{-1}$ are not discernible anymore. The effect of vibrational angular momentum coupling is to add an additional, but fairly small broadening of the absorption band, as the doubly degenerate modes both increase and decrease the anharmonic constants shared with the observed mode.

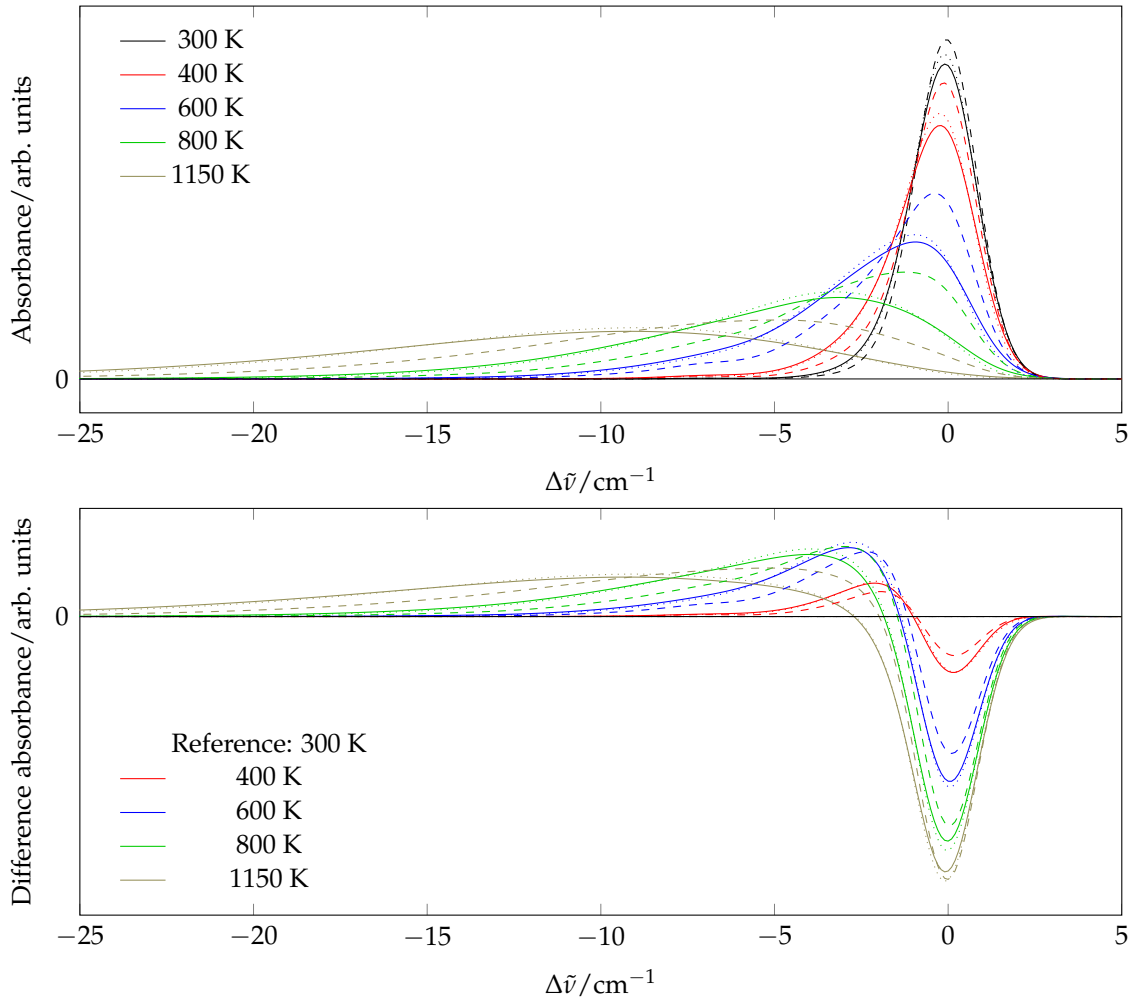


Figure 3.25: Original[2] (dashed) and recalculated (solid: including vibrational angular momentum coupling, dotted: no vibrational angular momentum coupling) spectra (top) and difference spectra (bottom) of the ν_{19} mode of benzene at selected temperatures.

3.5.2 Canonical and microcanonical ensembles of oscillators

After excitation by a short laser pulse and internal conversion to their electronic ground states, the state of the azulene derivatives has been conjectured to be best described as a vibrational wave packet.[124] Experimentally, this wave packet could not be detected, as internal conversion and the randomization of vibrational energy within the azulene moiety could not be separated.[7,69,80] Consequently, energy redistribution within the entire molecule, i.e. the azulene moiety and the aliphatic chain, and partial release of energy to the solvent will lead to a microcanonical ensemble of sample molecules, differing markedly from a thermal energy distribution (see Figure 3.26). As

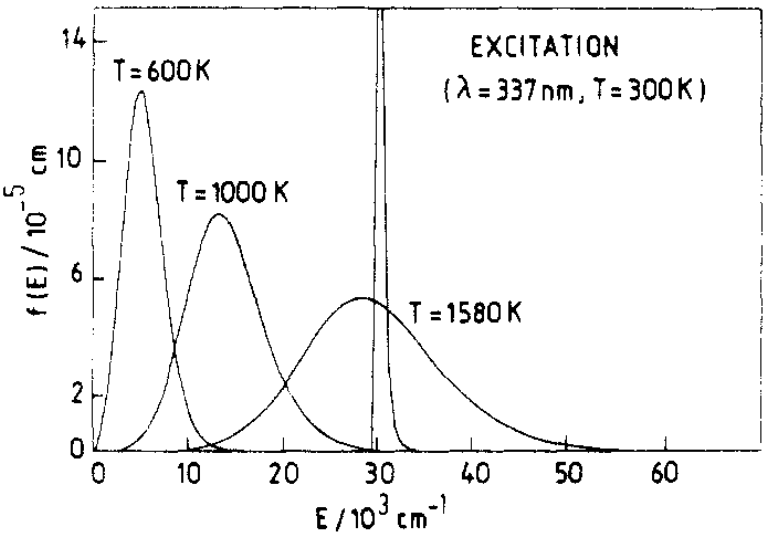


Figure 3.26: Distribution of total internal energy E in azulene for thermal ensembles and after laser excitation. Reprinted with permission from *The Journal of Physical Chemistry*, 1985, 89, No. 21, 4608–4612. © 2009 American Chemical Society.

energy is released to the solvent, the solute and solvent molecules will eventually reach thermal equilibrium at a slightly elevated temperature relative to their initial temperature.[18,72,78]

For a microcanonical ensemble of comparatively large systems of weakly coupled oscillators, such as the molecules investigated in this work or even just their azulene moieties by themselves, the energy distribution of an individual vibrational mode is very similar to that of the canonical case of the same average energy (see Figure 3.27).[140–142] This approximation breaks down for energies close to and above the average energy, where the microcanonical distribution declines sharply, and is thus poor in cases where the high energy part of the distribution is of interest, e.g. in overcoming a reaction barrier.[143] For spectral simulations, as described in section 3.5.1 and also for electronic spectra under certain conditions[80,140,144], the low energy part of the probability distribution, where the bulk resides for the case of the single oscillator, is of greatest importance.

For difference spectra, there is another subtle difference between canonical and microcanonical ensembles that pertains to the low energy part of the distribution. Occupation numbers for the microcanonical distribution decrease sharply to zero as $\langle E \rangle$ is approached, while for the canonical distribution they decline exponentially throughout. This leads to a smaller normalization constant for the microcanonical distribution and eventually a greater absolute population of states low in energy. For normalization

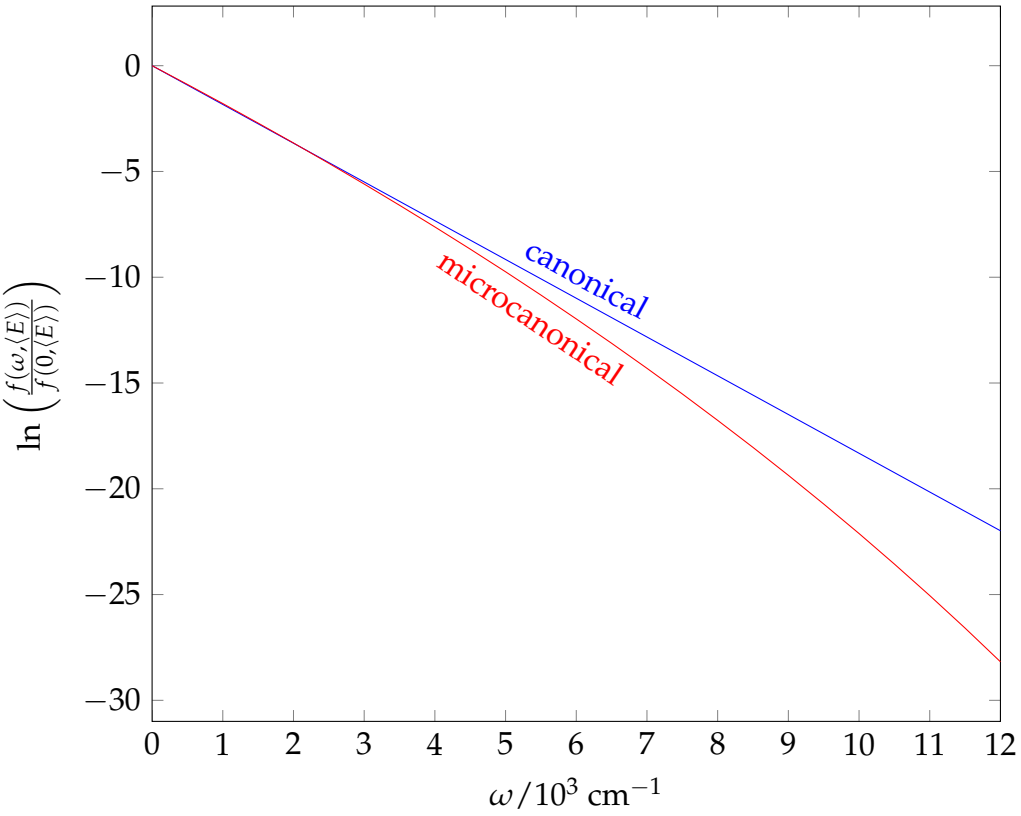


Figure 3.27: Energy distribution in a single oscillator of the azulene molecule. The average internal energy is $\langle E\rangle = 26\,200\ \mathrm{cm}^{-1}$, corresponding to the conditions after excitation from room temperature with a 610 nm laser pulse. $f(\omega, \langle E\rangle)$ is the classical energy distribution function for a harmonic oscillator in a canonical ensemble and a microcanonical ensemble, respectively. Adapted from [142].

against the ground state, as in Figure 3.27, this does not matter, of course, but for difference spectra, two spectra of different temperatures need to be employed, such that absolute occupation numbers do matter. In the classical case, the normalization constant of the microcanonical distribution is $(s-1)/(sk_BT)$[142], where s is the number of oscillators and k_B is Boltzmann's constant. For sufficiently large s, it approaches the canonical value of $(k_BT)^{-1}$, so that this difference can be neglected for the molecules discussed here.

For the present treatment, temperature shall hence take the place of average energy. This also remedies against the ambiguity of the width of the energy distribution: While the energy distribution shortly after excitation is certainly described well by the convolution of the initial thermal room temperature distribution and the exciting laser pulse[69], the precise evolution of the energy distribution is not evident. Not only will the average energy decrease, but the distribution will also broaden and possibly

change its shape, which would introduce additional parameters of uncertain magnitude.

3.5.3 Computational procedure

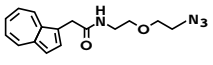
The approach followed here is inspired by Handy *et al.*'s description of their program "Spectro"[111,112,145,146], of which the most important working equations have already been laid out in subsection 3.3.1. The initial step of the protocol applied was the optimization of selected geometries using the program Ghemical and its built-in Tripos 5.2 force field[147,148]. Subsequently, further optimization with the chosen *ab initio* method was conducted, followed by a determination of normal modes by finite displacement and finally the necessary potential constants of third and fourth order, i.e. ϕ_{ijk} and ϕ_{ijkl} in the notation of subsection 3.3.1, were obtained in order to calculate the constants of anharmonicity x_{ij} pertaining to the modes of interest. All quantum chemical calculations were carried out using the Psi4[149] program at the df-MP2[150]/cc-pVDZ[151] level of theory, unless indicated otherwise, while calculations of normal modes, potential constants and anharmonic constants were performed using a home-built Python program, described in section D.2, using SciPy and NumPy[152,153]. The procedure of obtaining third and fourth order potential constants via finite displacements has been described earlier[112,116,154]. For the computation of harmonic force constants, a displacement of $\Delta x_i = 0.005$ Å was used, while for the displacement of dimensionless normal modes a value of $\Delta q_i = 0.1$ was chosen, because numerical difficulties occurred with the value of $\Delta q_i = 0.01$ recommended by Schneider and Thiel[154], although surprisingly little difference was generally found for the H_2O molecule (at most 0.19 cm^{-1} or 1% for x_{23}, but generally less than 0.1 cm^{-1}; values for the potential constants[145] and anharmonic constants[155] are available in the literature).

As the number of potential constants of fourth order generally scales as $\mathcal{O}(n^4)$ and the number of third order constants as $\mathcal{O}(n^3)$, where n is the number of normal modes, substantial reductions have to be made in order to make the task of computing them feasible. From Equation 3.4, it is evident that the number of constants of fourth order which are actually required to compute anharmonic constants scales only as $\mathcal{O}(n^2)$. Limiting the number of modes for which those constants are to be obtained reduces both the number of third and fourth order constants by one order. Finally, using analytic derivatives provided by the quantum chemistry program enables another reduction. As Psi4 only provides gradients as highest order analytic derivatives, the number of finite displacements from the equilibrium geometry required for obtaining anhar-

monic constants pertaining to selected normal modes can thus be reduced to $\mathcal{O}(n)$ at best. This constitutes, however, a manageable task, as opposed to the initial $\mathcal{O}(n^3)$ for the third order potential constants. Yet it should be noted that other programs offer out-of-the-box solutions to the computation of potential constants.[156]

It has to be noted, that this computation should by no means be mistaken to be exhaustive. The molecules at hand probably exist in a number of conformations that would require consideration, as alluded to by others[8,63]. Further, solvent effects are of course not included at any level of theory in the present treatment. Finally, from a theoretical point of view, the effect of different methods and basis sets would require deeper exploration. Despite these shortcomings, some semi-quantitative insight can certainly be gained.

3.5.4 Mode delocalization

Before analyzing the characteristics of the anharmonic potential, the localization properties of normal modes shall be discussed briefly. Here, the molecule  shall be considered, because the observed bands of the *N*-(azido-oligo ethylene glycol)-2-(1-azulenyl)-acetamides are better separated spectrally and because this is the smallest molecule in that group, thus keeping the computational effort manageable. As a simple criterion for localization, the participation ratio

$$p_k = 1 / \left(N \cdot \sum_{m,\alpha} (L_{km}^{\alpha})^4 \right)$$

was used in Figure 3.28, which tends to one if a mode is perfectly delocalized and to N^{-1}, with N being the number of atoms in the given molecule, if a mode is highly localized.[157]

Modes with a harmonic frequency ω_0 of more than 2000 cm^{-1} do not exhibit significant delocalization. The greatest degree of delocalization, on the contrary and as to be expected, can be found in modes of low frequency – especially less than 1000 cm^{-1}, but also up to 1500 cm^{-1} – which either extend over the larger parts of the molecule ("Delocalized") or are azulene skeletal modes. Of the smaller groups, the ethyleneoxy group at the azide end claims most delocalized modes, although this is certainly in part a result of the attribution of the ether oxygen atom to this group, but might also reflect its flexibility as some of its C-H-bending modes appear to be lower in frequency and more delocalized than those of the comparable ethylene group.

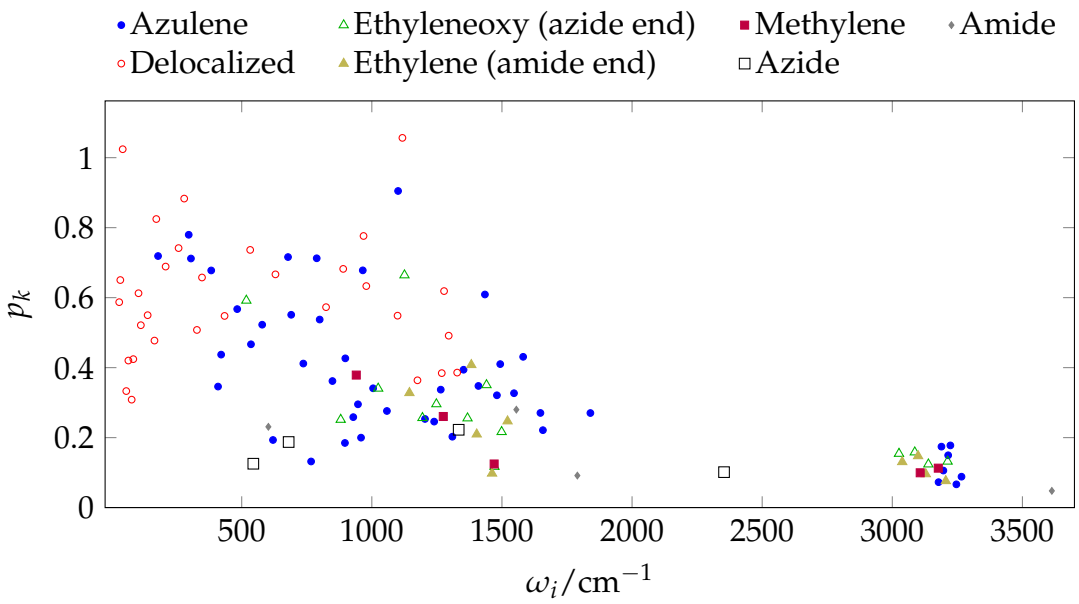
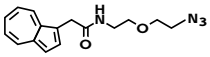
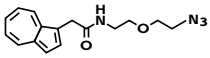


Figure 3.28: Participation ratios p_k of the normal modes of  . If the contribution of the atoms of a moiety to the norm of the normal mode vector, computed as $\sum_{m,\alpha} \left(L_{km}^{\alpha}\right)^2$, where $\alpha = x, y, z$ and m runs over all atoms of the respective moiety, was found to be greater than 0.5, that normal mode was attributed to the corresponding moiety.

Modes of the azide and amide group, which served as sensors in the experiments, are generally very localized. It may hence be inferred that the vibrational energy they report on is also confined to a small, well-defined volume.

A different approach to the representation of the normal mode coefficients can be found in Figure 3.1, where the distribution of normal modes over the molecule is illustrated.

3.5.5 Anharmonic constants

The anharmonicity constants x_{ij} found for  for four modes of interest – amide II, azulene ring distortion, amide I and azide asymmetric stretching – are shown in Figure 3.29. Generally, the vast majority of anharmonicity constants for all three modes discussed is negative, as to be expected. It is further evident that the azulene ring distortion mode (index 93) exhibits significant coupling only to other azulene modes, and particularly strong coupling to those of similar frequency (between 1200 and 1600 cm^{-1}), whose motion mainly involves bond stretching among the carbon

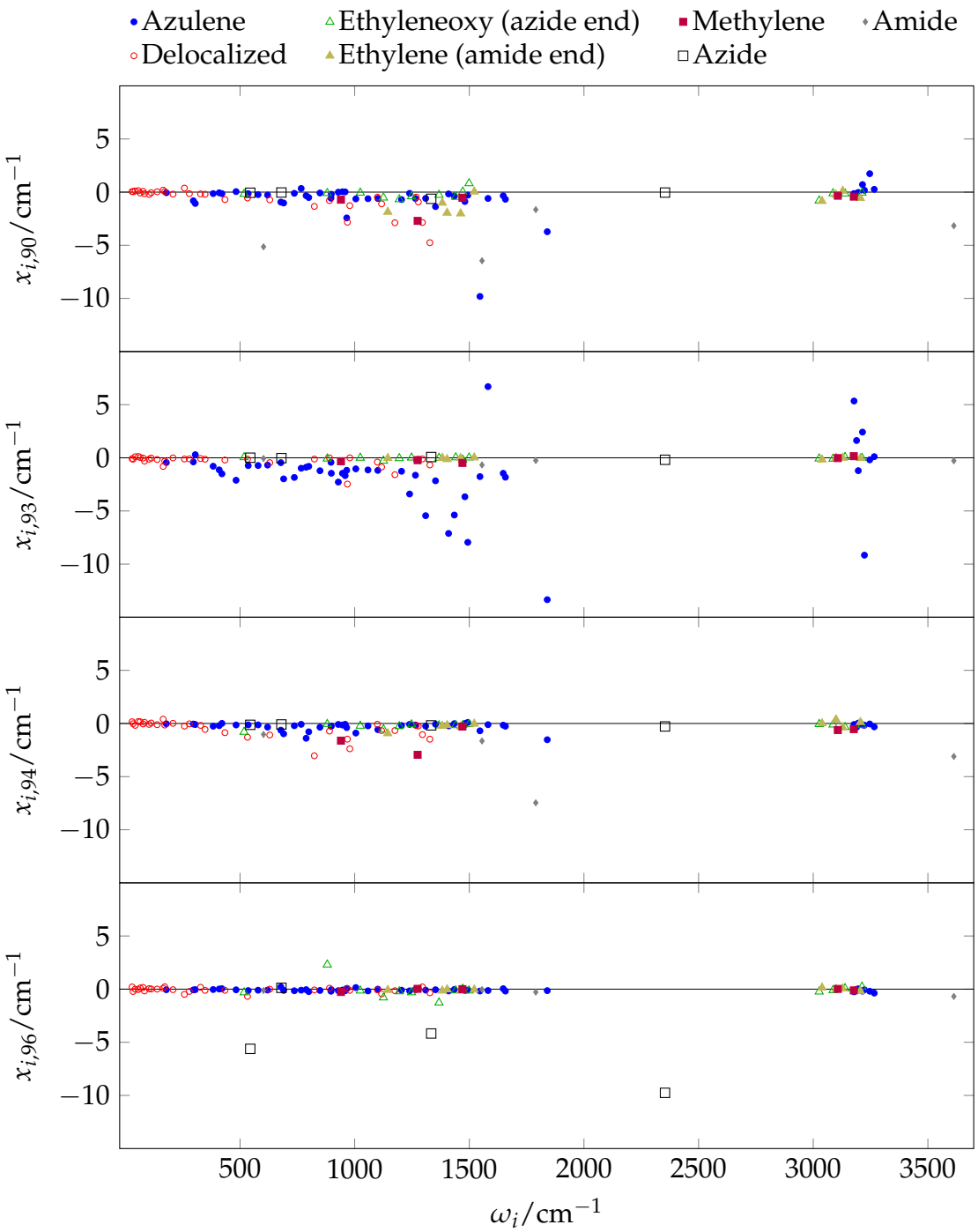
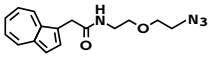


Figure 3.29: Anharmonic constants x_{ij} for selected modes j of  . The modes are from top to bottom: amide II ($j = 90$), azulene ring distortion ($j = 93$), amide I ($j = 94$) and asymmetric azide stretching ($j = 96$). Assignment of modes to moieties is the same as in Figure 3.28.

atoms forming the azulene skeleton. The diagonal anharmonicity is a mere $-1.8\ \mathrm{cm}^{-1}$, while the strongest coupling is to the highest frequency C-C stretching vibration. Some positive anharmonic constants were found for C-H stretching vibrations, but the very high frequencies of these modes allow one to expect them to have only little influence on the spectrum.

The anharmonic constants found for the amide I and II modes clearly manifest their role as reporters along the chain with appreciable coupling only to those azulene ring modes that are similar in frequency to the amide modes. At the same time, coupling to modes of the neighboring methylene and ethylene units, as well as to the amide unit itself and even some delocalized modes, is increased compared with the azulene ring distortion mode. Diagonal anharmonic constants are intermediate with -7.5 and $-6.5\ \mathrm{cm}^{-1}$ for the amide I and II modes, respectively, and in the case of the former it is the largest anharmonic constant, as to be expected for a strongly localized mode. Both values appear reasonable in comparison to previously measured and calculated values[158].

While none of the previously discussed modes have any significant anharmonic constants with the modes of the azide group, the asymmetric azide stretching (ν_3) mode itself couples almost exclusively to those. Other than that, anharmonic constants with only a few modes of the neighboring ethyleneoxy group are appreciable magnitude. The diagonal coupling of $-9.8\ \mathrm{cm}^{-1}$ is strongest but seems rather small in comparison with the experimental value of the local mode diagonal anharmonicity reported by Rubtsov for a comparable organic azide of $\Delta = 35\ \mathrm{cm}^{-1}$ and also with their theoretical value of $29.2\ \mathrm{cm}^{-1}$[66] (with $\Delta = -2x_{ii}$[158,159]). It is succeeded in magnitude by coupling to one of the azide bending (ν_2, $-5.6\ \mathrm{cm}^{-1}$) modes and finally to the symmetric strecthing mode (ν_1, $-4.2\ \mathrm{cm}^{-1}$). Coupling to the other bending mode, whose motion mainly involves the central nitrogen atom of the azide group and the carbon atom nearest to the azide group, is vanishingly small ($0.1\ \mathrm{cm}^{-1}$).

3.5.6 Simulated spectra

Transient spectra were simulated using the method laid out in section 3.5.1 and the anharmonic constants presented in Figure 3.29. The spectra are shown in Figure 3.30. Four aspects are apparent from these results. First, as was expected from the relative size of the positive and negative parts of the experimental signal, none of the modes is described well by the electrically harmonic approximation (see subsection 3.3.1). Instead, all absorptions of vibrationally hot species are less intense than the bleached

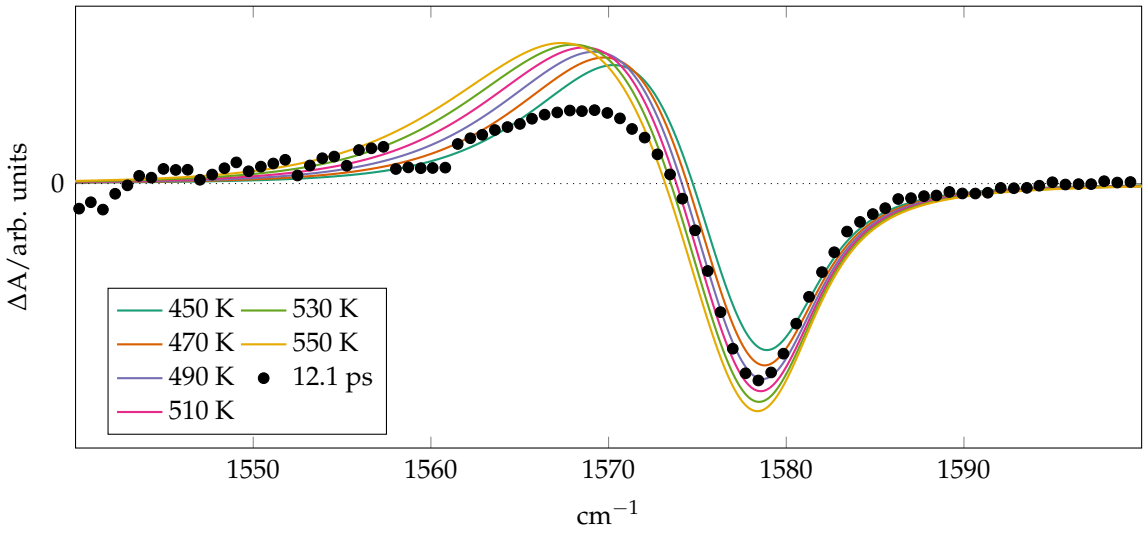
ground state spectrum. For the asymmetric azide stretching mode, this effect appears to be least severe.

Second, simulated spectra that approximately agree with the experimentally observed ones with regard to the relative shift of the positive and negative extrema of the signal are inconsistent with a canonical energy distribution. The experimental spectra shown in Figure 3.30 were all recorded at a delay of about 12 ps and ought to reflect the same "temperature" or average energy distribution, while the most similar model spectra – setting aside the mismatch due to electrical anharmonicity – correspond to only about 200 K above room temeperature for the azulene moiety, to about 700 K above room temperature for the amide I mode, and to more than 1000 K above room temperature in the case of the asymmetric azide stretching mode. Like the atypical behavior of the asymmetric azide stretching – and amide I modes, this is an indication that the randomization of vibrational energy in the molecule is not as complete as was assumed. This has been pointed out for other situations of IVR already.[160] Moreover the apparent temperature reported by the azide mode is larger than the initial temperature of the azulene unit after excitation of 1100 K[122], which can not be consolidated with a canonical distribution of the pump energy.

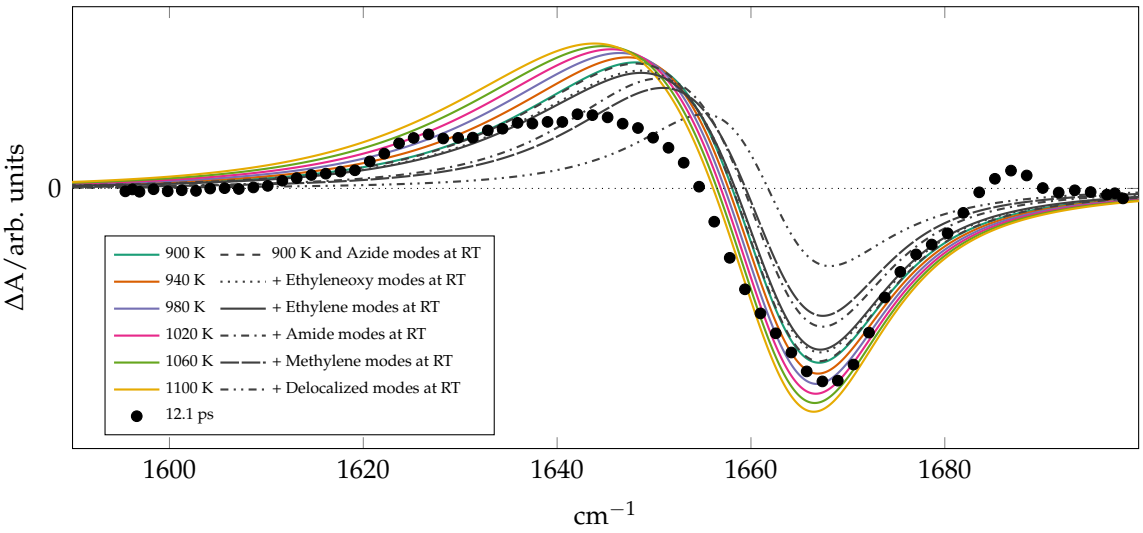
Third, as in the case of the ν_{19} mode of benzene, as anharmonic constants with negative sign prevail, vibrationally hot spectra are predicted to shift to lower frequencies. This is obviously a gradual shift, as observed experimentally for the azulene ring distortion mode (section 3.3.2). But the approach used to simulate vibrationally hot spectra cannot reflect the experimental observation of the amide, azide, and carbonyl modes (section 3.3.3, section 3.3.4, and section 3.3.4, respectively), which did not exhibit much of a spectral shift at all (with the exception of the slight shift of the amide I mode).

Fourth, the model does not offer insights into the observed peculiarities in the behavior of the amide mode. The amide I bleach – and to a lesser extent also the absorption – was found to exhibit a shift to greater frequencies at greater delay times. Intuitively, this might be thought of as an indication for the progress of randomization. In that case, earlier spectra would reflect a vibrationally hot azulene and a cold azide, while the groups in between would be either hot, cold, or intermediate.

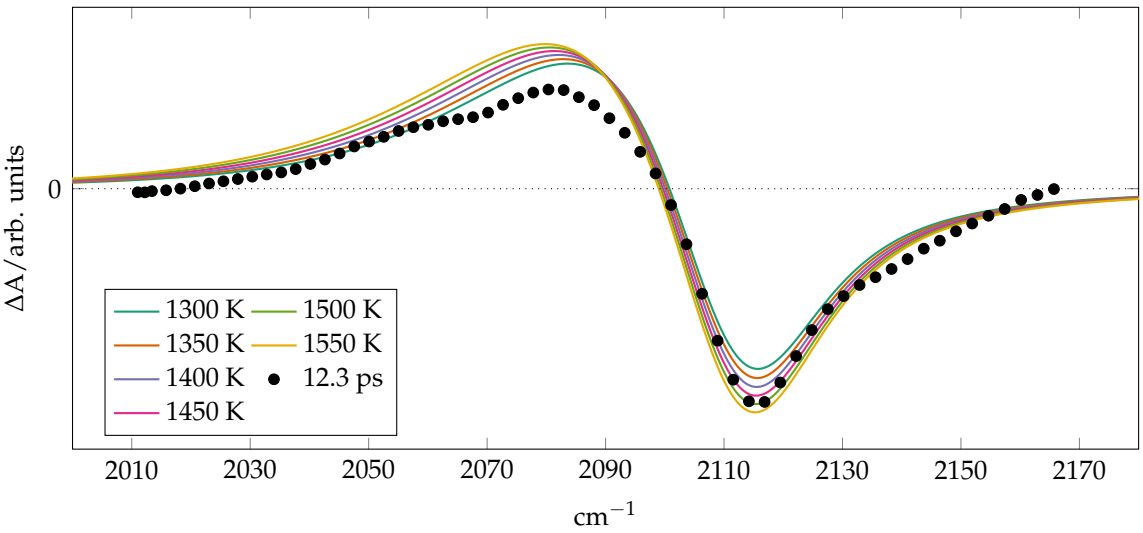
In an attempt to simulate these situations for the amide I spectrum (Figure 3.30(b)), groups were successively set to room temperature, starting with the azide group as the most distant one (dashed spectra in Figure 3.30(b)). The spectrum with only the azide group at room temperature is virtually identical to the original 900 K spectrum.



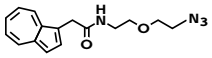
(a) Azulene ring distortion mode



(b) Amide I mode



(c) Azide asymmetric stretching mode

Figure 3.30: (facing page) Difference spectra of key absorption bands calculated based on anharmonic constants x_{ij}. Canonical ensembles of the indicated temperatures were assumed and phenomenological line shape functions fitted to the observed spectra were used. The geometry of  is given in Appendix B. Selected experimental data (•) are shown for comparison. The dashed spectra shown for the amide band were obtained by successively setting the "temperature" of the modes belonging to the listed groups to a room temperature value of 296 K; e.g. in the spectrum titled "+ Delocalized modes at RT", all but the azulene modes are at room temperature.

Additionally setting the neighboring ethyleneoxy group to room temperature has a visible, but small effect, which is – surprisingly – larger than that of adding the ethylene group (i.e. setting the entire polyethylene glycol chain and azide group to room temperature). "Cooling" the amide group itself yields a stronger response due to the larger anharmonic constants, while the methylene group has an effect on the amide mode similar to the ethyleneoxy group. The largest effect was achieved by decreasing the temperature of the modes classified as delocalized, which is owed to their large number and – on average – low frequencies. The "cooling" of all of these groups triggers a blue shift of the amide absorption, which results in a substantial shift of the positive signal to higher frequencies, but a much smaller one of the bleach.

This is the opposite of the experimental result in two respects: a red shift for presumably less randomized situations and a larger shift upon randomization for the bleach than for the positive part of the signal was observed in the experiment, while the model suggests a blue shift upon partially cooling the system and the shift is large for the positive part of the signal, while it is hardly visible for the bleach. These processes are thus either more subtle than assumed or the premises of the modeling are wrong.

3.6 Conclusions and Outlook

In this section, the findings of this study shall be summarized with regard to the energy loss of the azulene unit, energy transfer along molecular chain structures, IR bands as sensors for vibrational energy, and the description of IVR processes. The finding of earlier studies[7,69,75,80,81] that the loss of energy from the azulene unit tends to saturate with increasing chain length, has generally been confirmed. However, instead of the earlier assessment that energy loss rates for chains with more than four bonds

is "essentially constant"[7], the data of this study rather supports an ongoing asymptotic decline even for longer chains. While the earlier conclusion was essentially based on only two data points and supporting MD simulations, the result of the newer experiments does not strictly contradict it, considering that the molecular systems and experimental techniques employed differed substantially.

With regard to the signal arrival time at the terminal group of the chain, results obtained simultaneously by the Rubtsov group[8,63] were confirmed as well. Signal arrival times were found to be proportional to chain length and in good quantitative agreement, leading to the conclusion that energy transfer proceeds ballistically. The presented results differ in two respects from the Rubtsov group's findings: First, there is a hint at a fall-off behavior from proportionality for very short chains, as found in earlier studies[7] and MD simulations[69]. However, an insufficient number of chains and insufficiently short chains were investigated in this study to substantiate this hint. Second, polyethylene glycol based chains were found to exhibit systematically larger signal arrival times than found by Lin *et al.*[8,63] by a constant offset. Most likely, this is due to the transfer of vibrational energy from the modes of the azulene moiety to the chain being statistically unfavorable compared to the transfer from a single oscillator as in their experiments. At the same time, values for alkyl-based chains exhibited the same increase of arrival time with chain length, but were at the lower limit of the values reported by Lin *et al.* and thus offset from the polyethylene glycol-based chains by a nearly constant amount. Earlier studies[7,75,81] concluded that this was due to a disturbance of strongly delocalized normal modes by hetero atoms in the chain.

Closer inspection of the spectral response of the bands used for monitoring energy flow revealed that the assumption of nearly canonical energy distribution seems to hold well only for the azulene ring distortion mode. Amide I, azide asymmetric stretching, and carbonyl bands, on the other hand, show surprisingly little shift of the absorptions of the vibrationally excited molecules, indicating that a statistical distribution of energy in the vicinity of these modes is probably not reached, but instead transfer of energy of a limited number of some excited modes to the solvent prevails. The absorption of the amide I mode exhibits a particularly interesting shift after passing the maximum of its signal, which could not be explained on the basis of the shifts induced by anharmonic coupling to other groups of the respective molecules. It is conceivable that a closer inspection of individual modes instead of groups of modes is necessary, but the signal generally underscores the aptitude of the amide linker group to reflect the progress of IVR.

A simple diffusive-like model of energy transport has been discussed. Its deficien-

cies in describing the temporal profile of the experimental signals can well be explained by the difficulty to make sound assumptions about the nature of the transport. The fast rate of energy transport found, again, suggests it is not of diffusive, but rather of ballistic nature, as they are similar to the value of 450 m/s reported earlier[8]. The fact that energy transport in alkyl chains on gold surfaces is more than twice as fast (950 m/s[68]) might be owed to a greater degree of order in these systems.

Future research should focus on establishing a mode-based treatment of both the process of IVR in medium-sized and large molecules and its monitoring by IR spectroscopy. The approach taken here to analyze the spectral response of the different marker bands seems promising but requires improvement in several respects. First, anharmonic constants need to be determined reliably including a possible conformational distribution. It seems desirable to limit conformational flexibility in order to reduce uncertainties, and it is certainly necessary to study the dependence of the anharmonic constants on method and basis set. Second, a better understanding of the energy distribution is required. Simulations of the energy redistribution dynamics can help to determine how the energy distribution most likely differs from a canonical one. Recent progress has been achieved in this regard based on Marcus' electron transfer theory.[100] Third, experimental investigation of the anharmonic interactions between vibrational modes, as well as better-defined techniques of excitation, for instance of individual vibrational modes, are needed. Recent studies using 2DIR spectroscopy have made progress in both of these directions.[66,101]

Chapter 4

Interactions in weakly bound noble gas–halide clusters

In an effort to enhance the understanding of IVR in the azulene compounds which were investigated experimentally for this work, a computer program was designed to allow for the easy implementation of arbitrarily complex potential energy functions. The program is documented in section D.3. It was intended to be also employed in the analysis of instantaneous normal modes of the azulene derivatives described in the previous chapter, because the use of more complex potentials, such as the ones developed by Steele[161–163], in molecular dynamics programs is often not straightforward.

These efforts remained unfinished.

4.1 Model of the potential energy

For calculating electron affinities (EAs) of clusters comprised of single halide anions (henceforth designated "X^-") and multiple noble gas atoms (henceforth "Ng"), the approach suggested by Yourshaw and coworkers[3] was used. In this section the procedure used shall be described: broadly where Yourshaw's approach was exactly followed and in greater detail where modifications where introduced.

The potential energy functions of neutral and anionic states of an Ng_nX cluster are schematically depicted in Figure 4.1. While for the anion only its singlet ground state is of interest, neutral clusters possess three doubly degenerate electronic states: splitting between the $^2P_{1/2}$ and $^2P_{3/2}$ states occurs already in the absence of noble gas atoms due to spin-orbit interaction. The quadruple degeneracy of the latter state is further lifted

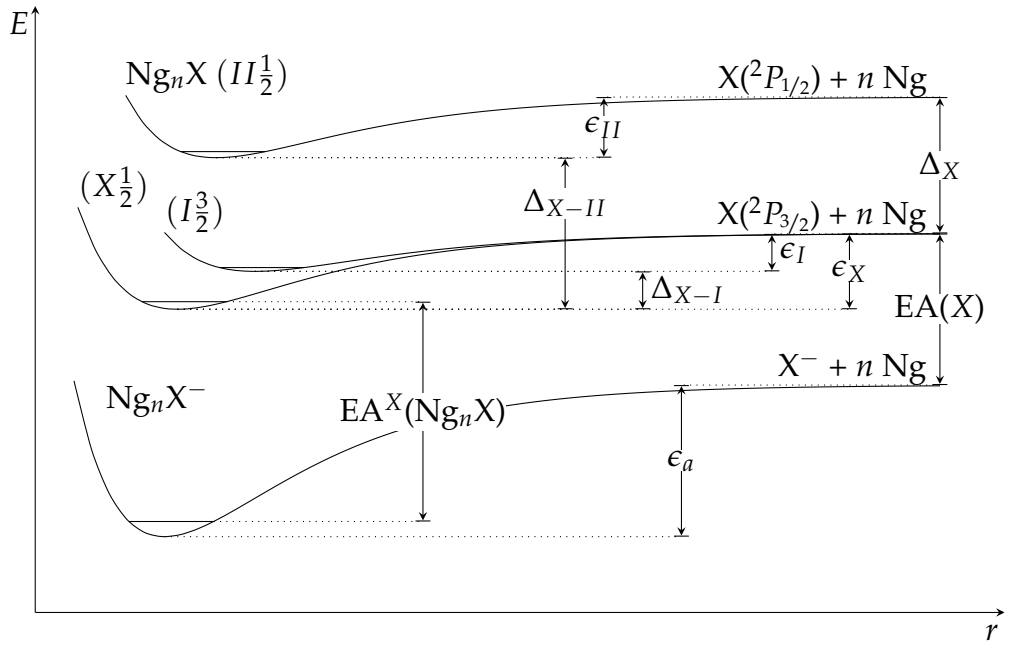


Figure 4.1: Schematic representation of the energy levels of a halide–noble gas system, adapted from [3].

as the noble gas atoms interact with its p-shell hole.[164]

The spin-orbit splitting of the neutral halide atom Δ_X and the electron affinity of its ground state $\mathrm{EA}(X)$ are readily available in the literature. In order to calculate the electron affinity of the cluster, it is now necessary to compute the zero-point energies of the anion and the desired neutral state as well as the potential energy well depth ϵ. Naturally, both of these tasks have to be preceded by a search for the global minimum energy geometry of the anionic cluster, which was conducted using the BFGS (Broyden-Fletcher-Goldfarb-Shanno) optimizer[165] of the GNU Scientific Library[166]. Zero-point energies E_{ZP} can then be found in the harmonic approximation as $E_{\mathrm{ZP}} \approx \frac{1}{2}\sum_i \omega_i$ from the harmonic frequencies ω_i, which in turn were obtained via finite differences of either energies or, where available, gradients of the potential (see subsection 3.3.1, different isotopic masses were disregarded for these calculations).

For geometry optimization, it was deviated from Yourshaw *et al.*'s simulated annealing scheme. Instead, several initial guess geometries were optimized directly using the BFGS algorithm. The initial guesses for an Ng$_n$X cluster were generated by adding a noble gas atom to the optimum geometry of the Ng$_{n-1}$X cluster on a sphere. The distance between the halide anion and the most distant noble gas atom was used as the radius of the sphere and occasionally doubled, if none of the first set of optimizations converged properly. Positions on the sphere where chosen in steps of usually

$\frac{\pi}{12}$ of the altitude angle θ with $0 \leqslant \theta \leqslant \pi$ and steps of $\frac{\pi}{6 \sin \theta}$ of the azimuth angle ϕ with $0 \leqslant \phi < 2\pi$ (note that while $\phi = 0$ is the same position as $\phi = 2\pi$, $\theta = 0$ and $\theta = \pi$ correspond to opposite poles of the sphere), always including both poles (totalling 188 initial guesses with the described parameters).

This scheme requires less effort for each initial guess as no simulated annealing needs to be conducted, although the guesses themselves may at times be rather unreasonable and the minimum search may hence take more iterations to converge or not converge at all. When comparing with Yourshaw *et al.*'s Ar_nI dataset[3], in all but two cases were the anionic minimum energies found using this approach. The two deviations were found for $Ar_{11}I^-$ and $Ar_{14}I^-$, where binding energies ϵ_a of 5647.2 and 7371.8 cm^{-1}, respectively, had been found earlier[3], while with the described scheme, those were 5677.9 and 7389.4 cm^{-1}, respectively. As the object is to find the global energetic minimum, the latter rather than the original values have to be regarded as correct, although there may still be lower ones as it was – naturally – refrained from performing an exhaustive search in the conformational space due to the NP-hardness of the optimization problem[167,168].

It has to be stressed, however, that this result does not stem from any insufficiency in the simulated annealing scheme. Rather, many-body effects had been explicitly excluded in the simulated annealing due to their computational cost and merely included to re-optimize minimum energy geometries found using additive potentials only. Indeed, the new global minimum structures prove to have less binding energy than those reported earlier, if only additive potential energy contributions are used.

Additionally, as mentioned above, only harmonic frequencies were used for the computation of the zero point energy, as opposed to Yourshaw *et al.*'s discrete variable representation procedure. Comparing to their Ar_nI dataset[3], there seems to be no obvious systematic trend that would suggest lower zero point energies due to anharmonicity effects. Generally, deviations due to the harmonic approximation were found to be negligible, given the accuracy of the experimental data available[9].

With these utilities available, the task at hand was to compute the potential well depth ϵ for the anionic and neutral states, or rather the potential energy functions $V(\underline{r})$.

4.1.1 Potential energy of neutral clusters

Neumark and coworkers[3] included three contributions to the potential energies of the neutral clusters: pairwise interactions among the noble gas atoms, "matrix-additive"

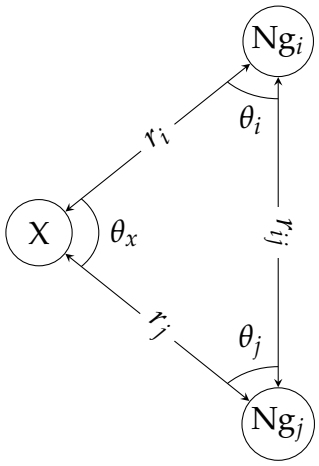


Figure 4.2: Lengths and angles for the triple dipole potential.

interactions between the halide atom and the noble gas atoms, and Axilrod-Teller triple-dipole interactions. Three-body exchange interactions were disregarded as their treatment proved too difficult and their contribution too small.[3]

Here, this treatment was followed in general, and the potentials and parameters used are listed in Table C.1, C.3, and C.2 for the interaction of the halide atom with the noble gas atoms and Table C.4 for the pairwise interaction of the noble gas atoms among themselves, respectively. For the Axilrod-Teller triple dipole term, Yourshaw's treatment rests on the definition of the potential V_{ddd}, the C_9 constant and the atomic average excitation energy η_k in the following form:

$$V_{ddd} = C_9 \frac{3 \cos\theta_x \, \cos\theta_i \, \cos\theta_j}{r_{ij}^3 \, r_i^3 \, r_j^3}$$

$$C_9 = \frac{3}{2} \alpha_x \alpha_i \alpha_j \frac{\eta_x \eta_i \eta_j \, (\eta_x + \eta_i + \eta_j)}{(\eta_x + \eta_i)(\eta_x + \eta_j)(\eta_i + \eta_j)}$$

$$\eta_k = \sqrt{\frac{N_k}{\alpha_k}}$$

where α_k is the dipole polarizability and N_k the effective number of electrons for the given particle, respectively.[3] All dimensions required are illustrated in Figure 4.2. Values of C_9 were computed from the dipole polarizabilities and effective numbers of electrons of the individual particles as given in Table C.6.

In their treatment of spin-orbit coupling in clusters of electrically neutral, i.e. open shell, halide atoms, Yourshaw *et al.*[3] largely relied on the model proposed by Lawrence and Apkarian[164]. The Hamiltonian of this model is restricted to additive

noble gas–noble gas interactions, anisotropic noble gas–halide interactions and spin-orbit coupling. The latter is assumed not to be influenced by the noble gas shell and the interaction between the halide and its noble gas shell is assumed not to impact on the interaction of the noble gas atoms among themselves.

In light nuclei, such as oxygen[169], spin-orbit coupling may of course be neglected and the problem is thus reduced to treating the splitting of angular momentum. Particularly with the heavier halides, however, spin-orbit interaction needs to be incorporated. It is convenient for this purpose to express the halide–noble gas Hamiltonian in the basis of the $|j\, m_j\rangle$-eigenstates of the open shell atom. In this representation, the spin-orbit Hamiltonian is simply a diagonal matrix that may be added to the halide–noble gas interaction Hamiltonian. Diagonalization of this combined Hamiltonian, as given by Lawrence and Apkarian[164], yields the eigenstates of the halide–noble gas interaction.

It is obvious, however, that an analytical solution of this problem is desirable in order to accelerate model calculations of the aforementioned systems. Yourshaw already commented on the considerable improvement in speed achieved thereby[3], but the solution obtained at the time still proved rather unwieldy[170]. For this work, Lawrence and Apkarian's Hamiltonian[164] was also used and a much simpler analytical solution to the problem of an atom with a single electron or hole in a P-state surrounded by closed-shell (S-state) atoms was derived.

Here, the following notation for the treatment of the anisotropic component of the interaction will be used

$$V_{xy} = \frac{3}{5}\sum_i \frac{x_i\, y_i}{r_i^2}\, V_2 \qquad V_{y^2-x^2} = \frac{3}{10}\sum_i \frac{y_i^2 - x_i^2}{r_i^2}\, V_2$$

$$V_{xz} = \frac{3}{5}\sum_i \frac{x_i\, z_i}{r_i^2}\, V_2 \qquad V_{z^2} = \frac{1}{10}\sum_i \left(\frac{3z_i^2}{r_i^2} - 1\right) V_2$$

$$V_{yz} = \frac{3}{5}\sum_i \frac{y_i\, z_i}{r_i^2}\, V_2$$

The respective sums run over all atoms and the isotropic part is $V_0 = (V_\Sigma + 2V_\Pi)/3$, while the anisotropic part of the potential is $V_2 = 5(V_\Sigma - V_\Pi)/3$, as usually defined[3,171]. The degeneracy of the ^2P state of the halide atom being lifted in the presence of a closed-shell noble gas atom, V_Σ and V_Π are the potentials of the two eigenstates of the diatomic case.[171,172]

With spin-orbit splitting included, the ^2P state is itself already non-degenerate,

yielding two terms, $^2P_{1/2}$ and $^2P_{3/2}$. While the former is of pure V_Π character, the latter bears both V_Σ and V_Π contributions, and it is split by the presence of a closed-shell noble gas atom, such that the states depicted in Figure 4.1 arise. In terms of these spectroscopic potentials, the isotropic component is then $V_0 = \frac{1}{3}\left(V_{X\frac{1}{2}} + V_{I\frac{3}{2}} + V_{II\frac{1}{2}}\right)$, whereas the anisotropic component is $V_2 = \frac{3}{5}\left(V_{X\frac{1}{2}} - 2V_{I\frac{3}{2}} + V_{II\frac{1}{2}}\right)$.[3,172]

The elements of the corresponding Hamiltonian in the $|l\, m_l\rangle$-basis can then be written as:[164,173,174]

$$V_{0,0} = \sum_i V_0 + 2V_{z^2} \qquad V_{1,1} = \sum_i V_0 - V_{z^2}$$
$$V_{1,0} = \sum_i V_0 + \frac{1}{\sqrt{2}}\left(-V_{xz} + i\, V_{yz}\right)$$
$$V_{1,-1} = \sum_i V_0 + V_{y^2-z^2} + i\, V_{xy}$$

From here on, the isotropic component V_0 will not be considered as it is simply additive. Using the Maple 12.01 analytical software[175] it was possible to obtain analytical solutions of the eigenvalues. As in Yourshaw's version[170], these proved to be rather bulky. Closer inspection, however, allowed for some significant simplifications to finally arrive at rather concise expressions.

The structures of the three doubly degenerate eigenvalues thus obtained were

$$E_{1,2} = -\frac{1}{2}\left(a + \frac{b}{a}\right) \pm i\frac{\sqrt{3}}{2}\left(a - \frac{b}{a}\right) \quad \text{and} \quad E_3 = a + \frac{b}{a}$$

with E_1 corresponding to the X state, E_2 to the I state, and E_3 to the II state, and where

$$b = \left(\frac{\Delta}{3}\right)^2 + \frac{V_{xy}^2 + V_{xz}^2 + V_{yz}^2 + V_{y^2-x^2}^2}{3} + V_{z^2}^2 \tag{4.1}$$

is obviously a positive, real number. Since the Hamiltonian is a Hermitian matrix, its eigenvalues are necessarily real. While $E_3 = a + b/a$ must already be real by itself, since it constitutes an eigenvalue of the Hamiltonian, $a - b/a$ must be purely imaginary in order to render the second summand in $E_{1,2}$ real and thus make $E_{1,2}$ itself real as well. Combining these conditions leads to $b/a = a^*$ and thus $b = |a|^2$. Introducing $a_r = \mathrm{Re}(a)$ and $a_i = \mathrm{Im}(a)$, it then follows immediately that

$$E_{1,2} = -a_r \mp \sqrt{3}\, a_i \qquad E_3 = 2a_r \tag{4.2}$$

At this point it proved advantageous to switch to polar representation, for which the modulus of a is readily available as $|a| = \sqrt{b}$. The associated polar angle ϕ_a then remains to be found. Closer inspection of the computer-generated a revealed that it is of the structure

$$a_r + i\,a_i \equiv |a|\,\exp(i\phi_a) = \sqrt[3]{c_r + \sqrt{\dots}} = \sqrt[3]{|c|}\,\exp(i\phi_c/3) \tag{4.3}$$

where

$$\begin{aligned} c_r = \left(\frac{\Delta}{3}\right)^3 + V_{z^2}\left(\frac{V_{xz}^2 + V_{yz}^2}{2} - V_{xy}^2 - V_{y^2-x^2}^2 + V_{z^2}^2\right) \\ + V_{xy}\,V_{xz}\,V_{yz} + \frac{V_{y^2-x^2}}{2}\left(V_{yz}^2 - V_{xz}^2\right) \end{aligned} \tag{4.4}$$

is also a real, but not necessarily positive number. The term abbreviated as $\sqrt{\dots}$ is the most cumbersome part of a. While the radicand is real, it is extremely tedious to verify that it is also negative.For all situations of interest, however, the opposite case of a positive radicand can be ruled out since it would lead to a real a and thus degenerate eigenvalues $E_{2,3}$. It should be mentioned, however, that in some cases where $\phi_a \to 0$ when the first shell of noble gas atoms around the halide atom is complete or nearly complete, one of the states could not be converged to a minimum energy geometry. This is probably due to near-degeneracy, and these states cannot be disentangled easily; using the solution from explicitly diagonalizing the Hamiltonian led to the same problem. For these cases, it would be desirable to obtain a non-periodic (see below) set of solutions.

Provided that $\sqrt{\dots}$ is imaginary, the angle of a in polar representation, ϕ_a can easily be obtained from eq. Equation 4.3:

$$\phi_a = \frac{1}{3}\arccos\frac{c_r}{|a|^3} = \frac{1}{3}\arccos\frac{c_r}{b^{\frac{3}{2}}} \tag{4.5}$$

Using some basic trigonometric operations, the eigenvalues in eq. Equation 4.2 can be cast into the form

$$E_n = 2\sqrt{b}\,\cos\left(\phi_a - \frac{4n\,\pi}{3}\right) \tag{4.6}$$

It may be worth mentioning that despite the constraint of $0 \leqslant \phi_a < \pi/3$ in the determination of the polar angle via eq. Equation 4.5, the periodicity of the eigenval-

ues in eq. Equation 4.6 warrants for covering the entire eigenvalue spectrum. Thus, Equations 4.1, 4.4, 4.5 and 4.6 as working equations establish a very concise scheme for treating the interaction of a single *P*-state atom with a number of closed shell atoms.

4.1.2 Potential energy of anionic clusters

The contributions to the potential energy used by Yourshaw *et al.*[3] were employed in a similar fashion. In detail, these included

- binary potentials as laid out in Table C.1 and Table C.4,
- Axilrod-Teller triple dipole interactions as described in the previous section on neutral clusters with the appropriate parameters from Table C.6,
- the interaction of the halide anion with the exchange quadrupole formed by pairs of closed-shell noble gas atoms using the parameters from Table C.5, and
- the interaction of the halide anion with those di- and quadrupole moments induced by dispersion.

For the latter, the expressions given by Yourshaw *et al.*[3] were simplified to yield

$$V_{\text{disp}} = q_x \sum_{i \neq q} \frac{C_6^{ii} B_i}{\alpha_i} r_i^{-3} \left\{ \frac{3}{2} \vec{r}_i \cdot \sum_{j \neq q,i} \vec{r}_{ij}\, r_{ij}^{-8} - \frac{1}{8} \sum_{j \neq q,i} \left(3 \left[\frac{\vec{r}_i \cdot \vec{r}_{ij}}{r_i\, r_{ij}} \right]^2 - 1 \right) r_{ij}^{-6} \right\}$$

where $\vec{r}_i$ is the vector from the noble gas atom to the halide and $\vec{r}_{ij}$ from the jth to the ith noble gas atom, analogous to the Axilrod-Teller potential in Figure 4.2. The corresponding particle parameters are given in Table C.6.

Finally, the quantitatively most important – except for the binary potentials – and simultaneously from a computational standpoint most costly contribution is induction nonadditivity. The treatment outlined by Yourshaw *et al.*[3] can be summarized as follows: The constituents of the cluster, i.e. the halide anion and the noble gas atoms, are characterized by point dipole and quadrupole moments, both of which are induced by the point charge of the halide as well as all the respective moments of the atoms. Terms higher than linear in either of these moments are neglected. The working equations for the determination of the induced moments are (originally labeled Eqs. (21)

through (24) in [3]):

$$E_{\alpha}^{(i)} = \sum_{j \neq i} \left(-T_{\alpha}^{(ij)} q_j + T_{\alpha\beta}^{(ij)} \mu_{\beta}^{(j)} - \frac{1}{3} T_{\alpha\beta\gamma}^{(ij)} \Theta_{\beta\gamma}^{(j)} \right) \tag{4.7a}$$

$$E_{\alpha\beta}^{(i)} = \sum_{j \neq i} \left(-T_{\alpha\beta}^{(ij)} q_j + T_{\alpha\beta\gamma}^{(ij)} \mu_{\gamma}^{(j)} - \frac{1}{3} T_{\alpha\beta\gamma\delta}^{(ij)} \Theta_{\gamma\delta}^{(j)} \right) \tag{4.7b}$$

$$\mu_{\alpha}^{(i)} = \alpha_i E_{\alpha}^{(i)} \tag{4.7c}$$

$$\Theta_{\alpha\beta}^{(i)} = C_i E_{\alpha\beta}^{(i)} \tag{4.7d}$$

where $E_{\alpha}^{(i)}$ and $E_{\alpha\beta}^{(i)}$ are the electric field and its gradient, respectively, at center i. Also, q_i, $\mu_{\alpha}^{(i)}$, and $\Theta_{\alpha\beta}^{(i)}$ are the charge, the α-component of the induced dipole moment and the $\alpha\beta$-component of the induced quadrupole moment of center i. Further, α_i and C_i are the dipole and quadrupole polarizabilities of the ith atom. Finally, $T_{\alpha\beta\cdots\nu}^{ij} = \nabla_\alpha \nabla_\beta \cdots \nabla_\nu R_{ij}^{-1}$ and $\underline{R}_{ij}$ is the vector from the jth to the ith atom in accordance with Buckingham's notation[176].

In their work, Yourshaw *et al.*[3] continue to describe an iterative scheme for the determination of the induced multipoles. This process could be confirmed to converge satisfactorily and can be sped up in successive computations at slightly modified geometries – i.e. in molecular dynamics calculations or geometry optimizations – by initiating the procedure with the multipole moments obtained for the last geometry. However, this still remains a very time-consuming procedure and, as an additional disadvantage, analytical derivatives are not available.

A more direct scheme, which also remedies the lack of analytical derivatives, was found to be more efficient. This broadly follows the polytensor formulation proposed by Applequist[177,178] and is essentially an extension along the lines of the treatment by Kong and Ponder[179,180].

Due to their linear form and incorporating the inter-dependence of the multipole moments, Equations 4.7a through 4.7d can be cast into a single matrix-vector equation as follows (vectors bear single and matrices double underlining for clarity):

$$\underline{\mu} = \underline{\underline{\alpha}} \left(\underline{q} + \underline{\underline{T}}\, \underline{\mu} \right) \tag{4.8}$$

$\underline{\underline{\alpha}}$ is a diagonal matrix containing the dipole and quadrupole polarizabilities. $\underline{\mu}$ is a vector whose elements are the symmetry-unique components of all dipole and quadrupole moments. The **T**-tensor elements coupling the charge – or, in the more

general case, charges – and the various multipoles constitute the elements of vector $\underline{q}$. Finally, the matrix $\underline{\underline{T}}$ contains all the remaining **T**-tensor components coupling the multipoles with each other. Details on the construction of these vectors and matrices shall be outlined below as some transformations are still to be considered.

At this point, it is obvious that transformations leading to

$$\underline{\mu} = \left(\underline{\underline{\alpha}}^{-1} - \underline{\underline{T}}\right)^{-1} \underline{q} \tag{4.9}$$

will provide the multipole components directly. With these known, Yourshaw *et al.*[3] continue to compute the energy of the charge-multipole interaction as

$$V_{\text{ind,total}} = \sum_j \sum_{i \neq j} q_j \left(\frac{1}{2} T_{\alpha}^{(ij)} \mu_{\alpha}^{(i)} + \frac{1}{6} T_{\alpha\beta}^{(ij)} \Theta_{\alpha\beta}^{(i)} \right) \tag{4.10}$$

which is essentially equation (27) in Yourshaw *et al.*'s work[3]. Note the change of indices, which eventually leads to a change of sign in the dipole-related summand.

Introducing a new vector $\underline{\tilde{q}}$, which differs from the original $\underline{q}$ in the weights $1/2$ for components of dipole moments and $1/6$ for those of quadrupole moments ($1/3$ in the case of symmetry-equivalent quadrupole moments), equation Equation 4.10 can be translated into matrix notation:

$$V_{\text{ind,total}} = \underline{\tilde{q}}^{\dagger}\, \underline{\mu} = \underline{\tilde{q}}^{\dagger} \left(\underline{\underline{\alpha}}^{-1} - \underline{\underline{T}}\right)^{-1} \underline{q} \tag{4.11}$$

As only real values occur, the dagger "†" is used only to denote transpose matrices and row vectors. Thus, the computation of the inductive contribution to the cluster energy can be viewed as a scalar product, or – when looking more closely – as a simple vector-matrix-vector operation.

The notation introduced here allows for making another simplification. Splitting off the weights of the $\underline{\tilde{q}}$-vector in a separate diagonal matrix $\underline{\underline{W}}$

$$\underline{\tilde{q}}^{\dagger} = \underline{q}^{\dagger}\, \underline{\underline{W}} \tag{4.12}$$

leads, when inserted into eq. Equation 4.11, to

$$\begin{aligned} V_{\text{ind,total}} &= \underline{q}^{\dagger}\ \underline{\underline{W}}\ \left(\underline{\underline{\alpha}}^{-1} - \underline{\underline{T}}\right)^{-1}\ \underline{q} \\ &= \underline{q}^{\dagger}\ \left(\underline{\underline{\alpha}}^{-1}\ \underline{\underline{W}}^{-1} - \underline{\underline{T}}\ \underline{\underline{W}}^{-1}\right)^{-1}\ \underline{q} \\ &= -\frac{1}{2}\,\underline{q}^{\dagger}\ \underline{\underline{D}}^{-1}\ \underline{q} \end{aligned} \tag{4.13}$$

Since $\underline{\underline{W}}$ is diagonal, inversion of this matrix is straightforward. The effect of including $\underline{\underline{W}}$ in $\underline{\underline{D}}$ is to essentially lift all of the weighting factors in $\underline{\underline{T}}$ mentioned above along with the factors arising with symmetry-equivalent elements, e.g. $\Theta^{(i)}_{xy}$ and $\Theta^{(i)}_{yx}$. All of these are now solely vested in the diagonal elements of $\underline{\underline{D}}$, which arise from $\underline{\underline{\alpha}}^{-1}\ \underline{\underline{W}}^{-1}$. The coupling matrix $\underline{\underline{D}}$ itself is then real-symmetric.

At this point, the expressions for the individual elements of the $\underline{\underline{D}}$-matrix shall be given explicitly. The diagonal elements can directly be obtained as reciprocal polarizabilities:

$$D_{\xi\xi} = \begin{cases} \frac{1}{\alpha_i} & \text{for a dipole component} \\ \frac{3}{C_i} & \text{for a symmetry-unique quadrupole component (e.g. } xx) \\ \frac{3}{2C_i} & \text{for a symmetry-non-unique quadrupole component (e.g. } xy) \end{cases}$$

where ξ denotes a multipole component, i.e. either a cartesian dipole component $\mu^{(i)}_{\alpha}$ or a component of the quadrupole tensor of an atom $\Theta^{(i)}_{\alpha\beta}$. The off-diagonal elements are either zero if referring to the same nucleus or respective elements of the **T**-tensor where those bearing a column index of a dipole component carry a negative sign:

$$D_{\xi\xi'} = \begin{cases} 0 & \text{for multipole components at the same center} \\ -T^{(ij)}_{\alpha\beta} & \text{if } \xi \text{ and } \xi' \text{ are dipole components} \\ -T^{(ij)}_{\alpha\beta\gamma} & \text{if } \xi \text{ belongs to a quadrupole and } \xi' \text{ to a dipole} \\ T^{(ij)}_{\alpha\beta\gamma} & \text{if } \xi \text{ belongs to a dipole and } \xi' \text{ to a quadrupole} \\ T^{(ij)}_{\alpha\beta\gamma\delta} & \text{if } \xi \text{ and } \xi' \text{ are quadrupole components} \end{cases}$$

Here, nucleus i belongs to ξ and j to ξ'.

With eq. Equation 4.13 it is also clear how partial derivatives with respect to any given coordinate ν can be obtained analytically:

$$\partial_\nu V_{\text{ind,total}} = -\underline{\tilde{\mu}}^\dagger \cdot \left(\partial_\nu \underline{q}\right) - \frac{1}{2}\underline{\tilde{\mu}}^\dagger \left(\partial_\nu \underline{\underline{D}}\right) \underline{\tilde{\mu}} \tag{4.14}$$

where $\underline{\tilde{\mu}} = \underline{\underline{D}}^{-1}\,\underline{q} = \left(\underline{q}\,\underline{\underline{D}}^{-1}\right)^\dagger$ was used, the "effective multipole moment vector". It is, of course, advantageous to store this latter vector in memory after energy calculations for computations of the gradient.

Finally performance aspects need to be discussed. In the original development of the underlying model Vesely pointed out the option of an algebraic treatment and explicitly mentioned the high symmetry of the matrix concerned.[181] Due to the large size of the matrix in the case at hand, the problem seemed intractable at the time, however. Presently and particularly for the problem of noble gas–halide clusters, both pathways are certainly well feasible.

In terms of theoretical scaling limits, both methods can be regarded – in worst case scenarios – as scaling as $\mathcal{O}(n^3)$. For the algebraic method, this result holds for the least optimized algorithms[182], while in the case of the iterative method it is based on the assumption that the number of iterations scales proportionally with the number of nuclei. The latter is certainly not the case if a suitable updating scheme is implemented. Regarding the former, however, it is known[183] that algorithms with better scaling properties exist. For this particular problem, it seems to be generally possible to assume that $\underline{\underline{D}}$ is positive-definite and that Cholesky-type methods may be used for better performance.[166,184] Only at unphysically short interatomic distances do non-positive-definite matrices occur.

For systems of rather limited size, such as the halide–noble gas clusters, overhead considerations are probably of greater importance than asymptotic scaling behavior. In this respect, the iterative scheme is inherently more prone to producing overhead, due to the up to multi-dimensional **T**-tensor elements. Recalculation at every step of the iteration will certainly be the worst alternative, but storage of and access to up to four- or five-dimensional arrays is also more costly than the two-dimensional matrices required for the algebraic solution.

4.2 Comparison of model and experimental results

For the systems investigated in [9], model calculations were performed, the results of which are shown in Figure 4.3. The difference in solvation energy ΔSE was calculated as

$$\Delta SE(n) = VDE(\mathrm{Ng}_n\mathrm{X}^-) - VDE(\mathrm{X}^-) \tag{4.15}$$

from the experimental values of VDE, the vertical detachment energy, i.e. the energy required to remove an electron from the anion. Thus, the difference in solvation energies can be interpreted as the increase of the energy needed for the removal of an electron from the anion caused by the presence of the noble gas atoms in the cluster. The theoretical counterpart is

$$\Delta SE(n) = V^{\mathrm{min}}_{\mathrm{neutral}} - V^{\mathrm{min}}_{\mathrm{anion}} + \sum_i \frac{1}{2}\left(\omega_i^{\mathrm{neutral}} - \omega_i^{\mathrm{anion}}\right) \tag{4.16}$$

where V^{min} is the depth of the potential energy well at (for the anionic cluster) or closest to (for the neutral cluster) the global energy minimum for the anionic cluster, and the sum represents the difference in zero point energies for the anion and the neutral cluster. Thus, $\Delta SE(n)$ with the X state as a reference corresponds to $\mathrm{EA}^X(\mathrm{Ng}_n\mathrm{X})$ in Figure 4.1.

The comparison of theory and experiment rests on the assumption that the electron detachment takes place mostly to the vibrational ground state of the neutral cluster. In other words, it is assumed that a minimum in the potential energy function of the neutral cluster exists for which the geometry of the cluster does not differ greatly from the anionic global minimum, such that the Franck–Condon factor for the 0-0 vibrational transition is the dominant one. This may well be a local minimum of the potential energy function of the neutral state. Additionally, the experiment was usually not able to resolve X and I states. Noticing that obvious deviations are substantially greater than what would be expected to be introduced by either of these simplifications, however, they appear acceptable.

It should be noted that some changes in the parameters used have been made, resulting in values differing somewhat from those presented in [9]. The potentials and parameters used here are listed in Appendix C. All model results exhibit shortcomings when compared to the experimental values, which shall be analyzed in this section.

Beginning with the fluorine systems, it is obvious that only for the case of $\mathrm{Ar}_n\mathrm{F}$

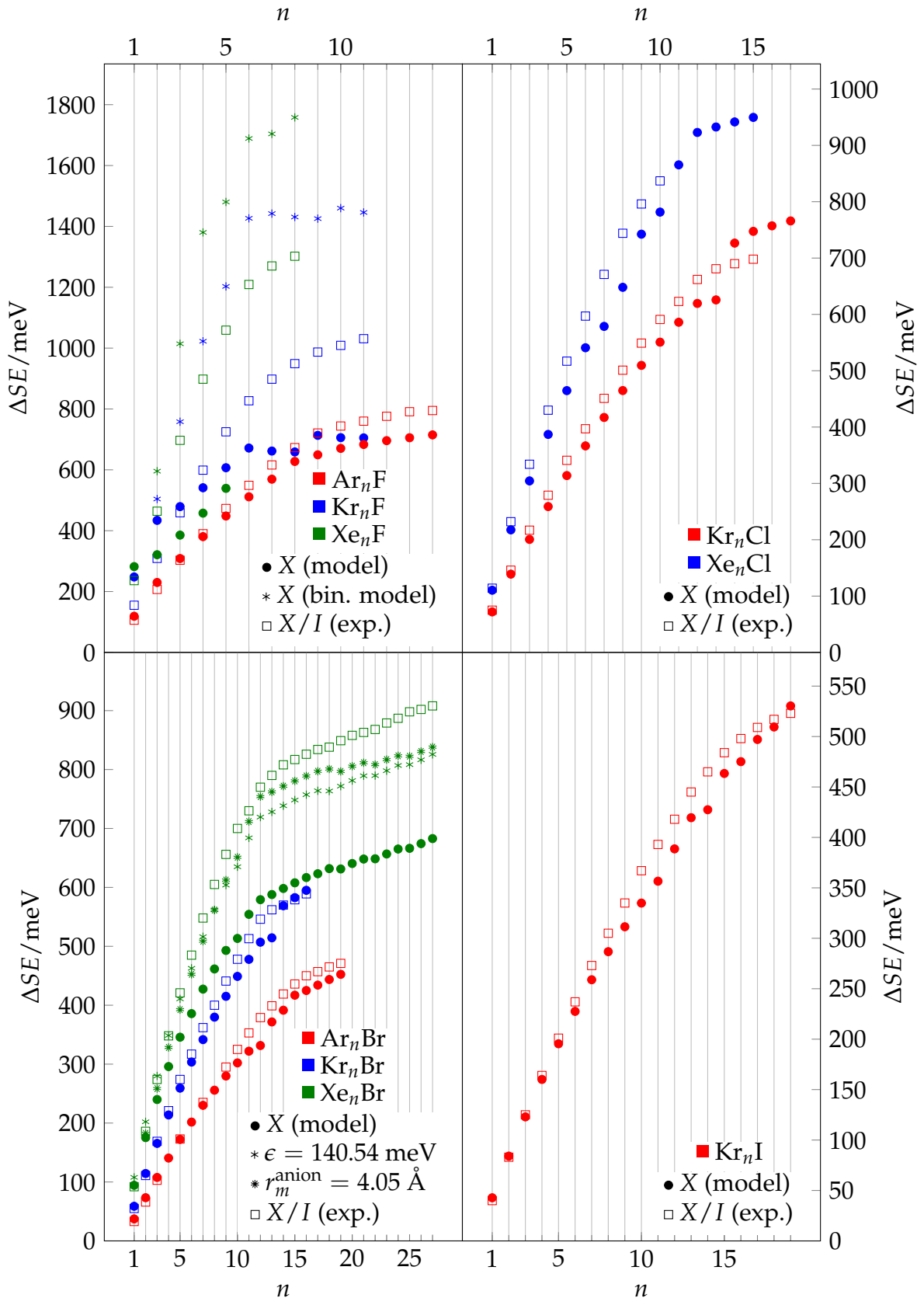

n
ΔSE/meV
Ar_nF
Kr_nF
Xe_nF
X (model)
X (bin. model)
X/I (exp.)
Kr_nCl
Xe_nCl
X (model)
X/I (exp.)
Ar_nBr
Kr_nBr
Xe_nBr
X (model)
ϵ = 140.54 meV
r_m^anion = 4.05 Å
X/I (exp.)
Kr_nI
X (model)
X/I (exp.)

Figure 4.3: (facing page) Comparison of experimental and model results for the difference in solvation energy ΔSE for various halide–noble gas systems. "Bin. model" refers to calculations without many-body effects. $\epsilon = 140.54$ meV and $r_m^{\text{anion}} = 4.05$ Å refer to changes in the Xe–Br^- binary potential parameters (see text).

could a qualitative agreement be achieved. Of course, xenon and krypton are well-known to form compounds with fluorine, while compounds for argon with fluorine and hydrogen have been found only fairly recently[185], a fact that underscores the strength of the interaction between fluorine and these noble gases. The very short equilibrium distances of the corresponding anionic potentials lead to an exaggeration of the largest repulsive contribution, induction nonadditivity. As already pointed out elsewhere[3], an uncertainty in equilibrium distance in the binary potentials translates into an uncertainty in energy from all non-additive potential contributions.

Additionally, an uncertainty in the shape of the more distant part of the potential certainly has a detrimental effect on the accuracy of the energy from the second solvation shell onward. For instance, the quadrupole contribution to the Xe–F^- binary potential (MMSV or Morse-Morse-Switching-van der Waals-type) has obviously not been determined appropriately, while the quadrupole contributions in the Ar–F^- and Kr–F^- potentials appear exaggerated. In the case of the latter two, the outermost noble gas atoms (see Figure 4.5) are close to or in the range of the multipole part of the MMSV potential (i.e. $r_{\text{Ng}-\text{X}} > x_2 \cdot r_m$) and indeed ΔSE for Kr_nF in the binary potential (Figure 4.3) level off much too strongly upon completion of the first shell of noble gas atoms ($n > 6$). However, it is obvious that strong deviations from the experimental values start much earlier: Even the values for clusters with only a single noble gas atom do not agree well with the experiment for XeF, and even more so for KrF, which is clearly due to a wrong well depth of the binary MMSV potential.

Figure 4.4 depicts the ratio of induction nonadditivity to the sum of all binary interactions between the halide anion and the surrounding noble gas atoms. The effect of the former starts out to be small for every combination of halide and noble gas and rises, to then level off as the first shell of noble gas atoms around the halide closes. With respect to the total energy, this corresponds to a decrease in relative contribution of induction nonadditivity, as the contribution of the noble gas–noble gas interactions becomes dominant. Generally, the value for a cluster with only two noble gas atoms is below 0.1, except for Kr_2F and Xe_2F, where it is roughly 0.15 and 0.22, respectively. Likewise, the upper limit for this ratio is generally about 0.25 to 0.3, while it is between 0.45 and 0.5 for Kr_nF and Xe_nF. The upper limit of Xe_nBr of 0.37 appears to be shared

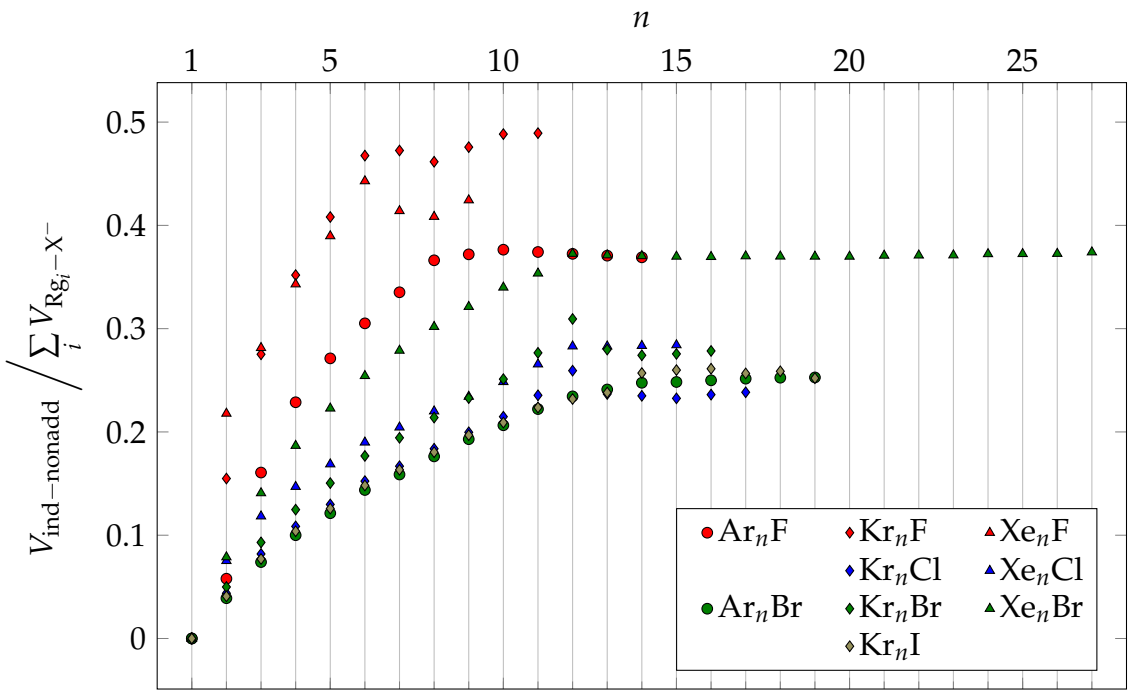


Figure 4.4: Ratio of induction nonadditivity to the sum of binary potential contributions between the halide anion and all noble gas atoms as a function of cluster size.

by Ar_nF and seems to be still within the validity of the assumptions for induction nonadditivity, given that $\Delta SE(n)$ does not become erratic as in the case of Xe_nF and Kr_nF after closure of the first solvent shell. A substantial contribution to the deviation from experimental values can thus certainly be attributed to overestimated induction nonadditivity in Xe_nF and Kr_nF.

Another large deviation was found for Xe_nBr-type clusters. Note that in the presentation of the experimental data given in [9], these were in rather good agreement with the experimental values due to a change of the anionic potential well depth from 126.92 to 140.54 meV, as shown in Figure 4.3. Obviously, however, this is not a good correction as the binary potential should agree with the experimental value of XeBr (i.e. the case of a single interaction partner) and increasing the binding energy of the binary potential necessarily leads to worse agreement for all small clusters.

An increase in the anionic equilibrium distance, on the other hand, can have a similar effect in correcting the deviations from the experimental values, as also shown in Figure 4.3, by decreasing the mostly repulsive contributions of the non-additive potential contributions. Changes in this parameter leave the energy of the smallest cluster unaffected, but the slope of ΔSE for clusters with a closed first solvent shell obviously decreases too strongly. Ultimately, the given experimental data is not suffi-

cient to determine which potential parameters require improvement and to what extent, especially when considering that the binary potentials of the neutral species offer yet more parameters. It may, however, be assumed that parameters pertaining to the non-additive contributions to the potential energy, e.g. dipole and quadrupole polarizabilities, are largely correct as this large deviation is not observable in other clusters containing either xenon or bromine. Similar problems appear to exist for the Ar_nF species, although to a much lesser degree.

For the remaining clusters, the greatest difficulty for the model appears to be to describe the closing of the first shell of noble gas atoms correctly. In Kr_nI, Ar_nBr, Kr_nBr, and Kr_nCl, the ΔSE value of the cluster n =14, 12, 13, and 13, respectively, exhibit an irregularly small increase in ΔSE. From the halide–noble gas distances shown in Figure 4.5, it can be inferred that these are due to some rearrangement within the not yet filled first solvation shell. For Kr_nBr and Kr_nCl, distances from the halide core have a larger spread in these instances, whereas in Kr_nI and Ar_nBr the spread in noble gas–halide distance is smaller in these cases than in the next larger and smaller clusters. The most straightforward explanation seems to be that the equilibrium distances of the corresponding halide–noble gas potentials are insufficiently accurate.

Additionally, in many cases (especially Kr_nI, Ar_nBr, Kr_nBr, Kr_nCl) the slope of ΔSE does not decrease after the completion of the first solvation shell as strongly as is the case for the experimental data. This indicates deficiencies in the long range part of the potential: most likely an overly attractive anionic potential, because it is the greatest contribution to impact on an increase of ΔSE. This is true, although the binary noble gas interactions are the largest component of the total energy of the anion. For the computation of ΔSE these are indeed practically completely eliminated as they also exist, with almost identical magnitude, in the neutral clusters (see Figure 4.6). In other words: their significance extends mostly to the determination of the cluster geometry.

Geometries are of interest in particular with regard to the size of the first solvation shell. From the plot of distances in the various clusters (Figure 4.5), it is apparent that for Kr_nCl, Kr_nBr, Xe_nCl, and Xe_nBr, a tendency to form icosahedral structures is prevalent. In two systems, namely Xe_nCl and Xe_nBr, the icosahedron constitutes the inner shell of noble gas atoms as further atoms are added to the second shell. For Xe_nBr even the start of a third shell can be seen. After it is energetically favorable for the first few noble gas atoms of the second shell to be located over the triangular faces of the icosahedron of the first shell, eventually forming a pentagon, it is then apparently more favorable to add the next xenon atom centered over that pentagon. It is thus also

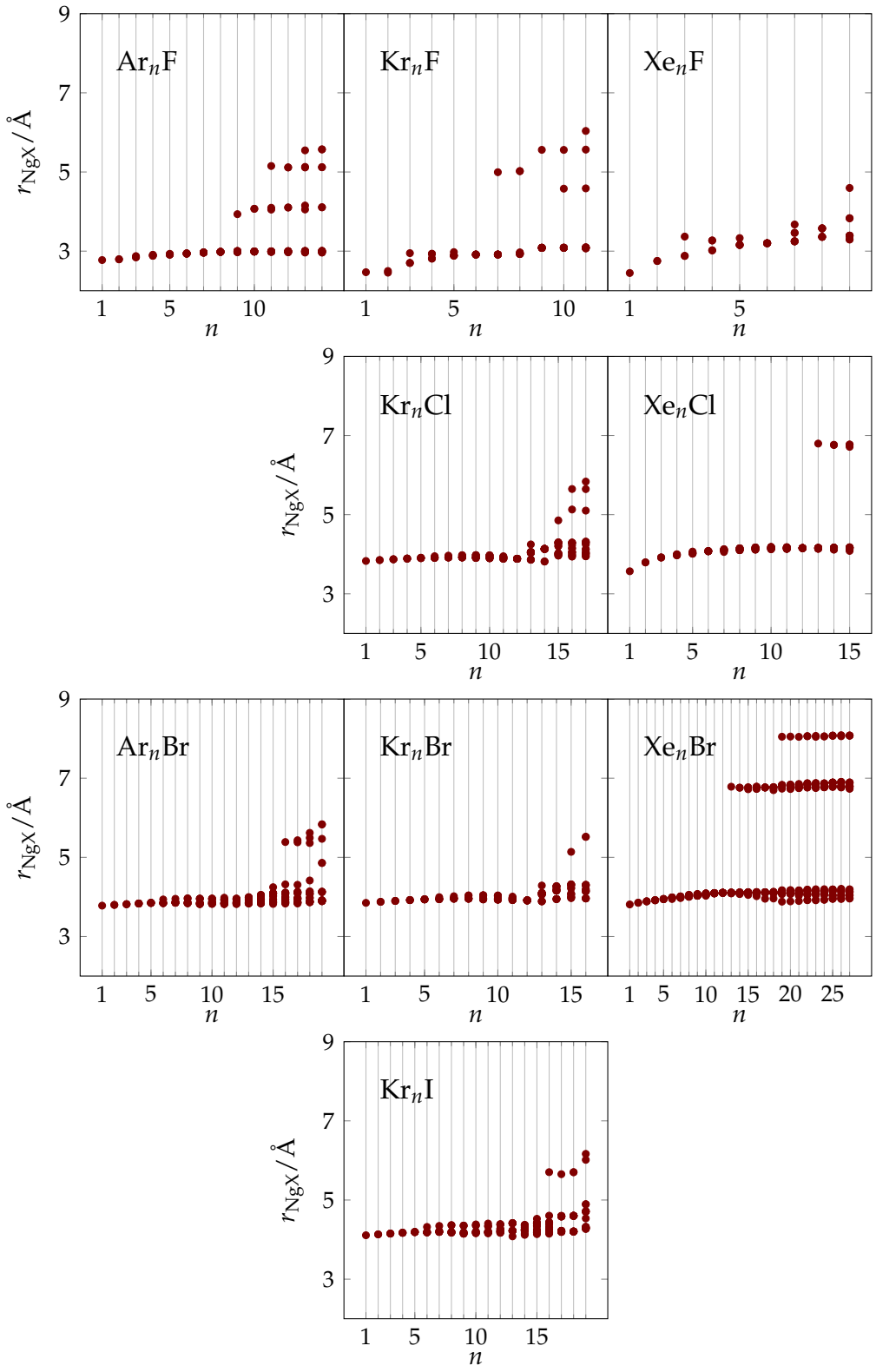


Figure 4.5: Halide–noble gas distances obtained from the model calculations for the respective anionic clusters.

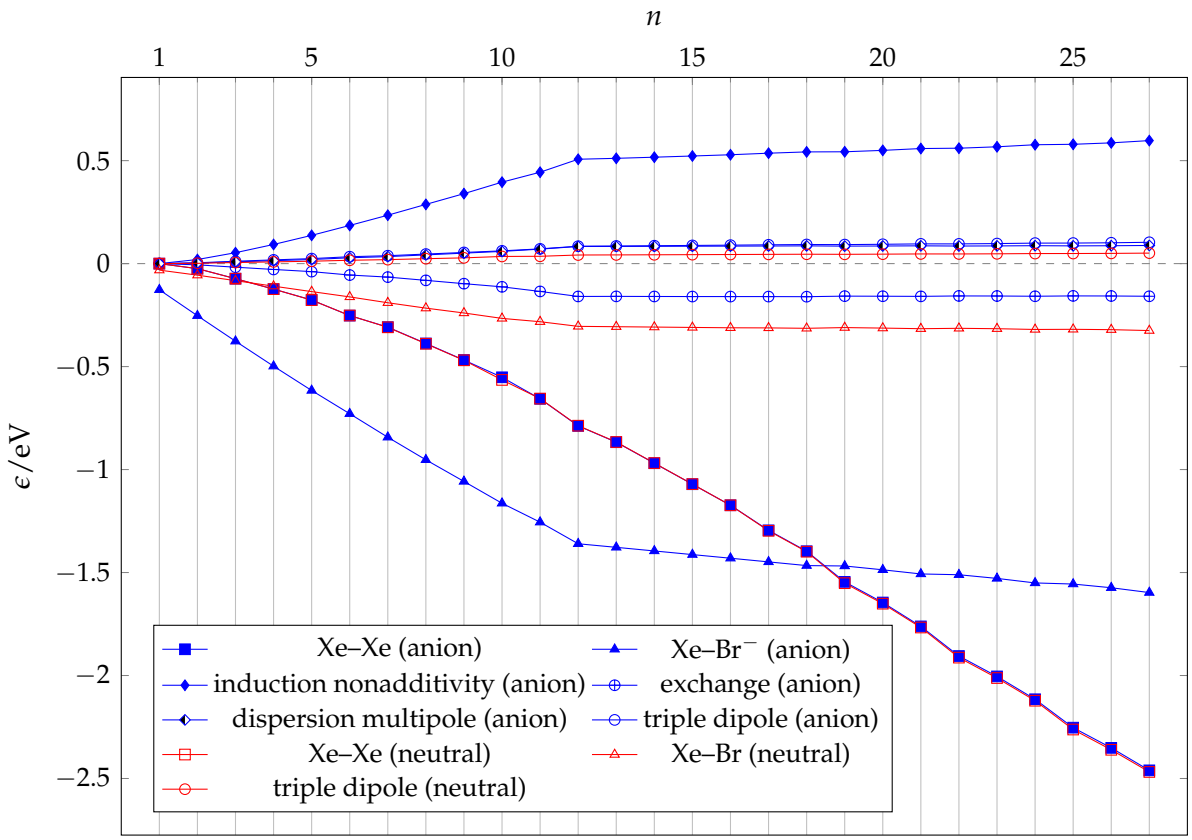


Figure 4.6: Contributions to the total cluster solvation energy (SE) for the Xe_nBr system (data for the neutral system is for the X state; the unmodified potential was used for the anion).

on top of one of the tips of the icosahedron. This is a manifestation of the growing importance of xenon-xenon interactions versus xenon-bromide interactions as a result of increasing shielding of the attraction of the anion.

For Kr_nCl and Kr_nBr, the preference of an icosahedral geometry is noticeable from a substantially smaller spread in halide–noble gas distances, as compared to both smaller as well as larger clusters. In both cases, the icosahedron is subsequently distorted to enable the first solvation shell to accommodate two more noble gas atoms and eventually less symmetrical polyhedrons form the first solvation shell. Contrary to what was assessed earlier[9] about structures of clusters with an incomplete first solvent shell in these systems, they do not resemble a capped square antiprism for $n = 10$, but rather *arachno*[186]-type icosahedrons (i.e. icosahedrons with two apex atoms missing).

In the cases of Ar_nBr and Kr_nI, shell closure occurs only with the 15th added noble gas atom, as the smaller ratio of van-der-Waals radii of noble gas and halide would

lead to expect.

Marking the opposite end of the scale, fluorine clusters prefer much smaller solvation shells, with Ar_8F being a square antiprism, of which subsequently ($n = 9$ and 10), caps are added to the squares and then the to the triangular faces ($n \geqslant 11$). Kr_6F and Xe_6F are both octahedral, but in both cases the addition of further noble gas atoms leads to distortion of the octahedron. While for Kr_nF ($n > 6$) subsequently added atoms are substantially farther from the halide, Xe_8F is of square antiprismatic shape, although this is again distorted greatly in Xe_9F.

This entire analysis of geometries, it should be noted, depends on the somewhat questionable (vide supra) accuracy of the equilibrium distances of the interaction potentials.

4.3 Conclusion

Improvements over the original work of Yourshaw *et al.*[3] have been achieved by finding a concise analytic solution of the Lawrence and Apkarian-Hamiltonian[164] and the adoption of a hypertensor[177]-like treatment of induction nonadditivity. The latter proved helpful for finding analytic derivatives of this energy contribution and thus avoiding the more costly numerical derivation.

While interesting insights into cluster geometries could be gleaned from model calculations, weaknesses of the underlying binary potentials were discovered by comparing the model results to experimental data. It seems that particularly the equilibrium distances of these potentials require greater accuracy, since many-body potential contributions are very sensitive to them. At the same time, the model may have general difficulties in describing larger clusters with a closed first solvent shell.

Bibliography

[1] Schade, M.; Moretto, A.; Crisma, M.; Toniolo, C. and Hamm, P.: Vibrational Energy Transport in Peptide Helices after Excitation of C–D Modes in Leu-d_{10}. *The Journal of Physical Chemistry B*, **2009**, *113(40)*, 13393–13397. DOI: 10.1021/jp906363a

[2] Hamm, P.; Ohline, S. M. and Zinth, W.: Vibrational cooling after ultrafast photoisomerization of azobenzene measured by femtosecond infrared spectroscopy. *The Journal of Chemical Physics*, **1997**, *106(2)*, 519–529. DOI: 10.1063/1.473392

[3] Yourshaw, I.; Zhao, Y. and Neumark, D. M.: Many-body effects in weakly bound anion and neutral clusters: Zero electron kinetic energy spectroscopy and threshold photodetachment spectroscopy of Ar_nBr^- (n=2–9) and Ar_nI^- (n=2–19). *The Journal of Chemical Physics*, **1996**, *105(2)*, 351–373. DOI: 10.1063/1.471893

[4] Hirata, Y. and Mataga, N.: Picosecond Dye-Laser Photolysis Study of Diphenylcyclopropenone in Solution - Formation of the Electronically Excited States of Diphenylacetylene. *Chemical Physics Letters*, **1992**, *193(4)*, 287–291. DOI: 10.1016/0009-2614(92)85669-2

[5] Nguyen, L. T.; De Proft, F.; Nguyen, M. T. and Geerlings, P.: Theoretical study of cyclopropenones and cyclopropenethiones: decomposition *via* intermediates. *Journal of the Chemical Society, Perkin Transactions 2*, **2001**, 6, 898–905. DOI: 10.1039/B100709M

[6] Vennekate, H.; Walter, A.; Fischer, D.; Schroeder, J. and Schwarzer, D.: Photodecarbonylation of Diphenylcyclopropenone — a Direct Pathway to Electronically Excited Diphenylacetylene? *Zeitschrift für Physikalische Chemie*, **2011**, *225(9-10)*, 1089–1104. DOI: 10.1524/zpch.2011.0164

[7] Schwarzer, D.; Kutne, P.; Schroder, C. and Troe, J.: Intramolecular vibrational energy redistribution in bridged azulene-anthracene compounds: Ballistic energy

transport through molecular chains. *The Journal of Chemical Physics*, **2004**, *121(4)*, 1754–1764. DOI: 10.1063/1.1765092

[8] Lin, Z.; Zhang, N.; Jayawickramarajah, J. and Rubtsov, I. V.: Ballistic energy transport along PEG chains: distance dependence of the transport efficiency. *Physical Chemistry Chemical Physics*, **2012**, 14, 10445–10454. DOI: 10.1039/C2CP40187H

[9] Kopczynski, M.: *Femtosekunden Photodetachment- Photoelektronenspektroskopie an isolierten und massenselektierten Halogen-Edelgas-Clustern*. PhD thesis, Georg-August-University, **2010**.

[10] Lenzer, T.; Furlanetto, M. R.; Pivonka, N. L. and Neumark, D. M.: Zero electron kinetic energy and threshold photodetachment spectroscopy of Xe_nI^- clusters (n=2–14): Binding, many-body effects, and structures. *The Journal of Chemical Physics*, **1999**, *110(14)*, 6714–6731. DOI: 10.1063/1.478577

[11] Lenzer, T.; Yourshaw, I.; Furlanetto, M. R.; Pivonka, N. L. and Neumark, D. M.: Characterization of Ar_nCl^- clusters (n = 2–15) using zero electron kinetic energy and partially discriminated threshold photodetachment spectroscopy. *The Journal of Chemical Physics*, **2001**, *115(8)*, 3578–3589. DOI: 10.1063/1.1388202

[12] Hamm, P.: Femtosecond IR Pump-Probe Spectroscopy of Nonlinear Energy Localization in Protein Models and Model Proteins. *Journal of Biological Physics*, **2009**, 35, 17–30. DOI: 10.1007/s10867-009-9126-3

[13] Crim, F. F.: Bond-Selected Chemistry: Vibrational State Control of Photodissociation and Bimolecular Reaction. *The Journal of Physical Chemistry*, **1996**, *100(31)*, 12725–12734. DOI: 10.1021/jp9604812

[14] Poloukhtine, A. and Popik, V.: Mechanism of the cyclopropenone decarbonylation reaction. A density functional theory and transient spectroscopy study. *The Journal of Physical Chemistry A*, **2006**, *110(5)*, 1749–1757. DOI: 10.1021/jp0563641

[15] Hamm, P.: Coherent effects in femtosecond infrared spectroscopy. *Chemical Physics*, **1995**, *200(3)*, 415–429. DOI: 10.1016/0301-0104(95)00262-6

[16] Reichardt, C.: *Aufklärung des photoinduzierten Zerfallsmechanismus von aromatischen Peroxoestern mittels Femtosekunden-IR-Spektroskopie*. PhD thesis, Georg-August-Universität Göttingen, **2008**.

[17] Vennekate, H.: *Untersuchung des photoinduzierten Zerfalls von Acetylbenzoyl- und Phthaloylperoxid mittels Femtosekunden-Infrarotspektroskopie*. Diplomarbeit, Georg-August-Universität Göttingen, **2008**.

[18] Schäfer, T.: *Untersuchung der Schwingungsenergierelaxation und Rotationsdynamik von assoziierten Flüssigkeiten über weite Dichte- und Temperaturbereiche*. PhD thesis, Georg-August-Universität Göttingen, **2009**.

[19] Fischer, D.: *Untersuchung der Dissoziation von Azido(cyclam-acetato)-Eisen(III)- hexafluorophosphat nach UV-Anregung mit Infrarot-Femtosekundenspektroskopie*. Bachelor's thesis, Georg-August-Universität Göttingen, **2011**.

[20] Auth, T.: *Untersuchungen zur Orientierungsrelaxation von Azido(cyclam-acetato)-Eisen(III)- hexafluorophosphat in Acetonitril*. Bachelor's thesis, Georg-August-Universität Göttingen, **2012**.

[21] Hamm, P.; Kaindl, R. A. and Stenger, J.: Noise suppression in femtosecond mid-infrared light sources. *Optics Letters*, **2000**, *25(24)*, 1798–1800. DOI: 10.1364/OL.25.001798

[22] Kaindl, R. A.; Wurm, M.; Reimann, K.; Hamm, P.; Weiner, A. M. and Woerner, M.: Generation, shaping, and characterization of intense femtosecond pulses tunable from 3 to 20 μm. *Journal of the Optical Society of America B*, **2000**, *17(12)*, 2086–2094. DOI: 10.1364/JOSAB.17.002086

[23] National Instruments Corporation: LabView 8.2.1, **2007**.

[24] Hamm, P.; Lim, M. and Hochstrasser, R. M.: Structure of the Amide I Band of Peptides Measured by Femtosecond Nonlinear-Infrared Spectroscopy. *The Journal of Physical Chemistry B*, **1998**, *102(31)*, 6123–6138. DOI: 10.1021/jp9813286

[25] Upitis, J. A. and Krol, A.: The Use of Diphenylcyclopropenone in the Treatment of Recalcitrant Warts. *Journal of Cutaneous Medicine and Surgery*, **2002**, *6(3)*, 214–217. DOI: 10.1007/s10227-001-0050-9

[26] Sotiriadis, D.; Patsatsi, A.; Lazaridou, E.; Kastanis, A.; Vakirlis, E. and Chrysomallis, F.: Topical immunotherapy with diphenylcyclopropenone in the treatment of chronic extensive alopecia areata. *Clinical and Experimental Dermatology*, **2007**, *32(1)*, 48–51. DOI: 10.1111/j.1365-2230.2006.02256.x

[27] Grabowski, J. J.; Simon, J. D. and Peters, K. S.: Heat of formation of diphenylcyclopropenone by photoacoustic calorimetry. *Journal of the American Chemical Society*, **1984**, *106(16)*, 4615–4616. DOI: 10.1021/ja00328a052

[28] Fessenden, R. W.; Carton, P. M.; Shimamori, H. and Scaiano, J. C.: Measurement of the dipole moments of excited states and photochemical transients by microwave dielectric absorption. *The Journal of Physical Chemistry*, **1982**, *86(19)*, 3803–3811. DOI: 10.1021/j100216a020

[29] Potts, K. T. and Baum, J. S.: Chemistry of cyclopropenones. *Chemical Reviews*, **1974**, *74(2)*, 189–213. DOI: 10.1021/cr60288a003

[30] Kandratsenka, A.; Schroeder, J.; Schwarzer, D. and Vikhrenko, V. S.: Molecular dynamics modeling of cooling of vibrationally highly excited carbon dioxide produced in the photodissociation of organic peroxides in solution. *Physical Chemistry Chemical Physics*, **2005**, *7(6)*, 1205–1213. DOI: 10.1039/b414623a

[31] Kim, K.: The integrated intensity of the carbon monoxide fundamental band. *Journal of Quantitative Spectroscopy and Radiative Transfer*, **1983**, *30(5)*, 413–416. DOI: 10.1016/0022-4073(83)90104-8

[32] Ewing, G. E.: Infrared Spectra of Liquid and Solid Carbon Monoxide. *The Journal of Chemical Physics*, **1962**, *37(10)*, 2250–2256. DOI: 10.1063/1.1732994

[33] Fayer, D.: *Ultrafast Infrared And Raman Spectroscopy*. Practical spectroscopy: Taylor & Francis, **2001**. ISBN: 978-0-8247-0451-3

[34] Hirata, Y.; Okada, T.; Mataga, N. and Nomoto, T.: Picosecond Time-Resolved Absorption-Spectrum Measurements of the Higher Excited Singlet-State of Diphenylacetylene in the Solution Phase. *The Journal of Physical Chemistry*, **1992**, *96(16)*, 6559–6563. DOI: 10.1021/j100195a011

[35] Takeuchi, S. and Tahara, T.: Femtosecond absorption study of photodissociation of diphenylcyclopropenone in solution: Reaction dynamics and coherent nuclear motion. *The Journal of Chemical Physics*, **2004**, *120(10)*, 4768–4776. DOI: 10.1063/1.1645778

[36] Ferrante, C.; Kensy, U. and Dick, B.: Does diphenylacetylene (tolan) fluoresce from its second excited singlet state? Semiempirical MO calculations and fluorescence quantum yield measurements. *The Journal of Physical Chemistry*, **1993**, *97(51)*, 13457–13463. DOI: 10.1021/j100153a008

[37] Amatatsu, Y. and Hosokawa, M.: Theoretical Study on the Photochemical Behavior of Diphenylacetylene in the Low-Lying Excited States. *The Journal of Physical Chemistry A*, **2004**, *108(46)*, 10238–10244. DOI: 10.1021/jp047308n

[38] Kellerer, B.; Hacker, H. H. and Brandmüller, J.: On the Structure of Azobenzene & Tolane in Solution: Raman spectra of Azobenzene, Azobenzene-d_{10}, *p*-,*p'*-Azobenzene-d_2, Azobenzene-^{15}N=^{15}N & Tolane. *Indian Journal of Pure and Applied Physics*, **1971**, 9, 903.

[39] Baranović, G.; Colombo, L. and Skare, D.: A valence force field for phenylalkynes: Part II. Fundamental frequencies of phenylacetylene and tolane and molecular conformation of tolane in solution. *Journal of Molecular Structure*, **1986**, *147(3–4)*, 275–300. DOI: 10.1016/0022-2860(86)80382-9

[40] Ishibashi, T. and Hamaguchi, H.: The central CC bond stretch frequencies and structure of S_2 and S_1 diphenylacetylene in solution studied by picosecond time-resolved CARS spectroscopy. *Chemical Physics Letters*, **1997**, *264(5)*, 551–555. DOI: 10.1016/S0009-2614(96)01368-1

[41] Ishibashi, T. and Hamaguchi, H.: Structure and Dynamics of S_2 and S_1 Diphenylacetylene in Solution Studied by Picosecond Time-Resolved CARS Spectroscopy. *The Journal of Physical Chemistry A*, **1998**, *102(13)*, 2263–2269. DOI: 10.1021/jp972809c

[42] Ishibashi, T.; Okamoto, H. and Hamaguchi, H.: Picosecond transient infrared spectra and structure of S_1 diphenylacetylene in solution. *Chemical Physics Letters*, **2000**, *325(1–3)*, 212–218. DOI: 10.1016/S0009-2614(00)00631-X

[43] Okuyama, K.; Hasegawa, T.; Ito, M. and Mikami, N.: Electronic spectra of tolane in a supersonic free jet: large-amplitude torsional motion. *The Journal of Physical Chemistry*, **1984**, *88(9)*, 1711–1716. DOI: 10.1021/j150653a010

[44] Tanizaki, Y.; Inoue, H.; Hoshi, T. and Shiraishi, J.: Localized and Delocalized Electronic Transitions in Diphenylacetylene, Stilbene and Diphenylbutadiene. *Zeitschrift für Physikalische Chemie*, **1971**, *74(1–2)*, 45–58. DOI: 10.1524/zpch.1971.74.1_2.045

[45] Yoshida, Z. and Miyahara, H.: Electronic Spectra and Structures of 3-Substituted 1,2-Diphenylcyclopropenyl Cations. *Bulletin of the Chemical Society of Japan*, **1972**, *45(6)*, 1919–1921. DOI: 10.1246/bcsj.45.1919

[46] Wilson, E. B.: The Normal Modes and Frequencies of Vibration of the Regular Plane Hexagon Model of the Benzene Molecule. *Physical Review*, **1934**, 45, 706–714. DOI: 10.1103/PhysRev.45.706

[47] Shimojima, A. and Takahashi, H.: Ab initio study of geometries and force fields of diphenylacetylene in the ground state, radical cation, radical anion, and lowest excited triplet state. *The Journal of Physical Chemistry*, **1993**, *97(36)*, 9103–9112. DOI: 10.1021/j100138a007

[48] Hiura, H. and Takahashi, H.: Time-resolved resonance Raman spectroscopy of diphenylacetylene: structures and dynamics of the lowest excited triplet state, radical cation, and radical anion. *The Journal of Physical Chemistry*, **1992**, *96(22)*, 8909–8915. DOI: 10.1021/j100201a040

[49] Schmidt, M. W.; Baldridge, K. K.; Boatz, J. A.; Elbert, S. T.; Gordon, M. S.; Jensen, J. H.; Koseki, S.; Matsunaga, N.; Nguyen, K. A.; Su, S.; Windus, T. L.; Dupuis, M. and Montgomery, J. A.: General atomic and molecular electronic structure system. *Journal of Computational Chemistry*, **1993**, *14(11)*, 1347–1363. DOI: 10.1002/jcc.540141112

[50] Stephens, P. J.; Devlin, F. J.; Chabalowski, C. F. and Frisch, M. J.: Ab Initio Calculation of Vibrational Absorption and Circular Dichroism Spectra Using Density Functional Force Fields. *The Journal of Physical Chemistry*, **1994**, *98(45)*, 11623–11627. DOI: 10.1021/j100096a001

[51] Krishnan, R.; Binkley, J. S.; Seeger, R. and Pople, J. A.: Self–consistent molecular orbital methods. XX. A basis set for correlated wave functions. *The Journal of Chemical Physics*, **1980**, *72(1)*, 650–654. DOI: 10.1063/1.438955

[52] Kuzmanich, G.; Natarajan, A.; Chin, K.; Veerman, M.; Mortko, C. and Garcia-Garibay, M.: Solid-state photodecarbonylation of diphenylcyclopropenone: A quantum chain process made possible by ultrafast energy transfer. *Journal of the American Chemical Society*, **2008**, *130(4)*, 1140–1141. DOI: 10.1021/ja078301x

[53] Kuzmanich, G.; Gard, M. N. and Garcia-Garibay, M. A.: Photonic Amplification by a Singlet-State Quantum Chain Reaction in the Photodecarbonylation of Crystalline Diarylcyclopropenones. *Journal of the American Chemical Society*, **2009**, *131(32)*, 11606–11614. DOI: 10.1021/ja9043449

[54] Doan, S. C.; Kuzmanich, G.; Gard, M. N.; Garcia-Garibay, M. A. and Schwartz, B. J.: Ultrafast Spectroscopic Observation of a Quantum Chain Reaction: The

Photodecarbonylation of Nanocrystalline Diphenylcyclopropenone. *The Journal of Physical Chemistry Letters*, **2011**, *3(1)*, 81–86. DOI: 10.1021/jz201360w

[55] Nesbitt, D. J. and Field, R. W.: Vibrational Energy Flow in Highly Excited Molecules: Role of Intramolecular Vibrational Redistribution. *The Journal of Physical Chemistry*, **1996**, *100(31)*, 12735–12756. DOI: 10.1021/jp960698w

[56] Carter, J. A.; Wang, Z.; Fujiwara, H. and Dlott, D. D.: Ultrafast Excitation of Molecular Adsorbates on Flash-Heated Gold Surfaces. *The Journal of Physical Chemistry A*, **2009**, *113(44)*, 12105–12114. DOI: 10.1021/jp906082u

[57] Segal, D.; Nitzan, A. and Hänggi, P.: Thermal conductance through molecular wires. *The Journal of Chemical Physics*, **2003**, *119(13)*, 6840–6855. DOI: 10.1063/1.1603211

[58] Nitzan, A.: Molecules Take the Heat. *Science*, **2007**, *317(5839)*, 759–760. DOI: 10.1126/science.1147011

[59] Manikandan, P.; Carter, J. A.; Dlott, D. D. and Hase, W. L.: Effect of Carbon Chain Length on the Dynamics of Heat Transfer at a Gold/Hydrocarbon Interface: Comparison of Simulation with Experiment. *The Journal of Physical Chemistry C*, **2011**, *115(19)*, 9622–9628. DOI: 10.1021/jp200672e

[60] Backus, E. H. G.; Bloem, R.; Pfister, R.; Moretto, A.; Crisma, M.; Toniolo, C. and Hamm, P.: Dynamical Transition in a Small Helical Peptide and Its Implication for Vibrational Energy Transport. *The Journal of Physical Chemistry B*, **2009**, *113(40)*, 13405–13409. DOI: 10.1021/jp904905d

[61] Schade, M.; Moretto, A.; Donaldson, P. M.; Toniolo, C. and Hamm, P.: Vibrational Energy Transport through a Capping Layer of Appropriately Designed Peptide Helices over Gold Nanoparticles. *Nano Letters*, **2010**, *10(8)*, 3057–3061. DOI: 10.1021/nl101580w

[62] Wang, Z.; Cahill, D. G.; Carter, J. A.; Koh, Y. K.; Lagutchev, A.; Seong, N.-H. and Dlott, D. D.: Ultrafast dynamics of heat flow across molecules. *Chemical Physics*, **2008**, *350(1–3)*, 31–44. DOI: 10.1016/j.chemphys.2007.12.017

[63] Lin, Z. and Rubtsov, I. V.: Constant-speed vibrational signaling along polyethyleneglycol chain up to 60-Å distance. *Proceedings of the National Academy of Sciences*, **2012**, *109(5)*, 1413–1418. DOI: 10.1073/pnas.1116289109

[64] Lepri, S.; Livi, R. and Politi, A.: Thermal conduction in classical low-dimensional lattices. *Physics Reports*, **2003**, *377(1)*, 1–80. DOI: 10.1016/S0370-1573(02)00558-6

[65] Schade, M. and Hamm, P.: Vibrational energy transport in the presence of intra-site vibrational energy redistribution. *The Journal of Chemical Physics*, **2009**, *131(4)*, 044511. DOI: 10.1063/1.3185152

[66] Lin, Z.; Keiffer, P. and Rubtsov, I. V.: A Method for Determining Small Anharmonicity Values from 2DIR Spectra Using Thermally Induced Shifts of Frequencies of High-Frequency Modes. *The Journal of Physical Chemistry B*, **2011**, *115(18)*, 5347–5353. DOI: 10.1021/jp1094189

[67] Rubtsova, N. I. and Rubtsov, I. V.: Ballistic energy transport via perfluoroalkane linkers. *Chemical Physics*, **2013**, 422, 16–21. DOI: 10.1016/j.chemphys.2013.01.026

[68] Wang, Z.; Carter, J. A.; Lagutchev, A.; Koh, Y. K.; Seong, N.-H.; Cahill, D. G. and Dlott, D. D.: Ultrafast Flash Thermal Conductance of Molecular Chains. *Science*, **2007**, *317(5839)*, 787–790. DOI: 10.1126/science.1145220

[69] Schröder, C.; Vikhrenko, V. and Schwarzer, D.: Molecular Dynamics Simulation of Heat Conduction through a Molecular Chain. *The Journal of Physical Chemistry A*, **2009**, *113(51)*, 14039–14051. DOI: 10.1021/jp903546h

[70] Backus, E. H. G.; Nguyen, P. H.; Botan, V.; Moretto, A.; Crisma, M.; Toniolo, C.; Zerbe, O.; Stock, G. and Hamm, P.: Structural Flexibility of a Helical Peptide Regulates Vibrational Energy Transport Properties. *The Journal of Physical Chemistry B*, **2008**, *112(48)*, 15487–15492. DOI: 10.1021/jp806403p

[71] Cahill, D. G.; Ford, W. K.; Goodson, K. E.; Mahan, G. D.; Majumdar, A.; Maris, H. J.; Merlin, R. and Phillpot, S. R.: Nanoscale thermal transport. *Journal of Applied Physics*, **2003**, *93(2)*, 793–818. DOI: 10.1063/1.1524305

[72] Botan, V.; Backus, E. H. G.; Pfister, R.; Moretto, A.; Crisma, M.; Toniolo, C.; Nguyen, P. H.; Stock, G. and Hamm, P.: Energy transport in peptide helices. *Proceedings of the National Academy of Sciences*, **2007**, *104(31)*, 12749–12754. DOI: 10.1073/pnas.0701762104

[73] Leitner, D. M.: Vibrational energy transfer and heat conduction in a one-dimensional glass. *Physical Review B*, **2001**, 64, 094201. DOI: 10.1103/PhysRevB.64.094201

[74] Fourier, J. B. J.: *The Analytical Theory of Heat*. Cambridge University Press, **1878**.

[75] Kutne, P.: *Zeitaufgelöste Untersuchngen zum intramolekularen Schwingungsenergiefluss durch molekulare Ketten*. PhD thesis, Georg-August-Universität Göttingen, **2003**.

[76] Kaviany, M.: *Essentials of Heat Transfer: Principles, Materials, and Applications*. Essentials of Heat Transfer: Principles, Materials, and Applications: Cambridge University Press, **2011**. ISBN: 978-1-107-01240-0

[77] Chen, G.: Phonon heat conduction in nanostructures. *International Journal of Thermal Sciences*, **2000**, *39(4)*, 471–480. DOI: 10.1016/S1290-0729(00)00202-7

[78] Backus, E. H. G.; Nguyen, P. H.; Botan, V.; Pfister, R.; Moretto, A.; Crisma, M.; Toniolo, C.; Stock, G. and Hamm, P.: Energy Transport in Peptide Helices: A Comparison between High- and Low-Energy Excitations. *The Journal of Physical Chemistry B*, **2008**, *112(30)*, 9091–9099. DOI: 10.1021/jp711046e

[79] Kab, G.; Schroder, C. and Schwarzer, D.: Intramolecular vibrational redistribution and energy relaxation in solution: A molecular dynamics approach. *Physical Chemistry Chemical Physics*, **2002**, 4, 271–278. DOI: 10.1039/B107256K

[80] Schwarzer, D.; Hanisch, C.; Kutne, P. and Troe, J.: Vibrational Energy Transfer in Highly Excited Bridged Azulene-Aryl Compounds: Direct Observation of Energy Flow through Aliphatic Chains and into the Solvent? *The Journal of Physical Chemistry A*, **2002**, *106(35)*, 8019–8028. DOI: 10.1021/jp0210576

[81] Schröder, C.: *Molekulardynamische Simulationen zum intra– und intermolekularen Schwingungsenergietransfer von ausgewählten Molekülen*. PhD thesis, Georg-August-Universität Göttingen, **2003**.

[82] Kühn, S.: *Intramolekulare Schwingungsenergieumverteilung in verbrückten Azulen-Anthrazen-Verbindungen*. PhD thesis, Georg-August-Universität Göttingen, **2007**.

[83] Schwarzer, D.; Troe, J. and Schroeder, J.: S_1-Lifetime of Azulene in Solution. *Berichte der Bunsengesellschaft für physikalische Chemie*, **1991**, *95(8)*, 933–934. DOI: 10.1002/bbpc.19910950814

[84] A. Ruth, A.; E.-K. Kim, d. and Hese, A.: The $S_0 \rightarrow S_1$ cavity ring-down absorption spectrum of jet-cooled azulene: dependence of internal conversion on the excess energy. *Physical Chemistry Chemical Physics*, **1999**, 1, 5121–5128. DOI: 10.1039/A906344G

[85] Wurzer, A.; Wilhelm, T.; Piel, J. and Riedle, E.: Comprehensive measurement of the S_1 azulene relaxation dynamics and observation of vibrational wavepacket motion. *Chemical Physics Letters*, **1999**, *299(3–4)*, 296–302. DOI: 10.1016/S0009-2614(98)01288-3

[86] Nägele, T.; Hoche, R.; Zinth, W. and Wachtveitl, J.: Femtosecond photoisomerization of *cis*-azobenzene. *Chemical Physics Letters*, **1997**, *272(5–6)*, 489–495. DOI: 10.1016/S0009-2614(97)00531-9

[87] Müller-Werkmeister, H. M. and Bredenbeck, J.: A donor-acceptor pair for the real time study of vibrational energy transfer in proteins. *Physical Chemistry Chemical Physics*, **2013**, accepted manuscript. DOI: 10.1039/C3CP54760D

[88] Carter, J. A.; Wang, Z. and Dlott, D. D.: Spatially Resolved Vibrational Energy Transfer in Molecular Monolayers. *The Journal of Physical Chemistry A*, **2008**, *112(16)*, 3523–3529. DOI: 10.1021/jp800278c

[89] Nguyen, P. H.; Park, S.-M. and Stock, G.: Nonequilibrium molecular dynamics simulation of the energy transport through a peptide helix. *The Journal of Chemical Physics*, **2010**, *132(2)*, 025102. DOI: 10.1063/1.3284742

[90] Deàk, J. C.; Pang, Y.; Sechler, T. D.; Wang, Z. and Dlott, D. D.: Vibrational Energy Transfer Across a Reverse Micelle Surfactant Layer. *Science*, **2004**, *306(5695)*, 473–476. DOI: 10.1126/science.1102074

[91] Kasyanenko, V. M.; Lin, Z.; Rubtsov, G. I.; Donahue, J. P. and Rubtsov, I. V.: Energy transport via coordination bonds. *The Journal of Chemical Physics*, **2009**, *131(15)*, 154508. DOI: 10.1063/1.3246862

[92] Kasyanenko, V. M.; Tesar, S. L.; Rubtsov, G. I.; Burin, A. L. and Rubtsov, I. V.: Structure Dependent Energy Transport: Relaxation-Assisted 2DIR Measurements and Theoretical Studies. *The Journal of Physical Chemistry B*, **2011**, *115(38)*, 11063–11073. DOI: 10.1021/jp2066315

[93] Naraharisetty, S. R. G.; Kasyanenko, V. M. and Rubtsov, I. V.: Bond connectivity measured via relaxation-assisted two-dimensional infrared spectroscopy. *The Journal of Chemical Physics*, **2008**, *128(10)*, 104502. DOI: 10.1063/1.2842071

[94] Naraharisetty, S. R. G.; Kasyanenko, V. M.; Zimmermann, J.; Thielges, M. C.; Romesberg, F. E. and Rubtsov, I. V.: C–D Modes of Deuterated Side Chain of Leucine as Structural Reporters via Dual-frequency Two-dimensional Infrared

Spectroscopy. *The Journal of Physical Chemistry B*, **2009**, *113(14)*, 4940–4946. DOI: 10.1021/jp8112446

[95] Carter, J. A.; Wang, Z. and Dlott, D. D.: Ultrafast Nonlinear Coherent Vibrational Sum-Frequency Spectroscopy Methods To Study Thermal Conductance of Molecules at Interfaces. *Accounts of Chemical Research*, **2009**, *42(9)*, 1343–1351. DOI: 10.1021/ar9000197

[96] Lagutchev, A. S.; Patterson, J. E.; Huang, W. and Dlott, D. D.: Ultrafast Dynamics of Self-Assembled Monolayers under Shock Compression: Effects of Molecular and Substrate Structure. *The Journal of Physical Chemistry B*, **2005**, *109(11)*, 5033–5044. DOI: 10.1021/jp0450742

[97] Schade, M. and Hamm, P.: Transition from {IVR} limited vibrational energy transport to bulk heat transport. *Chemical Physics*, **2012**, *393(1)*, 46–50. DOI: 10.1016/j.chemphys.2011.11.018

[98] Kurochkin, D. V.; Naraharisetty, S. R. G. and Rubtsov, I. V.: A relaxation-assisted 2D IR spectroscopy method. *Proceedings of the National Academy of Sciences*, **2007**, *104(36)*, 14209–14214. DOI: 10.1073/pnas.0700560104

[99] Davydov, A. S.: Solitons in Molecular Systems. *Physica Scripta*, **1979**, *20(3-4)*, 387–394. DOI: 10.1088/0031-8949/20/3-4/013

[100] Burin, A. L.; Tesar, S. L.; Kasyanenko, V. M.; Rubtsov, I. V. and Rubtsov, G. I.: Semiclassical Model for Vibrational Dynamics in Polyatomic Molecules: Investigation of Internal Vibrational Relaxation? *The Journal of Physical Chemistry C*, **2010**, *114(48)*, 20510–20517. DOI: 10.1021/jp104946m

[101] Tesar, S. L.; Kasyanenko, V. M.; Rubtsov, I. V.; Rubtsov, G. I. and Burin, A. L.: Theoretical Study of Internal Vibrational Relaxation and Energy Transport in Polyatomic Molecules. *The Journal of Physical Chemistry A*, **2013**, *117(2)*, 315–323. DOI: 10.1021/jp309481u

[102] Lieber, E.; Rao, C. N. R.; Chao, T. S. and Hoffman, C. W. W.: Infrared Spectra of Organic Azides. *Analytical Chemistry*, **1957**, *29(6)*, 916–918. DOI: 10.1021/ac60126a016

[103] Durig, D. T.; Durig, M. and Durig, J. R.: On the vibrational spectra and structural parameters of methyl, silyl, and germyl azide from theoretical predictions

and experimental data. *Spectrochimica Acta Part A*, **2005**, *61(7)*, 1287–1306. DOI: 10.1016/j.saa.2004.12.035

[104] Li, X.; Zhao, Y.; Cheng, L.; Yan, M.; Zheng, X.; Gao, Z. and Jiang, Z.: Enhanced ionic conductivity of poly(ethylene oxide) (PEO) electrolyte by adding mesoporous molecular sieve LiAlSBA. *Journal of Solid State Electrochemistry*, **2005**, *9(9)*, 609–615. DOI: 10.1007/s10008-004-0613-y

[105] Schwetlick, K.: *Organikum*. Wiley-VCH, Weinheim, 23rd edition edition, **2009**. ISBN: 978-3-527-32292-3

[106] Kalsi, P.: *Spectroscopy of organic compounds*. New Age International, 6th edition edition, **2004**. ISBN: 978-81-224-1543-8

[107] Wilson, E. B.; Decius, J. and Cross, P. C.: *Molecular Vibrations*. Dover Publications, Mineola, New York, **1980**. (first published in 1955 by McGraw-Hill Book Company, Inc.). ISBN: 978-0-486-63941-3

[108] Begue, D.; Baraille, I.; Garrain, P. A.; Dargelos, A. and Tassaing, T.: Calculation of IR frequencies and intensities in electrical and mechanical anharmonicity approximations: Application to small water clusters. *The Journal of Chemical Physics*, **2010**, *133(3)*, 034102. DOI: 10.1063/1.3457482

[109] Levine, I. N.: *Molecular Spectroscopy*. John Wiley & Sons, Inc., **1975**. ISBN: 978-0-471-53128-9

[110] Califano, S.: *Vibrational States*. John Wiley & Sons, Inc., **1976**. ISBN: 978-0-471-12996-7

[111] Gaw, J. F.; Willetts, A.; Green, W. H. and Handy, N. C.: Spectro: A Program for the Derivation of Spectroscopic Constants from Provided Quartic Force Fields and Cubic Dipole Fields. In *Advances in Molecular Vibrations and Collision Dynamics*, volume *1B*, pages 169–185: JAI Press Inc., **1991**. ISBN: 978-1-55938-295-3

[112] Green, W. H.; Jayatilaka, D.; Willetts, A.; Amos, R. D. and Handy, N. C.: The prediction of spectroscopic properties from quartic correlated force fields: HCCF, HFCO, SiH_3^+. *The Journal of Chemical Physics*, **1990**, *93(7)*, 4965–4981. DOI: 10.1063/1.458634

[113] Maslen, P. E.; Handy, N. C.; Amos, R. D. and Jayatilaka, D.: Higher analytic derivatives. IV. Anharmonic effects in the benzene spectrum. *The Journal of Chemical Physics*, **1992**, *97(6)*, 4233–4254. DOI: 10.1063/1.463926

[114] Mills, I. M.: Vibration–Rotation Structure in Asymmetric- and Symmetric-Top Molecules. In Rao, K. N. and Mathes, C. W. (editors): *Molecular Spectroscopy: Modern Research*, pages 115–140: Academic Press, **1972**. ISBN: 978-0-12-580640-4

[115] Dunham, J. L.: The Energy Levels of a Rotating Vibrator. *Physical Review*, **1932**, 41, 721–731. DOI: 10.1103/PhysRev.41.721

[116] Amos, R. D.; Handy, N. C.; Green, W. H.; Jayatilaka, D.; Willetts, A. and Palmieri, P.: Anharmonic vibrational properties of CH_2F_2 : A comparison of theory and experiment. *The Journal of Chemical Physics*, **1991**, *95(11)*, 8323–8336. DOI: 10.1063/1.461259

[117] Martin, J. M. L.; Lee, T. J.; Taylor, P. R. and François, J.: The anharmonic force field of ethylene, C_2H_4, by means of accurate *ab initio* calculations. *The Journal of Chemical Physics*, **1995**, *103(7)*, 2589–2602. DOI: 10.1063/1.469681

[118] Reichardt, C.; Schroeder, J. and Schwarzer, D.: Femtosecond IR Spectroscopy of Peroxycarbonate Photodecomposition: S_1-Lifetime Determines Decarboxylation Rate. *The Journal of Physical Chemistry A*, **2007**, *111(40)*, 10111–10118. DOI: 10.1021/jp0742968

[119] Kandratsenka, A.; Schroeder, J.; Schwarzer, D. and Vikhrenko, V. S.: Mode-specific energy absorption by solvent molecules during CO_2 vibrational cooling. *Physical Chemistry Chemical Physics*, **2007**, *9(14)*, 1688–1692. DOI: 10.1039/b618452a

[120] Gompertz, B.: On the Nature of the Function Expressive of the Law of Human Mortality, and on a New Mode of Determining the Value of Life Contingencies. *Philosophical Transactions of the Royal Society of London*, **1825**, 115, 513–583. DOI: 10.1098/rstl.1825.0026

[121] Schwarzer, D.; Troe, J.; Votsmeier, M. and Zerezke, M.: Collisional deactivation of vibrationally highly excited azulene in compressed liquids and supercritical fluids. *The Journal of Chemical Physics*, **1996**, *105(8)*, 3121–3131. DOI: 10.1063/1.472180

[122] Schultz, K. E.; Russell, D. J. and Harris, C. B.: The applicability of binary collision theories to complex molecules in simple liquids. *The Journal of Chemical Physics*, **1992**, *97(8)*, 5431–5438. DOI: 10.1063/1.463987

[123] Wild, W.; Seilmeier, A.; Gottfried, N. and Kaiser, W.: Ultrafast investigations of vibrationally hot molecules after internal conversion in solution. *Chemical Physics Letters*, **1985**, *119(4)*, 259–263. DOI: 10.1016/0009-2614(85)80413-9

[124] Bearpark, M. J.; Bernardi, F.; Clifford, S.; Olivucci, M.; Robb, M. A.; Smith, B. R. and Vreven, T.: The Azulene S_1 State Decays via a Conical Intersection: A CASSCF Study with MMVB Dynamics. *Journal of the American Chemical Society*, **1996**, *118(1)*, 169–175. DOI: 10.1021/ja9514555

[125] Milligan, D. E.: Infrared Spectroscopic Study of the Photolysis of Methyl Azide and Methyl-d_3 Azide in Solid Argon and Carbon Dioxide. *The Journal of Chemical Physics*, **1961**, *35(4)*, 1491–1497. DOI: 10.1063/1.1732070

[126] Tian, R.; Facelli, J. C. and Michl, J.: Vibrational and electronic spectra of matrix-isolated nitrogen trimer radical and azide. *The Journal of Physical Chemistry*, **1988**, *92(14)*, 4073–4079. DOI: 10.1021/j100325a018

[127] Wang, Z.; Pakoulev, A. and Dlott, D. D.: Watching Vibrational Energy Transfer in Liquids with Atomic Spatial Resolution. *Science*, **2002**, *296(5576)*, 2201–2203. DOI: 10.1126/science.1071293

[128] Stuchebrukhov, A. A. and Marcus, R. A.: Theoretical study of intramolecular vibrational relaxation of acetylenic CH vibration for $v = 1$ and 2 in large polyatomic molecules $(CX_3)_3YCCH$, where X=H or D and Y=C or Si. *The Journal of Chemical Physics*, **1993**, *98(8)*, 6044–6061. DOI: 10.1063/1.464843

[129] Yu, X. and Leitner, D. M.: Vibrational Energy Transfer and Heat Conduction in a Protein. *The Journal of Physical Chemistry B*, **2003**, *107(7)*, 1698–1707. DOI: 10.1021/jp026462b

[130] Wuttke, J.: lmfit – a C library for Levenberg-Marquardt least-squares minimization and curve fitting, **2008**. http://apps.jcns.fz-juelich.de/lmfit Release lmfit2.3.

[131] Marquardt, D.: An Algorithm for Least-Squares Estimation of Nonlinear Parameters. *Journal of the Society for Industrial and Applied Mathematics*, **1963**, *11(2)*, 431–441. DOI: 10.1137/0111030

[132] Hamm, P.: Electronic mail from November 3, 2013.

[133] Yao, S. and Overend, J.: Vibrational intensities—XXIII. The effect of anharmonicity on the temperature dependence of integrated band intensities. *Spectrochimica Acta Part A*, **1976**, *32(5)*, 1059–1065. DOI: 10.1016/0584-8539(76)80290-5

[134] Hamm, P.: Private communication.

[135] Beyer, T. and Swinehart, D. F.: Algorithm 448: number of multiply-restricted partitions. *Communications of the ACM*, **1973**, *16(6)*, 379. DOI: 10.1145/362248.362275

[136] Lauro, C. di and Mills, I.: Coriolis interactions about X-Y axes in symmetric tops. *Journal of Molecular Spectroscopy*, **1966**, *21(1–4)*, 386–413. DOI: 10.1016/0022-2852(66)90164-0

[137] Bernath, P. F.: Molecular Spectroscopy and Structure. In Cohen, E.; Lide, D. and Trigg, G. (editors): *A Physicist's Desk Reference*, chapter 16: Springer, 3rd Edition edition, **2000**. ISBN: 978-0-387-98973-0

[138] Atkinson, G. and Parmenter, C.: The 260 nm absorption spectrum of benzene: Selection rules and band contours of vibrational angular momentum components. *Journal of Molecular Spectroscopy*, **1978**, *73(1)*, 31–51. DOI: 10.1016/0022-2852(78)90196-0

[139] Strey, G. and Mills, I.: Anharmonic force field of acetylene. *Journal of Molecular Spectroscopy*, **1976**, *59(1)*, 103–115. DOI: 10.1016/0022-2852(76)90046-1

[140] Hippler, H.; Troe, J. and Wendelken, H. J.: UV absorption spectra of vibrationally highly excited toluene molecules. *The Journal of Chemical Physics*, **1983**, *78(9)*, 5351–5357. DOI: 10.1063/1.445488

[141] Shirts, R. B. and Shirts, M. R.: Deviations from the Boltzmann distribution in small microcanonical quantum systems: Two approximate one-particle energy distributions. *The Journal of Chemical Physics*, **2002**, *117(12)*, 5564–5575. DOI: 10.1063/1.1503306

[142] Brouwer, L. D.; Hippler, H.; Lindemann, L. and Troe, J.: Measurement of internal energies by hot ultraviolet absorption spectroscopy: spectra of excited azulene molecules. *The Journal of Physical Chemistry*, **1985**, *89(21)*, 4608–4612. DOI: 10.1021/j100267a039

[143] Quack, M.: Statistical mechanics and dynamics of molecular fragmentation. *Il Nuovo Cimento B Series 11*, **1981**, *63(1)*, 358–377. DOI: 10.1007/BF02721444

[144] Hippler, H.; Troe, J. and Wendelken, H. J.: Collisional deactivation of vibrationally highly excited polyatomic molecules. II. Direct observations for excited toluene. *The Journal of Chemical Physics*, **1983**, *78(11)*, 6709–6717. DOI: 10.1063/1.444670

[145] Jr., W. H. G.; Willetts, A.; Jayatilaka, D. and Handy, N. C.: Ab initio prediction of fundamental, overtone and combination band infrared intensities. *Chemical Physics Letters*, **1990**, *169(1–2)*, 127–137. DOI: 10.1016/0009-2614(90)85177-E

[146] Miller, W. H.; Hernandez, R.; Handy, N. C.; Jayatilaka, D. and Willetts, A.: Ab initio calculation of anharmonic constants for a transition state, with application to semiclassical transition state tunneling probabilities. *Chemical Physics Letters*, **1990**, *172(1)*, 62–68. DOI: 10.1016/0009-2614(90)87217-F

[147] Hassinen, T.; Hutchison, G.; Cruz, M.; Banck, M.; Rowley, C.; Brefort, J.; Leidert, D. and Peshov, V.: Ghemical-3.0.0. published under GPL, **2011**. (original copyright 1998) http://www.bioinformatics.org/ghemical.

[148] Clark, M.; Cramer, R. D. and Van Opdenbosch, N.: Validation of the general purpose tripos 5.2 force field. *Journal of Computational Chemistry*, **1989**, *10(8)*, 982–1012. DOI: 10.1002/jcc.540100804

[149] Turney, J. M.; Simmonett, A. C.; Parrish, R. M.; Hohenstein, E. G.; Evangelista, F. A.; Fermann, J. T.; Mintz, B. J.; Burns, L. A.; Wilke, J. J.; Abrams, M. L.; Russ, N. J.; Leininger, M. L.; Janssen, C. L.; Seidl, E. T.; Allen, W. D.; Schaefer, H. F.; King, R. A.; Valeev, E. F.; Sherrill, C. D. and Crawford, T. D.: Psi4: an open-source ab initio electronic structure program. *Wiley Interdisciplinary Reviews: Computational Molecular Science*, **2012**, *2(4)*, 556–565. DOI: 10.1002/wcms.93

[150] Werner, H.-J.; Manby, F. R. and Knowles, P. J.: Fast linear scaling second-order Møller-Plesset perturbation theory (MP2) using local and density fitting approximations. *The Journal of Chemical Physics*, **2003**, *118(18)*, 8149–8160. DOI: 10.1063/1.1564816

[151] Dunning, T. H.: Gaussian basis sets for use in correlated molecular calculations. I. The atoms boron through neon and hydrogen. *The Journal of Chemical Physics*, **1989**, *90(2)*, 1007–1023. DOI: 10.1063/1.456153

[152] Oliphant, T. E.: Python for Scientific Computing. *Computing in Science & Engineering*, **2007**, *9(3)*, 10–20. DOI: 10.1109/MCSE.2007.58

[153] Jones, E.; Oliphant, T.; Peterson, P. *et al.*: SciPy: Open source scientific tools for Python, **2001–2013**. `http://www.scipy.org`, Version 0.13.2.

[154] Schneider, W. and Thiel, W.: Anharmonic force fields from analytic second derivatives: Method and application to methyl bromide. *Chemical Physics Letters*, **1989**, *157(4)*, 367–373. DOI: `10.1016/0009-2614(89)87263-X`

[155] Mills, I. and Robiette, A.: On the relationship of normal modes to local modes in molecular vibrations. *Molecular Physics*, **1985**, *56(4)*, 743–765. DOI: `10.1080/00268978500102691`

[156] Frisch, M. J.; Trucks, G. W.; Schlegel, H. B.; Scuseria, G. E.; Robb, M. A.; Cheeseman, J. R.; Scalmani, G.; Barone, V.; Mennucci, B.; Petersson, G. A.; Nakatsuji, H.; Caricato, M.; Li, X.; Hratchian, H. P.; Izmaylov, A. F.; Bloino, J.; Zheng, G.; Sonnenberg, J. L.; Hada, M.; Ehara, M.; Toyota, K.; Fukuda, R.; Hasegawa, J.; Ishida, M.; Nakajima, T.; Honda, Y.; Kitao, O.; Nakai, H.; Vreven, T.; Montgomery, J. A.; Peralta, J. E.; Ogliaro, F.; Bearpark, M.; Heyd, J. J.; Brothers, E.; Kudin, K. N.; Staroverov, V. N.; Kobayashi, R.; Normand, J.; Raghavachari, K.; Rendell, A.; Burant, J. C.; Iyengar, S. S.; Tomasi, J.; Cossi, M.; Rega, N.; Millam, J. M.; Klene, M.; Knox, J. E.; Cross, J. B.; Bakken, V.; Adamo, C.; Jaramillo, J.; Gomperts, R.; Stratmann, R. E.; Yazyev, O.; Austin, A. J.; Cammi, R.; Pomelli, C.; Ochterski, J. W.; Martin, R. L.; Morokuma, K.; Zakrzewski, V. G.; Voth, G. A.; Salvador, P.; Dannenberg, J. J.; Dapprich, S.; Daniels, A. D.; Farkas, O.; Foresman, J. B.; Ortiz, J. V.; Cioslowski, J. and Fox, D. J.: Gaussian 09, **2009**.

[157] Sastry, S.; Deo, N. and Franz, S.: Spectral statistics of instantaneous normal modes in liquids and random matrices. *Physical Review E*, **2001**, 64, 016305. DOI: `10.1103/PhysRevE.64.016305`

[158] Wang, J. and Hochstrasser, R. M.: Anharmonicity of Amide Modes. *The Journal of Physical Chemistry B*, **2006**, *110(8)*, 3798–3807. DOI: `10.1021/jp0530092`

[159] Hamm, P. and Zanni, M.: *Concepts and Methods of 2D Infrared Spectroscopy*. Cambridge University Press, **2011**. ISBN: `978-1-107-00005-6`

[160] Beil, A.; Luckhaus, D.; Quack, M. and Stohner, J.: Intramolecular vibrational redistribution and unimolecular reaction: Concepts and new results on the femtosecond dynamics and statistics in CHBrClF. *Berichte der Bunsengesellschaft für physikalische Chemie*, **1997**, *101(3)*, 311–328. DOI: `10.1002/bbpc.19971010303`

[161] Steele, D.: The out-of-plane vibrations of azulene. *Journal of Molecular Spectroscopy*, **1965**, *15(3)*, 333–343. DOI: 10.1016/0022-2852(65)90149-9

[162] Steele, D.: The in-plane vibrations of azulene. *Spectrochimica Acta*, **1966**, *22(7)*, 1275–1282. DOI: 10.1016/0371-1951(66)80032-2

[163] Steele, D.: The in-plane vibrations of azulene. *Spectrochimica Acta Part A*, **1967**, *23(5)*, 1599. DOI: 10.1016/0584-8539(67)80384-2

[164] Lawrence, W. G. and Apkarian, V. A.: Many-body potentials of an open shell atom: Spectroscopy of spin–orbit transitions of iodine in crystalline Xe and Kr. *The Journal of Chemical Physics*, **1994**, *101(3)*, 1820–1831. DOI: 10.1063/1.467761

[165] Fletcher, R.: *Practical Methods of Optimization*. Wiley, 2nd edition, **1987**. ISBN: 978-0-471-91547-8

[166] GNU Scientific Library. http://www.gnu.org/software/gsl, Version 1.15.

[167] Wille, L. T. and Vennik, J.: Computational complexity of the ground-state determination of atomic clusters. *Journal of Physics A*, **1985**, *18(8)*, L419–L422. DOI: 10.1088/0305-4470/18/8/003

[168] Greenwood, G. W.: Revisiting the Complexity of Finding Globally Minimum Energy Configurations in Atomic Clusters. *Zeitschrift für Physikalische Chemie*, **1999**, *211(1)*, 105–114. DOI: 10.1524/zpch.1999.211.Part_1.105

[169] Danilychev, A. V. and Apkarian, V. A.: Atomic oxygen in crystalline Kr and Xe. II. Adiabatic potential energy surfaces. *The Journal of Chemical Physics*, **1994**, *100(8)*, 5556–5566. DOI: 10.1063/1.467174

[170] Yourshaw, I.: *Study of Pair and Many-Body Interactions in Rare-Gas Halide Atom Clusters Using Negative Ion Zero Electron Kinetic Energy (ZEKE) and Threshold Photodetachment Spectroscopy*. PhD thesis, University of California, Berkeley, **1998**.

[171] Aquilanti, V. and Grossi, G.: Angular momentum coupling schemes in the quantum mechanical treatment of *P*–state atom collisions. *The Journal of Chemical Physics*, **1980**, *73(3)*, 1165–1172. DOI: 10.1063/1.440270

[172] Aquilanti, V.; Liuti, G.; Pirani, F. and Vecchiocattivi, F.: Orientational and spin-orbital dependence of interatomic forces. *Journal of the Chemical Society, Faraday Transactions 2*, **1989**, 85, 955–964. DOI: 10.1039/F29898500955

[173] Balling, L. C. and Wright, J. J.: Use of dimer potentials to calculate the energy levels of alkali atoms in rare–gas matrices. *The Journal of Chemical Physics*, **1983**, *79(6)*, 2941–2944. DOI: 10.1063/1.446118

[174] Boatz, J. A. and Fajardo, M. E.: Monte Carlo simulations of the structures and optical absorption spectra of Na atoms in Ar clusters, surfaces, and solids. *The Journal of Chemical Physics*, **1994**, *101(5)*, 3472–3487. DOI: 10.1063/1.467532

[175] Maplesoft: Maple 12.01, **2008**. Build ID 363216.

[176] Buckingham, A. D.: Permanent and Induced Molecular Moments and Long-Range Intermolecular Forces. In Hirschfelder, J. O. (editor): *Advances in Chemical Physics*, pages 107–142: John Wiley & Sons, Inc., **2007**. ISBN: 978-0-470-14358-2

[177] Applequist, J.: Cartesian polytensors. *Journal of Mathematical Physics*, **1983**, *24(4)*, 736–741. DOI: 10.1063/1.525770

[178] Applequist, J.: A multipole interaction theory of electric polarization of atomic and molecular assemblies. *The Journal of Chemical Physics*, **1985**, *83(2)*, 809–826. DOI: 10.1063/1.449496

[179] Kong, Y. and Ponder, J. W.: Calculation of the reaction field due to off-center point multipoles. *The Journal of Chemical Physics*, **1997**, *107(2)*, 481–492. DOI: 10.1063/1.474409

[180] Kong, Y.: *Multipole Electrostatic Methods for Protein Modeling with Reaction Field Treatment*. PhD thesis, Washington University, **1997**.

[181] Vesely, F. J.: N-particle dynamics of polarizable Stockmayer-type molecules. *Journal of Computational Physics*, **1977**, *24(4)*, 361–371. DOI: 10.1016/0021-9991(77)90028-6

[182] Press, W.; Teukolsky, S.; Vetterling, W. and Flannery, B.: *Numerical Recipes in C*. Cambridge University Press, 2nd edition, **1992**.

[183] Strassen, V.: Gaussian elimination is not optimal. *Numerische Mathematik*, **1969**, 13, 354–356. DOI: 10.1007/BF02165411

[184] Baker, G.; Gunnels, J.; Morrow, G.; Riviere, B. and Geijn, R. van de: PLAPACK: high performance through high-level abstraction. In *Parallel Processing, 1998. Proceedings. 1998 International Conference on*, pages 414–422, **1998**.

[185] Khriachtchev, L.; Pettersson, M.; Runeberg, N.; Lundell, J. and Rasanen, M.: A stable argon compound. *Nature*, **2000**, 406, 874–876. DOI: 10.1038/35022551

[186] Wade, K.: The structural significance of the number of skeletal bonding electron-pairs in carboranes, the higher boranes and borane anions, and various transition-metal carbonyl cluster compounds. *Journal of the Chemical Society D*, **1971**, 15, 792–793. DOI: 10.1039/C29710000792

[187] Becker, C. H.; Casavecchia, P. and Lee, Y. T.: Crossed molecular beam studies on the interaction potentials for F(^2P) + Ne,Ar,Kr(^1S). *The Journal of Chemical Physics*, **1979**, *70(6)*, 2986–2990. DOI: 10.1063/1.437837

[188] Kirkpatrick, C. C. and Viehland, L. A.: Interaction potentials for the halide ion-rare gas systems. *Chemical Physics*, **1985**, *98(2)*, 221–231. DOI: 10.1016/0301-0104(85)80135-X

[189] Lenzer, T.; Yourshaw, I.; Furlanetto, M. R.; Reiser, G. and Neumark, D. M.: Zero electron kinetic energy spectroscopy of the $ArCl^-$ anion. *The Journal of Chemical Physics*, **1999**, *110(19)*, 9578–9586. DOI: 10.1063/1.478923

[190] Yourshaw, I.; Lenzer, T.; Reiser, G. and Neumark, D. M.: Zero electron kinetic energy spectroscopy of the $KrBr^-$, $XeBr^-$, and $KrCl^-$ anions. *The Journal of Chemical Physics*, **1998**, *109(13)*, 5247–5256. DOI: 10.1063/1.477141

[191] Zhao, Y.; Yourshaw, I.; Reiser, G.; Arnold, C. C. and Neumark, D. M.: Study of the $ArBr^-$, ArI^-, and KrI^- anions and the correspondingneutral van der Waals complexes by anion zero electron kinetic energy spectroscopy. *The Journal of Chemical Physics*, **1994**, *101(8)*, 6538–6551. DOI: 10.1063/1.468500

[192] Becker, C. H.; Casavecchia, P. and Lee, Y. T.: Crossed molecular beam studies on the interaction potential for F(^2P)+Xe(^1S). *The Journal of Chemical Physics*, **1978**, *69(6)*, 2377–2381. DOI: 10.1063/1.436921

[193] Aquilanti, V.; Cappelletti, D.; Lorent, V.; Luzzatti, E. and Pirani, F.: The ground and lowest excited states of XeCl by atomic beam scattering. *Chemical Physics Letters*, **1992**, *192(2–3)*, 153–160. DOI: 10.1016/0009-2614(92)85445-G

[194] Lenzer, T.; Yourshaw, I.; Furlanetto, M. R.; Pivonka, N. L. and Neumark, D. M.: Zero electron kinetic energy spectroscopy of the $XeCl^-$ anion. *The Journal of Chemical Physics*, **2002**, *116(10)*, 4170–4175. DOI: 10.1063/1.1450551

[195] Aziz, R. A. and Slaman, M.: The argon and krypton interatomic potentials revisited. *Molecular Physics*, **1986**, *58(4)*, 679–697. DOI: 10.1080/00268978600101501

[196] Dham, A. K.; Allnatt, A.; Meath, W. J. and Aziz, R. A.: The Kr-Kr potential energy curve and related physical properties; the XC and HFD-B potential models. *Molecular Physics*, **1989**, *67(6)*, 1291–1307. DOI: 10.1080/00268978900101821

[197] Dham, A. K.; Meath, W. J.; Allnatt, A.; Aziz, R. A. and Slaman, M.: XC and HFD-B potential energy curves for Xe-Xe and related physical properties. *Chemical Physics*, **1990**, *142(2)*, 173–189. DOI: 10.1016/0301-0104(90)89079-6

[198] Guillot, B.; Mountain, R. and Birnbaum, G.: Theoretical study of the three-body absorption spectrum in pure rare gas fluids. *Molecular Physics*, **1988**, *64(4)*, 747–757. DOI: 10.1080/00268978800100533

[199] Cohen, E. R.; Cvitaš, T.; Frey, J. G.; Holmström, B.; Kuchitsu, K.; Marquardt, R.; Mills, I.; Pavese, F.; Quack, M.; Stohner, J.; Strauss, H. L.; Takami, M.; Thor, A. J.; Homann, K. and Kallay, N.: *Quantities, Units and Symbols in Physical Chemistry*. IUPAC & RSC Publishing, 3rd Edition, 2nd Printing edition, **2008**.

[200] Hättig, C. and Hess, B. A.: TDMP2 Calculation of Dynamic Multipole Polarizabilities and Dispersion Coefficients of the Noble Gases Ar, Kr, Xe, and Rn. *The Journal of Physical Chemistry*, **1996**, *100(15)*, 6243–6248. DOI: 10.1021/jp9528121

[201] Aziz, R. A.: A highly accurate interatomic potential for argon. *The Journal of Chemical Physics*, **1993**, *99(6)*, 4518–4525. DOI: 10.1063/1.466051

[202] Maroulis, G. and Bishop, D. M.: On the electric polarisabilities of argon. *Journal of Physics B*, **1985**, *18(24)*, 4675–4682. DOI: 10.1088/0022-3700/18/24/012

[203] Maroulis, G. and Thakkar, A. J.: Quadrupole polarizabilities and hyperpolarizabilities of Kr and Xe from fourth-order many-body perturbation theory calculations. *The Journal of Chemical Physics*, **1988**, *89(12)*, 7320–7323. DOI: 10.1063/1.455313

[204] Hättig, C. and Heß, B. A.: TDMP2 calculation of dynamic multipole polarizabilities and dispersion coefficients for the halogen anions F^-, Cl^-, Br^- and I^-. *The Journal of Chemical Physics*, **1998**, *108(10)*, 3863–3870. DOI: 10.1063/1.475789

[205] Koutselos, A. D. and Mason, E. A.: Correlation and prediction of dispersion coefficients for isoelectronic systems. *The Journal of Chemical Physics*, **1986**, *85(4)*, 2154–2160. DOI: 10.1063/1.451108

[206] Zatsarinny, O.; Bartschat, K.; Mitroy, J. and Zhang, J.-Y.: Multipole polarizabilities and long-range interactions of the fluorine atom. *The Journal of Chemical Physics*, **2009**, *130(12)*, 124310. DOI: 10.1063/1.3098320

[207] Pellenq, R. J.-M. and Nicholson, D.: Intermolecular Potential Function for the Physical Adsorption of Rare Gases in Silicalite. *The Journal of Physical Chemistry*, **1994**, *98(50)*, 13339–13349. DOI: 10.1021/j100101a039

[208] ies), D. P. and/or its subsidiary(: Qt. http://qt-project.org/, http://qt.digia.com/, Version 4.8.4.

[209] Rathmann, U. and Wilgen, J.: Qwt – Qt Widgets for Technical Applications. http://qwt.sourceforge.net/, Version 6.1.0.

[210] Berg, I.: muparser – fast math parser library. http://muparser.beltoforion.de/, Version 2.2.3.

[211] Seward, J.: libbzip2. http://www.bzip.org/, Version 1.0.6.

Appendices

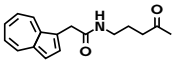
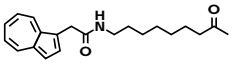
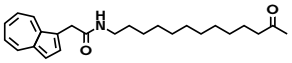
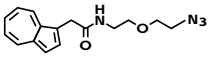
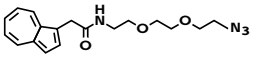
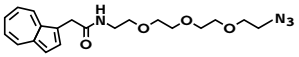
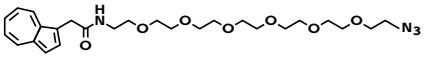
Appendix A

Material and equipment used

Item(s)	Supplier
Vitesse/Libra laser system	Coherent Inc., Santa Clara, CA, USA
Evolution laser system	Coherent Inc., Santa Clara, CA, USA
Gratings (stretcher, compressor)	Spectrogon AB, Täby, Sweden
Polychromator Chromex 250is	Bruker Optik GmbH, Ettlingen, Germany
Signal integrator IR-6416	InfraRed Associates Inc., Stuart, FL, USA
Detector MCT-32/2-10	InfraRed Associates Inc., Stuart, FL, USA
Delay generator DG535	Stanford Research Systems, Sunnyvale, CA, USA
Omnifit Labware, Omni-Lok	Diba Industries Inc., Danbury, CT, USA
Micropump GAX21. P8FS.B	IDEX Corp., Oak Harbor, WA, USA
Whatman GF 6 filters	Whatman GmbH, Dassel, Germany
Acetonitrile LiChrosolv	Merck KGaA, Darmstadt, Germany
d_3-Acetonitrile	Deutero GmbH, Kastellaun, Germany
Dichloromethane for analysis	Merck KGaA, Darmstadt, Germany
Diphenylacetylene (>99%)	abcr GmbH, Co. KG, Karlsruhe, Germany
Diphenylcyclopropenone (98%)†	Sigma-Aldrich, St. Louis, MO, USA

† further purified by column chromatography (ethyl acetate, silica gel) and subsequent recrystallization (petrol ether 60/80)[6]

All azulene compounds were kindly provided by Jens Schimpfhauser of the chemical synthesis facility of the Max-Planck-Institute for Biophysical Chemistry. Data on synthesis and analytical properties can be found under the following internal registration numbers:

Schi–096	
Schi–093	
Schi–097	
Schi–112	
Schi–111	
Schi–109	
Schi–110	

Appendix B

Geometry used for the computation of anharmonicity constants

Element	x [Å]	y [Å]	z [Å]	Element	x [Å]	y [Å]	z [Å]
C	1.98069	−2.78987	−1.36585	H	0.58160	−2.30860	−2.88983
C	1.13880	−1.89033	−2.04410	H	0.20971	−0.03620	−2.48161
C	0.91241	−0.52334	−1.79393	H	3.72612	−1.44644	1.31010
C	1.44207	0.28397	−0.78231	H	3.39440	−3.43755	0.08068
C	2.42360	−0.13018	0.28948	H	1.83270	3.08831	0.99083
C	3.01904	−1.38223	0.47183	H	3.32229	1.04797	1.95256
C	2.81931	−2.56880	−0.25919	H	−0.18176	2.04242	−2.22885
C	1.15521	1.65539	−0.56456	H	0.65627	3.48746	−1.58962
C	1.88392	2.07662	0.57834	H	−3.08096	2.78661	1.15502
C	2.65316	1.01163	1.09021	H	−3.75589	1.26316	0.49071
C	0.18514	2.53203	−1.31299	H	−2.79826	0.75015	2.70363
C	−0.95787	2.83576	−0.35037	H	−1.25078	1.55753	2.26092
C	−2.83102	1.78270	0.78479	H	−0.36320	−0.74324	2.77706
C	−2.11147	0.98331	1.86529	H	−1.94521	−1.60439	2.74019
C	−1.14004	−1.17346	2.11177	H	−0.16371	−3.09024	1.84554
C	−0.50804	−2.24347	1.23540	H	−1.76881	1.03813	−0.84789
O	−1.66046	−0.20321	1.22915	H	0.35436	−1.81870	0.70012
N	−1.98539	1.92196	−0.39079	N	−1.48174	−2.80596	0.27172
O	−0.92515	3.78531	0.42875	N	−1.79811	−2.04798	−0.65964
H	1.98623	−3.81282	−1.76137	N	−2.17479	−1.46462	−1.59420

Appendix C

Potential parameters for noble gas–halide clusters

Table C.1: Noble gas–halide binary interaction potentials

Halide–noble gas potentials

MMSV (Morse–Morse–switching function–van der Waals) type

$$\frac{V(x=r/r_m)}{\epsilon} = \begin{cases} \text{for } 0 < x \leqslant 1: & (\exp\left[\beta_1(1-x)\right]-1)^2 - 1 \\ \text{for } 1 < x \leqslant x_1: & M_2(x) = (\exp\left[\beta_2(1-x)\right]-1)^2 - 1 \\ \text{for } x_1 < x < x_2: & S(x)\, M_2(x) + [1-S(x)]\, W(x) \\ & \text{with } S(x) = \frac{1}{2}\left(\cos\frac{\pi\,(x-x_1)}{x_2-x_1}+1\right) \\ \text{for } x_2 \leqslant x: & W(x) = \begin{cases} -\frac{C_6}{\epsilon\, r_m^6}x^6 - \frac{C_8}{\epsilon\, r_m^8}x^8 & \text{for neutral species} \\ -\frac{B_4}{\epsilon\, r_m^4}x^4 - \frac{B_6}{\epsilon\, r_m^6}x^6 & \text{for anionic species} \end{cases} \end{cases}$$

	Ar ⋯ F[187], anionic potential fitted to [188]				Ar ⋯ Cl[189]			
	$X\frac{1}{2}$	$I\frac{3}{2}$	$II\frac{1}{2}$	anionic	$X\frac{1}{2}$	$I\frac{3}{2}$	$II\frac{1}{2}$	anionic
ϵ [meV]	12.1419	6.50461	6.50461	136.26393587	16.77	11.42	12.10	64.87
r_m [Å]	2.95	3.45	3.45	2.75263365	3.73	3.96	3.87	3.71
β_1	4.3	5.5	5.5	5.58563856	6.21	6.57	6.73	5.30
β_2	4.3	5.5	5.5	5.75637873	6.21	6.57	6.73	5.30
x_1	1.161	1.126	1.126	0.97470528	1.000	1.050	1.110	1.025
x_2	1.700	1.400	1.400	2.12944288	2.660	1.454	1.582	1.260
C_6 [eV Å^6]	12.619	13.226	13.226		43.4	46.3	44.9	
C_8 [eV Å^8]	55.0723	55.0723	55.0723		308.3	333.6	320.9	
B_4 [eV Å^4]				3.90144312				11.8
B_6 [eV Å^6]				314.49622971				98.1
	Ar ⋯ Br[3]				Ar ⋯ I[3]			
	$X\frac{1}{2}$	$I\frac{3}{2}$	$II\frac{1}{2}$	anionic	$X\frac{1}{2}$	$I\frac{3}{2}$	$II\frac{1}{2}$	anionic
ϵ [meV]	16.5	11.5	14.0	54.4	18.8	13.9	16.0	45.8
r_m [Å]	3.73	3.94	3.89	3.78	3.95	4.18	4.11	4.07
β_1	6.80	7.72	6.70	5.10	7.15	7.25	6.90	5.70
β_2	6.50	7.10	6.35	4.45	6.18	6.30	6.40	4.45
x_1	1.02	1.012	1.01	1.065	1.01	1.04	1.04	1.08

Not all figures significant.

	Ar ⋯ Br (continued)				Ar ⋯ I (continued)			
	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic
x_2	1.59	1.63	1.58	1.66	1.62	1.62	1.64	1.62
C_6 [eV Å^6]	65.2	70.2	68.8		98.4	98.4	98.4	
C_8 [eV Å^8]	379	379	379		715	715	715	
B_4 [eV Å^4]				12.5				12.8
B_6 [eV Å^6]				120.5				162
	Kr ⋯ F[187], anionic potential fitted to [188]				Kr ⋯ Cl[190]			
	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic
ϵ [meV]	13.4429	6.72143	6.72143	270.84735950	22.01	22.01	22.01	95.70
r_m [Å]	3.0	3.6	3.6	2.46779731	3.75	3.75	3.75	3.83
β_1	4.3	5.5	5.5	5.28747801	5.49	5.49	5.49	5.70
β_2	4.3	5.5	5.5	3.87876607	5.70	5.70	5.70	4.40
x_1	1.161	1.126	1.126	1.16362943	1.30	1.30	1.30	1.30
x_2	1.700	1.500	1.500	1.95820171	1.90	1.90	1.90	2.50
C_6 [eV Å^6]	20.9882	21.9856	21.9856		60.8	60.8	60.8	
C_8 [eV Å^8]	97.5692	97.5692	97.5692		473.0	473.0	473.0	
B_4 [eV Å^4]				2.85665922				17.91
B_6 [eV Å^6]				76.25465037				138.0
	Kr ⋯ Br[190]				Kr ⋯ I[191]			
	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic
ϵ [meV]	19.90	13.10	15.70	79.5	23.85	16.70	20.20	67.2
r_m [Å]	3.90	4.15	4.03	3.85	4.05	4.32	4.20	4.11
β_1	5.70	7.20	7.00	4.62	5.80	6.72	6.38	5.44
β_2	6.72	8.00	7.20	4.62	6.12	6.48	6.25	4.55
x_1	1.02	1.05	1.05	1.04	1.08	1.11	1.09	1.04
x_2	1.70	1.65	1.85	1.50	1.77	1.64	1.70	1.54
C_6 [eV Å^6]	86.6	92.7	89.6		141	141	141	
C_8 [eV Å^8]	740.0	801.0	771.0		908	908	908	
B_4 [eV Å^4]				17.91				20.2
B_6 [eV Å^6]				165.0				238
	Xe ⋯ F[192], anionic potential fitted to [188]				Xe ⋯ Br[190]			
	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic	$X\,^2\!\Sigma^+_{1/2}$	$I\,^2\Pi_{3/2}$	$II\,^2\Pi_{1/2}$	anionic
ϵ [meV]	145.66	6.93825	6.93825	426.13387586	31.53	25.52	25.52	126.92
r_m [Å]	2.293	3.80	3.80	2.44762175	3.82	4.00	4.00	3.81
β_1	8.5	7.5	7.5	5.24277817	4.35	6.42	6.42	3.50
β_2	6.8	6.0	6.0	3.69023462	7.41	7.00	7.00	5.30
x_1	1.102	1.116	1.116	1.1 (fixed)	1.01	1.03	1.03	1.03
x_2	1.950	1.500	1.500	1.6 (fixed)	2.00	1.60	1.60	1.60
C_6 [eV Å^6]	30.4849	32.5231	32.5231		129.0	133.0	133.0	
C_8 [eV Å^8]	162.182	162.182	162.182		1270.0	1320.0	1320.0	
B_4 [eV Å^4]				31.29777817				28.98
B_6 [eV Å^6]				0.00150266				271.0

Not all figures significant.

Table C.2: Xenon-chlorine binary interaction potential

XeCl neutral binary potential[193]

ESMSV (exponential–spline–Morse–spline–van der Waals) type

$$\frac{V_0(x=r/r_m)}{\epsilon} = \begin{cases} \text{for } 0 < x \leqslant x_1: & A\,\exp\left[-\alpha\,(x-1)\right] \\ \text{for } x_1 < x < x_2: & \exp\left\{a_1 + (x-x_1)\left[a_2 + (x-x_2)\left\{a_3 + (x-x_1)a_4\right\}\right]\right\} \\ \text{for } x_2 \leqslant x \leqslant x_3: & \exp\left[-2\beta\,(x-1)\right] - 2\exp\left[-\beta\,(x-1)\right] \\ \text{for } x_3 < x < x_4: & b_1 + (x-x_3)\left\{b_2 + (x-x_4)\left[b3 + (x-x_3)b_4\right]\right\} \\ \text{for } x_4 \leqslant x: & -C_0/\left(\epsilon\,R^6\,x^6\right) \end{cases}$$

Values of a_i and b_i are determined such that the function and its first derivative matches the neighboring functions.

ϵ [meV]	r_m [Å]	β	C_0 [eV Å^6]	A
20.287	4.06	6.12	118.427	0.362
α	x_1	x_2	x_3	x_4
14.50	0.6600	0.800	1.1100	1.5100

$$V_2(r) = -A_2\exp\left(-\alpha_2\,r - \beta_2\,r^2\right) + C_2\,r^{-6}$$

A_2 [eV]	α_2 [Å^{-1}]	β_2 [Å^{-2}]	C_2 [eV Å^6]
4918.85	2.4977	0.12	21.317

Table C.3: Xenon-chloride binary interaction potential

XeCl anion binary potential[194]

MMSV type

$$\frac{V(x=r/r_m)}{\epsilon} = \begin{cases} \text{for } 0 < x \leqslant 1: & \left(\exp\left[\beta_1(1-x)\right]-1\right)^2 - 1 \\ \text{for } 1 < x \leqslant x_1: & M_2(x) = \left(\exp\left[\beta_2(1-x)\right]-1\right)^2 - 1 \\ \text{for } x_1 < x < x_2: & S(x)\,M_2(x) + \left[1-S(x)\right]W(x) \\ & \text{with } S(x) = \frac{1}{2}\left(\cos\frac{\pi\,(x-x_1)}{x_2-x_1}+1\right) \\ \text{for } x_2 \leqslant x: & X(x) = -\frac{B_4}{\epsilon\,r_m^4}x^4 - \frac{B_6}{\epsilon\,r_m^6}x^6 \end{cases}$$

ϵ [meV]	r_m [Å]	β_1	β_2	x_1	x_2	B_4 [eV Å^4]	B_6 [eV Å^6]
145.82	3.57	4.90	2.30	1.05	1.80	28.98	239.54

Table C.4: Noble gas–noble gas binary interaction potentials

Noble gas potentials
HFD-B2 (Hartree–Fock–dispersion) type

$$\frac{V(x=r/r_m)}{\epsilon} = A^* \exp\left(-\alpha^* x + \beta^* x^2\right) - F(x) \sum_{i=0}^{2} \frac{c_{2i+6}}{x^{2i+6}}$$

$$\text{with } F(x) = \begin{cases} \exp\left(-\left({}^{D}/_{x} - 1\right)^2\right) \text{ for } x < D \\ F(x) = 1 \text{ otherwise} \end{cases}$$

	Ar $\cdots$ Ar[195]	Kr $\cdots$ Kr[196]	Xe $\cdots$ Xe[197]
$A^*/10^5$	2.26210716	0.697353915	0.544087277
α^*	10.77874743	8.38802216	7.52958289
c_6^*	1.10785136	1.06136003	1.00555220
c_8^*	0.56072459	0.56845577	0.58359858
c_{10}^*	0.34602794	0.42605480	0.47378306
β^*	−1.8122004	−2.79611543	−3.3390428
β	−0.128422	−0.1738	−0.17520
D	1.36	1.2080	1.114
ϵ [meV]	12.34208816	17.34669012	24.36981603
r_m [Å]	3.7565	4.011	4.3656
σ [Å]	3.3527	3.5709	3.8924

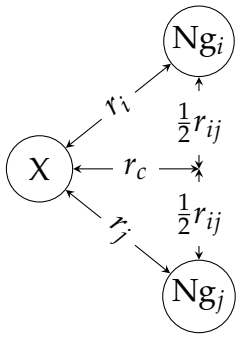
Not all figures significant.

Table C.5: Parameters for the exchange contribution to the potential energy

Exchange potential[3]

$$V_{\mathrm{exc}}(r_i, r_j, r_c, r_{ij}) = \sum_{i<j} \left[\exp\left(\beta^2 r_{ij}^2/2\right) - 1\right]^{-1} \cdot [\mathrm{erf}\left(\beta r_i\right)/r_i + \mathrm{erf}\left(\beta r_j\right)/r_j - 2\,\mathrm{erf}\left(\beta r_c\right)/r_c]$$

in atomic units



	Ar	Kr	Xe
β [Å^{-1}]	0.927[11]	0.84[198]	0.765[10]
Cut off if $r_{ij} > \ldots$	6.50 Å[3]	6.94 Å	7.55 Å

Original cut-off value for Ar from [3], other values scaled according to equilibrium distances.

Table C.6: Particle properties used for the induction nonadditivity, triple dipole, and dispersion contributions to the potential energy

Particle parameters

	m [Da][199]	α_d [a_0^3]	$C = \frac{1}{2}\alpha_q$ [a_0^5] *	$C_6\,B/\alpha$ [$e\,a_0^8$]	N	Δ [cm^{-1}]
Ar	39.948	11.15[200]	26.39[200]	808.1[11,201,202]	5.90[11]	
Kr	83.798	16.85[200]	48.375[200]	2674 [190,195,203]	6.309[190]	
Xe	131.293	27.17[200]	101.4[200]	8954 [194,203]	7.253[10]	
F^-	18.9984032	16.65[204]	71.285[204]		4.449[205]	
F	18.9984032	3.49[206]			4.080[207]	404.[187]
Cl^-[11]	35.453	36.997[204]	199.85[204]		5.9	
Cl[11]	35.453	14.63			4.2	882.4
Br^-[3]	79.904	49.05[204]	294.1[204]		6.70	
Br[3]	79.904	20.6			6.2	3685
I^-[10]	126.90447	69.34[204]	495.0[204]		7.253	
I[10]	126.90447	33.1			6.5	7603.15[3]

* As in Yourshaw *et al.*'s original work[3], Buckingham's convention is used.

Appendix D

Computer Programs

D.1 Data processing

A graphical program, largely based on the Qt –[208] and Qwt libraries[209], was used for processing the pump probe data recorded with the apparatus described in section 1.2. The program also made use of the libraries muparser[210], GSL[166], and libbz2[211]. The files `src/lmfit/lmmin.h` and `src/lmfit/lmmin.c` are – with minuscule changes – taken from the lmfit library[130].

An exemplary screenshot of the program is shown in Figure D.1. At the bottom, the "Log" window is visible, in which data from log files can be viewed and managed. The "Data" window in the left half of the picture allows for the organization and processing of individual spectra. It further includes numerous tools for manipulating the data, including a "drag-and-drop" actions to be performed directly with the plotted data. Finally, the "Meta" window allowed for combining various data sets, e.g. to generate kinetic traces as shown in the screenshot.

Heavy emphasis was put on making as many actions as possible feasible through "drag-and-drop" techniques. A further design goal was an unlimited and saveable history function, allowing the user to even undo actions taken in a previous session. Encapsulation of data objects and their extension strictly at their end were adhered to in order to allow for a maximum of inherent backward, as well as some forward compatibility in the file format.

It has to be noted, however, that the program still requires some improvement and is in a developmental stage. In the following, first some of the Qt-related helper files shall be laid out:

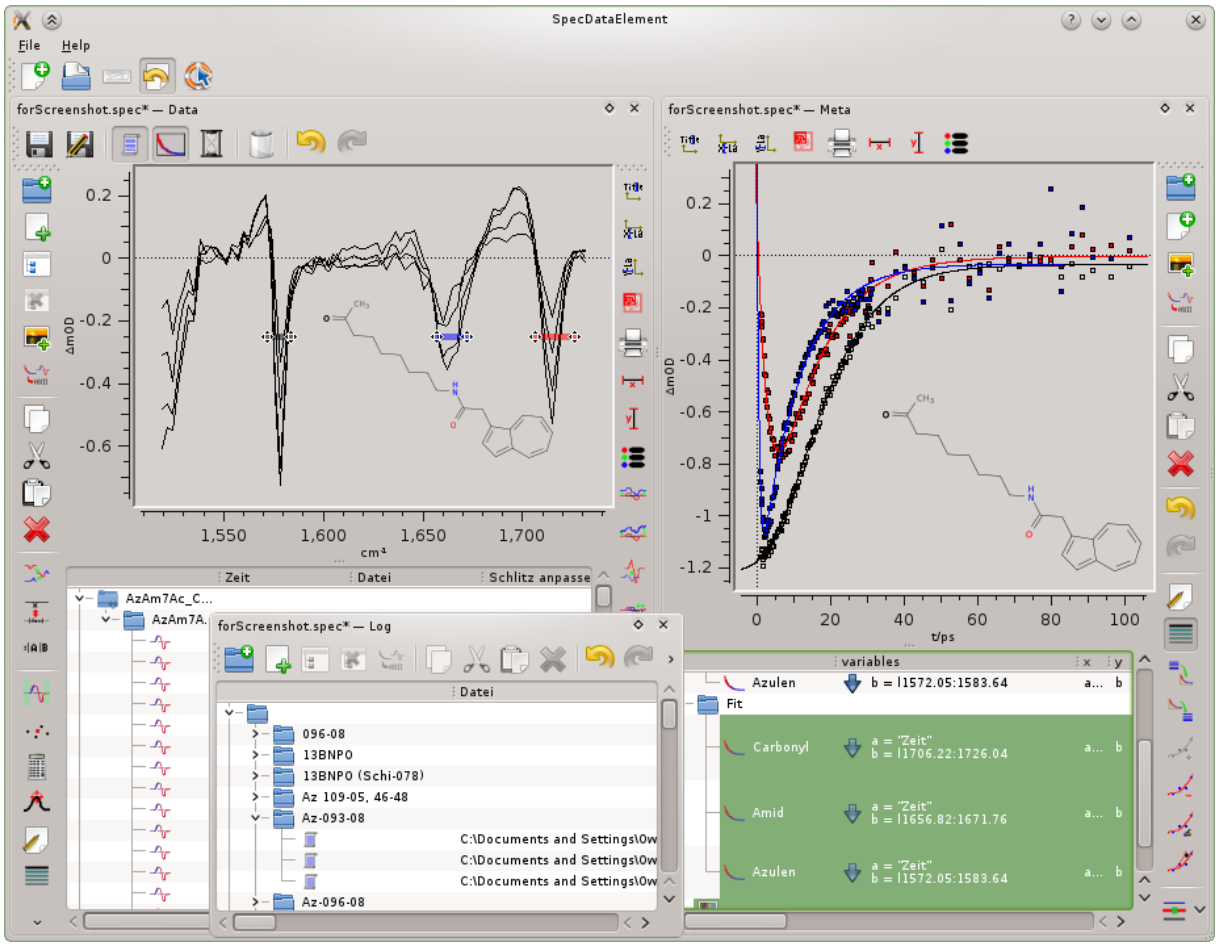


Figure D.1: Screenshot of the data processing program.

src/src.pro

```
include(prefix.pri)

SOURCES += main.cpp \
    cutbyintensitydialog.cpp \
    specappwindow.cpp \
    specplotwidget.cpp \
    utility-functions.cpp \
    kinetic/speckineticwidget.cpp \
    log/speclogentryitem.cpp \
    log/speclogmessage.cpp \
    model/exportdialog.cpp \
    model/exportlayoutitem.cpp \
    model/exportformatitem.cpp \
    model/specdelegate.cpp \
    model/specview.cpp \
    model/specdescriptor.cpp \
    model/specmodel.cpp \
    model/specmodelitem.cpp \
    model/specfolderitem.cpp \
    plot/canvaspicker.cpp \
    plot/speczoomer.cpp \
    plot/specplot.cpp \
    plot/specrange.cpp \
    plot/speccanvasitem.cpp \
    spectral/specdataitem.cpp \
    spectral/specdatapoint.cpp \
```

```
    spectral/specdataview.cpp \
    actionlib/specactionlibrary.cpp \
    actionlib/commands/specundocommand.cpp \
    actionlib/actions/specundoaction.cpp \
    actionlib/actions/specdeleteaction.cpp \
    actionlib/commands/specdeletecommand.cpp \
    actionlib/actions/specaddfolderaction.cpp \
    actionlib/commands/specaddfoldercommand.cpp \
    model/specgenealogy.cpp \
    actionlib/commands/specmanageitemscommand.cpp \
    actionlib/commands/specmovecommand.cpp \
    actionlib/actions/specimportspecaction.cpp \
    model/specviewstate.cpp \
    log/speclogmodel.cpp \
    log/speclogview.cpp \
    actionlib/actions/speccopyaction.cpp \
    actionlib/actions/specpasteaction.cpp \
    actionlib/actions/speccutaction.cpp \
    plot/specspectrumplot.cpp \
    textEditor/specsimpletextedit.cpp \
    actionlib/actions/changeplotstyleaction.cpp \
    actionlib/commands/specstylecommand.cpp \
    actionlib/actions/spectreeaction.cpp \
    actionlib/commands/specmulticommand.cpp \
    actionlib/actions/specmergeaction.cpp \
    actionlib/actions/specremovedataaction.cpp \
    actionlib/actions/specaveragedataaction.cpp \
    actionlib/commands/specexchangedatacommand.cpp \
    model/specsvgitem.cpp \
    actionlib/actions/specaddsvgitem.cpp \
    actionlib/actions/specprintplotaction.cpp \
    actionlib/commands/specresizesvgcommand.cpp \
    kinetic/specmetaitem.cpp \
    kinetic/specmetamodel.cpp \
    kinetic/specmetaview.cpp \
    actionlib/actions/specnewmetaitemaction.cpp \
    actionlib/commands/specmanageconnectionscommand.cpp \
    actionlib/commands/specaddconnectionscommand.cpp \
    actionlib/commands/specdeleteconnectionscommand.cpp \
    actionlib/actions/specaddconnectionsaction.cpp \
    model/specmimeconverter.cpp \
    log/speclogtodataconverter.cpp \
    kinetic/specmetaparser.cpp \
    kinetic/specmetavariable.cpp \
    kinetic/specintegralvariable.cpp \
    kinetic/specxvariable.cpp \
    kinetic/specdescriptorvariable.cpp \
    kinetic/specyvariable.cpp \
    kinetic/specminvariable.cpp \
    kinetic/specmaxvariable.cpp \
    kinetic/specmetadelegate.cpp \
    kinetic/specmetarange.cpp \
    log/speclogwidget.cpp \
    actionlib/commands/speceditdescriptorcommand.cpp \
    plot/specplotstyle.cpp \
    specstreamable.cpp \
    actionlib/commands/specmetarangecommand.cpp \
    model/specgenericmimeconverter.cpp \
    model/spectextmimeconverter.cpp \
    spectral/dataitemproperties.cpp \
    kinetic/metaitemproperties.cpp \
    model/svgitemproperties.cpp \
    model/specsvgunitbutton.cpp \
    specsplitter.cpp \
    actionlib/commands/specplotlabelcommand.cpp \
    actionlib/actions/speclabelaction.cpp \
    specprofiler.cpp \
    actionlib/actions/genericexportaction.cpp \
    actionlib/actions/specitemaction.cpp \
    actionlib/actions/specrequiresitemaction.cpp \
    model/specmimetextexporter.cpp \
    lmfit/lmmin.c \
    kinetic/specfitcurve.cpp \
    actionlib/commands/specexchangefitcurvecommand.cpp \
    actionlib/actions/specitempropertiesaction.cpp \
```

```
    actionlib/actions/specaddfitaction.cpp \
    actionlib/actions/specconductfitaction.cpp \
    actionlib/actions/specremovefitaction.cpp \
    actionlib/commands/spectogglefitstylecommand.cpp \
    actionlib/actions/spectogglefitstyleaction.cpp \
    actionlib/speccommandgenerator.cpp \
    actionlib/actions/specselectconnectedaction.cpp \
    actionlib/specworkerthread.cpp \
    actionlib/commands/specdescriptorflagscommand.cpp \
    actionlib/actions/specsetmultilineaction.cpp \
    actionlib/commands/specdeletedescriptorcommand.cpp \
    actionlib/commands/specrenamedescriptorcommand.cpp \
    actionlib/actions/specdescriptoreditaction.cpp \
    utilities/bzipiodevice.cpp \
    model/specmimefileimporter.cpp \
    asciiexporter.cpp \
    kinetic/specminposvariable.cpp \
    kinetic/specmaxposvariable.cpp \
    actionlib/actions/spectiltmatrixaction.cpp \
    actionlib/actions/specexchangedescriptorxdialog.cpp \
    spectral/specpefile.cpp \
    actionlib/actions/specspectrumcalculatoraction.cpp \
    actionlib/actions/specformulavalidator.cpp \
    actionlib/actions/spectrumcalculatordialog.cpp \
    specdockwidget.cpp \
    actionlib/actions/specflattentreeaction.cpp \
    actionlib/actions/specnormalizeaction.cpp \
    actionlib/actions/specmergedialog.cpp \
    actionlib/actions/specconnectionsaction.cpp \
    model/textedit.cpp \
    actionlib/commands/specsingleitemcommand.cpp \
    actionlib/commands/specmultipleitemcommand.cpp \
    specshortcutdialog.cpp \
    actionlib/commands/specexchangefiltercommand.cpp \
    model/specdatapointfilter.cpp \
    actionlib/actions/specfitaction.cpp \
    plot/specfiltergenerator.cpp \
    model/specdescriptorcomparisoncriterion.cpp

HEADERS += cutbyintensitydialog.h \
    specappwindow.h \
    specplotwidget.h \
    names.h \
    utility-functions.h \
    kinetic/speckineticwidget.h \
    log/speclogentryitem.h \
    log/speclogmessage.h \
    model/exportdialog.h \
    model/exportlayoutitem.h \
    model/exportformatitem.h \
    model/specdelegate.h \
    model/specview.h \
    model/specdescriptor.h \
    model/specmodel.h \
    model/specmodelitem.h \
    model/specfolderitem.h \
    plot/canvaspicker.h \
    plot/speczoomer.h \
    plot/specplot.h \
    plot/specrange.h \
    plot/speccanvasitem.h \
    spectral/specdataitem.h \
    spectral/specdatapoint.h \
    spectral/specdataview.h \
    actionlib/specactionlibrary.h \
    actionlib/commands/specundocommand.h \
    actionlib/actions/specundoaction.h \
    actionlib/actions/specdeleteaction.h \
    actionlib/commands/specdeletecommand.h \
    actionlib/actions/specaddfolderaction.h \
    actionlib/commands/specaddfoldercommand.h \
    model/specgenealogy.h \
    actionlib/commands/specmanageitemscommand.h \
    actionlib/commands/specmovecommand.h \
    actionlib/actions/specimportspecaction.h \
```

```
    model/specviewstate.h \
    log/speclogmodel.h \
    log/speclogview.h \
    actionlib/actions/speccopyaction.h \
    actionlib/actions/specpasteaction.h \
    actionlib/actions/speccutaction.h \
    plot/specspectrumplot.h \
    textEditor/specsimpletextedit.h \
    actionlib/actions/changeplotstyleaction.h \
    actionlib/commands/specstylecommand.h \
    actionlib/actions/spectreeaction.h \
    actionlib/commands/specmulticommand.h \
    actionlib/actions/specmergeaction.h \
    actionlib/actions/specremovedataaction.h \
    actionlib/actions/specaveragedataaction.h \
    actionlib/commands/specexchangedatacommand.h \
    model/specsvgitem.h \
    actionlib/actions/specaddsvgitem.h \
    actionlib/actions/specprintplotaction.h \
    actionlib/commands/specresizesvgcommand.h \
    kinetic/specmetaitem.h \
    kinetic/specmetamodel.h \
    kinetic/specmetaview.h \
    actionlib/actions/specnewmetaitemaction.h \
    actionlib/commands/specmanageconnectionscommand.h \
    actionlib/commands/specaddconnectionscommand.h \
    actionlib/commands/specdeleteconnectionscommand.h \
    actionlib/actions/specaddconnectionsaction.h \
    model/specmimeconverter.h \
    log/speclogtodataconverter.h \
    kinetic/specmetaparser.h \
    kinetic/specmetavariable.h \
    kinetic/specintegralvariable.h \
    kinetic/specxvariable.h \
    kinetic/specdescriptorvariable.h \
    kinetic/specyvariable.h \
    kinetic/specminvariable.h \
    kinetic/specmaxvariable.h \
    kinetic/specmetadelegate.h \
    kinetic/specmetarange.h \
    log/speclogwidget.h \
    actionlib/commands/speceditdescriptorcommand.h \
    plot/specplotstyle.h \
    specstreamable.h \
    actionlib/commands/specmetarangecommand.h \
    model/specgenericmimeconverter.h \
    model/spectextmimeconverter.h \
    spectral/dataitemproperties.h \
    kinetic/metaitemproperties.h \
    model/svgitemproperties.h \
    model/specsvgunitbutton.h \
    specsplitter.h \
    actionlib/commands/specplotlabelcommand.h \
    actionlib/actions/speclabelaction.h \
    specprofiler.h \
    actionlib/actions/genericexportaction.h \
    actionlib/actions/specitemaction.h \
    actionlib/actions/specrequiresitemaction.h \
    model/specmimetextexporter.h \
    lmfit/lmmin.h \
    kinetic/specfitcurve.h \
    actionlib/commands/specexchangefitcurvecommand.h \
    actionlib/actions/specitempropertiesaction.h \
    actionlib/actions/specaddfitaction.h \
    actionlib/actions/specconductfitaction.h \
    actionlib/actions/specremovefitaction.h \
    actionlib/commands/spectogglefitstylecommand.h \
    actionlib/actions/spectogglefitstyleaction.h \
    actionlib/speccommandgenerator.h \
    actionlib/actions/specselectconnectedaction.h \
    actionlib/specworkerthread.h \
    actionlib/commands/specdescriptorflagscommand.h \
    actionlib/actions/specsetmultilineaction.h \
    actionlib/commands/specdeletedescriptorcommand.h \
    actionlib/commands/specrenamedescriptorcommand.h \
```

```
    actionlib/actions/specdescriptoreditaction.h \
    utilities/bzipiodevice.h \
    model/specmimefileimporter.h \
    asciiexporter.h \
    kinetic/specminposvariable.h \
    kinetic/specmaxposvariable.h \
    actionlib/actions/spectiltmatrixaction.h \
    actionlib/actions/specexchangedescriptorxdialog.h \
    spectral/specpefile.h \
    actionlib/actions/specspectrumcalculatoraction.h \
    actionlib/actions/specformulavalidator.h \
    actionlib/actions/spectrumcalculatordialog.h \
    specdockwidget.h \
    actionlib/actions/specflattentreeaction.h \
    actionlib/actions/specnormalizeaction.h \
    actionlib/actions/specmergedialog.h \
    actionlib/actions/specconnectionsaction.h \
    model/textedit.h \
    actionlib/commands/specsingleitemcommand.h \
    actionlib/commands/specmultipleitemcommand.h \
    specshortcutdialog.h \
    actionlib/commands/specexchangefiltercommand.h \
    model/specdatapointfilter.h \
    actionlib/actions/specfitaction.h \
    plot/specfiltergenerator.h \
    model/specdescriptorcomparisoncriterion.h

INCLUDEPATH += kinetic \
        log \
        model \
        plot \
        spectral \
        actionlib \
        actionlib/actions \
        actionlib/commands \
        lmfit \
        utilities

TEMPLATE = app
CONFIG += warn_on \
    thread \
    qt

unix {
INCLUDEPATH += . \
        /home/hendrik/Programme/qwt-6.1.0/include
LIBS += -L/home/hendrik/Programme/qwt-6.1.0/lib \
        -lqwt \
        -lmuparser \
        -lbz2 \
        -lgsl \
        -lgslcblas
}
win32 {
INCLUDEPATH += . \
    C:/builds/Qwt-6.0.1/qwt \
    C:/builds/muparser_v2_2_2/include
LIBS += -LC:/builds/Qwt-6.0.1/lib \
    -lqwt \
    C:/builds/muparser_v2_2_2/lib/libmuparser.a
DEFINES += WIN32BUILD \
    QT_DLL \
    QWT_DLL
DEPENDPATH += C:/Users/Hendrik/Downloads/muparser_v2_2_2/lib
RESOURCES += oxygen.qrc
}
CONFIG += qwt

TARGET = data_element

RESOURCES += icons/application.qrc

QT += svg
```

```
#QMAKE_CXXFLAGS += -std=c++0x

FORMS += \
    spectral/dataitemproperties.ui \
    kinetic/metaitemproperties.ui \
    model/svgitemproperties.ui \
    actionlib/actions/specexchangedescriptorxdialog.ui \
    actionlib/actions/spectrumcalculatordialog.ui \
    actionlib/actions/specnormalizeactiondialog.ui \
    actionlib/actions/specmergedialog.ui \
    actionlib/actions/averagedialog.ui \
    specshortcutdialog.ui

OTHER_FILES += icons/oxygen/index.theme \
    icons/oxygen/geticons.py

CODECFORTR = UTF-8

DEFINES += DOUBLEDEVIATIONCORRECTION=1
DEFINES += NUMBEROFFRACTIONBITSINDOUBLE=52
#DEFINES += DEBUGCOMMANDREADER
```

src/prefix.pri

```
unix {
        OBJECTS_DIR = ../build/objects
        MOC_DIR = ../build/moc
        DESTDIR = ../build/target
        DLLDESTDIR = ../build/target
        UI_DIR = ../build/uic
        UI_HEADERS_DIR = ../build/uic-headers
        UI_SOURCES_DIR = ../build/uic-src
        LIBS += -L../build/target
        RCC_DIR = ../build/resources
        build_pass:CONFIG(release) {
                TARGET = data_element-release
        }
}
win32 {
        OBJECTS_DIR = ../winbuild/objects
        MOC_DIR = ../winbuild/moc
        DESTDIR = ../winbuild/target
        DLLDESTDIR = ../winbuild/target
        UI_DIR = ../winbuild/uic
        UI_HEADERS_DIR = ../winbuild/uic-headers
        UI_SOURCES_DIR = ../winbuild/uic-src
        LIBS += -L../winbuild/target
        RCC_DIR = ../winbuild/resources
        build_pass:CONFIG(release) {
                TARGET = data_element-release.exe
        }
}



DEFINES += FILECHECKRANDOMNUMBER=469222828 FILECHECKCOMPRESSNUMBER=469333828

DEFINES += GITSHA1HASH=$$sprintf("%1", $$system(git rev-parse HEAD))
```

src/oxygen.qrc

```
<!DOCTYPE RCC><RCC version="1.0">
<qresource prefix="icons/oxygen">
        <file alias="index.theme">icons/oxygen/index.theme</file>

        <file alias="16x16/document-revert.png">icons/oxygen/document-revert.png</file>
        <file alias="16x16/document-new.png">icons/oxygen/document-new.png</file>
        <file alias="16x16/edit-copy.png">icons/oxygen/edit-copy.png</file>
        <file alias="16x16/edit-undo.png">icons/oxygen/edit-undo.png</file>
        <file alias="16x16/edit-delete.png">icons/oxygen/edit-delete.png</file>
        <file alias="16x16/document-open.png">icons/oxygen/document-open.png</file>
        <file alias="16x16/document-properties.png">icons/oxygen/document-properties.png</file>
```

```
        <file alias="16x16/document-print.png">icons/oxygen/document-print.png</file>
        <file alias="16x16/dialog-warning.png">icons/oxygen/dialog-warning.png</file>
        <file alias="16x16/go-down.png">icons/oxygen/go-down.png</file>
        <file alias="16x16/image-x-generic.png">icons/oxygen/image-x-generic.png</file>
        <file alias="16x16/accessories-calculator.png">icons/oxygen/accessories-calculator.png</file>
        <file alias="16x16/go-up.png">icons/oxygen/go-up.png</file>
        <file alias="16x16/insert-image.png">icons/oxygen/insert-image.png</file>
        <file alias="16x16/document-save.png">icons/oxygen/document-save.png</file>
        <file alias="16x16/edit-cut.png">icons/oxygen/edit-cut.png</file>
        <file alias="16x16/format-text-italic.png">icons/oxygen/format-text-italic.png</file>
        <file alias="16x16/format-text-bold.png">icons/oxygen/format-text-bold.png</file>
        <file alias="16x16/user-trash.png">icons/oxygen/user-trash.png</file>
        <file alias="16x16/help-contextual.png">icons/oxygen/help-contextual.png</file>
        <file alias="16x16/folder-new.png">icons/oxygen/folder-new.png</file>
        <file alias="16x16/edit-paste.png">icons/oxygen/edit-paste.png</file>
        <file alias="16x16/document-save-as.png">icons/oxygen/document-save-as.png</file>
        <file alias="16x16/edit-redo.png">icons/oxygen/edit-redo.png</file>
</qresource>
</RCC>
```

src/icons/application.qrc

```
<RCC>
    <qresource prefix="/">
        <file>addZeroRange.png</file>
        <file>changetitle.png</file>
        <file>changexlabel.png</file>
        <file>changeylabel.png</file>
        <file>data.png</file>
        <file>deleteZeroRange.png</file>
        <file>export.png</file>
        <file>fromKinetic.png</file>
        <file>individualZero.png</file>
        <file>kinetic.png</file>
        <file>kineticwindow.png</file>
        <file>lineWidth.png</file>
        <file>merge.png</file>
        <file>offline.png</file>
        <file>offset.png</file>
        <file>scale.png</file>
        <file>toKinetic.png</file>
        <file>zeroCorrection.png</file>
        <file>color.png</file>
        <file>cbi.png</file>
        <file>ave.png</file>
        <file>mave.png</file>
        <file>multiminus.png</file>
        <file>logs.png</file>
        <file>fixXAxis.png</file>
        <file>fixYAxis.png</file>
        <file>xshift.png</file>
        <file>lineStyle.png</file>
        <file>resizeImage.png</file>
        <file>moveIndicator.png</file>
        <file>legend.png</file>
        <file>exchangex.png</file>
        <file>normalize.png</file>
        <file>addFit.png</file>
        <file>doFit.png</file>
        <file>editDescriptors.png</file>
        <file>styleFit.png</file>
        <file>removeFit.png</file>
        <file>zeroRange.png</file>
        <file>multiline.png</file>
    </qresource>
</RCC>
```

The icons used for the program were:

 src/icons/addFit.png
 src/icons/addZeroRange.png
 src/icons/ave.png
 src/icons/cbi.png
 src/icons/changetitle.png
 src/icons/changexlabel.png
 src/icons/changeylabel.png
 src/icons/color.png
 src/icons/data.png
 src/icons/deleteZeroRange.png
 src/icons/doFit.png
 src/icons/editDescriptors.png
 src/icons/exchangex.png
 src/icons/export.png
 src/icons/fixXAxis.png
 src/icons/fixYAxis.png
 src/icons/fromKinetic.png
 src/icons/individualZero.png
 src/icons/kinetic.png
 src/icons/kineticwindow.png
 src/icons/legend.png
 src/icons/lineStyle.png
 src/icons/lineWidth.png
 src/icons/logs.png
 src/icons/mave.png
 src/icons/merge.png
 src/icons/moveIndicator.png
 src/icons/multiline.png
 src/icons/multiminus.png
 src/icons/normalize.png
 src/icons/offline.png
 src/icons/offset.png
 src/icons/removeFit.png
 src/icons/resizeImage.png
 src/icons/scale.png
 src/icons/styleFit.png
 src/icons/toKinetic.png
 src/icons/xshift.png
 src/icons/zeroCorrection.png
 src/icons/zeroRange.png

Additional icons can be taken from the built-in scheme of the system:

src/icons/oxygen/index.theme

```
[Icon Theme]
Name=Oxygen
Comment=Oxygen Theme
Inherits=default
Directories=16x16

[16x16]
Size=16
```

src/icons/oxygen/geticons.py

```
# -*- coding: utf-8 -*-
"""
Created on Sat Sep  1 11:41:26 2012

@author: hendrik
"""

import subprocess
from subprocess import Popen as Popen
from subprocess import STDOUT as STDOUT
from subprocess import PIPE as PIPE
```

```
12
def executeCommand(command):
    p = Popen(command, shell=True, stdin=PIPE, stdout=PIPE, stderr=STDOUT, close_fds=True)
    return p.stdout.read()

17 findIconNamesCmd = "grep␣fromTheme␣`find␣../../␣-name␣\"*cpp\"`"
iconNames = [i[i.find("\"",i.find("fromTheme"))+1:i.find("\"",i.find("\"",i.find("fromTheme"))+1)]+"►
    .png" for i in executeCommand(findIconNamesCmd).split("\n") ][:-1]

print "<!DOCTYPE␣RCC><RCC␣version=\"1.0\">\n<qresource␣prefix=\"icons/oxygen\">\n\t<file␣alias=\"►
    index.theme\">icons/oxygen/index.theme</file>\n"

22 for iconName in set(iconNames):
    findIconCmd = "find␣/usr/share/icons/oxygen/16x16/␣-name␣" + iconName
    fileName = executeCommand(findIconCmd)
    executeCommand("cp␣" + fileName[:-1] + "␣./")
    print "\t<file␣alias=\"16x16/" + iconName + "\">icons/oxygen/" +\
27  iconName + "</file>"

print "</qresource>\n</RCC>"
```

The core C++ and user interface files of the program are:

src/actionlib/actions/averagedialog.ui

```
1  <?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>averageDialog</class>
 <widget class="QDialog" name="averageDialog">
  <property name="geometry">
6  <rect>
    <x>0</x>
    <y>0</y>
    <width>249</width>
    <height>214</height>
11  </rect>
  </property>
  <property name="windowTitle">
   <string>Average</string>
  </property>
16  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="QWidget" name="widget" native="true">
     <layout class="QVBoxLayout" name="verticalLayout_3">
      <item>
21     <widget class="QRadioButton" name="byPointNumber">
        <property name="text">
         <string>Average by point index</string>
        </property>
        <property name="checked">
26       <bool>true</bool>
        </property>
       </widget>
      </item>
      <item>
31     <widget class="QSpinBox" name="number">
        <property name="minimum">
         <number>2</number>
        </property>
       </widget>
36    </item>
      <item>
       <widget class="QCheckBox" name="running">
        <property name="text">
         <string>Running average</string>
41       </property>
       </widget>
      </item>
      <item>
       <widget class="QLabel" name="runningLabel">
46       <property name="enabled">
         <bool>false</bool>
        </property>
```

```
        <property name="text">
         <string>Points will be taken symmetrically from both sides (if available)</string>
        </property>
        <property name="wordWrap">
         <bool>true</bool>
        </property>
        <property name="indent">
         <number>30</number>
        </property>
       </widget>
      </item>
      <item>
       <widget class="Line" name="line">
        <property name="orientation">
         <enum>Qt::Horizontal</enum>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QRadioButton" name="byTolerance">
        <property name="text">
         <string>Average by x value</string>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QDoubleSpinBox" name="toleranceEdit">
        <property name="enabled">
         <bool>false</bool>
        </property>
       </widget>
      </item>
      <item>
       <widget class="QCheckBox" name="rightToLeft">
        <property name="enabled">
         <bool>false</bool>
        </property>
        <property name="text">
         <string>Invert direction (start from larger x values)</string>
        </property>
       </widget>
      </item>
     </layout>
     <zorder>byPointNumber</zorder>
     <zorder>byTolerance</zorder>
     <zorder>running</zorder>
     <zorder>number</zorder>
     <zorder>rightToLeft</zorder>
     <zorder>runningLabel</zorder>
     <zorder>toleranceEdit</zorder>
     <zorder>line</zorder>
    </widget>
   </item>
   <item>
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="enabled">
      <bool>true</bool>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections>
  <connection>
   <sender>running</sender>
   <signal>toggled(bool)</signal>
   <receiver>runningLabel</receiver>
   <slot>setEnabled(bool)</slot>
   <hints>
    <hint type="sourcelabel">
     <x>60</x>
```

```
      <y>62</y>
     </hint>
     <hint type="destinationlabel">
      <x>89</x>
      <y>90</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>byTolerance</sender>
    <signal>toggled(bool)</signal>
    <receiver>toleranceEdit</receiver>
    <slot>setEnabled(bool)</slot>
    <hints>
     <hint type="sourcelabel">
      <x>49</x>
      <y>123</y>
     </hint>
     <hint type="destinationlabel">
      <x>59</x>
      <y>154</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>byTolerance</sender>
    <signal>toggled(bool)</signal>
    <receiver>rightToLeft</receiver>
    <slot>setEnabled(bool)</slot>
    <hints>
     <hint type="sourcelabel">
      <x>33</x>
      <y>118</y>
     </hint>
     <hint type="destinationlabel">
      <x>43</x>
      <y>161</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>byPointNumber</sender>
    <signal>toggled(bool)</signal>
    <receiver>number</receiver>
    <slot>setEnabled(bool)</slot>
    <hints>
     <hint type="sourcelabel">
      <x>66</x>
      <y>11</y>
     </hint>
     <hint type="destinationlabel">
      <x>108</x>
      <y>40</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>byPointNumber</sender>
    <signal>toggled(bool)</signal>
    <receiver>running</receiver>
    <slot>setEnabled(bool)</slot>
    <hints>
     <hint type="sourcelabel">
      <x>136</x>
      <y>17</y>
     </hint>
     <hint type="destinationlabel">
      <x>136</x>
      <y>63</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>byPointNumber</sender>
    <signal>toggled(bool)</signal>
```

```
   <receiver>runningLabel</receiver>
   <slot>setEnabled(bool)</slot>
   <hints>
    <hint type="sourcelabel">
     <x>194</x>
     <y>14</y>
    </hint>
    <hint type="destinationlabel">
     <x>192</x>
     <y>89</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>averageDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>236</x>
     <y>204</y>
    </hint>
    <hint type="destinationlabel">
     <x>240</x>
     <y>208</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>averageDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>222</x>
     <y>198</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>208</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/actionlib/actions/changeplotstyleaction.h

```
#ifndef CHANGEPLOTSTYLEACTION_H
#define CHANGEPLOTSTYLEACTION_H
#include "specrequiresitemaction.h"
#include "specdataview.h"
#include "specstylecommand.h"
class specGenealogy ;
class changePlotStyleAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit changePlotStyleAction(QObject* parent = 0);
        const std::type_info& possibleParent() { return typeid(specView) ; }
private:
        specUndoCommand* generateUndoCommand() ;
        QMenu* lineColorMenu,
               *symbolMenu,
               *symbolInnerColorMenu,
               *symbolOuterColorMenu,
               *symbolSizeMenu,
               *lineWidthMenu,
               *penStyleMenu ;
        QVector<Qt::GlobalColor> colors ;
        QVector<double> sizes ;
        QVector<QAction*> symbolActions,
```

```
                lineWidthActions,
                lineColorActions,
                symbolInnerColorActions,
                symbolOuterColorActions,
                symbolSizeActions,
                penStyleActions ;
        QColor getColor(int index) ;
        double getSize(int index, bool& ok) ;
        QObject* currentTrigger ;
        QAction* currentAction ;
private slots:
        void actionTriggered(QAction*) ;
};
#endif
```

src/actionlib/actions/changeplotstyleaction.cpp

```
#include "changeplotstyleaction.h"
#include <QDialog>
#include <QGridLayout>
#include <QVBoxLayout>
#include "specgenealogy.h"
#include <QColorDialog>
#include <qwt_symbol.h>
#include <QPainter>
#include "specstylecommand.h"
#include <QInputDialog>
#include "specdataitem.h"
changePlotStyleAction::changePlotStyleAction(QObject* parent) :
        specRequiresItemAction(parent),
        currentTrigger(0)
{
        setIcon(QIcon(":/lineStyle.png"));
        setToolTip(tr("Plot Style")) ;
        setText(tr("Set plot style")) ;
        setWhatsThis(tr("Plot Style -- In this menu you will find commands for modifying the ►
            appearance of plots (colors, line widths, symbols).\nClick the \"triple circle\" icon at►
            the top of a list for defining a color/size/width yourself."));
        QWidget* p = qobject_cast<QWidget*> (parent) ;
        setMenu(new QMenu(p)) ;
        lineColorMenu = new QMenu("Line color", menu()) ;
        symbolMenu = new QMenu("Symbol", menu()) ;
        symbolInnerColorMenu = new QMenu("Symbol Inner Color", menu()) ;
        symbolOuterColorMenu = new QMenu("Symbol Outer Color", menu()) ;
        symbolSizeMenu = new QMenu("Symbol Size", menu()) ;
        lineWidthMenu = new QMenu("Line Width", menu()) ;
        penStyleMenu = new QMenu("Line Style", menu()) ;
        menu()->addMenu(lineColorMenu) ;
        menu()->addMenu(lineWidthMenu) ;
        menu()->addMenu(symbolMenu) ;
        menu()->addMenu(symbolInnerColorMenu) ;
        menu()->addMenu(symbolOuterColorMenu) ;
        menu()->addMenu(symbolSizeMenu) ;
        menu()->addMenu(penStyleMenu) ;
        colors << Qt::black << Qt::red << Qt::green << Qt::blue << Qt::yellow << Qt::magenta << Qt::►
            cyan << Qt::darkYellow ;
        sizes << 1 << 2 << 2.5 << 3 << 5 << 7 << 10 ;
        QSize iconSize(16, 16) ;
        QPixmap icon(iconSize) ;
        QPainter painter(&icon) ;
        QwtSymbol symbol ;
        symbol.setSize(iconSize) ;
        icon.fill(Qt::white);
        int l = icon.size().width() / 5 ;
        painter.drawEllipse(l, l, l, l);
        painter.drawEllipse(2 * l, 2 * l, l, l);
        painter.drawEllipse(3 * l, 3 * l, l, l);
        QPixmap customPixmap(icon) ;
        QStringList symbolNames ;
        symbolNames << "none"
                    << "circle"
                    << "square"
                    << "diamond"
                    << "triangle"
```

```
                        << "inverted triangle"
                        << "up triangle"
                        << "left triangle"
                        << "right triangle"
                        << "cross"
                        << "x cross"
                        << "horizontal line"
                        << "vertical line"
                        << "star 1"
                        << "star 2"
                        << "hexagon" ;
        for(int i = -1 ; i < QwtSymbol::Hexagon + 1 ; ++i)
        {
                symbol.setStyle(QwtSymbol::Style(i)) ;
                symbol.setSize(iconSize/1.2);
                icon.fill(Qt::white);
                symbol.drawSymbol(&painter, QPointF(iconSize.width() / 2., iconSize.height() / 2.)) ;
                QAction* newSymbolAction = new QAction(icon, "Symbol: " + symbolNames[i + 1], this) ;
                symbolActions << newSymbolAction ;
        }
        lineColorActions << new QAction(customPixmap, "", this) ;
        symbolInnerColorActions << new QAction(customPixmap, "", this) ;
        symbolOuterColorActions << new QAction(customPixmap, "", this) ;
        for(QVector<Qt::GlobalColor>::iterator i = colors.begin() ; i != colors.end() ; ++i)
        {
                icon.fill(*i) ;
                lineColorActions << new QAction(icon, "", this) ;
                symbolInnerColorActions << new QAction(icon, "", this) ;
                symbolOuterColorActions << new QAction(icon, "", this) ;
        }
        lineWidthActions << new QAction(customPixmap, "Plot line width: custom...", this) ;
        symbolSizeActions << new QAction(customPixmap, "Plot mark size: custom...", this) ;
        for(QVector<double>::iterator i = sizes.begin() ; i != sizes.end() ; ++i)
        {
                lineWidthActions << new QAction("Plot line width " + QString::number(*i), this) ;
                symbolSizeActions << new QAction("Plot mark size " + QString::number(*i), this) ;
        }
        penStyleActions << new QAction(tr("no line"), this) <<
                        new QAction(tr("solid"), this) <<
                        new QAction(tr("dashed"), this) <<
                        new QAction(tr("dotted"), this) <<
                        new QAction(tr("dash-dotted"), this) <<
                        new QAction(tr("dash-dot-dotted"), this) ;
        foreach(QAction* lineAction, penStyleActions)
                lineAction->setText(tr("Plot line style: ") + lineAction->text());
        symbolMenu->addActions(symbolActions.toList());
        lineColorMenu->addActions(lineColorActions.toList()) ;
        lineWidthMenu->addActions(lineWidthActions.toList()) ;
        symbolInnerColorMenu->addActions(symbolInnerColorActions.toList()) ;
        symbolOuterColorMenu->addActions(symbolOuterColorActions.toList());
        symbolSizeMenu->addActions(symbolSizeActions.toList());
        penStyleMenu->addActions(penStyleActions.toList()) ;
        connect(symbolMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(QAction*))) ;
        connect(lineColorMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(QAction*))) ;
        connect(lineWidthMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(QAction*))) ;
        connect(symbolInnerColorMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(►
            QAction*))) ;
        connect(symbolOuterColorMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(►
            QAction*))) ;
        connect(symbolSizeMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(QAction*))) ;
        connect(penStyleMenu, SIGNAL(triggered(QAction*)), this, SLOT(actionTriggered(QAction*))) ;
}
QColor changePlotStyleAction::getColor(int index)
{
        if(!index) return QColorDialog::getColor() ;
        return colors[index - 1] ;
}
double changePlotStyleAction::getSize(int index, bool& ok)
{
        if(!index) return QInputDialog::getDouble(0, "Value?", "Value:", 1, 0, 100, 1, &ok) ;
        return sizes[index - 1] ;
}
void changePlotStyleAction::actionTriggered(QAction* action)
{
        currentTrigger = sender() ;
```

```
        currentAction = action ;
        execute() ;
}
specUndoCommand* changePlotStyleAction::generateUndoCommand()
{
        specStyleCommand* newCommand = 0 ;
        if(!currentTrigger || !currentAction) return 0 ;
        specDataItem item ;
        if(currentTrigger == lineColorMenu)
        {
                QColor color = getColor(lineColorActions.indexOf(currentAction)) ;
                if(!color.isValid()) return 0;
                newCommand = generateStyleCommand(specStreamable::penColorCommandId) ;
                item.setPenColor(color) ;
                newCommand->setText(tr("Change␣line␣color"));
        }
        if(currentTrigger == symbolInnerColorMenu)
        {
                QColor color = getColor(symbolInnerColorActions.indexOf(currentAction)) ;
                if(!color.isValid()) return 0;
                newCommand = generateStyleCommand(specStreamable::symbolBrushColorCommandId) ;
                item.setSymbolBrushColor(color) ;
                newCommand->setText(tr("Change␣symbol␣inner␣color"));
        }
        if(currentTrigger == symbolOuterColorMenu)
        {
                QColor color = getColor(symbolOuterColorActions.indexOf(currentAction)) ;
                if(!color.isValid()) return 0;
                newCommand = generateStyleCommand(specStreamable::symbolPenColorCommandId) ;
                item.setSymbolPenColor(color) ;
                newCommand->setText(tr("Change␣symbol␣outer␣color"));
        }
        bool ok = true;
        if(currentTrigger == lineWidthMenu)
        {
                double value = getSize(lineWidthActions.indexOf(currentAction), ok) ;
                if(!ok) return 0;
                newCommand = generateStyleCommand(specStreamable::lineWidthCommandId) ;
                item.setLineWidth(value) ;
                newCommand->setText(tr("Change␣line␣width"));
        }
        if(currentTrigger == symbolSizeMenu)
        {
                double value = getSize(symbolSizeActions.indexOf(currentAction), ok) ;
                if(!ok) return 0;
                newCommand = generateStyleCommand(specStreamable::symbolSizeCommandId) ;
                item.setSymbolSize(value) ;
                newCommand->setText(tr("Change␣symbol␣size"));
        }
        if(currentTrigger == symbolMenu)
        {
                newCommand = generateStyleCommand(specStreamable::symbolStyleCommandId) ;
                int symbolStyle = symbolActions.indexOf(currentAction) - 1 ;
                item.setSymbolStyle(symbolStyle);
                newCommand->setText(tr("Change␣symbol␣type"));
        }
        if(currentTrigger == penStyleMenu)
        {
                newCommand = generateStyleCommand(specStreamable::penStyleCommandId) ;
                item.setPenStyle(penStyleActions.indexOf(currentAction));
                newCommand->setText(tr("Change␣line␣style"));
        }
        if(!newCommand) return 0;
        newCommand->obtainStyle(&item) ;
        newCommand->setParentObject(model) ;
        newCommand->setItems(pointers) ;
        return newCommand ;
}
```

src/actionlib/actions/genericexportaction.h

```
#ifndef GENERICEXPORTACTION_H
#define GENERICEXPORTACTION_H
#include "specrequiresitemaction.h"
```

```
class exportDialog ;
class genericExportAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit genericExportAction(QObject* parent = 0);
        ~genericExportAction() ;
private:
        specUndoCommand* generateUndoCommand() ;
        exportDialog* exportFormat ;
};
#endif
```

src/actionlib/actions/genericexportaction.cpp

```
#include "genericexportaction.h"
#include "exportdialog.h"
#include <QFile>
#include <QMessageBox>
#include <QFileDialog>
genericExportAction::genericExportAction(QObject* parent) :
        specRequiresItemAction(parent),
        exportFormat(new exportDialog(0))
{
        setIcon(QIcon(":/export.png")) ;
        setToolTip(tr("Export␣to␣ASCII")) ;
        setWhatsThis(tr("Export␣selected␣items␣to␣ASCII␣file.")) ;
        setText(tr("Export␣ASCII..."));
        setShortcut(tr("e"));
}
genericExportAction::~genericExportAction()
{
        delete exportFormat;
}
specUndoCommand* genericExportAction::generateUndoCommand()
{
        QFile exportFile(QFileDialog::getSaveFileName(0, "File␣name", "", "ASCII␣files␣(*.asc)")) ;
        if(exportFile.fileName() == "") return 0 ;
        exportFormat->setDataTypes(model->dataTypes()) ;
        exportFormat->setDescriptors(model->descriptors()) ;
        if(exportFormat->exec() != QDialog::Accepted) return 0 ;
        if(!exportFile.open(QIODevice::WriteOnly | QIODevice::Text))
        {
                QMessageBox::critical(0, tr("Cannot␣open␣file"), tr("Cannot␣open␣file\n") + ►
                    exportFile.fileName(), QMessageBox::Ok) ;
                return 0 ;
        }
        QTextStream out(&exportFile) ;
        QList<QPair<bool, QString> > headerFormat = exportFormat->headerFormat() ;
        QList<QPair<spec::value, QString> > dataFormat = exportFormat->dataFormat() ;
        if(selection.isEmpty())
                selection << QModelIndex() ;
        QList<specModelItem*> pointers = model->pointerList(selection) ;
        foreach(specModelItem * item, pointers)
        item->exportData(headerFormat, dataFormat, out) ;
        exportFile.close() ;
        return 0 ;
}
```

src/actionlib/actions/specaddconnectionsaction.h

```
#ifndef SPECADDCONNECTIONSACTION_H
#define SPECADDCONNECTIONSACTION_H
#include "specconnectionsaction.h"
class specAddConnectionsAction : public specConnectionsAction
{
        Q_OBJECT
public:
        explicit specAddConnectionsAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
        bool dataViewRequirement, metaViewRequirement, changing ;
```

```
        void checkRequirements() ;
13  private slots:
        void changeByDataView() ;
        void changedByMetaView() ;
    };
    #endif
```

src/actionlib/actions/specaddconnectionsaction.cpp

```
    #include "specaddconnectionsaction.h"
    #include "specmetaview.h"
 3  #include "specplotwidget.h"
    #include "specaddconnectionscommand.h"
    #include "specmulticommand.h"
    #include "specmetaitem.h"
    specAddConnectionsAction::specAddConnectionsAction(QObject* parent) :
 8      specConnectionsAction(parent),
        dataViewRequirement(false),
        metaViewRequirement(false),
        changing(false)
    {
13      this->setIcon(QIcon(":/toKinetic.png")) ;
        setToolTip(tr("Connect␣current␣to␣selected."));
        setWhatsThis(tr("Connects␣the␣current␣item␣in␣the␣meta␣dock␣window␣with␣the␣selected␣items␣►
            in␣the␣data␣and␣meta␣dock␣window.\nThis␣is␣the␣essential␣action␣for␣enabling␣a␣meta␣item►
            ␣to␣obtain␣data␣for␣processing␣from␣other␣items."));
        setText(tr("Connect␣to␣selected")) ;
        setShortcut(tr("Ctrl+Alt+c"));
18      specMetaView* view = dynamic_cast<specMetaView*>(parent) ;
        connect(this, SIGNAL(changed()), this, SLOT(changedByMetaView())) ;
        if(view && view->getDataView() && view->getDataView()->selectionModel())
                connect(view->getDataView()->selectionModel(), SIGNAL(selectionChanged(►
                    QItemSelection, QItemSelection)),
                        this, SLOT(changeByDataView())) ;
23  }
    void specAddConnectionsAction::changedByMetaView()
    {
        if(changing) return ;
        metaViewRequirement = isEnabled() ;
28      checkRequirements() ;
    }
    void specAddConnectionsAction::changeByDataView()
    {
        if (!requirements()) return ;
33      if(!dataView) return ;
        dataViewRequirement = dataView->model() && !dataView->getSelection().isEmpty() ;
        checkRequirements() ;
    }
    void specAddConnectionsAction::checkRequirements()
38  {
        changing = true ;
        setEnabled(metaViewRequirement && dataViewRequirement);
        changing = false ;
    }
43  specUndoCommand* specAddConnectionsAction::generateUndoCommand()
    {
        specMultiCommand* command = new specMultiCommand ;
        command->setParentObject(model);
        QList<specModelItem*> originalSelection = dataView->model()->pointerList(dataView->►
            getSelection()) ;
48      foreach(specModelItem * item, pointers)
        (new specAddConnectionsCommand(command))
        ->setItems(dynamic_cast<specMetaItem*>(item), originalSelection) ;
        command->setText(tr("Connected␣")
                         + QString::number(pointers.size())
53                       + tr("␣meta␣items␣to␣")
                         + QString::number(originalSelection.size())
                         + tr("␣data␣items")) ;
        return command ;
    }
```

src/actionlib/actions/specaddfitaction.h

```
#ifndef SPECADDFITACTION_H
#define SPECADDFITACTION_H
#include "specfitaction.h"
class specAddFitAction : public specFitAction
{
        Q_OBJECT
public:
        explicit specAddFitAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
        bool negateFitRequirement() const ;
};
#endif
```

src/actionlib/actions/specaddfitaction.cpp

```
#include "specaddfitaction.h"
#include "specmetaitem.h"
#include "specfitcurve.h"
#include "specexchangefitcurvecommand.h"
#include "specmulticommand.h"
specAddFitAction::specAddFitAction(QObject* parent) :
        specFitAction(parent)
{
        setIcon(QIcon(":/addFit.png")) ;
        setToolTip(tr("Add␣fit")) ;
        setWhatsThis(tr("Adds␣a␣fit␣to␣the␣current␣item.")) ;
        setText(tr("Add␣fit")) ;
        setShortcut(tr("Ctrl+F"));
}
specUndoCommand* specAddFitAction::generateUndoCommand()
{
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setText(tr("Add␣fit␣curve")) ;
        parentCommand->setParentObject(model) ;
        foreach(specModelItem * pointer, pointers)
        {
                specMetaItem* item = (specMetaItem*) pointer ;
                if(item->getFitCurve()) continue ;
                (new specExchangeFitCurveCommand(parentCommand))
                ->setup(pointer, new specFitCurve) ;
        }
        if(!parentCommand->childCount())
        {
                delete parentCommand ;
                return 0 ;
        }
        return parentCommand ;
}
bool specAddFitAction::negateFitRequirement() const
{
        return true ;
}
```

src/actionlib/actions/specaddfolderaction.h

```
#ifndef SPECADDFOLDERACTION_H
#define SPECADDFOLDERACTION_H
#include "specitemaction.h"
class specAddFolderAction : public specItemAction
{
        Q_OBJECT
public:
        explicit specAddFolderAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specaddfolderaction.cpp

```
#include "specaddfolderaction.h"
#include "specdataview.h"
#include "specgenealogy.h"
#include "specaddfoldercommand.h"
specAddFolderAction::specAddFolderAction(QObject* parent)
        : specItemAction(parent)
{
        this->setIcon(QIcon::fromTheme("folder-new"));
        setToolTip(tr("Create␣folder")) ;
        setWhatsThis(tr("Creates␣a␣new␣folder␣that␣may␣then␣be␣filled␣with␣items."));
        setText(tr("Add␣folder"));
        setShortcut(tr("f"));
}
specUndoCommand* specAddFolderAction::generateUndoCommand()
{
        specFolderItem* newItem = new specFolderItem ;
        if(! model->insertItems(QList<specModelItem*>() << newItem, insertionIndex, insertionRow))
        {
                delete newItem ;
                return 0 ;
        }
        specAddFolderCommand* command = new specAddFolderCommand ;
        command->setParentObject(model) ;
        command->setItem(newItem) ;
        command->setText(tr("Add␣folder")) ;
        return command ;
}
```

src/actionlib/actions/specaddsvgitem.h

```
#ifndef SPECADDSVGITEMACTION_H
#define SPECADDSVGITEMACTION_H
#include "specitemaction.h"
class specAddSVGItemAction : public specItemAction
{
        Q_OBJECT
public:
        explicit specAddSVGItemAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specaddsvgitem.cpp

```
#include "specaddsvgitem.h"
#include "specdataview.h"
#include "specgenealogy.h"
#include "specaddfoldercommand.h"
#include "specsvgitem.h"
#include <QFile>
#include <QFileDialog>
#include <QByteArray>
#include <QtSvg>
specAddSVGItemAction::specAddSVGItemAction(QObject* parent) :
        specItemAction(parent)
{
        setIcon(QIcon::fromTheme("insert-image")) ;
        setToolTip(tr("Add␣SVG␣image")) ;
        setWhatsThis(tr("Insert␣an␣image␣to␣be␣displayed␣as␣an␣annotation␣to␣the␣plot␣canvas.␣␣►
             Currently,␣SVG␣images␣are␣supported.")) ;
        setText(tr("Add␣SVG␣picture␣item...")) ;
        setShortcut(tr("i"));
}
specUndoCommand* specAddSVGItemAction::generateUndoCommand()
{
        QFile input(QFileDialog::getOpenFileName(parentWidget(),
                    "Select␣SVG␣file",
                    "",
                    "SVG␣images␣(*.svg)")) ;
```

```
        if(!(input.exists() && input.open(QIODevice::ReadOnly)))
                return 0;
        QByteArray fileContent(input.readAll()) ;
        specSVGItem* newItem = new specSVGItem() ;
        newItem->setImage(fileContent) ;
        int row = 0 ;
        if(!currentItem->isFolder())
        {
                row = currentIndex.row() + 1 ;
                currentIndex = currentIndex.parent() ;
        }
        if(! model->insertItems(QList<specModelItem*>() << newItem, currentIndex, row))
        {
                delete newItem ;
                return 0;
        }
        specAddFolderCommand* command = new specAddFolderCommand ;
        command->setParentObject(model) ;
        command->setItem(newItem) ;
        command->setText(tr("Add␣SVG␣item")) ;
        return command ;
}
```

src/actionlib/actions/specaveragedataaction.h

```
#ifndef SPECAVERAGEDATAACTION_H
#define SPECAVERAGEDATAACTION_H
#include "specrequiresitemaction.h"
class QDialog ;
namespace Ui
{
        class averageDialog ;
}
class specAverageDataAction : public specRequiresDataItemAction
{
        Q_OBJECT
public:
        explicit specAverageDataAction(QObject* parent = 0);
        ~specAverageDataAction() ;
        const std::type_info& possibleParent() ;
protected:
        specUndoCommand* generateUndoCommand() ;
private:
        QDialog* dialog ;
        Ui::averageDialog* ui ;
private slots:
        void runningToggled() ;
};
#endif
```

src/actionlib/actions/specaveragedataaction.cpp

```
#include "specaveragedataaction.h"
#include <QDialog>
#include <QSpinBox>
#include <QCheckBox>
#include <QVBoxLayout>
#include <QDialogButtonBox>
#include <QLabel>
#include <QHBoxLayout>
#include "specmulticommand.h"
#include "specexchangedatacommand.h"
#include <cmath>
#include "specdataview.h"
#include "utility-functions.h"
#include <QRadioButton>
#include "ui_averagedialog.h"
specAverageDataAction::specAverageDataAction(QObject* parent) :
        specRequiresDataItemAction(parent),
        dialog(new QDialog),
        ui(new Ui::averageDialog)
{
```

```
        setIcon(QIcon(":/ave.png")) ;
        setToolTip(tr("Average␣Data")) ;
        setWhatsThis(tr("Smooth␣data␣by␣averaging.␣␣You␣can␣choose␣between␣plainly␣averaging␣any␣►
            number␣of␣data␣points␣or␣calculating␣a␣moving␣average."));
        setText(tr("Average␣data...")) ;
        setShortcut(tr("Ctrl+Shift+a"));
        ui->setupUi(dialog);
        connect(ui->running, SIGNAL(toggled(bool)), this, SLOT(runningToggled())) ;
        ui->toleranceEdit->setMaximum(INFINITY);
}
specAverageDataAction::~specAverageDataAction()
{
        delete ui ;
        delete dialog ;
}
const std::type_info& specAverageDataAction::possibleParent()
{
        return typeid(specDataView) ;
}
class tolerantComparison
{
private:
        double tolerance ;
public:
        tolerantComparison(double d) : tolerance(d) {}
        bool operator()(const specDataPoint& a, const specDataPoint& b)
        {
                return fabs(a.nu - b.nu) <= tolerance ;
        }
};
void specAverageDataAction::runningToggled()
{
        ui->number->setMinimum(ui->running->isChecked() ? 1 : 2);
}
specUndoCommand* specAverageDataAction::generateUndoCommand()
{
        if(!dialog->exec())
                return 0 ;
        int numAverages = ui->number->value() ;
        if(ui->byPointNumber->isChecked() && numAverages == 1 && !ui->running->isChecked()) return 0 ;
        specMultiCommand* groupCommand = new specMultiCommand ;
        groupCommand->setParentObject(model) ;
        groupCommand->setMergeable(false) ;
        if(ui->byPointNumber->isChecked())
        {
                foreach(QModelIndex index, selection)
                {
                        specDataItem* item = dynamic_cast<specDataItem*>(view->model()->itemPointer(►
                            index)) ;
                        if(!item) continue ;
                        QVector<specDataPoint> oldData(item->allData()), newData ;
                        item->applyCorrection(oldData);
                        if(ui->running->isChecked())
                        {
                                for(int i = 0 ; i < oldData.size() ; ++i)
                                {
                                        int limit = qMin(i + numAverages + 1, oldData.size()) ;
                                        specDataPoint dataPoint ;
                                        for(int j = qMax(0, i - numAverages) ; j < limit ; ++j)
                                                dataPoint += oldData[j] ;
                                        dataPoint /= limit - qMax(0, i - numAverages) ;
                                        newData << dataPoint ;
                                }
                        }
                        else
                        {
                                int avs = std::ceil(double(oldData.size()) / numAverages) ;
                                for(int i = 0 ; i < avs ; ++i)
                                {
                                        specDataPoint dataPoint ;
                                        int limit = qMin(oldData.size(), (i + 1) * numAverages) ;
                                        for(int j = i * numAverages ; j < limit ; ++ j)
                                                dataPoint += oldData[j] ;
                                        dataPoint /= limit - i * numAverages ;
                                        newData << dataPoint ;
```

```
                                }
                        }
                        item->reverseCorrection(newData);
                        specExchangeDataCommand* command = new specExchangeDataCommand(groupCommand) ;
                        command->setItem(item, newData);
                }
                groupCommand->setText(tr("Average data (")
                                + QString::number(ui->number->value())
                                + tr(" points")
                                + (ui->running->isChecked() ? tr(", running)") : tr(")"))) ;
        }
        else if(ui->byTolerance->isChecked())
        {
                groupCommand->setText(tr("Averaged data with a tolerance of ")
                                + ui->toleranceEdit->text()
                                + (ui->rightToLeft->isChecked() ? tr(", inverse x direction") ►
                                    : tr("."))) ;
                tolerantComparison comp(ui->toleranceEdit->text().toDouble()) ;
                foreach(QModelIndex index, selection)
                {
                        specDataItem* item = dynamic_cast<specDataItem*>(model->itemPointer(index)) ;
                        if(!item) continue ;
                        specExchangeDataCommand* command = new specExchangeDataCommand(groupCommand) ;
                        command->setParentObject(model) ;
                        QVector<specDataPoint> newData, oldData(item->allData()) ;
                        if(ui->rightToLeft->isChecked())
                        {
                                newData.reserve(oldData.size());
                                std::reverse_copy(oldData.begin(), oldData.end(), std::back_inserter►
                                    (newData)) ;
                                oldData.clear();
                                oldData.swap(newData) ;
                        }
                        averageToNew(oldData.begin(), oldData.end(), comp, std::back_inserter(►
                            newData)) ;
                        command->setItem(item, newData) ;
                }
        }
        return groupCommand ;
}
```

src/actionlib/actions/specconductfitaction.h

```
#ifndef SPECCONDUCTFITACTION_H
#define SPECCONDUCTFITACTION_H
#include "specfitaction.h"
class specConductFitAction : public specFitAction
{
        Q_OBJECT
public:
        explicit specConductFitAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specconductfitaction.cpp

```
#include "specconductfitaction.h"
#include "specmetaitem.h"
#include "specfitcurve.h"
#include "speceditdescriptorcommand.h"
#include "specmulticommand.h"
specConductFitAction::specConductFitAction(QObject* parent) :
        specFitAction(parent)
{
        setIcon(QIcon(":/doFit.png")) ;
        setToolTip(tr("Conduct fit")) ;
        setWhatsThis(tr("Conducts fits for the selected meta items.")) ;
        setText(tr("Fit")) ;
        setShortcut(tr("Ctrl+Alt+f"));
}
```

```
specUndoCommand* specConductFitAction::generateUndoCommand()
{
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setText(tr("Conduct␣fitting")) ;
        parentCommand->setParentObject(model);
        foreach(specModelItem * modelItem, pointers)
        {
                specMetaItem* item = dynamic_cast<specMetaItem*>(modelItem) ;
                if(!item) continue;
                if(!(item->getFitCurve())) continue ;
                QString oldDescriptor = item->descriptor(tr("Fit␣variables"), true) ;
                int oldActiveLine = item->activeLine(tr("Fit␣variables")) ;
                item->conductFit();
                QString newDescriptor = item->descriptor(tr("Fit␣variables"), true) ;
                specEditDescriptorCommand* editCommand = new specEditDescriptorCommand(parentCommand) ;
                editCommand->setItem(item, tr("Fit␣variables"), newDescriptor, item->activeLine(tr("Fit␣variables"))) ;
                item->changeDescriptor(tr("Fit␣variables"), oldDescriptor) ;
                item->setActiveLine(tr("Fit␣variables"), oldActiveLine) ;
        }
        return parentCommand ;
}
```

src/actionlib/actions/specconnectionsaction.h

```
#ifndef SPECCONNECTIONSACTION_H
#define SPECCONNECTIONSACTION_H
#include "specrequiresitemaction.h"
class specMetaView ;
class specConnectionsAction : public specRequiresMetaItemAction
{
        Q_OBJECT
private:
        bool specificRequirements() ;
protected:
        specView* dataView ;
        specMetaView* metaView ;
public:
        explicit specConnectionsAction(QObject* parent = 0);
        const std::type_info& possibleParent() ;
};
#endif
```

src/actionlib/actions/specconnectionsaction.cpp

```
#include "specconnectionsaction.h"
#include "specmetaview.h"
specConnectionsAction::specConnectionsAction(QObject* parent) :
        specRequiresMetaItemAction(parent),
        dataView(0),
        metaView(0)
{
}
bool specConnectionsAction::specificRequirements()
{
        return (metaView = qobject_cast<specMetaView*> (view))
               && (dataView = metaView->getDataView()) ;
}
const std::type_info& specConnectionsAction::possibleParent()
{
        return typeid(specMetaView) ;
}
```

src/actionlib/actions/speccopyaction.h

```
#ifndef SPECCOPYACTION_H
#define SPECCOPYACTION_H
#include "specrequiresitemaction.h"
class specCopyAction : public specRequiresItemAction
```

```
{
        Q_OBJECT
public:
        explicit specCopyAction(QObject* parent = 0);
        static void copyToClipboard(specModel*, const QModelIndexList& selection) ;
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/speccopyaction.cpp

```
#include "speccopyaction.h"
#include "specview.h"
#include <QApplication>
#include <QClipboard>
specCopyAction::specCopyAction(QObject* parent) :
        specRequiresItemAction(parent)
{
        this->setIcon(QIcon::fromTheme("edit-copy")) ;
        setToolTip(tr("Copy␣selected␣items")) ;
        setWhatsThis(tr("Copy␣items␣to␣clipboard.␣␣May␣be␣pasted␣in␣this␣program␣or␣in␣any␣other␣►
            that␣understands␣plain␣text.")) ;
        setText(tr("Copy")) ;
        setShortcut(tr("Ctrl+c"));
}
specUndoCommand* specCopyAction::generateUndoCommand()
{
        copyToClipboard(model, selection) ;
        return 0 ;
}
void specCopyAction::copyToClipboard(specModel* model, const QModelIndexList& selection)
{
        QApplication::clipboard()->setMimeData(model->mimeData(selection));
}
```

src/actionlib/actions/speccutaction.h

```
#ifndef SPECCUTACTION_H
#define SPECCUTACTION_H
#include "specdeleteaction.h"
class specCutAction : public specDeleteAction
{
        Q_OBJECT
public:
        explicit specCutAction(QObject* parent = 0);
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/speccutaction.cpp

```
#include "speccutaction.h"
#include <QApplication>
#include <QClipboard>
#include "speccopyaction.h"
specCutAction::specCutAction(QObject* parent) :
        specDeleteAction(parent)
{
        this->setIcon(QIcon::fromTheme("edit-cut")) ;
        setToolTip(tr("Cut␣selected␣items")) ;
        setWhatsThis(tr("Cut␣items␣to␣clipboard.␣␣Removes␣orginal␣items␣from␣this␣file.␣␣May␣be␣►
            pasted␣in␣this␣program␣or␣in␣any␣other␣that␣understands␣plain␣text.")) ;
        setText(tr("Cut")) ;
        setShortcut(QKeySequence("Ctrl+x"));
}
specUndoCommand* specCutAction::generateUndoCommand()
{
        specCopyAction::copyToClipboard(model, selection) ;
```

```
        return specDeleteAction::generateUndoCommand() ;
}
```

src/actionlib/actions/specdeleteaction.h

```
#ifndef SPECDELETEACTION_H
#define SPECDELETEACTION_H
#include "specrequiresitemaction.h"
class specDeleteAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit specDeleteAction(QObject* parent = 0);
        static specUndoCommand* command(specModel* model, QList<specModelItem*>& selection, ►
            specUndoCommand* parentsParent = 0) ;
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specdeleteaction.cpp

```
#include "specdeleteaction.h"
#include "specdeletecommand.h"
#include "specgenealogy.h"
#include "specmetamodel.h"
#include "specmulticommand.h"
#include "specdeleteconnectionscommand.h"
#include <QMap>
#include "speclogmodel.h"
#include "specmetaitem.h"
#include "specdataview.h"
specDeleteAction::specDeleteAction(QObject* parent) :
        specRequiresItemAction(parent)
{
        this->setIcon(QIcon::fromTheme("edit-delete"));
        setToolTip(tr("Delete")) ;
        setWhatsThis(tr("Deletes␣selected␣items.")) ;
        setText(tr("Delete")) ;
        setShortcut(Qt::Key_Delete);
}
specUndoCommand* specDeleteAction::command(specModel* model, QList<specModelItem*>& selection, ►
    specUndoCommand* parentsParent)
{
        specMultiCommand* parentCommand = 0 ;
        QString descriptionText = tr("Delete␣") + QString::number(selection.size()) + tr("␣items.") ;
        if(!qobject_cast<specLogModel*> (model))
        {
                parentCommand = new specMultiCommand(parentsParent) ;
                parentCommand->setParentObject(model) ;
                parentCommand->setText(descriptionText) ;
                parentCommand->setMergeable(false) ;
                selection = specModel::expandFolders(selection, true) ;
                QMap<specMetaItem*, QList<specModelItem*> > toDisconnect ;
                foreach(specModelItem * item, selection)
                {
                        specMetaItem* metaItem = dynamic_cast<specMetaItem*>(item) ;
                        if(metaItem)
                                toDisconnect[metaItem] << metaItem->serverList() ;
                        foreach(specMetaItem * metaItem, item->clientList())
                        toDisconnect[metaItem] << item ;
                }
                foreach(specMetaItem * metaItem, toDisconnect.keys())
                (new specDeleteConnectionsCommand(parentCommand))->
                setItems(metaItem, toDisconnect[metaItem]) ;
        }
        specDeleteCommand* command = new specDeleteCommand(parentCommand) ;
        command->setParentObject(model);
        command->setItems(selection) ;
        if(!parentCommand)
        {
                command->setText(descriptionText) ;
```

```
                return command ;
        }
        return parentCommand ;
}
specUndoCommand* specDeleteAction::generateUndoCommand()
{
        return command(model, pointers) ;
}
```

src/actionlib/actions/specdescriptoreditaction.h

```
#ifndef SPECDESCRIPTOREDITACTION_H
#define SPECDESCRIPTOREDITACTION_H
#include <QDialog>
#include <QValidator>
class QLineEdit ;
class QCheckBox ;
class stringListEntryWidget : public QWidget
{
        Q_OBJECT
private:
        QLineEdit* newValue ;
        QCheckBox* Active ;
private slots:
        void checkStateChanged(int) ;
public:
        explicit stringListEntryWidget(QString content, QWidget* parent = 0) ;
        void setAllWidgets(QList<stringListEntryWidget*> widgets) ;
        QString content() const ;
        QString oldContent() const ;
        bool active() const ;
signals:
        void contentChanged() ;
};
class stringListValidator : public QValidator
{
private:
        QStringList* forbiddenList ;
public:
        stringListValidator(QWidget* parent = 0) ;
        void fixup(QString&) const ;
        State validate(QString&, int&) const ;
        void setForbidden(const QStringList& fl) const ;
        ~stringListValidator() ;
};
class stringListEntryValidator : public stringListValidator
{
        Q_OBJECT
        QList<stringListEntryWidget*> allWidgets ;
public:
        stringListEntryValidator(QList<stringListEntryWidget*> allWidgets, stringListEntryWidget* ►
            parent = 0) ;
        State validate(QString&, int&) const ;
};
class stringListChangeDialog : public QDialog
{
        Q_OBJECT
        QList<stringListEntryWidget*> widgets ;
public:
        explicit stringListChangeDialog(QStringList strings, QWidget* parent = 0);
        QStringList inactive() const ;
        QMap<QString, QString> active() const ;
};
#include "specundoaction.h"
class specDescriptorEditAction : public specUndoAction
{
        Q_OBJECT
public:
        explicit specDescriptorEditAction(QObject* parent = 0) ;
private:
        void execute() ;
        const std::type_info& possibleParent() ;
};
#endif
```

src/actionlib/actions/specdescriptoreditaction.cpp

```
#include "specdescriptoreditaction.h"
#include <QHBoxLayout>
#include <QCheckBox>
#include <QLineEdit>
#include <QLabel>
#include <QVBoxLayout>
#include <QScrollArea>
#include <QDialogButtonBox>
#include "specmulticommand.h"
#include "specdeletedescriptorcommand.h"
#include "specrenamedescriptorcommand.h"
stringListEntryValidator::stringListEntryValidator(QList<stringListEntryWidget*> aW, ►
    stringListEntryWidget* parent)
        : stringListValidator(parent),
          allWidgets(aW)
{}
stringListValidator::stringListValidator(QWidget* parent)
        : QValidator(parent),
          forbiddenList(new QStringList)
{}
stringListValidator::~stringListValidator()
{
        delete forbiddenList ;
}
QValidator::State stringListValidator::validate(QString& s, int& pos) const
{
        Q_UNUSED(pos)
        return forbiddenList->contains(s) ? Intermediate : Acceptable ;
}
void stringListValidator::setForbidden(const QStringList& fl) const
{
        *forbiddenList = fl ;
}
QValidator::State stringListEntryValidator::validate(QString& s, int& pos) const
{
        Q_UNUSED(pos)
        stringListEntryWidget* parentPointer = qobject_cast<stringListEntryWidget*> (parent()) ;
        if(!parentPointer || !parentPointer->active()) return Acceptable ;
        QStringList forbiddenStrings ;
        forbiddenStrings << QString() ;
        foreach(stringListEntryWidget * widget, allWidgets)
        if(widget != parentPointer &&
                widget->active())
                forbiddenStrings << widget->content() ;
        setForbidden(forbiddenStrings);
        return stringListValidator::validate(s, pos) ;
}
void stringListValidator::fixup(QString& s) const
{
        int n = 0 ;
        int count = 0 ;
        QString original = s ;
        while(validate(s, n) != Acceptable)
                s = original + QString::number(count++) ;
}
stringListEntryWidget::stringListEntryWidget(QString content, QWidget* parent)
        : QWidget(parent)
{
        QHBoxLayout* layout = new QHBoxLayout(this) ;
        Active = new QCheckBox(content, this) ;
        Active->setChecked(true) ;
        newValue = new QLineEdit(content, this) ;
        layout->addWidget(Active) ;
        layout->addStretch();
        layout->addWidget(newValue) ;
        connect(Active, SIGNAL(stateChanged(int)), this, SLOT(checkStateChanged(int))) ;
        setSizePolicy(QSizePolicy::Fixed, QSizePolicy::Fixed);
}
bool stringListEntryWidget::active() const
{
```

```
        return Active->isChecked() ;
}
QString stringListEntryWidget::oldContent() const
{
        return Active->text() ;
}
QString stringListEntryWidget::content() const
{
        return newValue->text() ;
}
void stringListEntryWidget::setAllWidgets(QList<stringListEntryWidget*> widgets)
{
        disconnect(0, 0, this, SLOT(verify())) ;
        newValue->setValidator(new stringListEntryValidator(widgets, this)) ;
}
void stringListEntryWidget::checkStateChanged(int s)
{
        newValue->setEnabled(Qt::Checked == s);
        if(newValue->isEnabled())
        {
                QString content = newValue->text() ;
                if(newValue->validator())
                        newValue->validator()->fixup(content) ;
                if(newValue->text() != content)
                {
                        newValue->selectAll();
                        newValue->insert(content);
                }
        }
}
stringListChangeDialog::stringListChangeDialog(QStringList strings, QWidget* parent) :
        QDialog(parent)
{
        QScrollArea* scrollArea = new QScrollArea(this) ;
        QWidget* areaWidget = new QWidget(scrollArea) ;
        QVBoxLayout* layout = new QVBoxLayout(this), *scrollLayout = new QVBoxLayout(areaWidget) ;
        layout->addWidget(scrollArea) ;
        foreach(QString string, strings)
        {
                stringListEntryWidget* widget = new stringListEntryWidget(string, this) ;
                widgets << widget ;
                scrollLayout->addWidget(widget) ;
        }
        foreach(stringListEntryWidget * widget, widgets)
        widget->setAllWidgets(widgets);
        QDialogButtonBox* buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::►
             Cancel, Qt::Horizontal, this) ;
        scrollArea->setWidget(areaWidget);
        layout->addWidget(buttonBox) ;
        connect(buttonBox, SIGNAL(accepted()), this, SLOT(accept())) ;
        connect(buttonBox, SIGNAL(rejected()), this, SLOT(reject())) ;
}
QStringList stringListChangeDialog::inactive() const
{
        QStringList result ;
        foreach(stringListEntryWidget * widget, widgets)
        if(!widget->active())
                result << widget->content() ;
        return result ;
}
QMap<QString, QString> stringListChangeDialog::active() const
{
        QMap<QString, QString> map ;
        foreach(stringListEntryWidget * widget, widgets)
        if(widget->active())
                map[widget->oldContent()] = widget->content() ;
        return map ;
}
specDescriptorEditAction::specDescriptorEditAction(QObject* parent)
        : specUndoAction(parent)
{
        setIcon(QIcon(":/editDescriptors.png")) ;
        setToolTip(tr("Edit␣columns")) ;
        setWhatsThis(tr("Use␣this␣action␣to␣edit/delete␣column␣headers."));
        setText(tr("Edit␣columns")) ;
```

```
}
void specDescriptorEditAction::execute()
{
        specView* view = qobject_cast<specView*> (parent()) ;
        if(!view) return ;
        specModel* model = view->model() ;
        if(!model) return ;
        QStringList descriptors = model->descriptors() ;
        descriptors.removeAll("") ;
        stringListChangeDialog dialog(descriptors) ;
        dialog.exec() ;
        if(dialog.result() != QDialog::Accepted) return ;
        QStringList toDelete = dialog.inactive() ;
        QMap<QString, QString> toExchange = dialog.active() ;
        foreach(const QString & key, toExchange.keys())
        if(toExchange[key] == key)
                toExchange.remove(key) ;
        specMultiCommand* command = new specMultiCommand ;
        command->setText(tr("Modify␣column␣heads")) ;
        command->setParentObject(model);
        foreach(const QString & key, toDelete)
        {
                specDeleteDescriptorCommand* deleteCommand = new specDeleteDescriptorCommand(command►
                    , key) ;
                deleteCommand->setParentObject(model) ;
        }
        specRenameDescriptorCommand* renameCommand = new specRenameDescriptorCommand(command) ;
        renameCommand->setRenamingMap(toExchange) ;
        renameCommand->setParentObject(model) ;
        if(!library)
        {
                command->redo();
                delete command ;
        }
        else
                library->push(command) ;
}
const std::type_info& specDescriptorEditAction::possibleParent()
{
        return typeid(specView) ;
}
```

src/actionlib/actions/specexchangedescriptorxdialog.h

```
#ifndef SPECEXCHANGEDESCRIPTORXDIALOG_H
#define SPECEXCHANGEDESCRIPTORXDIALOG_H
#include <QDialog>
class stringListValidator ;
namespace Ui
{
        class specExchangeDescriptorXDialog;
}
class specExchangeDescriptorXDialog : public QDialog
{
        Q_OBJECT
public:
        explicit specExchangeDescriptorXDialog(QWidget* parent = 0);
        QString descriptorToConvert() const ;
        QString newDescriptor() const ;
        void setDescriptors(const QStringList&) ;
        ~specExchangeDescriptorXDialog();
private slots:
        void on_columnToConvert_currentIndexChanged(const QString& arg1) ;
private:
        Ui::specExchangeDescriptorXDialog* ui;
        QStringList entries ;
        stringListValidator* validator ;
};
#endif
```

src/actionlib/actions/specexchangedescriptorxdialog.cpp

```
#include "specexchangedescriptorxdialog.h"
#include "ui_specexchangedescriptorxdialog.h"
#include "specdescriptoreditaction.h"
specExchangeDescriptorXDialog::specExchangeDescriptorXDialog(QWidget* parent) :
        QDialog(parent),
        ui(new Ui::specExchangeDescriptorXDialog),
        validator(new stringListValidator(this))
{
        ui->setupUi(this);
        ui->xName->setValidator(validator);
        QString newLabel(tr("old x")) ;
        validator->fixup(newLabel) ;
        ui->xName->setText(newLabel);
}
specExchangeDescriptorXDialog::~specExchangeDescriptorXDialog()
{
        delete ui;
}
void specExchangeDescriptorXDialog::on_columnToConvert_currentIndexChanged(const QString& current)
{
        QStringList listContent(entries) ;
        listContent.removeOne(current) ;
        validator->setForbidden(listContent);
        ui->xName->setText(ui->xName->text());
}
void specExchangeDescriptorXDialog::setDescriptors(const QStringList& ds)
{
        QString current = ui->columnToConvert->currentText() ;
        ui->columnToConvert->clear();
        ui->columnToConvert->addItems(ds) ;
        if(ds.contains(current))
                ui->columnToConvert->setCurrentIndex(ds.indexOf(current));
}
QString specExchangeDescriptorXDialog::descriptorToConvert() const
{
        return ui->columnToConvert->currentText() ;
}
QString specExchangeDescriptorXDialog::newDescriptor() const
{
        return ui->xName->text() ;
}
```

src/actionlib/actions/specexchangedescriptorxdialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>specExchangeDescriptorXDialog</class>
 <widget class="QDialog" name="specExchangeDescriptorXDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>305</width>
    <height>86</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Preferred" vsizetype="Minimum">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>Choose columns</string>
  </property>
  <layout class="QFormLayout" name="formLayout">
   <item row="0" column="0">
    <widget class="QLabel" name="columnToConvertLabel">
     <property name="text">
      <string>Column to convert to x:</string>
     </property>
     <property name="buddy">
      <cstring>columnToConvert</cstring>
     </property>
```

```
    </widget>
   </item>
   <item row="0" column="1">
    <widget class="QComboBox" name="columnToConvert"/>
   </item>
   <item row="1" column="0">
    <widget class="QLabel" name="xNameLabel">
     <property name="text">
      <string>Name for old x:</string>
     </property>
     <property name="buddy">
      <cstring>xName</cstring>
     </property>
    </widget>
   </item>
   <item row="1" column="1">
    <widget class="QLineEdit" name="xName"/>
   </item>
   <item row="2" column="0" colspan="2">
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>specExchangeDescriptorXDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>252</x>
     <y>81</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>specExchangeDescriptorXDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>300</x>
     <y>81</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/actionlib/actions/specfitaction.h

```
#ifndef SPECFITACTION_H
#define SPECFITACTION_H
#include "specrequiresitemaction.h"
class specFitAction : public specRequiresMetaItemAction
{
```

```
        Q_OBJECT
protected:
        virtual bool negateFitRequirement() const ;
        bool specificRequirements() ;
public:
        explicit specFitAction(QObject *parent = 0);
};
#endif
```

src/actionlib/actions/specfitaction.cpp

```
#include "specfitaction.h"
#include "specmetaitem.h"
specFitAction::specFitAction(QObject *parent) :
        specRequiresMetaItemAction(parent)
{
}
bool specFitAction::negateFitRequirement() const
{
        return false ;
}
bool specFitAction::specificRequirements()
{
        assembleSelection();
        foreach(specModelItem* p, pointers)
        {
                specMetaItem *mi = dynamic_cast<specMetaItem*>(p) ;
                if (!p) continue ;
                bool hasFitCurve = mi->hasFitCurve() ;
                if ((hasFitCurve || negateFitRequirement())
                                && !(hasFitCurve && negateFitRequirement())) return true ;
        }
        return false ;
}
```

src/actionlib/actions/specflattentreeaction.h

```
#ifndef SPECFLATTENTREEACTION_H
#define SPECFLATTENTREEACTION_H
#include "specrequiresitemaction.h"
class specFlattenTreeAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit specFlattenTreeAction(QObject* parent = 0);
protected:
        specUndoCommand* generateUndoCommand() ;
        QList<specStreamable::type> requiredTypes() const ;
};
#endif
```

src/actionlib/actions/specflattentreeaction.cpp

```
#include "specflattentreeaction.h"
#include "specmulticommand.h"
#include "specmovecommand.h"
#include "specdeleteaction.h"
specFlattenTreeAction::specFlattenTreeAction(QObject* parent) :
        specRequiresItemAction(parent)
{
        QPixmap base(QIcon::fromTheme("view-list-tree").pixmap(32,32)) ;
        QPainter painter(&base) ;
        painter.drawPixmap(5,5,20,20,QIcon::fromTheme("edit-delete").pixmap(20,20));
        setIcon(QIcon(base)) ;
        setToolTip(tr("Flatten␣directories")) ;
        setWhatsThis(tr("Flattens␣all␣selected␣directories␣(one␣level)␣and␣places␣the␣actual␣data␣►
            items␣in␣their␣places."));
}
specUndoCommand* specFlattenTreeAction::generateUndoCommand()
{
```

```
        model->eliminateChildren(selection) ;
        QList<specModelItem*> foldersToFlatten ;
        qSort(selection) ;
        foreach(specModelItem * item, model->pointerList(selection))
        if(item->isFolder())
                foldersToFlatten << item ;
        if(foldersToFlatten.isEmpty()) return 0 ;
        specMultiCommand* command = new specMultiCommand ;
        command->setParentObject(model) ;
        foreach(specModelItem * item, foldersToFlatten)
        {
                specFolderItem* folder = dynamic_cast<specFolderItem*>(item) ;
                if(!folder) continue ;
                specMoveCommand* moveCommand = new specMoveCommand(command) ;
                QModelIndex folderIndex = model->index(folder) ;
                QModelIndexList containedIndexes = model->indexList(folder->childrenList()) ;
                moveCommand->setItems(containedIndexes,
                                      model->parent(folderIndex),
                                      folderIndex.row());
                moveCommand->redo();
        }
        specUndoCommand* deleteCommand = specDeleteAction::command(model, foldersToFlatten, command) ;
        deleteCommand->redo();
        command->undo();
        command->setText(tr("Flattened␣") + QString::number(foldersToFlatten.size()) + tr("␣folders.►
            "));
        return command ;
}
QList<specStreamable::type> specFlattenTreeAction::requiredTypes() const
{
        return QList<specStreamable::type>() << specStreamable::folder ;
}
```

src/actionlib/actions/specformulavalidator.h

```
#ifndef SPECFORMULAVALIDATOR_H
#define SPECFORMULAVALIDATOR_H
#include <QValidator>
#include <QStringList>
#include <QRegExp>
class specFormulaValidator : public QValidator
{
        Q_OBJECT
private:
        QRegExp variables ;
        static int openCloseDifference(const QString& s) ;
public:
        explicit specFormulaValidator(const QRegExp&, QObject* parent = 0);
        void fixup(QString&) const;
        State validate(QString&, int&) const ;
signals:
        void evaluationError(const QString&) const ;
};
#endif
```

src/actionlib/actions/specformulavalidator.cpp

```
#include "specformulavalidator.h"
#include <muParser.h>
#include <map>
#include <QStringList>
#include <QDebug>
#include "specspectrumcalculatoraction.h"
specFormulaValidator::specFormulaValidator(const QRegExp& v, QObject* parent)
        : QValidator(parent),
          variables(v)
{
}
int specFormulaValidator::openCloseDifference(const QString& s)
{
        return s.count("(") - s.count(")") ;
}
```

```
void specFormulaValidator::fixup(QString& s) const
{
        int num = openCloseDifference(s) ;
        if(num < 0) s.prepend(QString(-num, '(')) ;
        if(num > 0) s.append(QString(num, ')')) ;
}
double modulo(double a, double b)
{
        return int (a) % int (b) ;
}
QValidator::State specFormulaValidator::validate(QString& expr, int& pos) const
{
        Q_UNUSED(pos)
        QString s(expr) ;
        specSpectrumCalculatorAction::descriptorNames(s) ;
        if(openCloseDifference(s))
        {
                emit evaluationError("Parantheses not balanced.");
                return Intermediate ;
        }
        mu::Parser parser ;
        parser.DefineOprt("%", modulo, 6);
        std::map<mu::string_type, mu::value_type*> usedVariables ;
        try
        {
                parser.SetExpr(s.toStdString());
                usedVariables = parser.GetUsedVar() ;
        }
        catch(mu::Parser::exception_type& e)
        {
                emit evaluationError(tr("muParser error: ")
                                     + QString::fromStdString(e.GetMsg())
                                     + (e.GetToken().empty() ? QString() :
                                        "\nToken: \""
                                        + QString::fromStdString(e.GetToken())));
                return Intermediate ;
        }
        for(std::map<mu::string_type, mu::value_type*>::iterator
                i = usedVariables.begin() ; i != usedVariables.end() ; ++i)
        {
                QString varName(QString::fromStdString(i->first)) ;
                if(!variables.exactMatch(varName))
                {
                        emit evaluationError(tr("Illegal variable name: ") + varName) ;
                        return Intermediate ;
                }
        }
        emit evaluationError(QString()) ;
        return Acceptable ;
}
```

src/actionlib/actions/specimportspecaction.h

```
#ifndef SPECIMPORTSPECACTION_H
#define SPECIMPORTSPECACTION_H
#include "specitemaction.h"
class QFile ;
class specModelItem ;
class specImportSpecAction : public specItemAction
{
        Q_OBJECT
        QStringList filters ;
public:
        explicit specImportSpecAction(QObject* parent = 0);
        void setFilters(const QStringList& f) ;
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specimportspecaction.cpp

```
#include "specimportspecaction.h"
#include <QFileDialog>
#include "utility-functions.h"
#include "specaddfoldercommand.h"
#include "specview.h"
#include <QMessageBox>
specImportSpecAction::specImportSpecAction(QObject* parent) :
        specItemAction(parent)
{
        setIcon(QIcon::fromTheme("archive-insert"));
        setToolTip(tr("Import␣files")) ;
        setWhatsThis(tr("Import␣files.␣␣Use␣this␣button␣to␣get␣started."));
        setText(tr("Import␣file...")) ;
        setShortcut(tr("Ctrl+F"));
}
specUndoCommand* specImportSpecAction::generateUndoCommand()
{
        int row = 0 ;
        if(!currentItem->isFolder())
        {
                row = currentIndex.row() + 1 ;
                currentIndex = currentIndex.parent() ;
        }
        QStringList fileNames = QFileDialog::getOpenFileNames(view, tr("Files␣to␣import")) ;
        if(fileNames.isEmpty()) return 0 ;
        QList<specModelItem*> importedItems ;
        for(int i = 0 ; i < fileNames.size() ; ++i)
        {
                QList<specModelItem*> (*importFunction)(QFile&) = fileFilter(fileNames[i]);
                if(!model->acceptableImportFunctions().contains(importFunction))
                {
                        QMessageBox::critical(0, tr("Cannot␣import"), tr("Cannot␣import␣this␣type␣of►
                            ␣data␣here:") + fileNames[i], QMessageBox::Ok) ;
                        continue ;
                }
                QFile fileToImport(fileNames[i]) ;
                fileToImport.open(QFile::ReadOnly | QFile::Text) ;
                importedItems << importFunction(fileToImport) ;
        }
        if(importedItems.isEmpty()) return 0 ;
        if(! model->insertItems(importedItems, currentIndex, row))
        {
                foreach(specModelItem* item, importedItems)
                        delete item ;
                return 0 ;
        }
        specAddFolderCommand* command = new specAddFolderCommand ;
        command->setParentObject(model) ;
        command->setItems(importedItems) ;
        command->setText((fileNames.size() > 1 ? tr("Import␣data␣(files:") : tr("Import␣data␣(file:"))
                         + fileNames.join(",␣") + ")") ;
        return command ;
}
void specImportSpecAction::setFilters(const QStringList& f)
{
        filters = f ;
}
```

src/actionlib/actions/specitemaction.h

```
#ifndef SPECITEMACTION_H
#define SPECITEMACTION_H
#include "specundoaction.h"
class specView ;
class specModel ;
class specItemAction : public specUndoAction
{
        Q_OBJECT
public:
        explicit specItemAction(QObject* parent = 0) ;
        virtual const std::type_info& possibleParent() ;
        virtual QList<specStreamable::type> requiredTypes() const ;
        bool requirements() ;
protected:
```

```
        virtual bool specificRequirements() { return true ;}
        virtual bool postProcessingRequirements() const { return true ; }
        void execute();
        specView* view ;
        specModel* model ;
        QModelIndex currentIndex, insertionIndex ;
        QModelIndexList selection ;
        QList<specModelItem*> pointers ;
        specModelItem* currentItem ;
        int insertionRow ;
        void expandSelectedFolders(QList<specModelItem*>& items, QList<specModelItem*>& folders) ;
        virtual specUndoCommand* generateUndoCommand() = 0 ;
        void assembleSelection() ;
        void getInsertionIndex() ;
};
#endif
```

src/actionlib/actions/specitemaction.cpp

```
#include "specitemaction.h"
#include "specview.h"
#include "specmodel.h"
#include "specmulticommand.h"
specItemAction::specItemAction(QObject* parent)
        : specUndoAction(parent),
          view(),
          model(0),
          currentItem(0),
          insertionRow(0)
{
}
const std::type_info& specItemAction::possibleParent()
{
        return typeid(specView) ;
}
void specItemAction::assembleSelection()
{
        QList<specStreamable::type> types = requiredTypes() ;
        selection = view->getSelection() ;
        pointers.clear();
        if(!types.isEmpty())
        {
                foreach(specModelItem * item, model->pointerList(selection))
                if(types.contains(item->typeId()))
                        pointers << item ;
        }
}
void specItemAction::getInsertionIndex()
{
        if(currentItem && !currentItem->isFolder())
        {
                insertionRow = currentIndex.row() + 1 ;
                insertionIndex = currentIndex.parent() ;
        }
        else
        {
                insertionRow = 0 ;
                insertionIndex = currentIndex ;
        }
}
void specItemAction::execute()
{
        if(!requirements()) return ;
        assembleSelection();
        getInsertionIndex();
        if(!postProcessingRequirements()) return ;
        specUndoCommand* command = generateUndoCommand();
        if(!command) return ;
        if(dynamic_cast<specMultiCommand*>(command) && !command->childCount())
        {
                delete command ;
                return ;
        }
        if(!library)
```

```
        {
                command->redo();
                delete command ;
                return ;
        }
        library->push(command) ;
}
void specItemAction::expandSelectedFolders(QList<specModelItem*>& items, QList<specModelItem*>& ►
    folders)
{
        if(!model) return ;
        folders.clear();
        items = model->pointerList(selection) ;
        for(int i = 0 ; i < items.size() ; ++i)
        {
                if(items[i]->isFolder())
                {
                        for(int j = 0 ; j < items[i]->children() ; ++j)
                                items << ((specFolderItem*)(items[i]))->child(j) ;
                        folders << items.takeAt(i--) ;
                }
        }
}
QList<specStreamable::type> specItemAction::requiredTypes() const
{
        return QList<specStreamable::type>() ;
}
bool specItemAction::requirements()
{
        if(!(view = qobject_cast<specView*> (parent()))) return false ;
        if(!(model = view->model())) return false ;
        currentIndex = view->currentIndex() ;
        currentItem = model->itemPointer(currentIndex) ;
        return specificRequirements() ;
}
```

src/actionlib/actions/specitempropertiesaction.h

```
#ifndef SPECITEMPROPERTIESACTION_H
#define SPECITEMPROPERTIESACTION_H
#include "specrequiresitemaction.h"
class specItemPropertiesAction : public specItemAction
{
        Q_OBJECT
public:
        explicit specItemPropertiesAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
        bool specificRequirements() ;
};
#endif
```

src/actionlib/actions/specitempropertiesaction.cpp

```
#include "specitempropertiesaction.h"
#include "specdataitem.h"
#include "specsvgitem.h"
#include "specmetaitem.h"
specItemPropertiesAction::specItemPropertiesAction(QObject* parent) :
        specItemAction(parent)
{
        setIcon(QIcon::fromTheme("document-properties")) ;
        setToolTip(tr("Edit␣item␣properties")) ;
        setWhatsThis(tr("Displays␣an␣item-specific␣dialog␣to␣let␣you␣edit␣the␣current␣item's␣►
            properties")) ;
        setText(tr("Item␣properties...")) ;
        setShortcut(tr("Ctrl+i"));
}
specUndoCommand* specItemPropertiesAction::generateUndoCommand()
{
        if(!currentItem) return 0 ;
        return currentItem->itemPropertiesAction(model) ;
```

```
}
bool specItemPropertiesAction::specificRequirements()
{
        return (dynamic_cast<specDataItem*>(currentItem)
                || dynamic_cast<specMetaItem*>(currentItem)
                || dynamic_cast<specSVGItem*>(currentItem)) ;
}
```

src/actionlib/actions/speclabelaction.h

```
#ifndef SPECLABELACTION_H
#define SPECLABELACTION_H
#include "specundoaction.h"
class specSimpleTextEdit ;
class specLabelAction : public specUndoAction
{
        Q_OBJECT
private:
        void execute() ;
protected:
        virtual QString textToEdit() = 0 ;
        virtual QString labelText() const = 0 ;
        virtual specStreamable::type commandId() = 0 ;
public:
        explicit specLabelAction(QObject* parent = 0) ;
        const std::type_info& possibleParent() ;
};
class specTitleAction : public specLabelAction
{
        Q_OBJECT
public:
        explicit specTitleAction(QObject* parent = 0) ;
private:
        QString textToEdit() ;
        specStreamable::type commandId() ;
        QString labelText() const ;
};
class specXLabelAction : public specLabelAction
{
        Q_OBJECT
public:
        explicit specXLabelAction(QObject* parent = 0) ;
private:
        QString textToEdit() ;
        specStreamable::type commandId() ;
        QString labelText() const ;
};
class specYLabelAction : public specLabelAction
{
        Q_OBJECT
public:
        explicit specYLabelAction(QObject* parent = 0) ;
private:
        QString textToEdit() ;
        specStreamable::type commandId() ;
        QString labelText() const ;
};
#endif
```

src/actionlib/actions/speclabelaction.cpp

```
#include "speclabelaction.h"
#include "textEditor/specsimpletextedit.h"
#include <QDialog>
#include <QVBoxLayout>
#include <QDialogButtonBox>
#include "specplotlabelcommand.h"
specLabelAction::specLabelAction(QObject* parent)
        : specUndoAction(parent)
{
}
specTitleAction::specTitleAction(QObject* parent)
```

```
        : specLabelAction(parent)
{
        setIcon(QIcon(":/changetitle.png")) ;
        setToolTip("Change␣Plot␣Title") ;
        setWhatsThis("Allows␣you␣to␣edit␣the␣plot's␣title.");
        setText(tr("Change␣plot␣title...")) ;
        setShortcut(tr("Ctrl+T"));
}
specXLabelAction::specXLabelAction(QObject* parent)
        : specLabelAction(parent)
{
        setIcon(QIcon(":/changexlabel.png")) ;
        setToolTip("Change␣X␣Label") ;
        setWhatsThis("Allows␣you␣to␣edit␣the␣plot's␣x␣axis␣label.");
        setText(tr("Plot␣x␣label...")) ;
        setShortcut(tr("Ctrl+Alt+x"));
}
specYLabelAction::specYLabelAction(QObject* parent)
        : specLabelAction(parent)
{
        setIcon(QIcon(":/changeylabel.png")) ;
        setToolTip("Change␣Y␣Label") ;
        setWhatsThis("Allows␣you␣to␣edit␣the␣plot's␣y␣axis␣label.");
        setText(tr("Plot␣y␣label...")) ;
        setShortcut(tr("Ctrl+Alt+y"));
}
const std::type_info& specLabelAction::possibleParent()
{
        return typeid(specPlot) ;
}
void specLabelAction::execute()
{
        QString text(textToEdit()) ;
        QDialog dialog ;
        dialog.setWindowTitle(tr("Edit␣Label")) ;
        dialog.setLayout(new QVBoxLayout) ;
        specSimpleTextEdit* editor = new specSimpleTextEdit(&dialog) ;
        editor->setText(text) ;
        dialog.layout()->addWidget(editor) ;
        QDialogButtonBox* buttons = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::►
            Cancel) ;
        dialog.layout()->addWidget(buttons);
        connect(buttons, SIGNAL(accepted()), &dialog, SLOT(accept())) ;
        connect(buttons, SIGNAL(rejected()), &dialog, SLOT(reject())) ;
        if(QDialog::Accepted != dialog.exec() || editor->getText() == text) return ;
        text = editor->getText() ;
        specPlotLabelCommand* command = generatePlotLabelCommand(commandId()) ;
        command->setParentObject(parent()) ;
        command->setLabelText(text) ;
        command->setText(tr("Edited␣") + labelText());
        library->push(command) ;
}
QString specTitleAction::textToEdit()
{
        return ((specPlot*) parent())->title().text() ;
}
QString specXLabelAction::textToEdit()
{
        return ((specPlot*) parent())->axisTitle(QwtPlot::xBottom).text() ;
}
QString specYLabelAction::textToEdit()
{
        return ((specPlot*) parent())->axisTitle(QwtPlot::yLeft).text() ;
}
specStreamable::type specTitleAction::commandId()
{
        return specStreamable::plotTitleCommandId ;
}
specStreamable::type specXLabelAction::commandId()
{
        return specStreamable::plotXLabelCommandId ;
}
specStreamable::type specYLabelAction::commandId()
{
        return specStreamable::plotYLabelCommandId ;
```

```
}
QString specTitleAction::labelText() const
{
        return tr("title") ;
}
QString specXLabelAction::labelText() const
{
        return tr("x␣axis␣label") ;
}
QString specYLabelAction::labelText() const
{
        return tr("y␣axis␣label") ;
}
```

src/actionlib/actions/specmergeaction.h

```
#ifndef SPECMERGEACTION_H
#define SPECMERGEACTION_H
#include "specrequiresitemaction.h"
#include "specdataview.h"
class specMergeDialog ;
class specMergeAction : public specRequiresDataItemAction
{
        Q_OBJECT
private:
        QModelIndexList allChildren(const QModelIndex& parent) const ;
public:
        explicit specMergeAction(QObject* parent = 0);
        ~specMergeAction() ;
        const std::type_info& possibleParent() { return typeid(specDataView) ; }
private:
        specUndoCommand* generateUndoCommand() ;
        specMergeDialog* dialog ;
};
#endif
```

src/actionlib/actions/specmergeaction.cpp

```
#include "specmergeaction.h"
#include "specdeleteaction.h"
#include "specaddfoldercommand.h"
#include "specmulticommand.h"
#include "specspectrumplot.h"
#include "names.h"
#include "specprofiler.h"
#include <QProgressDialog>
#include "specdataitem.h"
#include "specmergedialog.h"
#include "specworkerthread.h"
#include "specfiltergenerator.h"
class mergeActionThread : public specWorkerThread
{
private:
        specDescriptorComparisonCriterion::container criteria ;
        spec::correctionMode spectralAdaptation ;
        QList<specModelItem*> items ;
        QList<specModelItem*> toBeDeleted ;
        QList<specModelItem*> toInsert ;
        bool cleanUp()
        {
                if(!toTerminate) return false ;
                items.clear();
                toBeDeleted.clear();
                foreach (specModelItem* item, toInsert)
                        delete item ;
                toInsert.clear();
                return true ;
        }
        void mergeItems(specDataItem* newItem, const specDataItem* other) const
        {
                if (!newItem || !other) return ;
                specDataItem* correctedItem = 0 ;
```

```
                if(spectralAdaptation != spec::noCorrection)
                {
                        QMap<double, double> reference = newItem->dataMap() ;
                        if (!reference.isEmpty())
                        {
                                specFilterGenerator filterGenerator ;
                                filterGenerator.calcOffset();
                                if (spectralAdaptation == spec::offsetAndSlope)
                                        filterGenerator.calcSlope();
                                filterGenerator.setReference(reference) ;
                                correctedItem = new specDataItem(*other) ;
                                specDataPointFilter filter = filterGenerator.generateFilter(►
                                    correctedItem) ;
                                if (filter.valid())
                                {
                                        correctedItem->addDataFilter(filter) ;
                                        other = correctedItem ;
                                }
                        }
                }
                *newItem += *other ;
                delete correctedItem ;
        }
        bool itemsAreEqual(specModelItem* first, specModelItem* second, const QList<stringDoublePair►
            >& criteria) ;
public:
        mergeActionThread(const QList<specModelItem*>& itms,
                          const specDescriptorComparisonCriterion::container& crit,
                          spec::correctionMode sadap)
                : specWorkerThread(itms.size()),
                  criteria(crit),
                  spectralAdaptation(sadap),
                  items(itms)
        {
        }
        ~mergeActionThread() { cleanUp() ; }
#define PASSLISTFUNCTIONMACRO(FUNCTIONNAME,LISTNAME) \
QList<specModelItem*> FUNCTIONNAME() { \
                QList<specModelItem*> result = LISTNAME; \
                LISTNAME.clear(); \
                return result ; }
        PASSLISTFUNCTIONMACRO(getItemsToDelete, toBeDeleted)
        PASSLISTFUNCTIONMACRO(getItemsToInsert, toInsert)
        void run()
        {
                emit progressValue(0);
                specProfiler profiler("run␣merge␣command␣thread:") ;
                int total = items.size() ;
                while(!items.isEmpty())
                {
                        emit progressValue(total - items.size());
                        if(cleanUp()) return ;
                        specModelItem *comparisonItem = items.first() ;
                        QList<specModelItem*> toMergeWith ;
                        QList<specModelItem*>::iterator i = items.begin() ;
                        while (i != items.end())
                        {
                                if (specDescriptorComparisonCriterion::itemsEqual(comparisonItem, *i►
                                    , criteria))
                                {
                                        toMergeWith << *i ;
                                        i = items.erase(i) ;
                                }
                                else
                                        ++i ;
                        }
                        if (toMergeWith.size() == 1) continue ;
                        specDataItem* newItem = new specDataItem() ;
                        foreach(specModelItem* other, toMergeWith)
                                mergeItems(newItem, dynamic_cast<specDataItem*>(other));
                        newItem->flatten();
                        toBeDeleted << toMergeWith ;
                        toInsert << newItem ;
                }
                emit progressValue(total) ;
```

```
                finish() ;
        }
};
specMergeAction::specMergeAction(QObject* parent)
        : specRequiresDataItemAction(parent),
          dialog(new specMergeDialog(0))
{
        setIcon(QIcon(":/merge.png")) ;
        setToolTip(tr("Merge␣items")) ;
        setWhatsThis(tr("Merge␣selected␣data␣items.␣␣You␣may␣define␣criteria␣and␣processing␣options␣►
            for␣merging.")) ;
        setText(tr("Merge␣items...")) ;
        setShortcut(tr("Ctrl+M"));
}
specMergeAction::~specMergeAction()
{
        delete dialog ;
}
specUndoCommand* specMergeAction::generateUndoCommand()
{
        specProfiler profiler("Prepare␣merge␣commands:") ;
        if(selection.size() < 2) return 0 ;
        specDescriptorComparisonCriterion::container criteria ;
        spec::correctionMode spectralAdaptation;
        dialog->setDescriptors(model->descriptors(), model->descriptorProperties()) ;
        if(dialog->exec() != QDialog::Accepted) return 0 ;
        dialog->getMergeCriteria(criteria, spectralAdaptation);
        mergeActionThread mat(pointers, criteria, spectralAdaptation) ;
        mat.run();
        QList<specModelItem*> toInsert = mat.getItemsToInsert() ;
        if (!model->insertItems(toInsert, insertionIndex, insertionRow))
        {
                foreach(specModelItem* item, toInsert)
                        delete item ;
                return 0 ;
        }
        QList<specModelItem*> toDelete = mat.getItemsToDelete() ;
        specMultiCommand* Command = new specMultiCommand ;
        Command->setMergeable(false) ;
        QString description = tr("Merge␣")
                            + QString::number(toDelete.size())
                            + tr("␣items␣to␣")
                            + QString::number(toInsert.size())
                            + tr("␣items.") ;
        if(spectralAdaptation != spec::noCorrection)
                description += tr("␣Items␣aligned␣prior␣to␣merging␣(offset")
                             + (spectralAdaptation == spec::offsetAndSlope ?
                                     tr("␣and␣slope") : QString())
                             + tr(").") ;
        QStringList criteriaDescription ;
        foreach(const specDescriptorComparisonCriterion& comparison, criteria)
                criteriaDescription << comparison.descriptor()
                                    + (comparison.isNumeric() ?
                                              "␣(" + QString::number(comparison.tolerance()) + ")"
                                            : QString());
        if(!criteriaDescription.isEmpty())
                (description += tr("␣Criteria␣(Tolerance):␣")) += criteriaDescription.join(",␣") ;
        Command->setText(description) ;
        specAddFolderCommand* insertionCommand = new specAddFolderCommand(Command) ;
        Command->setParentObject(model) ;
        insertionCommand->setItems(toInsert) ;
        specDeleteAction::command(model, toDelete, Command) ;
        return Command ;
}
```

src/actionlib/actions/specmergedialog.h

```
#ifndef SPECMERGEDIALOG_H
#define SPECMERGEDIALOG_H
#include <QDialog>
#include <QAbstractTableModel>
#include <QItemDelegate>
#include "names.h"
#include "specdescriptorcomparisoncriterion.h"
```

```
namespace Ui
{
        class specMergeDialog;
}
typedef QPair<QString, double> stringDoublePair ;
class mergeModel : public QAbstractTableModel
{
        Q_OBJECT
private:
        class mergeModelPrivate ;
        mergeModelPrivate* d ;
public:
        mergeModel(QObject* parent = 0) ;
        ~mergeModel() ;
        void setDescriptors(const QStringList& descriptors, const QList<spec::descriptorFlags>& ►
            descriptorProperties) ;
        specDescriptorComparisonCriterion::container getMergeCriteria() const ;
        int rowCount(const QModelIndex& parent) const ;
        int columnCount(const QModelIndex& parent) const ;
        QVariant data(const QModelIndex& index, int role) const ;
        QVariant headerData(int section, Qt::Orientation orientation, int role) const ;
        bool setData(const QModelIndex& index, const QVariant& value, int role) ;
        Qt::ItemFlags flags(const QModelIndex& index) const ;
};
class specMergeDialog : public QDialog
{
        Q_OBJECT
public:
        explicit specMergeDialog(QWidget* parent = 0);
        ~specMergeDialog();
        void setDescriptors(const QStringList& descriptors, const QList<spec::descriptorFlags>& ►
            descriptorProperties) ;
        void getMergeCriteria(specDescriptorComparisonCriterion::container &toCompare, spec::►
            correctionMode &spectralAdaptation) const ;
private:
        Ui::specMergeDialog* ui;
        mergeModel* mm ;
};
class mergeDelegate : public QItemDelegate
{
        Q_OBJECT
public:
        explicit mergeDelegate(QObject* parent = 0) ;
        QWidget* createEditor(QWidget* parent, const QStyleOptionViewItem& option, const QModelIndex►
            & index) const;
        void setEditorData(QWidget* editor, const QModelIndex& index) const;
        void setModelData(QWidget* editor, QAbstractItemModel* model, const QModelIndex& index) const;
        void updateEditorGeometry(QWidget* editor, const QStyleOptionViewItem& option, const ►
            QModelIndex& index) const;
};
#endif
```

src/actionlib/actions/specmergedialog.cpp

```
#include "specmergedialog.h"
#include "ui_specmergedialog.h"
#include <QSet>
#include "names.h"
#include <QItemDelegate>
#include <QDoubleSpinBox>
class mergeModel::mergeModelPrivate
{
public:
        struct mergeItem : specDescriptorComparisonCriterion
        {
                bool enabled ;
        public:
                mergeItem()
                        : specDescriptorComparisonCriterion("", false),
                          enabled(false)
                {}
                mergeItem(const QString& s, bool n)
                        : specDescriptorComparisonCriterion(s, n),
                          enabled(false)
```

```
                {}
22              bool isEnabled() const { return enabled ; }
                void enable(bool a = true) { enabled = a ; }
        };
        QList<mergeItem> items ;
};
27 mergeModel::mergeModel(QObject* parent)
        : QAbstractTableModel(parent),
          d(new mergeModelPrivate)
{}
mergeModel::~mergeModel()
32 {
        delete d ;
}
int mergeModel::rowCount(const QModelIndex& parent) const
{
37      Q_UNUSED(parent)
        return d->items.size() ;
}
int mergeModel::columnCount(const QModelIndex& parent) const
{
42      Q_UNUSED(parent)
        return 2 ;
}
QVariant mergeModel::headerData(int section, Qt::Orientation orientation, int role) const
{
47      if(role != Qt::DisplayRole) return QVariant() ;
        if(orientation != Qt::Horizontal) return QVariant() ;
        switch(section)
        {
                case 0:
52                      return tr("Matching␣criterion") ;
                case 1:
                        return tr("Tolerance") ;
                default:
                        return QVariant() ;
57      }
}
Qt::ItemFlags mergeModel::flags(const QModelIndex& index) const
{
        if(index.column() == 0) return Qt::ItemIsUserCheckable | Qt::ItemIsEnabled;
62      if(index.column() == 1 && d->items[index.row()].isNumeric())
                return Qt::ItemIsEditable | Qt::ItemIsEnabled ;
        return Qt::NoItemFlags ;
}
QVariant mergeModel::data(const QModelIndex& index, int role) const
67 {
        if(role == Qt::CheckStateRole && index.column() == 0)
                return d->items[index.row()].isEnabled() ? Qt::Checked : Qt::Unchecked ;
        if(role != Qt::DisplayRole)
                return QVariant() ;
72      mergeModelPrivate::mergeItem item = d->items[index.row()] ;
        switch(index.column())
        {
                case 0:
                        return item.descriptor();
77              case 1:
                        return item.isNumeric() ? QVariant(item.tolerance()) : QVariant() ;
                default:
                        return QVariant() ;
        }
82 }
bool mergeModel::setData(const QModelIndex& index, const QVariant& value, int role)
{
        if(index.column() == 0 && Qt::CheckStateRole == role)
        {
87              d->items[index.row()].enable(value.toBool()) ;
                emit dataChanged(index, index) ;
                return true ;
        }
        if(index.column() == 1 && Qt::EditRole == role)
92      {
                d->items[index.row()].setTolerance(value.toDouble()) ;
                emit dataChanged(index, index) ;
                return true ;
```

```
        }
        return false ;
}
specDescriptorComparisonCriterion::container mergeModel::getMergeCriteria() const
{
        specDescriptorComparisonCriterion::container result ;
        foreach(mergeModelPrivate::mergeItem item, d->items)
                if(item.isEnabled())
                        result << item ;
        return result ;
}
void mergeModel::setDescriptors(const QStringList& descriptors, const QList<spec::descriptorFlags>& ►
    descriptorProperties)
{
        beginResetModel();
        QList<mergeModelPrivate::mergeItem> newItems ;
        for (int i = 0 ; i < qMin(descriptors.size(), descriptorProperties.size()) ; ++i)
        {
                const QString& descriptor = descriptors[i] ;
                bool isNumeric = descriptorProperties[i] & spec::numeric ;
                foreach(mergeModelPrivate::mergeItem item, d->items)
                {
                        if (item.descriptor() == descriptor)
                        {
                                item.setNumeric(isNumeric);
                                newItems << item ;
                                break ;
                        }
                }
                if (newItems.size() == i)
                        newItems << mergeModelPrivate::mergeItem(descriptor, isNumeric) ;
        }
        d->items.swap(newItems) ;
        endResetModel();
}
mergeDelegate::mergeDelegate(QObject* parent)
        : QItemDelegate(parent)
{}
QWidget* mergeDelegate::createEditor(QWidget* parent, const QStyleOptionViewItem& option, const ►
    QModelIndex& index) const
{
        Q_UNUSED(option)
        Q_UNUSED(index)
        QDoubleSpinBox* spinBox = new QDoubleSpinBox(parent) ;
        spinBox->setMinimum(0);
        spinBox->setDecimals(4);
        return spinBox ;
}
void mergeDelegate::setModelData(QWidget* editor, QAbstractItemModel* model, const QModelIndex& ►
    index) const
{
        QDoubleSpinBox* spinBox = qobject_cast<QDoubleSpinBox*> (editor) ;
        if(!spinBox) return ;
        model->setData(index, spinBox->value()) ;
}
void mergeDelegate::setEditorData(QWidget* editor, const QModelIndex& index) const
{
        QDoubleSpinBox* spinBox = qobject_cast<QDoubleSpinBox*> (editor) ;
        if(!spinBox) return ;
        spinBox->setValue(index.data().toDouble());
}
void mergeDelegate::updateEditorGeometry(QWidget* editor, const QStyleOptionViewItem& option, const ►
    QModelIndex& index) const
{
        Q_UNUSED(index)
        editor->setGeometry(option.rect);
}
specMergeDialog::specMergeDialog(QWidget* parent) :
        QDialog(parent),
        ui(new Ui::specMergeDialog),
        mm(new mergeModel(this))
{
        ui->setupUi(this);
        ui->criteria->setModel(mm) ;
        ui->criteria->setItemDelegateForColumn(1, new mergeDelegate(ui->criteria));
```

```
}
void specMergeDialog::setDescriptors(const QStringList& descriptors, const QList<spec::►
    descriptorFlags>& descriptorProperties)
{
        mm->setDescriptors(descriptors, descriptorProperties);
        ui->criteria->horizontalHeader()->setResizeMode(QHeaderView::Interactive);
        ui->criteria->resizeColumnsToContents();
        ui->criteria->horizontalHeader()->setResizeMode(QHeaderView::Fixed);
        ui->criteria->setFixedWidth(ui->criteria->sizeHint().width() + 15);
}
void specMergeDialog::getMergeCriteria(specDescriptorComparisonCriterion::container& toCompare, spec►
    ::correctionMode& spectralAdaptation) const
{
        spectralAdaptation = spec::noCorrection ;
        if (ui->useOffset->isChecked())
        {
                spectralAdaptation = spec::offset ;
                if (ui->useSlope->isChecked())
                        spectralAdaptation = spec::offsetAndSlope ;
        }
        toCompare = mm->getMergeCriteria() ;
}
specMergeDialog::~specMergeDialog()
{
        delete ui;
}
```

src/actionlib/actions/specmergedialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>specMergeDialog</class>
 <widget class="QDialog" name="specMergeDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>298</width>
    <height>327</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Minimum" vsizetype="Preferred">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>Merge</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="QLabel" name="description">
     <property name="text">
      <string>Matching items will be merged</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QTableView" name="criteria">
     <property name="sizePolicy">
      <sizepolicy hsizetype="Expanding" vsizetype="Expanding">
       <horstretch>0</horstretch>
       <verstretch>0</verstretch>
      </sizepolicy>
     </property>
     <property name="alternatingRowColors">
      <bool>true</bool>
     </property>
     <property name="showGrid">
      <bool>false</bool>
     </property>
     <property name="gridStyle">
      <enum>Qt::DotLine</enum>
```

```
     </property>
    </widget>
   </item>
   <item>
    <widget class="QCheckBox" name="useOffset">
     <property name="text">
      <string>Align items before merging</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QCheckBox" name="useSlope">
     <property name="enabled">
      <bool>false</bool>
     </property>
     <property name="text">
      <string>Use offset and slope for alignment</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>specMergeDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>252</x>
     <y>322</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>specMergeDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>293</x>
     <y>322</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>useOffset</sender>
   <signal>toggled(bool)</signal>
   <receiver>useSlope</receiver>
   <slot>setEnabled(bool)</slot>
   <hints>
    <hint type="sourcelabel">
     <x>66</x>
     <y>261</y>
```

```
    </hint>
    <hint type="destinationlabel">
     <x>71</x>
     <y>278</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/actionlib/actions/specnewmetaitemaction.h

```
#ifndef SPECNEWMETAITEMACTION_H
#define SPECNEWMETAITEMACTION_H
#include "specitemaction.h"
class specNewMetaItemAction : public specItemAction
{
        Q_OBJECT
public:
        explicit specNewMetaItemAction(QObject* parent = 0);
        const std::type_info& possibleParent() ;
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specnewmetaitemaction.cpp

```
#include "specnewmetaitemaction.h"
#include "kinetic/specmetaview.h"
#include "specmetamodel.h"
#include "specaddfoldercommand.h"
#include "specmetaitem.h"
specNewMetaItemAction::specNewMetaItemAction(QObject* parent) :
        specItemAction(parent)
{
        this->setIcon(QIcon::fromTheme("document-new")) ;
        setToolTip(tr("New item"));
        setWhatsThis(tr("Create a new item in this dock window's list."));
        setText(tr("New meta item")) ;
        setShortcut(tr("Ctrl+Shift+N"));
}
const std::type_info& specNewMetaItemAction::possibleParent()
{
        return typeid(specMetaView) ;
}
specUndoCommand* specNewMetaItemAction::generateUndoCommand()
{
        specMetaView* view = qobject_cast<specMetaView*> (specItemAction::view) ;
        if(!view) return 0 ;
        int row = 0 ;
        if(!currentItem->isFolder())
        {
                row = currentIndex.row() + 1 ;
                currentIndex = currentIndex.parent() ;
        }
        specMetaItem* pointer = new specMetaItem ;
        pointer->setModels(model, view->getDataView()->model());
        if(! model->insertItems(QList<specModelItem*>() << pointer, currentIndex, row))
        {
                delete pointer ;
                return 0;
        }
        specAddFolderCommand* command = new specAddFolderCommand ;
        command->setParentObject(model) ;
        command->setItem(pointer) ;
        command->setText(tr("Add meta item")) ;
        return command ;
}
```

src/actionlib/actions/specnormalizeaction.h

```
#ifndef SPECNORMALIZEACTION_H
#define SPECNORMALIZEACTION_H
#include "specrequiresitemaction.h"
class QDialog ;
namespace Ui
{
        class specNormalizeActionDialog ;
}
class specNormalizeAction : public specRequiresDataItemAction
{
        Q_OBJECT
public:
        explicit specNormalizeAction(QObject* parent = 0);
        ~specNormalizeAction() ;
        const std::type_info& possibleParent() ;
protected:
        specUndoCommand* generateUndoCommand() ;
private:
        QDialog* uiDialog ;
        Ui::specNormalizeActionDialog* ui ;
};
#endif
```

src/actionlib/actions/specnormalizeaction.cpp

```
#include "specnormalizeaction.h"
#include "ui_specnormalizeactiondialog.h"
#include <QDoubleValidator>
#include <QDialog>
#include "specmulticommand.h"
#include "specexchangefiltercommand.h"
#include "specdataview.h"
#include "specdataitem.h"
specNormalizeAction::specNormalizeAction(QObject* parent) :
        specRequiresDataItemAction(parent),
        uiDialog(new QDialog),
        ui(new Ui::specNormalizeActionDialog)
{
        setIcon(QIcon(":/normalize.png")) ;
        setToolTip(tr("Normalize␣spectrum")) ;
        setText(tr("Normalize...")) ;
        setWhatsThis(tr("Shift␣x␣and␣scale␣y␣for␣minimum␣or␣maximum␣to␣desired␣position/value.")) ;
        ui->setupUi(uiDialog) ;
        ui->xValue->setValidator(new QDoubleValidator) ;
        ui->yValue->setValidator(new QDoubleValidator) ;
}
specNormalizeAction::~specNormalizeAction()
{
        delete ui ;
        delete uiDialog ;
}
const std::type_info& specNormalizeAction::possibleParent()
{
        return typeid(specDataView) ;
}
specUndoCommand* specNormalizeAction::generateUndoCommand()
{
        if(uiDialog->exec() == QDialog::Rejected) return 0 ;
        if(!ui->scaleYValue->isChecked() && !ui->shiftXValue->isChecked()) return 0 ;
        specMultiCommand* command = new specMultiCommand ;
        command->setParentObject(model) ;
        foreach(specModelItem * item, pointers)
        {
                if(!item->dataSize()) continue ;
                if(!dynamic_cast<specDataItem*>(item)) continue ;
                size_t extremum = 0 ;
                double previousValue = item->sample(0).y() ;
                if(ui->minimum->isChecked())
                {
                        for(size_t i = 0 ; i < item->dataSize() ; ++i)
                                if(previousValue > item->sample(i).y())
                                        extremum = i, previousValue = item->sample(i).y() ;
```

```
                }
                else
                {
                        for(size_t i = 0 ; i < item->dataSize() ; ++i)
                                if(previousValue < item->sample(i).y())
                                        extremum = i, previousValue = item->sample(i).y() ;
                }
                specExchangeFilterCommand* moveCommand = new specExchangeFilterCommand(command, true) ;
                moveCommand->setRelativeFilter(
                    specDataPointFilter(0, 0,
                                        ui->scaleYValue->isChecked() ? ui->yValue->text().toDouble() / previousValue : 1,
                                        ui->shiftXValue->isChecked() ? ui->xValue->text().toDouble() - item->sample(extremum).x() : 0));
                moveCommand->setItem(item);
        }
        command->setText(tr("Normalized ")
                         + QString::number(command->childCount())
                         + tr(" spectra by their ")
                         + (ui->minimum->isChecked() ? tr("minima") : tr("maxima"))
                         + tr(" to")
                         + (ui->shiftXValue->isChecked() ? tr(" x = ") + ui->xValue->text() : QString())
                         + (ui->scaleYValue->isChecked() ? tr(" y = ") + ui->yValue->text() : QString())
                         + tr("."));
        return command ;
}
```

src/actionlib/actions/specnormalizeactiondialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>specNormalizeActionDialog</class>
 <widget class="QDialog" name="specNormalizeActionDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>228</width>
    <height>112</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Normalize</string>
  </property>
  <layout class="QGridLayout" name="gridLayout_2">
   <item row="0" column="0">
    <widget class="QRadioButton" name="minimum">
     <property name="text">
      <string>Minimum</string>
     </property>
     <property name="checked">
      <bool>true</bool>
     </property>
    </widget>
   </item>
   <item row="0" column="1">
    <widget class="QRadioButton" name="maximum">
     <property name="text">
      <string>Maximum</string>
     </property>
    </widget>
   </item>
   <item row="1" column="0" colspan="2">
    <layout class="QGridLayout" name="gridLayout">
     <item row="0" column="0">
      <widget class="QCheckBox" name="shiftXValue">
       <property name="text">
        <string>Shift x value</string>
       </property>
      </widget>
     </item>
```

```
     <item row="0" column="1">
      <widget class="QLineEdit" name="xValue">
       <property name="enabled">
        <bool>false</bool>
       </property>
       <property name="text">
        <string>0.0</string>
       </property>
      </widget>
     </item>
     <item row="1" column="0">
      <widget class="QCheckBox" name="scaleYValue">
       <property name="text">
        <string>Scale y value</string>
       </property>
       <property name="checked">
        <bool>true</bool>
       </property>
      </widget>
     </item>
     <item row="1" column="1">
      <widget class="QLineEdit" name="yValue">
       <property name="text">
        <string>1.0</string>
       </property>
      </widget>
     </item>
    </layout>
   </item>
   <item row="2" column="0">
    <spacer name="horizontalSpacer">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="sizeHint" stdset="0">
      <size>
       <width>66</width>
       <height>20</height>
      </size>
     </property>
    </spacer>
   </item>
   <item row="2" column="1">
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <tabstops>
  <tabstop>buttonBox</tabstop>
  <tabstop>minimum</tabstop>
  <tabstop>maximum</tabstop>
  <tabstop>shiftXValue</tabstop>
  <tabstop>xValue</tabstop>
  <tabstop>scaleYValue</tabstop>
  <tabstop>yValue</tabstop>
 </tabstops>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>specNormalizeActionDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>223</x>
     <y>97</y>
    </hint>
```

```
    <hint type="destinationlabel">
     <x>157</x>
     <y>111</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>specNormalizeActionDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>223</x>
     <y>103</y>
    </hint>
    <hint type="destinationlabel">
     <x>227</x>
     <y>111</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>shiftXValue</sender>
   <signal>toggled(bool)</signal>
   <receiver>xValue</receiver>
   <slot>setEnabled(bool)</slot>
   <hints>
    <hint type="sourcelabel">
     <x>57</x>
     <y>33</y>
    </hint>
    <hint type="destinationlabel">
     <x>147</x>
     <y>40</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>scaleYValue</sender>
   <signal>toggled(bool)</signal>
   <receiver>yValue</receiver>
   <slot>setEnabled(bool)</slot>
   <hints>
    <hint type="sourcelabel">
     <x>38</x>
     <y>66</y>
    </hint>
    <hint type="destinationlabel">
     <x>125</x>
     <y>63</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/actionlib/actions/specpasteaction.h

```
#ifndef SPECPASTEACTION_H
#define SPECPASTEACTION_H
#include "specitemaction.h"
class specPasteAction : public specItemAction
{
        Q_OBJECT
private:
        bool specificRequirements() ;
private slots:
        void checkClipboard() ;
public:
        explicit specPasteAction(QObject* parent = 0);
protected:
        specUndoCommand* generateUndoCommand() ;
};
```

```
#endif
```

src/actionlib/actions/specpasteaction.cpp

```
#include "specpasteaction.h"
#include "specview.h"
#include "specgenealogy.h"
#include "specaddfoldercommand.h"
#include <QMessageBox>
#include <QApplication>
#include <QClipboard>
specPasteAction::specPasteAction(QObject* parent) :
        specItemAction(parent)
{
        this->setIcon(QIcon::fromTheme("edit-paste")) ;
        setToolTip(tr("Paste")) ;
        setWhatsThis(tr("Paste␣data␣from␣clipboard,␣if␣possible")) ;
        setText(tr("Paste")) ;
        setShortcut(tr("Ctrl+V"));
        connect(QApplication::clipboard(), SIGNAL(dataChanged()), this, SLOT(checkClipboard())) ;
        checkClipboard();
}
void specPasteAction::checkClipboard()
{
        if (!requirements()) return ;
        setEnabled(model->mimeAcceptable(QApplication::clipboard()->mimeData())) ;
}
bool specPasteAction::specificRequirements()
{
        return isEnabled() ;
}
specUndoCommand* specPasteAction::generateUndoCommand()
{
        specFolderItem* parentFolder = dynamic_cast<specFolderItem*>(model->itemPointer(►
            insertionIndex)) ;
        if(!parentFolder) return 0 ;
        QList<specModelItem*> oldChildren = parentFolder->childrenList() ;
        if(!model->dropMimeData(QApplication::clipboard()->mimeData(), Qt::CopyAction, insertionRow,►
             0, insertionIndex))
                QMessageBox::critical(0, tr("Paste␣error"), tr("Could␣not␣paste␣correctly."));
        return 0 ;
}
```

src/actionlib/actions/specprintplotaction.h

```
#ifndef SPECPRINTPLOTACTION_H
#define SPECPRINTPLOTACTION_H
#include "specundoaction.h"
class QPrinter ;
class specPrintPlotAction : public specUndoAction
{
        Q_OBJECT
public:
        explicit specPrintPlotAction(QObject* parent = 0);
        ~specPrintPlotAction() ;
        const std::type_info& possibleParent() ;
private:
        void execute() ;
        QPrinter* printer ;
};
#endif
```

src/actionlib/actions/specprintplotaction.cpp

```
#include "specprintplotaction.h"
#include "specplot.h"
#include <QPrinter>
#include <QPrintDialog>
#include <qwt_plot_renderer.h>
#include <QMessageBox>
```

```
#ifndef WIN32BUILD
#include <csignal>
#endif
specPrintPlotAction::specPrintPlotAction(QObject* parent) :
        specUndoAction(parent),
        printer(0)
{
#ifndef WIN32BUILD
        signal(SIGPIPE, SIG_IGN) ;
#endif
        printer = new QPrinter ;
        specPlot* plot = (specPlot*) parentWidget() ;
        QSize plotSize = plot->size() ;
        double aspectRatio = (double) plotSize.width() / plotSize.height() ;
        printer->setOrientation(aspectRatio > 1. ? QPrinter::Landscape : QPrinter::Portrait) ;
        setIcon(QIcon::fromTheme("document-print"));
        setToolTip(tr("Print␣plot")) ;
        setWhatsThis(tr("Print␣this␣dock␣window's␣plot␣as␣currently␣displayed.")) ;
        setText(tr("Print␣plot...")) ;
        setShortcut(tr("Ctrl+P"));
}
const std::type_info& specPrintPlotAction::possibleParent()
{
        return typeid(specPlot) ;
}
specPrintPlotAction::~specPrintPlotAction()
{
        delete printer ;
}
void specPrintPlotAction::execute()
{
        specPlot* plot = (specPlot*) parentWidget() ;
        QSize plotSize = plot->size() ;
        double aspectRatio = (double) plotSize.width() / plotSize.height() ;
        QPrintDialog dialog(printer) ;
        if(dialog.exec())
        {
                QRectF pageSize = printer->pageRect(QPrinter::Millimeter) ;
                double printRatio = pageSize.width() / pageSize.height() ;
                qreal top, left, bottom, right ;
                printer->getPageMargins(&left, &top, &right, &bottom, QPrinter::Millimeter);
                if(QMessageBox::question(0, "Keep␣Aspect", "Preserve␣aspect␣ratio?", QMessageBox::►
                    Yes | QMessageBox::No, QMessageBox::Yes) != QMessageBox::Yes)
                {
                        plot->resize(printer->pageRect().size()) ;
                        plot->replot();
                }
                else
                {
                        if(printRatio > aspectRatio)
                        {
                                double frameAdd = (pageSize.width() - pageSize.height() * ►
                                    aspectRatio) / 2.;
                                printer->setPageMargins(left + frameAdd, top, right + frameAdd, ►
                                    bottom, QPrinter::Millimeter);
                        }
                        else if(printRatio < aspectRatio)
                        {
                                double frameAdd = (pageSize.height() - pageSize.width() / ►
                                    aspectRatio) / 2. ;
                                printer->setPageMargins(left, top + frameAdd, right, bottom + ►
                                    frameAdd, QPrinter::Millimeter);
                        }
                }
                QwtPlotRenderer renderer ;
                renderer.setDiscardFlag(QwtPlotRenderer::DiscardCanvasBackground, true) ;
                renderer.setLayoutFlags(QwtPlotRenderer::FrameWithScales) ;
                renderer.setDiscardFlags(QwtPlotRenderer::DiscardBackground
                                         | QwtPlotRenderer::DiscardCanvasBackground);
                renderer.renderTo(plot, *printer) ;
                printer->setPageMargins(left, top, right, bottom, QPrinter::Millimeter) ;
        }
        plot->resize(plotSize) ;
}
```

src/actionlib/actions/specremovedataaction.h

```
#ifndef SPECREMOVEDATAACTION_H
#define SPECREMOVEDATAACTION_H
#include "specrequiresitemaction.h"
class specRemoveDataAction : public specRequiresDataItemAction
{
        Q_OBJECT
public:
        explicit specRemoveDataAction(QObject* parent = 0);
        const std::type_info& possibleParent() ;
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specremovedataaction.cpp

```
#include "specremovedataaction.h"
#include "specmulticommand.h"
#include "specexchangedatacommand.h"
#include "cutbyintensitydialog.h"
#include "specdataview.h"
specRemoveDataAction::specRemoveDataAction(QObject* parent)
        : specRequiresDataItemAction(parent)
{
        setIcon(QIcon(":/cbi.png")) ;
        setToolTip(tr("Remove␣data")) ;
        setWhatsThis(tr("Remove␣data␣from␣the␣selected␣items.␣␣You␣will␣be␣prompted␣to␣determined␣►
            which␣data␣points␣to␣delete␣by␣setting␣up␣ranges␣along␣the␣x␣axis."));
        setText(tr("Remove␣data␣by␣range...")) ;
        setShortcut(tr("Ctrl+r"));
}
const std::type_info& specRemoveDataAction::possibleParent()
{
        return typeid(specDataView) ;
}
specUndoCommand* specRemoveDataAction::generateUndoCommand()
{
        cutByIntensityDialog* dialog = new cutByIntensityDialog(parentWidget()) ;
        dialog->assignSpectra(pointers) ;
        if(dialog->exec() == QDialog::Rejected)
        {
                delete dialog ;
                return 0 ;
        }
        specMultiCommand* groupCommand = new specMultiCommand ;
        groupCommand->setParentObject(model) ;
        groupCommand->setMergeable(false) ;
        QList<specRange*> ranges = dialog->ranges() ;
        foreach(QModelIndex index, selection)
        {
                specDataItem* item = dynamic_cast<specDataItem*>(model->itemPointer(index)) ;
                if(!item) continue ;
                specExchangeDataCommand* command = new specExchangeDataCommand(groupCommand) ;
                command->setParentObject(model) ;
                command->setItem(item, item->getDataExcept(ranges)) ;
        }
        QStringList rangeStrings ;
        foreach(specRange * range, ranges)
        rangeStrings << QString::number(range->minValue()) + "--" + QString::number(range->maxValue►
            ()) ;
        groupCommand->setText(tr("Delete␣data␣points␣in␣ranges:␣") + rangeStrings.join(",␣") + tr(".►
            ")) ;
        delete dialog ;
        if(ranges.isEmpty())
        {
                delete groupCommand ;
                return 0 ;
        }
        return groupCommand ;
}
```

src/actionlib/actions/specremovefitaction.h

```
#ifndef SPECREMOVEFITACTION_H
#define SPECREMOVEFITACTION_H
#include "specfitaction.h"
class specRemoveFitAction : public specFitAction
{
        Q_OBJECT
public:
        explicit specRemoveFitAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specremovefitaction.cpp

```
#include "specremovefitaction.h"
#include "specmetaitem.h"
#include "specfitcurve.h"
#include "specexchangefitcurvecommand.h"
#include "specmulticommand.h"
specRemoveFitAction::specRemoveFitAction(QObject* parent) :
        specFitAction(parent)
{
        setIcon(QIcon(":/removeFit.png")) ;
        setToolTip(tr("Remove fit")) ;
        setWhatsThis(tr("Removes the fit from the current item.")) ;
        setText(tr("Remove fit")) ;
        setShortcut(tr("Ctrl+Shift+F"));
}
specUndoCommand* specRemoveFitAction::generateUndoCommand()
{
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setText(tr("Remove fit curve"));
        parentCommand->setParentObject(model) ;
        foreach(specModelItem * modelItem, pointers)
        {
                specMetaItem* item = dynamic_cast<specMetaItem*>(modelItem) ;
                if(!item) continue ;
                if(!item->getFitCurve()) continue;
                (new specExchangeFitCurveCommand(parentCommand))
                ->setup(item, 0) ;
        }
        if(!parentCommand->childCount())
        {
                delete parentCommand ;
                return 0 ;
        }
        return parentCommand ;
}
```

src/actionlib/actions/specrequiresitemaction.h

```
#ifndef SPECREQUIRESITEMACTION_H
#define SPECREQUIRESITEMACTION_H
#include "specitemaction.h"
class specRequiresItemAction : public specItemAction
{
        Q_OBJECT
public:
        explicit specRequiresItemAction(QObject* parent = 0);
protected:
        bool specificRequirements() ;
        bool postProcessingRequirements() const ;
private:
        QList<specStreamable::type> requiredTypes() const ;
};
class specRequiresMetaItemAction : public specRequiresItemAction
{
        Q_OBJECT
private:
```

```
        QList<specStreamable::type> requiredTypes() const ;
public:
        explicit specRequiresMetaItemAction(QObject* parent = 0) ;
};
class specRequiresDataItemAction : public specRequiresItemAction
{
        Q_OBJECT
private:
        QList<specStreamable::type> requiredTypes() const ;
public:
        explicit specRequiresDataItemAction(QObject* parent = 0) ;
};
#endif
```

src/actionlib/actions/specrequiresitemaction.cpp

```
#include "specrequiresitemaction.h"
specRequiresItemAction::specRequiresItemAction(QObject* parent) :
        specItemAction(parent)
{
}
specRequiresDataItemAction::specRequiresDataItemAction(QObject* parent)
        : specRequiresItemAction(parent)
{}
specRequiresMetaItemAction::specRequiresMetaItemAction(QObject* parent)
        : specRequiresItemAction(parent)
{}
bool specRequiresItemAction::specificRequirements()
{
        return view->selectionModel()->hasSelection() ;
}
bool specRequiresItemAction::postProcessingRequirements() const
{
        return !pointers.isEmpty() ;
}
QList<specStreamable::type> specRequiresItemAction::requiredTypes() const
{
        return QList<specStreamable::type>()
                << specStreamable::dataItem
                << specStreamable::folder
                << specStreamable::logEntry
                << specStreamable::sysEntry
                << specStreamable::metaItem
                << specStreamable::svgItem ;
}
QList<specStreamable::type> specRequiresDataItemAction::requiredTypes() const
{
        return QList<specStreamable::type>()
                << specStreamable::dataItem ;
}
QList<specStreamable::type> specRequiresMetaItemAction::requiredTypes() const
{
        return QList<specStreamable::type>()
                << specStreamable::metaItem ;
}
```

src/actionlib/actions/specselectconnectedaction.h

```
#ifndef SPECSELECTCONNECTEDACTION_H
#define SPECSELECTCONNECTEDACTION_H
#include "specconnectionsaction.h"
class specMetaView ;
class specSelectConnectedAction : public specConnectionsAction
{
        Q_OBJECT
private:
        specUndoCommand* generateUndoCommand() ;
public:
        explicit specSelectConnectedAction(QObject* parent = 0);
};
#endif
```

src/actionlib/actions/specselectconnectedaction.cpp

```
#include "specselectconnectedaction.h"
#include "specmetaview.h"
#include "specmetaitem.h"
#include "specviewstate.h"
specSelectConnectedAction::specSelectConnectedAction(QObject* parent) :
        specConnectionsAction(parent)
{
        setIcon(QIcon(":/fromKinetic.png"));
        setText(tr("Select␣connected␣items"));
        setWhatsThis(tr("Selects␣the␣items␣connected␣to␣the␣current␣meta␣item.")) ;
        setToolTip(tr("Select␣connected␣items"));
        setShortcut(tr("Ctrl+s"));
}
specUndoCommand* specSelectConnectedAction::generateUndoCommand()
{
        foreach(specModelItem * pointer, pointers)
        {
                specMetaItem* metaPointer = (specMetaItem*) pointer ;
                QModelIndexList dataItems, metaItems ;
                metaPointer->connectedItems(dataItems, metaItems) ;
                metaView->selectionModel()->select(specViewState::indexesToSelection(metaItems, (►
                    specModel*) metaView->model()), QItemSelectionModel::Select) ;
                dataView->selectionModel()->select(specViewState::indexesToSelection(dataItems, ►
                    metaView->getDataView()->model()), QItemSelectionModel::ClearAndSelect) ;
        }
        return 0 ;
}
```

src/actionlib/actions/specsetmultilineaction.h

```
#ifndef SPECSETMULTILINEACTION_H
#define SPECSETMULTILINEACTION_H
#include "specrequiresitemaction.h"
class specSetMultilineAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit specSetMultilineAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/specsetmultilineaction.cpp

```
#include "specsetmultilineaction.h"
#include "specdescriptorflagscommand.h"
specSetMultilineAction::specSetMultilineAction(QObject* parent) :
        specRequiresItemAction(parent)
{
        setIcon(QIcon(":/multiline.png")) ;
        setToolTip(tr("Toggle␣showing␣all␣lines"));
        setWhatsThis(tr("Switches␣showing␣all␣lines␣of␣this␣descriptor␣on␣or␣off."));
        setText(tr("Show␣all␣lines"));
        setCheckable(true) ;
        setShortcut(tr("Ctrl+M"));
}
specUndoCommand* specSetMultilineAction::generateUndoCommand()
{
        if(!currentIndex.isValid()) return 0 ;
        specDescriptorFlagsCommand* command = new specDescriptorFlagsCommand ;
        command->setParentObject(model) ;
        QString key = model->descriptors() [currentIndex.column()] ;
        foreach(specModelItem * item, pointers)
        command->addItem(item, key, item->descriptorProperties(key)
                        ^ spec::multiline) ;
        command->setText(tr("Toggle␣multiline")) ;
        return command ;
}
```

src/actionlib/actions/specspectrumcalculatoraction.h

```
#ifndef SPECSPECTRUMCALCULATORACTION_H
#define SPECSPECTRUMCALCULATORACTION_H
#include "specrequiresitemaction.h"
class spectrumCalculatorDialog ;
class specSpectrumCalculatorAction : public specRequiresDataItemAction
{
        Q_OBJECT
private:
        spectrumCalculatorDialog* dialog ;
public:
        explicit specSpectrumCalculatorAction(QObject* parent = 0);
        ~specSpectrumCalculatorAction() ;
        const std::type_info& possibleParent() ;
protected:
        specUndoCommand* generateUndoCommand() ;
public:
        static QStringList descriptorNames(QString& expression) ;
        static QString parameterName(int i) ;
};
#endif
```

src/actionlib/actions/specspectrumcalculatoraction.cpp

```
#include "specspectrumcalculatoraction.h"
#include <QList>
#include "specmodelitem.h"
#include "spectrumcalculatordialog.h"
#include "specmulticommand.h"
#include "specexchangedatacommand.h"
#include <muParser.h>
#include <QMessageBox>
#include "specdataview.h"
#include <QRegExp>
inline double modulo(double a, double b)
{
        return int (a) % int (b) ;
}
QString specSpectrumCalculatorAction::parameterName(int i)
{
        return "p" + QString::number(i) ;
}
QStringList specSpectrumCalculatorAction::descriptorNames(QString& teststring)
{
        QRegExp re("\"([^\"]|\\\\\")*\"") ;
        QStringList descriptorNames ;
        int i = 0 ;
        int index = 0 ;
        while((index = re.indexIn(teststring)) != -1)
        {
                descriptorNames << re.cap().mid(1, re.cap().length() - 2) ;
                teststring.remove(index, re.matchedLength()) ;
                teststring.insert(index, parameterName(i++)) ;
        }
        return descriptorNames ;
}
specSpectrumCalculatorAction::specSpectrumCalculatorAction(QObject* parent) :
        specRequiresDataItemAction(parent),
        dialog(new spectrumCalculatorDialog)
{
        setIcon(QIcon::fromTheme("accessories-calculator")) ;
        setToolTip(tr("Data␣calculator")) ;
        setText(tr("Data␣Calculator")) ;
        setWhatsThis(tr("Perform␣mathematical␣operations␣on␣x-␣and␣y-data."));
}
specSpectrumCalculatorAction::~specSpectrumCalculatorAction()
{
        delete dialog ;
}
const std::type_info& specSpectrumCalculatorAction::possibleParent()
{
        return typeid(specDataView) ;
}
```

```
specUndoCommand* specSpectrumCalculatorAction::generateUndoCommand()
{
        QList<specModelItem*> items, folders ;
        expandSelectedFolders(items, folders);
        if(items.isEmpty()) return 0 ;
        if(!dialog->exec()) return 0 ;
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setParentObject(model) ;
        parentCommand->setMergeable(false) ;
        mu::Parser xParser, yParser ;
        QString xFormula(dialog->xFormula()), yFormula(dialog->yFormula()) ;
        QStringList xDescriptors(descriptorNames(xFormula)), yDescriptors(descriptorNames(yFormula)) ;
        try
        {
                xParser.DefineOprt("%", modulo, 6);
                yParser.DefineOprt("%", modulo, 6);
                xParser.SetExpr(xFormula.toStdString()) ;
                yParser.SetExpr(yFormula.toStdString()) ;
        }
        catch(...)
        {
                QMessageBox::critical(0, tr("Failed␣evaluation"), tr("Failed␣parsing␣the␣given␣►
                    formulae.")) ;
                return 0;
        }
        int failCount = 0 ;
        foreach(specModelItem * item, items)
        {
                size_t count = item->dataSize() ;
                double* oldX = new double[count],
                *oldY = new double[count],
                *newX = new double[count],
                *newY = new double[count];
                for(size_t i = 0 ; i < count ; ++i)
                {
                        QPointF point = item->sample(i) ;
                        oldX[i] = point.x() ;
                        oldY[i] = point.y() ;
                }
                bool evaluationOk = false ;
                QVector<specDataPoint> newData ;
                try
                {
                        xParser.ClearVar();
                        yParser.ClearVar();
                        xParser.ClearConst();
                        yParser.ClearConst();
                        xParser.DefineVar("x", oldX) ;
                        yParser.DefineVar("x", oldX) ;
                        xParser.DefineVar("y", oldY) ;
                        yParser.DefineVar("y", oldY) ;
                        for(int i = 0 ; i < xDescriptors.size() ; ++i)
                                xParser.DefineConst(parameterName(i).toStdString(), item->►
                                    descriptorValue(xDescriptors[i])) ;
                        for(int i = 0 ; i < yDescriptors.size() ; ++i)
                                yParser.DefineConst(parameterName(i).toStdString(), item->►
                                    descriptorValue(yDescriptors[i])) ;
                        xParser.Eval(newX, count) ;
                        yParser.Eval(newY, count) ;
                        for(size_t i = 0 ; i < count ; ++i)
                                newData << specDataPoint(newX[i], newY[i], 0.) ;
                        evaluationOk = true ;
                }
                catch(...) {}
                delete[] oldX ;
                delete[] oldY ;
                delete[] newX ;
                delete[] newY ;
                failCount += !evaluationOk ;
                if(evaluationOk)
                {
                        if(specDataItem* dataItem = dynamic_cast<specDataItem*>(item))
                                dataItem->reverseCorrection(newData) ;
                        specExchangeDataCommand* command = new specExchangeDataCommand(parentCommand►
                            ) ;
```

```
                        command->setParentObject(model) ;
                        command->setItem(item, newData);
                }
        }
        if(failCount)
                QMessageBox::warning(0, tr("Failed to evaluate"), tr("Failed to evaluate new data ►
                    for ")
                                + QString::number(failCount)
                                + tr(" items. These items remained unchanged.")) ;
        parentCommand->setText(tr("Arithmetic operation on data. New x: ")
                        + dialog->xFormula() + ". New y: "
                        + dialog->yFormula() + ".");
        return parentCommand ;
}
```

src/actionlib/actions/spectiltmatrixaction.h

```
#ifndef SPECTILTMATRIXACTION_H
#define SPECTILTMATRIXACTION_H
#include "specrequiresitemaction.h"
class specExchangeDescriptorXDialog ;
class specTiltMatrixAction : public specRequiresDataItemAction
{
        Q_OBJECT
public:
        explicit specTiltMatrixAction(QObject* parent = 0);
        ~specTiltMatrixAction() ;
protected:
        specUndoCommand* generateUndoCommand() ;
private:
        specExchangeDescriptorXDialog* dialog ;
};
#endif
```

src/actionlib/actions/spectiltmatrixaction.cpp

```
#include "spectiltmatrixaction.h"
#include "specmulticommand.h"
#include "specdeleteaction.h"
#include <QSet>
#include "specdataitem.h"
#include <QMap>
#include "specaddfoldercommand.h"
#include "specdeletecommand.h"
#include "specexchangedescriptorxdialog.h"
specTiltMatrixAction::~specTiltMatrixAction()
{
        delete dialog ;
}
specTiltMatrixAction::specTiltMatrixAction(QObject* parent) :
        specRequiresDataItemAction(parent),
        dialog(new specExchangeDescriptorXDialog)
{
        setIcon(QIcon(":/exchangex.png")) ;
        setToolTip(tr("Exchange x data for descriptive field")) ;
        setWhatsThis(tr("Generates new items from the selected items' data, simultaneously turning ►
            the x-axis data into a description field and exchanging it for a different description ►
            field."));
        setText(tr("Exchange x...")) ;
        setShortcut(tr("Ctrl+j")) ;
}
class descriptorPreparationObject
{
        QSet<QString> values ;
        spec::descriptorFlags flagValue ;
public:
        void addDescriptor(const QString& value, const spec::descriptorFlags& Flags)
        {
                values << value ;
                flagValue &= Flags ;
        }
        QString descriptorValue() const
```

```
        {
                QStringList valueList(values.toList()) ;
                valueList.sort();
                return valueList.join("\n") ;
        }
        spec::descriptorFlags flags() const
        { return values.size() > 1 ? (flagValue & ~spec::numeric) : flagValue ; }
        specDescriptor descriptor() const
        { return specDescriptor(descriptorValue(), flags()) ; }
};
class dataItemPreparationObject
{
        QVector<specDataPoint> dataPoints ;
        QMap<QString, descriptorPreparationObject> description ;
public:
        void addDataPoint(double x, double y) { dataPoints << specDataPoint(x, y, 0) ; }
        void addDescriptor(const QString& key, const QString& value, const spec::descriptorFlags& ►
            flags)
        {
                description[key].addDescriptor(value, flags) ;
        }
        void addItem(const specModelItem* item, double x, double y)
        {
                addDataPoint(x, y) ;
                foreach(const QString & key, item->descriptorKeys())
                addDescriptor(key, item->descriptor(key, true), item->descriptorProperties(key));
        }
        specDataItem* dataItem() const
        {
                QHash<QString, specDescriptor> descriptionHash ;
                foreach(const QString & key, description.keys())
                descriptionHash[key] = description[key].descriptor() ;
                return new specDataItem(dataPoints, descriptionHash) ;
        }
};
specUndoCommand* specTiltMatrixAction::generateUndoCommand()
{
        QList<specModelItem*> items, toBeDeletedFolders ;
        expandSelectedFolders(items, toBeDeletedFolders);
        if(items.isEmpty()) return 0 ;
        dialog->setDescriptors(model->descriptors()) ;
        if(QDialog::Accepted != dialog->exec()) return 0 ;
        QString newDescriptor = dialog->newDescriptor(),
                conversionDescriptor = dialog->descriptorToConvert() ;
        QMap<double, dataItemPreparationObject> newData ;
        foreach(specModelItem * item, items)
        {
                item->revalidate();
                for(size_t i = 0 ; i < item->dataSize() ; ++i)
                        newData[item->sample(i).x()].addItem(item, item->descriptorValue(►
                            conversionDescriptor), item->sample(i).y()) ;
        }
        QList<specModelItem*> newItems ;
        foreach(const double & key, newData.keys())
        {
                newData[key].addDescriptor(newDescriptor,
                                           QString::number(key),
                                           spec::numeric);
                newItems << newData[key].dataItem() ;
        }
        foreach(specModelItem * item, newItems)
        item->deleteDescriptor(conversionDescriptor) ;
        specModelItem* parent = currentItem ;
        while(parent->parent())
        {
                if(toBeDeletedFolders.contains(parent))
                {
                        insertionRow = parent->parent()->childNo(parent) ;
                        insertionIndex = model->index(parent->parent()) ;
                }
                parent = parent->parent() ;
        }
        if(!model->insertItems(newItems, insertionIndex, insertionRow)) return 0 ;
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setParentObject(model) ;
```

```
        parentCommand->setMergeable(false) ;
        specAddFolderCommand* insertionCommand = new specAddFolderCommand(parentCommand) ;
        insertionCommand->setItems(newItems);
        items << toBeDeletedFolders ;
        specDeleteAction::command(model, items, parentCommand) ;
        parentCommand->setText(tr("Exchange␣\"") +
                               conversionDescriptor +
                               tr("\"␣and␣\"") +
                               newDescriptor +
                               tr("\"␣in␣") +
                               QString::number(newItems.size()) +
                               tr("␣items.")) ;
        return parentCommand ;
}
```

src/actionlib/actions/spectogglefitstyleaction.h

```
#ifndef SPECTOGGLEFITSTYLEACTION_H
#define SPECTOGGLEFITSTYLEACTION_H
#include "specfitaction.h"
class specToggleFitStyleAction : public specFitAction
{
        Q_OBJECT
public:
        explicit specToggleFitStyleAction(QObject* parent = 0);
private:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/spectogglefitstyleaction.cpp

```
#include "spectogglefitstyleaction.h"
#include "specmetaitem.h"
#include "spectogglefitstylecommand.h"
specToggleFitStyleAction::specToggleFitStyleAction(QObject* parent) :
        specFitAction(parent)
{
        setIcon(QIcon(":/styleFit.png")) ;
        setToolTip(tr("Switch␣styling␣to/from␣fit")) ;
        setWhatsThis(tr("Switches␣the␣styling␣options␣to␣affect␣the␣fit␣connected␣with␣this␣meta␣►
            item␣or␣the␣meta␣item␣itself.")) ;
        setText(tr("Switch␣styling␣to/from␣fit")) ;
}
specUndoCommand* specToggleFitStyleAction::generateUndoCommand()
{
        specToggleFitStyleCommand* command = new specToggleFitStyleCommand ;
        command->setText(tr("Toggle␣styling␣to␣fit")) ;
        command->setParentObject(model) ;
        command->setItem(currentItem) ;
        return command ;
}
```

src/actionlib/actions/spectreeaction.h

```
#ifndef SPECTREEACTION_H
#define SPECTREEACTION_H
#include "specrequiresitemaction.h"
class specTreeAction : public specRequiresItemAction
{
        Q_OBJECT
public:
        explicit specTreeAction(QObject* parent = 0);
protected:
        specUndoCommand* generateUndoCommand() ;
};
#endif
```

src/actionlib/actions/spectreeaction.cpp

```
#include "spectreeaction.h"
#include "specmulticommand.h"
#include "specaddfoldercommand.h"
#include "specmovecommand.h"
#include "specdeleteaction.h"
#include "specworkerthread.h"
class treeActionThread : public specWorkerThread
{
private:
        typedef QList<specModelItem*> itemPointerList ;
        specUndoCommand* Command ;
        specFolderItem* treeRoot ;
        specModel* model ;
        specModelItem* currentItem ;
        itemPointerList items, toBeDeletedFolders ;
        bool cleanUp()
        {
                if(!toTerminate) return false ;
                delete Command ;
                Command = 0 ;
                delete treeRoot ;
                treeRoot = 0 ;
                return true ;
        }
public:
        void run()
        {
                emit progressValue(0) ;
                typedef QPair<specFolderItem*, itemPointerList> destination ;
                typedef QList<destination> destinationList ;
                treeRoot = new specFolderItem(0, tr("Tree")) ;
                items = items.toSet().toList() ;
                int columnCount = model->columnCount(QModelIndex()) ;
                QStringList descriptors ;
                for(int i = 0 ; i < columnCount ; ++i)
                        descriptors << model->headerData(i, Qt::Horizontal).toString() ;
                destinationList folderQueue, finalPositions ;
                folderQueue << qMakePair(treeRoot, items) ;
                emit progressValue(5);
                foreach(QString descriptor, descriptors)
                {
                        if(cleanUp()) return ;
                        destinationList nextQueue ;
                        foreach(destination folder, folderQueue)
                        {
                                QMap<QString, itemPointerList > newCategories ;
                                foreach(specModelItem * item, folder.second)
                                newCategories[item->descriptor(descriptor, true)] += item ;
                                if(newCategories.size() < 2)
                                        nextQueue << folder ;
                                else
                                {
                                        destination thisParentsDirectChildren ;
                                        thisParentsDirectChildren.first = folder.first ;
                                        itemPointerList thisParentsFolderChildren ;
                                        foreach(const QString & newCategory, newCategories.keys())
                                        {
                                                if(newCategories[newCategory].size() == 1)
                                                {
                                                        thisParentsDirectChildren.second << ►
                                                             newCategories[newCategory] ;
                                                        continue ;
                                                }
                                                specFolderItem* newFolder = new specFolderItem(0, ►
                                                     newCategory) ;
                                                nextQueue << qMakePair(newFolder, newCategories[►
                                                     newCategory]) ;
                                                thisParentsFolderChildren << newFolder ;
                                        }
                                        finalPositions << thisParentsDirectChildren ;
                                        folder.first->addChildren(thisParentsFolderChildren) ;
                                }
                        }
```

```
                        folderQueue.swap(nextQueue) ;
                        emit progressValue(85 * descriptors.indexOf(descriptor) / descriptors.size());
73              }
                finalPositions << folderQueue ;
                emit progressValue(90);
                Command = new specMultiCommand ;
                Command->setText(tr("Form␣tree␣of␣items")) ;
78              Command->setParentObject(model) ;
                int row = 0 ;
                while(toBeDeletedFolders.contains(currentItem) || !currentItem->isFolder())
                {
                        row = currentItem->parent()->childNo(currentItem) ;
83                      currentItem = currentItem->parent() ;
                }
                if(! model->insertItems(QList<specModelItem*>() << treeRoot, model->index(►
                    currentItem), row))
                {
                        cleanUp();
88                      return ;
                }
                specAddFolderCommand* insertTree = new specAddFolderCommand(Command) ;
                insertTree->setParentObject(model) ;
                insertTree->setItem(treeRoot);
93              treeRoot = 0 ;
                specMoveCommand* moveCommand = new specMoveCommand(Command) ;
                moveCommand->setParentObject(model) ;
                foreach(destination d, finalPositions)
                {
98                      if(d.second.isEmpty()) continue ;
                        QModelIndexList indexes = model->indexList(d.second) ;
                        moveCommand->setItems(indexes, model->index(d.first), d.first->children());
                }
                specDeleteAction::command(model, toBeDeletedFolders, Command) ;
103             if(cleanUp()) return ;
                emit progressValue(100);
        }
        specUndoCommand* command()
        {
108             specUndoCommand* c = Command ;
                Command = 0 ;
                return c ;
        }
        treeActionThread(specModel* Model, const itemPointerList& Items, const itemPointerList& ►
            Folders, specModelItem* CurrentItem)
113             : specWorkerThread(100),
                  Command(0),
                  treeRoot(0),
                  model(Model),
                  currentItem(CurrentItem),
118               items(Items),
                  toBeDeletedFolders(Folders)
        {
        }
};
123 specTreeAction::specTreeAction(QObject* parent) :
        specRequiresItemAction(parent)
{
        setIcon(QIcon::fromTheme("view-list-tree")) ;
        setToolTip(tr("Set␣up␣a␣directory␣tree")) ;
128     setWhatsThis(tr("Generates␣a␣tree␣of␣directories␣and␣moves␣the␣selected␣items␣into␣those␣►
            directories.␣␣The␣columns␣in␣this␣dock␣window's␣list␣will␣be␣used␣from␣left␣to␣right␣to␣►
            establish␣the␣level␣of␣folders␣within␣the␣tree."));
        setText(tr("Build␣directory␣tree")) ;
        setShortcut(tr("Ctrl+T"));
}
specUndoCommand* specTreeAction::generateUndoCommand()
133 {
        QList<specModelItem*> items, folders ;
        expandSelectedFolders(items, folders) ;
        treeActionThread tat(model, items, folders, currentItem) ;
        tat.run();
138     return tat.command() ;
}
```

src/actionlib/actions/spectrumcalculatordialog.h

```
#ifndef SPECTRUMCALCULATORDIALOG_H
#define SPECTRUMCALCULATORDIALOG_H
#include <QDialog>
class QStringListModel ;
namespace Ui
{
        class spectrumCalculatorDialog;
}
class spectrumCalculatorDialog : public QDialog
{
        Q_OBJECT
public:
        explicit spectrumCalculatorDialog(QWidget* parent = 0);
        ~spectrumCalculatorDialog();
        QString xFormula() const ;
        QString yFormula() const ;
private slots:
        void on_toNm_clicked();
        void on_toAbsorption_clicked();
        void on_toTransmittance_clicked();
        void errorChanged(const QString&) ;
        void syncFormulae() const ;
private:
        Ui::spectrumCalculatorDialog* ui;
        QStringListModel* xFormulae, *yFormulae ;
        void syncFormula(const QString& formula, QStringListModel* model) const ;
};
#endif
```

src/actionlib/actions/spectrumcalculatordialog.cpp

```
#include "spectrumcalculatordialog.h"
#include "ui_spectrumcalculatordialog.h"
#include "specformulavalidator.h"
#include "QStringList"
#include <QCompleter>
#include <QStringListModel>
spectrumCalculatorDialog::spectrumCalculatorDialog(QWidget* parent) :
        QDialog(parent),
        ui(new Ui::spectrumCalculatorDialog),
        xFormulae(new QStringListModel(this)),
        yFormulae(new QStringListModel(this))
{
        ui->setupUi(this);
        QRegExp xy("(x|y|p\\d+)") ;
        specFormulaValidator* xValidator = new specFormulaValidator(xy, ui->formulaX) ;
        specFormulaValidator* yValidator = new specFormulaValidator(xy, ui->formulaY) ;
        ui->formulaX->setValidator(xValidator);
        ui->formulaY->setValidator(yValidator) ;
        connect(xValidator, SIGNAL(evaluationError(QString)), ui->errorsX, SLOT(setText(QString))) ;
        connect(yValidator, SIGNAL(evaluationError(QString)), ui->errorsY, SLOT(setText(QString))) ;
        connect(xValidator, SIGNAL(evaluationError(QString)), this, SLOT(errorChanged(QString))) ;
        connect(yValidator, SIGNAL(evaluationError(QString)), this, SLOT(errorChanged(QString))) ;
        connect(this, SIGNAL(accepted()), this, SLOT(syncFormulae())) ;
        ui->formulaX->setCompleter(new QCompleter(xFormulae, ui->formulaX)) ;
        ui->formulaY->setCompleter(new QCompleter(yFormulae, ui->formulaY));
        ui->formulaX->setText("x") ;
        ui->formulaY->setText("y");
}
void spectrumCalculatorDialog::syncFormulae() const
{
        syncFormula(ui->formulaX->text(), xFormulae) ;
        syncFormula(ui->formulaY->text(), yFormulae) ;
}
void spectrumCalculatorDialog::syncFormula(const QString& formula, QStringListModel* model) const
{
        QStringList formulae = model->stringList() ;
        if(formulae.contains(formula)) return ;
        formulae << formula ;
        model->setStringList(formulae) ;
}
spectrumCalculatorDialog::~spectrumCalculatorDialog()
```

```
{
        delete ui;
}
void spectrumCalculatorDialog::on_toNm_clicked()
{
        ui->formulaX->setText("1e7/x") ;
}
void spectrumCalculatorDialog::on_toAbsorption_clicked()
{
        ui->formulaY->setText("2-log(y)");
}
void spectrumCalculatorDialog::on_toTransmittance_clicked()
{
        ui->formulaY->setText("10^-y");
}
QString spectrumCalculatorDialog::xFormula() const
{
        return ui->formulaX->text() ;
}
QString spectrumCalculatorDialog::yFormula() const
{
        return ui->formulaY->text() ;
}
void spectrumCalculatorDialog::errorChanged(const QString& error)
{
        ui->buttonBox->button(QDialogButtonBox::Ok)->setEnabled(error.isEmpty());
}
```

src/actionlib/actions/spectrumcalculatordialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>spectrumCalculatorDialog</class>
 <widget class="QDialog" name="spectrumCalculatorDialog">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>356</width>
    <height>167</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Recalculate</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_3">
   <item>
    <layout class="QHBoxLayout" name="horizontalLayout">
     <item>
      <layout class="QVBoxLayout" name="verticalLayout">
       <item>
        <widget class="QLabel" name="formulaXLabel">
         <property name="text">
          <string>Formula for x values:</string>
         </property>
         <property name="buddy">
          <cstring>formulaX</cstring>
         </property>
        </widget>
       </item>
       <item>
        <widget class="QLineEdit" name="formulaX"/>
       </item>
       <item>
        <widget class="QLabel" name="errorsX">
         <property name="text">
          <string/>
         </property>
        </widget>
       </item>
       <item>
        <widget class="QLabel" name="formulaYLabel">
         <property name="text">
          <string>Formula for y values:</string>
```

```
          </property>
          <property name="buddy">
47         <cstring>formulaY</cstring>
          </property>
         </widget>
        </item>
        <item>
52       <widget class="QLineEdit" name="formulaY"/>
        </item>
        <item>
         <widget class="QLabel" name="errorsY">
          <property name="text">
57         <string/>
          </property>
         </widget>
        </item>
       </layout>
62    </item>
      <item>
       <layout class="QVBoxLayout" name="verticalLayout_2">
        <item>
         <widget class="QPushButton" name="toNm">
67        <property name="text">
           <string>nm/cm-1 conversion</string>
          </property>
         </widget>
        </item>
72      <item>
         <spacer name="verticalSpacer">
          <property name="orientation">
           <enum>Qt::Vertical</enum>
          </property>
77        <property name="sizeHint" stdset="0">
           <size>
            <width>20</width>
            <height>40</height>
           </size>
82        </property>
         </spacer>
        </item>
        <item>
         <widget class="QPushButton" name="toAbsorption">
87        <property name="text">
           <string>To absorption</string>
          </property>
         </widget>
        </item>
92      <item>
         <widget class="QPushButton" name="toTransmittance">
          <property name="text">
           <string>To transmittance</string>
          </property>
97        </widget>
        </item>
       </layout>
      </item>
     </layout>
102  </item>
     <item>
      <widget class="QDialogButtonBox" name="buttonBox">
       <property name="orientation">
        <enum>Qt::Horizontal</enum>
107    </property>
       <property name="standardButtons">
        <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
       </property>
      </widget>
112  </item>
    </layout>
   </widget>
   <tabstops>
    <tabstop>formulaX</tabstop>
117 <tabstop>toNm</tabstop>
    <tabstop>formulaY</tabstop>
    <tabstop>toAbsorption</tabstop>
```

```
  <tabstop>toTransmittance</tabstop>
  <tabstop>buttonBox</tabstop>
 </tabstops>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>spectrumCalculatorDialog</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>222</x>
     <y>152</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>166</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>spectrumCalculatorDialog</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>290</x>
     <y>158</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>166</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/actionlib/actions/specundoaction.h

```
#ifndef SPECUNDOACTION_H
#define SPECUNDOACTION_H
#include <QAction>
#include <typeinfo>
#include "specview.h"
#include "specplot.h"
#include "specactionlibrary.h"
class specUndoAction : public QAction
{
        Q_OBJECT
public:
        explicit specUndoAction(QObject* parent = 0);
        virtual const std::type_info& possibleParent() = 0;
        void setLibrary(specActionLibrary*) ;
protected:
        specActionLibrary* library ;
        virtual void execute() = 0 ;
        bool event(QEvent *) ;
private slots:
        void gotTrigger() ;
};
#endif
```

src/actionlib/actions/specundoaction.cpp

```
#include "specundoaction.h"
#include <QEvent>
specUndoAction::specUndoAction(QObject* parent) :
        QAction(parent)
{
```

```
        setShortcutContext(Qt::WidgetShortcut);
        if(QWidget* w = qobject_cast<QWidget*> (parent))
                w->addAction(this);
}
void specUndoAction::gotTrigger()
{
        if(!parent()) return ;
        execute() ;
}
void specUndoAction::setLibrary(specActionLibrary* lib)
{
        if(lib == 0)
                disconnect(this, SIGNAL(triggered()), this, SLOT(gotTrigger())) ;
        else
                connect(this, SIGNAL(triggered()), this, SLOT(gotTrigger())) ;
        library = lib ;
}
bool specUndoAction::event(QEvent *e)
{
        if (e->type() == QEvent::ActionChanged)
        {
                QString description = toolTip().section("\t",0,0) ;
                if (!shortcut().isEmpty())
                        description += QString("\t\t<br><small>") + shortcut().toString() + "</small►
                            >";
                setToolTip(description) ;
        }
        return QAction::event(e) ;
}
```

src/actionlib/commands/specaddconnectionscommand.h

```
#ifndef SPECADDCONNECTIONSCOMMAND_H
#define SPECADDCONNECTIONSCOMMAND_H
#include "specmanageconnectionscommand.h"
class specAddConnectionsCommand : public specManageConnectionsCommand
{
private:
        void doIt();
        void undoIt();
        type typeId() const { return specStreamable::newConnectionsCommandId ; }
        void processServers(specMetaItem* client, QList<specModelItem*>& servers) const ;
public:
        specAddConnectionsCommand(specUndoCommand* parent = 0);
};
#endif
```

src/actionlib/commands/specaddconnectionscommand.cpp

```
#include "specaddconnectionscommand.h"
#include "specmetaitem.h"
specAddConnectionsCommand::specAddConnectionsCommand(specUndoCommand* parent)
        : specManageConnectionsCommand(parent)
{
}
void specAddConnectionsCommand::doIt()
{
        restore();
}
void specAddConnectionsCommand::undoIt()
{
        take();
}
void specAddConnectionsCommand::processServers(specMetaItem* client, QList<specModelItem*>& servers)►
     const
{
        servers = (servers.toSet() - client->serverList().toSet()).toList() ;
}
```

src/actionlib/commands/specaddfoldercommand.h

```
#ifndef SPECADDFOLDERCOMMAND_H
#define SPECADDFOLDERCOMMAND_H
#include "specmanageitemscommand.h"
#include "specmodel.h"
#include "specgenealogy.h"
#include "names.h"
class specAddFolderCommand : public specManageItemsCommand
{
private:
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::newFolderCommandId ;}
public:
        specAddFolderCommand(specUndoCommand* parent = 0);
};
#endif
```

src/actionlib/commands/specaddfoldercommand.cpp

```
#include "specaddfoldercommand.h"
specAddFolderCommand::specAddFolderCommand(specUndoCommand* parent)
        : specManageItemsCommand(parent)
{
}
void specAddFolderCommand::doIt()
{
        restoreItems();
}
void specAddFolderCommand::undoIt()
{
        takeItems();
}
```

src/actionlib/commands/specdeletecommand.h

```
#ifndef SPECDELETECOMMAND_H
#define SPECDELETECOMMAND_H
#include "specmodel.h"
#include "specmanageitemscommand.h"
#include <QVector>
#include <QPair>
#include <QHash>
#include "specgenealogy.h"
#include "names.h"
class specDeleteCommand : public specManageItemsCommand
{
private:
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::deleteCommandId; }
public:
        explicit specDeleteCommand(specUndoCommand* parent = 0);
};
#endif
```

src/actionlib/commands/specdeletecommand.cpp

```
#include "specdeletecommand.h"
specDeleteCommand::specDeleteCommand(specUndoCommand* parent)
        : specManageItemsCommand(parent)
{
}
void specDeleteCommand::doIt()
{
        takeItems();
}
void specDeleteCommand::undoIt()
{
        restoreItems(); ;
}
```

src/actionlib/commands/specdeleteconnectionscommand.h

```
#ifndef SPECDELETECONNECTIONSCOMMAND_H
#define SPECDELETECONNECTIONSCOMMAND_H
#include "specmanageconnectionscommand.h"
class specDeleteConnectionsCommand : public specManageConnectionsCommand
{
private:
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::deleteConnectionsCommandId ; }
        void processServers(specMetaItem* client, QList<specModelItem*>& servers) const ;
public:
        specDeleteConnectionsCommand(specUndoCommand* parent) ;
};
#endif
```

src/actionlib/commands/specdeleteconnectionscommand.cpp

```
#include "specdeleteconnectionscommand.h"
#include "specmetaitem.h"
specDeleteConnectionsCommand::specDeleteConnectionsCommand(specUndoCommand* parent)
        : specManageConnectionsCommand(parent)
{
}
void specDeleteConnectionsCommand::doIt()
{
        take() ;
}
void specDeleteConnectionsCommand::undoIt()
{
        restore() ;
}
void specDeleteConnectionsCommand::processServers(specMetaItem* client, QList<specModelItem*>& ►
    servers) const
{
        servers = (servers.toSet() & (client->serverList().toSet())).toList() ;
}
```

src/actionlib/commands/specdeletedescriptorcommand.h

```
#ifndef SPECDELETEDESCRIPTORCOMMAND_H
#define SPECDELETEDESCRIPTORCOMMAND_H
#include "specundocommand.h"
#include "specdescriptor.h"
class specDeleteDescriptorCommand : public specUndoCommand
{
private:
        QList<specDescriptor> contents ;
        QString key ;
        qint16 position ;
        spec::descriptorFlags flags ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::deleteDescriptorCommandId ; }
        void parentAssigned();
public:
        specDeleteDescriptorCommand(specUndoCommand* parent = 0, QString key = "");
};
#endif
```

src/actionlib/commands/specdeletedescriptorcommand.cpp

```
#include "specdeletedescriptorcommand.h"
#include "specmodel.h"
specDeleteDescriptorCommand::specDeleteDescriptorCommand(specUndoCommand* parent, QString k)
        : specUndoCommand(parent),
          key(k),
```

```
          position(0)
{
}
void specDeleteDescriptorCommand::writeCommand(QDataStream& out) const
{
        out << position << (qint8) flags << key << contents ;
}
void specDeleteDescriptorCommand::readCommand(QDataStream& in)
{
        qint8 f ;
        in >> position >> f >> key >> contents ;
        flags = (spec::descriptorFlags) f ;
}
#define DELETEDESCRIPTORFUNCTIONMACRO(fname,code) void specDeleteDescriptorCommand::fname() \
        { \
                specModel* myModel = qobject_cast<specModel*>(parentObject()) ; \
                if (!myModel) return ; \
                myModel->signalBeginReset(); \
                code ; \
        }
DELETEDESCRIPTORFUNCTIONMACRO(doIt, myModel->deleteDescriptor(key))
DELETEDESCRIPTORFUNCTIONMACRO(undoIt, QListIterator<specDescriptor> it(contents) ; myModel->►
    restoreDescriptor(position, flags, it, key))
DELETEDESCRIPTORFUNCTIONMACRO(parentAssigned, if(contents.isEmpty()) \
{
\
myModel->dumpDescriptor(contents, key) ; \
        position = myModel->descriptors().indexOf(key) ; \
        flags = myModel->descriptorProperties() [position] ; \
})
```

src/actionlib/commands/specdescriptorflagscommand.h

```
#ifndef SPECDESCRIPTORFLAGSCOMMAND_H
#define SPECDESCRIPTORFLAGSCOMMAND_H
#include "specundocommand.h"
#include "specgenealogy.h"
class specDescriptorFlagsCommand : public specUndoCommand
{
private:
        typedef QPair<specGenealogy, qint8> itemPropertyPair ;
        typedef QVector<itemPropertyPair> itemsContainer ;
        itemsContainer items ;
        QString key ;
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        type typeId() const { return specStreamable::descriptorFlagsCommand ; }
        void parentAssigned();
public:
        explicit specDescriptorFlagsCommand(specUndoCommand* parent = 0) ;
        void setItems(QList<specModelItem*>&, QString key, spec::descriptorFlags f) ;
        void addItem(specModelItem* item, QString Key, spec::descriptorFlags f) ;
};
#endif
```

src/actionlib/commands/specdescriptorflagscommand.cpp

```
#include "specdescriptorflagscommand.h"
specDescriptorFlagsCommand::specDescriptorFlagsCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
void specDescriptorFlagsCommand::setItems(QList<specModelItem*>& list, QString Key, spec::►
    descriptorFlags f)
{
        key = Key ;
        items.clear();
        if(list.isEmpty()) return ;
        foreach(specModelItem * item, list)
        items << itemPropertyPair(specGenealogy(item, model()), f) ;
```

```
}
void specDescriptorFlagsCommand::addItem(specModelItem* item, QString Key, spec::descriptorFlags f)
{
        if(!item) return ;
        key = Key ;
        items << itemPropertyPair(specGenealogy(item, model()), f) ;
}
void specDescriptorFlagsCommand::undoIt()
{
        doIt() ;
}
void specDescriptorFlagsCommand::doIt()
{
        for(itemsContainer::iterator i = items.begin() ; i != items.end() ; ++i)
        {
                specModelItem* item = i->first.firstItem() ;
                if(!item) continue ;
                spec::descriptorFlags temp = item->descriptorProperties(key) ;
                item->setDescriptorProperties(key, spec::descriptorFlags(i->second)) ;
                i->second = temp ;
                if(specModel* model = i->first.model())
                        model->signalChanged(model->index(item, model->descriptors().indexOf(key))) ;
        }
}
void specDescriptorFlagsCommand::parentAssigned()
{
        specModel* model = qobject_cast<specModel*> (parentObject()) ;
        for(itemsContainer::iterator i = items.begin() ; i != items.end() ; ++i)
                i->first.setModel(model) ;
}
void specDescriptorFlagsCommand::writeCommand(QDataStream& out) const
{
        out << items << key ;
}
void specDescriptorFlagsCommand::readCommand(QDataStream& in)
{
        in >> items >> key ;
}
```

src/actionlib/commands/speceditdescriptorcommand.h

```
#ifndef SPECEDITDESCRIPTORCOMMAND_H
#define SPECEDITDESCRIPTORCOMMAND_H
#include "specsingleitemcommand.h"
class specEditDescriptorCommand : public specSingleItemCommand<specModelItem>
{
private:
        QStringList previousContent ;
        QString descriptor ;
        QVector<int> previousActiveLine ;
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        type typeId() const {return specStreamable::editDescriptorCommandId ;}
        QString description() const ;
        void generateDescription() ;
public:
        explicit specEditDescriptorCommand(specUndoCommand* parent = 0) ;
        void setItem(specModelItem*, QString descriptor,
                     QString newContent, int activeLine = 0) ;
};
#endif
```

src/actionlib/commands/speceditdescriptorcommand.cpp

```
#include "speceditdescriptorcommand.h"
#include <QQueue>
specEditDescriptorCommand::specEditDescriptorCommand(specUndoCommand* parent)
        : specSingleItemCommand(parent)
{
}
```

```
void specEditDescriptorCommand::setItem(specModelItem* i, QString desc,
                                        QString newContent, int activeLine)
{
        previousContent.clear();
        previousActiveLine.clear();
        previousContent << newContent ;
        previousActiveLine << activeLine ;
        if(!i) return ;
        specSingleItemCommand::setItem(i) ;
        descriptor = desc ;
        generateDescription();
}
void specEditDescriptorCommand::undoIt()
{
        doIt() ;
}
void specEditDescriptorCommand::doIt()
{
        specModelItem* pointer = itemPointer() ;
        if(!pointer || previousActiveLine.isEmpty() || previousContent.isEmpty()) return ;
        QVector<int>::iterator previousActiveLineIterator(previousActiveLine.begin()) ;
        QStringList::iterator previousContentIterator(previousContent.begin()) ;
        QStringList newContent ;
        QVector<int> newActiveLines ;
        QQueue<specModelItem*> items ;
        items.enqueue(pointer) ;
        while(!items.isEmpty())
        {
                pointer = items.dequeue() ;
                QString currentContent = pointer->descriptor(descriptor, true) ;
                int currentLine = pointer->activeLine(descriptor) ;
                if(pointer->changeDescriptor(descriptor, *previousContentIterator))
                {
                        pointer->setActiveLine(descriptor, *previousActiveLineIterator) ;
                        newContent << currentContent ;
                        newActiveLines << currentLine ;
                        if(previousActiveLineIterator + 1 != previousActiveLine.end())
                                ++ previousActiveLineIterator;
                        if(previousContentIterator + 1 != previousContent.end())
                                ++ previousContentIterator ;
                }
                else if(pointer->isFolder())
                {
                        specFolderItem* folder = dynamic_cast<specFolderItem*>(pointer) ;
                        if(!folder) continue ;
                        for(int i = 0 ; i < folder->children() ; ++i)
                                items.enqueue(folder->child(i)) ;
                }
        }
        specModel* model = qobject_cast<specModel*> (parentObject()) ;
        if(model)
                model->signalChanged(model->index(itemPointer())) ;
        previousContent.swap(newContent) ;
        previousActiveLine.swap(newActiveLines);
}
void specEditDescriptorCommand::writeCommand(QDataStream& out) const
{
        out << previousContent
            << descriptor
            << previousActiveLine ;
        writeItem(out);
}
void specEditDescriptorCommand::readCommand(QDataStream& in)
{
        in >> previousContent
           >> descriptor
           >> previousActiveLine ;
        readItem(in);
        generateDescription();
}
QString specEditDescriptorCommand::description() const
{
        return QString() ;
}
void specEditDescriptorCommand::generateDescription()
```

```
{
        setText(QObject::tr("Changed␣entry␣in␣\"")
                + descriptor
                + QObject::tr("\".")) ;
}
```

src/actionlib/commands/specexchangedatacommand.h

```
#ifndef SPECEXCHANGEDATACOMMAND_H
#define SPECEXCHANGEDATACOMMAND_H
#include "specsingleitemcommand.h"
#include "specdataitem.h"
class specExchangeDataCommand : public specSingleItemCommand<specDataItem>
{
private:
        QVector<specDataPoint> data ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in);
        void doIt();
        void undoIt() ;
        type typeId() const { return specStreamable::exchangeDataCommandId ; }
public:
        explicit specExchangeDataCommand(specUndoCommand* parent = 0) ;
        void setItem(specModelItem*, const QVector<specDataPoint>& newData) ;
};
#endif
```

src/actionlib/commands/specexchangedatacommand.cpp

```
#include "specexchangedatacommand.h"
#include "qwt_plot.h"
specExchangeDataCommand::specExchangeDataCommand(specUndoCommand* parent)
        : specSingleItemCommand(parent)
{
}
void specExchangeDataCommand::setItem(specModelItem* p, const QVector<specDataPoint>& newData)
{
        data = newData ;
        specSingleItemCommand::setItem(p) ;
}
void specExchangeDataCommand::undoIt()
{
        doIt() ;
}
void specExchangeDataCommand::doIt()
{
        specDataItem* pointer = itemPointer() ;
        if(!pointer) return ;
        pointer->swapData(data);
}
void specExchangeDataCommand::writeCommand(QDataStream& out) const
{
        out << data ;
        writeItem(out) ;
}
void specExchangeDataCommand::readCommand(QDataStream& in)
{
        in >> data ;
        readItem(in) ;
}
```

src/actionlib/commands/specexchangefiltercommand.h

```
#ifndef SPECEXCHANGEFILTERCOMMAND_H
#define SPECEXCHANGEFILTERCOMMAND_H
#include "specsingleitemcommand.h"
#include "specmodelitem.h"
#include "specdatapointfilter.h"
class specDataItem ;
class specExchangeFilterCommand : public specSingleItemCommand<specModelItem>
```

```
{
private:
        specDataPointFilter relativeFilter ;
        QVector<specDataPointFilter> absoluteFilters ;
        bool relative ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        QVector<specDataPointFilter> collectFilters(const QVector<specDataItem*>& items) const ;
        void applyAbsoluteFilters(const QVector<specDataItem*>& items) const ;
        type typeId() const ;
        void generateDescription() ;
        QString description() const;
        void undoIt() ;
        void doIt() ;
public:
        explicit specExchangeFilterCommand(specUndoCommand* parent = 0, bool relative = false) ;
        bool mergeWith(const QUndoCommand* other) ;
        void setRelativeFilter(const specDataPointFilter& filter) ;
        void setAbsoluteFilter(const specDataPointFilter& filter) ;
        bool mergeable(const specUndoCommand* other) ;
};
#endif
```

src/actionlib/commands/specexchangefiltercommand.cpp

```
#include "specexchangefiltercommand.h"
#include "specmulticommand.h"
#include "specdataitem.h"
specExchangeFilterCommand::specExchangeFilterCommand(specUndoCommand* parent, bool rel)
        : specSingleItemCommand<specModelItem> (parent),
          relative(rel)
{}
QVector<specDataPointFilter> specExchangeFilterCommand::collectFilters(const QVector<specDataItem*>&►
      items) const
{
        QVector<specDataPointFilter> filters ;
        foreach(specDataItem * item, items)
        filters << item->dataFilter() ;
        return filters ;
}
void specExchangeFilterCommand::applyAbsoluteFilters(const QVector<specDataItem*>& items) const
{
        for(int i = 0 ; i < qMin(absoluteFilters.size(), items.size()) ; ++i)
                items[i]->setDataFilter(absoluteFilters[i]) ;
}
void specExchangeFilterCommand::doIt()
{
        specModelItem* item = itemPointer() ;
        if(!item) return ;
        QVector<specDataItem*> items = item->findDescendants<specDataItem*>() ;
        QVector<specDataPointFilter> oldFilters = collectFilters(items) ;
        if(relative)
                item->addDataFilter(relativeFilter) ;
        else
                applyAbsoluteFilters(items);
        absoluteFilters.swap(oldFilters) ;
        relative = false ;
        generateDescription();
}
void specExchangeFilterCommand::undoIt()
{
        if(relative)
                relativeFilter = -relativeFilter ;
        doIt() ;
}
void specExchangeFilterCommand::writeCommand(QDataStream& out) const
{
        if(relative)
                out << relativeFilter.getSlope()
                    << relativeFilter.getOffset()
                    << relativeFilter.getFactor()
                    << relativeFilter.getXShift() ;
        else
                out << absoluteFilters ;
```

```
        writeItem(out) ;
}
void specExchangeFilterCommand::readCommand(QDataStream& in)
{
        if(relative)
        {
                double slope = 0, offset = 0, scale = 1, shift = 0 ;
                in >> slope >> offset >> scale >> shift ;
                relativeFilter = specDataPointFilter(offset, slope, scale, shift) ;
        }
        else
                in >> absoluteFilters ;
        readItem(in) ;
        generateDescription();
}
QString specExchangeFilterCommand::description() const
{
        return QString() ;
}
bool specExchangeFilterCommand::mergeable(const specUndoCommand* other)
{
        return !relative
               && !(((specExchangeFilterCommand*) other)->relative)
               && (((specExchangeFilterCommand*) other)->itemPointer()) == itemPointer() ;
}
bool specExchangeFilterCommand::mergeWith(const QUndoCommand* ot)
{
        if(!parentObject()) return false ;
        const specExchangeFilterCommand* other = (const specExchangeFilterCommand*) ot ;
        if(!mergeable(other)) return false ;
        generateDescription();
        return true ;
}
void specExchangeFilterCommand::setRelativeFilter(const specDataPointFilter& f)
{
        relative = true ;
        relativeFilter = f ;
}
void specExchangeFilterCommand::setAbsoluteFilter(const specDataPointFilter& filter)
{
        relative = false ;
        specModelItem* item = itemPointer() ;
        absoluteFilters = (item ?
                           QVector<specDataPointFilter> (item->findDescendants<specDataItem*>().size►
                                (), filter)
                           : QVector<specDataPointFilter> (1, filter)) ;
}
void specExchangeFilterCommand::generateDescription()
{
        if(relative)
                setText(QObject::tr("Add␣filter.") + relativeFilter.description()) ;
        else
        {
                if(absoluteFilters.size() == 1)
                        setText(QObject::tr("Exchange␣filter.␣") + absoluteFilters.first().►
                             description()) ;
                else
                        setText(QObject::tr("Exchange␣") + QString::number(absoluteFilters.size()) +►
                              QObject::tr("␣filters.")) ;
        }
}
specStreamable::type specExchangeFilterCommand::typeId() const
{
        return relative ?
               specStreamable::movePlotCommandId
               : specStreamable::exchangeFilterCommandId ;
}
```

src/actionlib/commands/specexchangefitcurvecommand.h

```
#ifndef SPECEXCHANGEFITCURVECOMMAND_H
#define SPECEXCHANGEFITCURVECOMMAND_H
#include "specsingleitemcommand.h"
#include "specmetaitem.h"
```

```
class specFitCurve ;
class specExchangeFitCurveCommand : public specSingleItemCommand<specMetaItem>
{
private:
        specFitCurve* curve ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::exchangeFitCommand ; }
public:
        explicit specExchangeFitCurveCommand(specUndoCommand* parent = 0) ;
        void setup(specModelItem*, specFitCurve*) ;
        ~specExchangeFitCurveCommand() ;
};
#endif
```

src/actionlib/commands/specexchangefitcurvecommand.cpp

```
#include "specexchangefitcurvecommand.h"
#include "specfitcurve.h"
specExchangeFitCurveCommand::specExchangeFitCurveCommand(specUndoCommand* parent)
        : specSingleItemCommand(parent),
          curve(0)
{
}
void specExchangeFitCurveCommand::setup(specModelItem* i, specFitCurve* c)
{
        if(curve) delete curve ;
        curve = c ;
        setItem(i) ;
}
void specExchangeFitCurveCommand::undoIt()
{
        doIt() ;
}
void specExchangeFitCurveCommand::doIt()
{
        specMetaItem* pointer = itemPointer() ;
        if(!pointer) return ;
        model()->signalBeginReset();
        curve = pointer->setFitCurve(curve);
        if(curve) curve->detach();
        model()->signalChanged(model()->index(itemPointer()));
}
void specExchangeFitCurveCommand::writeCommand(QDataStream& out) const
{
        out << quint8((bool) curve) ;
        writeItem(out);
        if(curve) out << *curve ;
}
void specExchangeFitCurveCommand::readCommand(QDataStream& in)
{
        delete curve ;
        quint8 hasCurve(0) ;
        in >> hasCurve ;
        readItem(in) ;
        if(hasCurve)
        {
                curve = new specFitCurve ;
                in >> *curve ;
        }
        else
                curve = 0 ;
}
specExchangeFitCurveCommand::~specExchangeFitCurveCommand()
{
        delete curve ;
}
```

src/actionlib/commands/specitemcommand.h

```
#ifndef SPECITEMCOMMAND_H
#define SPECITEMCOMMAND_H
#include "specundocommand.h"
#include "specgenealogy.h"
class specMultipleItemCommand : public specUndoCommand
{
protected:
        void writeCommand(QDataStream& out) ;
        virtual void setItems(QList<specModelItem*>&) ;
public:
        specMultipleItemCommand(specUndoCommand* parent = 0) ;
        virtual ~specMultipleItemCommand() ;
};
class specSingleItemCommand : public specUndoCommand
{
protected:
        specGenealogy item ;
        void writeCommand(QDataStream& out) ;
        virtual void setItem(specModelItem*) ;
public:
        specSingleItemCommand(specUndoCommand *parent = 0);
        virtual ~specSingleItemCommand() ;
};
#endif
```

src/actionlib/commands/specitemcommand.cpp

```
#include "specitemcommand.h"
specSingleItemCommand::specSingleItemCommand(specUndoCommand *parent)
        : specUndoCommand(parent)
{
}
```

src/actionlib/commands/specmanageconnectionscommand.h

```
#ifndef SPECMANAGECONNECTIONSCOMMAND_H
#define SPECMANAGECONNECTIONSCOMMAND_H
#include "specmultipleitemcommand.h"
class specMetaModel ;
class specManageConnectionsCommand : public specMultipleItemCommand
{
private:
        specGenealogy target ;
        specMetaItem* targetPointer() ;
        specMetaModel* targetModel ;
        bool sameModel ;
        void writeCommand(QDataStream& out) const;
        void readCommand(QDataStream& in) ;
        void parentAssigned();
protected:
        virtual void processServers(specMetaItem* client, QList<specModelItem*>& servers) const = 0 ;
        void take() ;
        void restore() ;
public:
        explicit specManageConnectionsCommand(specUndoCommand* parent = 0);
        virtual void setItems(specMetaItem* client, QList<specModelItem*> servers) ;
};
#endif
```

src/actionlib/commands/specmanageconnectionscommand.cpp

```
#include "specmanageconnectionscommand.h"
#include "specmetaitem.h"
#include "specmetamodel.h"
specManageConnectionsCommand::specManageConnectionsCommand(specUndoCommand* parent)
        : specMultipleItemCommand(parent),
          targetModel(0)
{
}
void specManageConnectionsCommand::setItems(specMetaItem* client, QList<specModelItem*> servers)
```

```
{
        if(!parentObject()) return ;
        if(!targetModel)
                setParentObject(parentObject());
        processServers(client, servers) ;
        target = specGenealogy(client, targetModel) ;
        qSort(servers.begin(), servers.end(), specModel::lessThanItemPointer) ;
        specMultipleItemCommand::setItems(servers) ;
}
void specManageConnectionsCommand::restore()
{
        specMetaItem* client = targetPointer() ;
        foreach(specModelItem * pointer, itemPointers())
        client->connectServer(pointer);
}
void specManageConnectionsCommand::take()
{
        specMetaItem* client = targetPointer() ;
        foreach(specModelItem * pointer, itemPointers())
        pointer->disconnectClient(client) ;
}
void specManageConnectionsCommand::writeCommand(QDataStream& out) const
{
        out << itemCount() << target ;
        writeItems(out) ;
}
void specManageConnectionsCommand::readCommand(QDataStream& in)
{
        qint32 toRead ;
        in >> toRead >> target ;
        readItems(in, toRead) ;
}
void specManageConnectionsCommand::parentAssigned()
{
        if(!parentObject()) return ;
        targetModel = qobject_cast<specMetaModel*> (parentObject()) ;
        if(targetModel)
        {
                setParentObject(targetModel->getDataModel()) ;
                return ;
        }
        targetModel = model()->getMetaModel() ;
        target.setModel(targetModel);
        specMultipleItemCommand::parentAssigned() ;
}
specMetaItem* specManageConnectionsCommand::targetPointer()
{
        return (specMetaItem*)(target.firstItem()) ;
}
```

src/actionlib/commands/specmanagedatacommand.h

```
#ifndef SPECMANAGEDATACOMMAND_H
#define SPECMANAGEDATACOMMAND_H
#include "specundocommand.h"
#include "specgenealogy.h"
#include "specmodel.h"
#include "specdataitem.h"
class specManageDataCommand : public specUndoCommand
{
private:
        specGenealogy *item ;
        QVector<int> toTake ;
        QVector<specDataPoint> taken ;
public:
        explicit specManageDataCommand(specUndoCommand *parent = 0) ;
        void setItem(const QModelIndex&, const QVector<int>&) ;
        bool ok() ;
        QDataStream& write(QDataStream &out) const ;
        QDataStream& read(QDataStream &in) ;
protected:
        void take() ;
        void insert() ;
};
```

```
#endif
```

src/actionlib/commands/specmanageitemscommand.h

```
#ifndef SPECMANAGEITEMSCOMMAND_H
#define SPECMANAGEITEMSCOMMAND_H
#include "specmultipleitemcommand.h"
class specManageItemsCommand : public specMultipleItemCommand
{
public:
        explicit specManageItemsCommand(specUndoCommand* parent = 0);
        void setItems(QList<specModelItem*>&) ;
        void setItem(specModelItem* pointer) ;
};
#endif
```

src/actionlib/commands/specmanageitemscommand.cpp

```
#include "specmanageitemscommand.h"
#include <QAbstractItemView>
specManageItemsCommand::specManageItemsCommand(specUndoCommand* parent) :
        specMultipleItemCommand(parent)
{
}
void specManageItemsCommand::setItems(QList<specModelItem*>& pointers)
{
        specModel::eliminateChildren(pointers) ;
        qSort(pointers.begin(), pointers.end(), specModel::lessThanItemPointer) ;
        specMultipleItemCommand::setItems(pointers) ;
}
void specManageItemsCommand::setItem(specModelItem* pointer)
{
        setItems(QList<specModelItem*>() << pointer) ;
}
```

src/actionlib/commands/specmetarangecommand.h

```
#ifndef SPECMETARANGECOMMAND_H
#define SPECMETARANGECOMMAND_H
#include "specsingleitemcommand.h"
class specMetaRangeCommand : public specSingleItemCommand<specMetaItem>
{
private:
        double oldX, oldY, newX, newY ;
        int rangeNo, pointNo, variableNo ;
        void undoIt() ;
        void doIt() ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        type typeId() const { return specStreamable::metaRangeCommand ; }
public:
        specMetaRangeCommand(specUndoCommand* parent = 0);
        bool mergeWith(const QUndoCommand* other) ;
        void setItem(specMetaItem*, int VariableIndex, int rangeIndex, int pointNo, double newX, ►
            double newY) ;
        bool mergeable(const specUndoCommand* other) ;
};
#endif
```

src/actionlib/commands/specmetarangecommand.cpp

```
#include "specmetarangecommand.h"
#include "specmetaitem.h"
specMetaRangeCommand::specMetaRangeCommand(specUndoCommand* parent)
        : specSingleItemCommand(parent)
{
}
void specMetaRangeCommand::doIt()
```

```
{
        itemPointer()->setRange(variableNo, rangeNo, pointNo, newX, newY) ;
        specModel* model = qobject_cast<specModel*> (parentObject());
        if(model)
                model->signalChanged(model->index(itemPointer(), model->descriptors().indexOf("►
                    variables")));
}
void specMetaRangeCommand::undoIt()
{
        qSwap(newX, oldX) ;
        qSwap(newY, oldY) ;
        doIt() ;
        qSwap(newX, oldX) ;
        qSwap(newY, oldY) ;
}
bool specMetaRangeCommand::mergeWith(const QUndoCommand* ot)
{
        if(!parentObject()) return false ;
        const specMetaRangeCommand* other = (const specMetaRangeCommand*) ot ;
        if(!(this->itemPointer() && other->itemPointer())) return false ;
        if(other->variableNo != variableNo) return false ;
        if(other->rangeNo != rangeNo) return false ;
        if(other->pointNo != pointNo) return false ;
        newX = other->newX ;
        newY = other->newY ;
        return true ;
}
void specMetaRangeCommand::setItem(specMetaItem* i, int variableIndex, int rangeIndex, int point, ►
    double nX, double nY)
{
        specSingleItemCommand::setItem(i) ;
        pointNo = point ;
        variableNo = variableIndex ;
        rangeNo = rangeIndex ;
        newX = nX ;
        newY = nY ;
        i->getRangePoint(variableNo, rangeNo, pointNo, oldX, oldY) ;
}
void specMetaRangeCommand::writeCommand(QDataStream& out) const
{
        out << oldX << oldY << newX << newY << rangeNo << pointNo << variableNo ;
        writeItem(out) ;
}
void specMetaRangeCommand::readCommand(QDataStream& in)
{
        in >> oldX >> oldY >> newX >> newY >> rangeNo >> pointNo >> variableNo ;
        readItem(in);
}
bool specMetaRangeCommand::mergeable(const specUndoCommand* other)
{
        return (((specMetaRangeCommand*) other)->itemPointer()) == itemPointer() ;
}
```

src/actionlib/commands/specmovecommand.h

```
#ifndef SPECMOVECOMMAND_H
#define SPECMOVECOMMAND_H
#include "specundocommand.h"
#include "specmodelitem.h"
#include <QModelIndex>
#include <QVector>
class specModel ;
class specView ;
class specMoveCommand : public specUndoCommand
{
public:
        class moveUnit
        {
        private:
                QVector<int> firstItemIndex ;
                QVector<int> parentIndex ;
                QVector<specModelItem*> items ;
                specFolderItem* parent ;
                int count ;
```

```
                int row ;
        public:
                void pointersToIndexes(specModel*) ;
                void indexesToPointers(specModel*) ;
                moveUnit(QModelIndexList&, const QModelIndex& target, int& r, specModel*) ;
                moveUnit() ;
                void moveIt() ;
                friend QDataStream& operator<< (QDataStream&, const moveUnit&) ;
                friend QDataStream& operator>> (QDataStream&, moveUnit&) ;
        };
private:
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const;
        void readCommand(QDataStream& in) ;
        void finish() ;
        bool prepare() ;
        type typeId() const { return specStreamable::moveItemsCommandId ; }
        QVector<moveUnit> moveUnits ;
        bool refresh() ;
        specView* view ;
        specModel* model ;
public:
        void setItems(QModelIndexList& sources, const QModelIndex& target, int row) ;
        specMoveCommand(specUndoCommand* parent = 0) ;
};
QDataStream& operator<< (QDataStream&, const specMoveCommand::moveUnit&) ;
QDataStream& operator>> (QDataStream&, specMoveCommand::moveUnit&) ;
#endif
```

src/actionlib/commands/specmovecommand.cpp

```
#include "specmovecommand.h"
#include "specmodel.h"
#include "specview.h"
void specMoveCommand::moveUnit::pointersToIndexes(specModel* model)
{
        if(!model || items.isEmpty() || parent == 0 || items.first() == 0)
        {
                firstItemIndex.clear();
                parentIndex.clear();
                count = 0 ;
                return ;
        }
        parentIndex = model->hierarchy(parent) ;
        firstItemIndex = model->hierarchy(items.first()) ;
        items.clear();
        parent = 0 ;
}
void specMoveCommand::moveUnit::indexesToPointers(specModel* model)
{
        if(!model || firstItemIndex.isEmpty())
        {
                items.clear();
                parent = 0 ;
                return ;
        }
        parent = dynamic_cast<specFolderItem*>(model->itemPointer(parentIndex)) ;
        items.clear();
        QVector<int> prefix(firstItemIndex) ;
        for(int i = 0 ; i < count ; ++i)
        {
                items << model->itemPointer(prefix) ;
                prefix.first() ++ ;
        }
        firstItemIndex.clear();
        parentIndex.clear();
}
void specMoveCommand::moveUnit::moveIt()
{
        specFolderItem* oldParent = items.first()->parent() ;
        int oldRow = oldParent->childNo(items.first()) ;
        foreach(specModelItem * item, items)
        item->setParent(0);
```

```
        parent->addChildren(items.toList(), row) ;
        row = oldRow ;
        parent = oldParent ;
}
QDataStream& operator<< (QDataStream& out, const specMoveCommand::moveUnit& unit)
{
        return out << unit.firstItemIndex << unit.parentIndex << unit.count << unit.row ;
}
QDataStream& operator>> (QDataStream& in, specMoveCommand::moveUnit& unit)
{
        return in >> unit.firstItemIndex >> unit.parentIndex >> unit.count >> unit.row ;
}
specMoveCommand::moveUnit::moveUnit(QModelIndexList& list, const QModelIndex& target, int& r, ►
    specModel* model)
        : parent(0),
          count(0),
          row(r)
{
        if(list.isEmpty() || !model) return ;
        int nextRow = list.first().row() ;
        int firstRow = nextRow ;
        firstItemIndex = model->hierarchy(list.first()) ;
        if(firstItemIndex.isEmpty()) return ;
        parentIndex = model->hierarchy(target) ;
        const QModelIndex parentModelIndex(list.first().parent()) ;
        while(!list.isEmpty() &&
                list.first().row() == nextRow++ &&
                list.first().parent() == parentModelIndex)
        {
                count ++ ;
                list.takeFirst() ;
        }
        if(parentModelIndex == target && row > firstRow)
                row = qMax(row - count, firstRow) ;
        r = row + count ;
}
specMoveCommand::moveUnit::moveUnit()
        : parent(0),
          count(0),
          row(0)
{
}
specMoveCommand::specMoveCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
void specMoveCommand::setItems(QModelIndexList& sources, const QModelIndex& target, int row)
{
        if(!refresh()) return ;
        while(!sources.isEmpty())
                moveUnits << moveUnit(sources, target, row, model) ;
}
bool specMoveCommand::refresh()
{
        return (model = qobject_cast<specModel*> (parentObject())) ;
}
bool specMoveCommand::prepare()
{
        if(!refresh()) return false ;
        for(QVector<moveUnit>::iterator i = moveUnits.begin() ; i != moveUnits.end() ; ++i)
                i->indexesToPointers(model) ;
        model->signalBeginReset();
        return true ;
}
void specMoveCommand::finish()
{
        for(QVector<moveUnit>::iterator i = moveUnits.begin() ; i != moveUnits.end() ; ++i)
                i->pointersToIndexes(model) ;
}
void specMoveCommand::doIt()
{
        if(!prepare()) return ;
        for(int i = 0 ; i < moveUnits.size() ; ++i)
                moveUnits[i].moveIt() ;
        finish() ;
```

```
}
void specMoveCommand::undoIt()
{
        if(!prepare()) return ;
        for(int i = moveUnits.size() ; i > 0 ; --i)
                moveUnits[i - 1].moveIt() ;
        finish() ;
}
void specMoveCommand::writeCommand(QDataStream& out) const
{
        out << moveUnits ;
}
void specMoveCommand::readCommand(QDataStream& in)
{
        in >> moveUnits ;
}
```

src/actionlib/commands/specmulticommand.h

```
#ifndef SPECMULTICOMMAND_H
#define SPECMULTICOMMAND_H
#include "specundocommand.h"
#include "names.h"
#include "speccommandgenerator.h"
class specMultiCommand : public specUndoCommand
{
private:
        bool mayMerge ;
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const;
        void readCommand(QDataStream& in) ;
        type typeId() const { return specStreamable::multiCommandId ; }
        void parentAssigned();
        specStreamable* factory(const type& t) const ;
        specCommandGenerator commandGenerator ;
public:
        explicit specMultiCommand(specUndoCommand* parent = 0);
        void setMergeable(bool mergeable = true) ;
        bool mergeWith(const QUndoCommand* other) ;
};
#endif
```

src/actionlib/commands/specmulticommand.cpp

```
#include "specmulticommand.h"
#include "specactionlibrary.h"
#include "speccommandgenerator.h"
specMultiCommand::specMultiCommand(specUndoCommand* parent)
        : specUndoCommand(parent),
          mayMerge(true),
          commandGenerator(this)
{
}
void specMultiCommand::setMergeable(bool mergeable)
{
        mayMerge = mergeable ;
}
void specMultiCommand::doIt()
{
        QUndoCommand::redo() ;
}
void specMultiCommand::undoIt()
{
        QUndoCommand::undo() ;
}
void specMultiCommand::writeCommand(QDataStream& out) const
{
        out << mayMerge << qint32(childCount()) ;
        for(int i = 0 ; i < childCount() ; ++i)
                out << * ((specUndoCommand*) child(i)) ;
}
```

```
void specMultiCommand::readCommand(QDataStream& in)
{
        qint32 children ;
        in >> mayMerge >> children ;
        for(int i = 0 ; i < children ; ++i)
                produceItem(in) ;
}
bool specMultiCommand::mergeWith(const QUndoCommand* other)
{
        if(!parentObject()) return false ;
        if(!mayMerge) return false ;
        if(other->childCount() != childCount()) return false ;
        bool mergeable = true ;
        for(int i = 0 ; i < childCount() ; ++i)
        {
                if(child(i)->id() != other->child(i)->id())
                        return false ;
                mergeable = mergeable && ((specUndoCommand*) child(i))->mergeable((specUndoCommand*)►
                     other->child(i)) ;
        }
        if(!mergeable) return false ;
        for(int i = 0 ; i < childCount() ; ++i)
                ((specUndoCommand*) child(i))->mergeWith((const specUndoCommand*) other->child(i)) ;
        setText(other->text());
        return true ;
}
void specMultiCommand::parentAssigned()
{
        specUndoCommand* childPointer = 0 ;
        for(int i = 0 ; i < childCount() ; ++i)
                if((childPointer = dynamic_cast<specUndoCommand*>(const_cast<QUndoCommand*>(child(i)►
                    ))))
                        childPointer->setParentObject(parentObject()) ;
}
specStreamable* specMultiCommand::factory(const type& t) const
{
        return commandGenerator.commandById(t) ;
}
```

src/actionlib/commands/specmultipleitemcommand.h

```
#ifndef SPECMULTIPLEITEMCOMMAND_H
#define SPECMULTIPLEITEMCOMMAND_H
#include "specundocommand.h"
#include "specgenealogy.h"
class specMultipleItemCommand : public specUndoCommand
{
private:
        QVector<specGenealogy> items ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in);
protected:
        void writeItems(QDataStream&) const ;
        void readItems(QDataStream&, qint32 = -1) ;
        void takeItems() ;
        void restoreItems() ;
        void parentAssigned();
        qint32 itemCount() const ;
        QSet<specFolderItem*> parents() const;
        QList<specModelItem*> itemPointers() ;
        void addItems(QList<specModelItem*>&) ;
        void clearItems() ;
        QList<specModelItem*> firstItems() ;
public:
        specMultipleItemCommand(specUndoCommand* parent = 0) ;
        virtual void setItems(QList<specModelItem*>&) ;
};
#endif
```

src/actionlib/commands/specmultipleitemcommand.cpp

```
#include "specmultipleitemcommand.h"
```

```
specMultipleItemCommand::specMultipleItemCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
void specMultipleItemCommand::clearItems()
{
        items.clear();
}
void specMultipleItemCommand::addItems(QList<specModelItem*>& list)
{
        specModel* m = model() ;
        if(!m) return ;
        while(!list.isEmpty())
                items << specGenealogy(list, m) ;
}
void specMultipleItemCommand::setItems(QList<specModelItem*>& list)
{
        clearItems();
        addItems(list) ;
}
QList<specModelItem*> specMultipleItemCommand::firstItems()
{
        QList<specModelItem*> l ;
        for(int i = 0 ; i < items.size() ; ++i)
                l << items[i].firstItem() ;
        return l ;
}
QList<specModelItem*> specMultipleItemCommand::itemPointers()
{
        QList<specModelItem*> l ;
        for(int i = 0 ; i < items.size() ; ++i)
                l << items[i].items().toList() ;
        return l ;
}
qint32 specMultipleItemCommand::itemCount() const
{
        return items.size() ;
}
void specMultipleItemCommand::writeCommand(QDataStream& out) const
{
        out << itemCount() ;
        writeItems(out);
}
void specMultipleItemCommand::readCommand(QDataStream& in)
{
        qint32 num = 0;
        in >> num;
        readItems(in, num);
}
void specMultipleItemCommand::writeItems(QDataStream& out) const
{
        for(int i = 0 ; i < items.size() ; ++i)
                out << items[i] ;
}
void specMultipleItemCommand::takeItems()
{
        specModel* m = model() ;
        if(!m) return ;
        m->signalBeginReset();
        for(int i = 0 ; i < items.size() ; ++i)
                items[i].takeItems();
}
void specMultipleItemCommand::restoreItems()
{
        specModel* m = model() ;
        if(!m) return ;
        m->signalBeginReset();
        for(int i = 0 ; i < items.size() ; i++)
                items[i].returnItems();
}
void specMultipleItemCommand::readItems(QDataStream& in, qint32 n)
{
        items.clear();
        items.resize(n);
        for(int i = 0 ; i < n ; ++i)
```

```
                in >> items[i] ;
}
void specMultipleItemCommand::parentAssigned()
{
        specModel* model = dynamic_cast<specModel*>(parentObject()) ;
        Q_ASSERT(model) ;
        for(int i = 0 ; i < items.size() ; ++i)
                items[i].setModel(model) ;
}
QSet<specFolderItem*> specMultipleItemCommand::parents() const
{
        QSet<specFolderItem*> ps ;
        for(int i = 0 ; i < items.size() ; ++i)
                ps << items[i].parent() ;
        ps.remove(0) ;
        return ps ;
}
```

src/actionlib/commands/specplotlabelcommand.h

```
#ifndef SPECPLOTLABELCOMMAND_H
#define SPECPLOTLABELCOMMAND_H
#include "specundocommand.h"
class specPlotLabelCommand : public specUndoCommand
{
private:
        void undoIt();
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
protected:
        QString text ;
public:
        specPlotLabelCommand(specUndoCommand* parent = 0) ;
        void setLabelText(const QString&) ;
};
specPlotLabelCommand* generatePlotLabelCommand(specStreamable::type id, specUndoCommand* parent = 0) ;
class specPlotTitleCommand : public specPlotLabelCommand
{
private:
        void doIt() ;
        type typeId() const { return specStreamable::plotTitleCommandId ; }
public:
        specPlotTitleCommand(specUndoCommand* parent = 0) ;
};
class specPlotYLabelCommand : public specPlotLabelCommand
{
private:
        void doIt() ;
        type typeId() const { return specStreamable::plotYLabelCommandId ; }
public:
        specPlotYLabelCommand(specUndoCommand* parent = 0) ;
};
class specPlotXLabelCommand : public specPlotLabelCommand
{
private:
        void doIt() ;
        type typeId() const { return specStreamable::plotXLabelCommandId ; }
public:
        specPlotXLabelCommand(specUndoCommand* parent = 0) ;
};
#endif
```

src/actionlib/commands/specplotlabelcommand.cpp

```
#include "specplotlabelcommand.h"
#include "specplot.h"
specPlotLabelCommand::specPlotLabelCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
void specPlotLabelCommand::setLabelText(const QString& t)
{
```

```
        text = t ;
}
void specPlotLabelCommand::undoIt()
{
        doIt() ;
}
void specPlotLabelCommand::writeCommand(QDataStream& out) const
{
        out << text ;
}
void specPlotLabelCommand::readCommand(QDataStream& in)
{
        in >> text ;
}
void specPlotTitleCommand::doIt()
{
        specPlot* plot = (specPlot*) parentObject() ;
        QString temp = plot->title().text() ;
        plot->setTitle(text) ;
        text = temp ;
}
void specPlotYLabelCommand::doIt()
{
        specPlot* plot = (specPlot*) parentObject() ;
        QString temp = plot->axisTitle(QwtPlot::yLeft).text() ;
        plot->setAxisTitle(QwtPlot::yLeft, text) ;
        text = temp ;
}
void specPlotXLabelCommand::doIt()
{
        specPlot* plot = (specPlot*) parentObject() ;
        QString temp = plot->axisTitle(QwtPlot::xBottom).text() ;
        plot->setAxisTitle(QwtPlot::xBottom, text) ;
        text = temp ;
}
specPlotTitleCommand::specPlotTitleCommand(specUndoCommand* parent)
        : specPlotLabelCommand(parent)
{
}
specPlotXLabelCommand::specPlotXLabelCommand(specUndoCommand* parent)
        : specPlotLabelCommand(parent)
{
}
specPlotYLabelCommand::specPlotYLabelCommand(specUndoCommand* parent)
        : specPlotLabelCommand(parent)
{
}
specPlotLabelCommand* generatePlotLabelCommand(specStreamable::type id, specUndoCommand* parent)
{
        switch(id)
        {
                case(specStreamable::plotTitleCommandId) : return new specPlotTitleCommand(parent) ;
                case(specStreamable::plotYLabelCommandId) : return new specPlotYLabelCommand(parent) ;
                case(specStreamable::plotXLabelCommandId) : return new specPlotXLabelCommand(parent) ;
                default: ;
        }
        return 0 ;
}
```

src/actionlib/commands/specrenamedescriptorcommand.h

```
#ifndef SPECRENAMEDESCRIPTORCOMMAND_H
#define SPECRENAMEDESCRIPTORCOMMAND_H
#include "specundocommand.h"
#include <QMap>
#include <QString>
class specRenameDescriptorCommand : public specUndoCommand
{
        QMap<QString, QString> map ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
        void doIt();
        void undoIt() ;
        type typeId() const { return specStreamable::renameDescriptorCommandId ; }
```

```
public:
        specRenameDescriptorCommand(specUndoCommand* parent = 0);
        void setRenamingMap(QMap<QString, QString> map) ;
};
#endif
```

src/actionlib/commands/specrenamedescriptorcommand.cpp

```
#include "specrenamedescriptorcommand.h"
#include "specmodel.h"
specRenameDescriptorCommand::specRenameDescriptorCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
void specRenameDescriptorCommand::setRenamingMap(QMap<QString, QString> m)
{
        map.swap(m);
}
void specRenameDescriptorCommand::writeCommand(QDataStream& out) const
{
        out << map ;
}
void specRenameDescriptorCommand::readCommand(QDataStream& in)
{
        in >> map ;
}
void specRenameDescriptorCommand::doIt()
{
        specModel* myModel = qobject_cast<specModel*> (parentObject()) ;
        if(!myModel) return ;
        myModel->signalBeginReset();
        myModel->renameDescriptors(map) ;
        QMap<QString, QString> inverted ;
        foreach(const QString & key, map.keys())
        inverted[map[key]] = key ;
        map.swap(inverted);
}
void specRenameDescriptorCommand::undoIt()
{
        doIt() ;
}
```

src/actionlib/commands/specresizesvgcommand.h

```
#ifndef SPECRESIZESVGCOMMAND_H
#define SPECRESIZESVGCOMMAND_H
#include "specsingleitemcommand.h"
#include "specsvgitem.h"
class specGenealogy ;
class specResizeSVGcommand : public specSingleItemCommand<specSVGItem>
{
private:
        specSVGItem::bounds other ;
        specSVGItem::SVGCornerPoint anchor ;
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const;
        void readCommand(QDataStream& in) ;
        type typeId() const { return specStreamable::resizeSVGCommandId ; }
public:
        explicit specResizeSVGcommand(specUndoCommand* parent = 0) ;
        void setItem(specModelItem* item, const specSVGItem::bounds&,
                     specSVGItem::SVGCornerPoint anchor = specSVGItem::undefined) ;
        bool mergeable(const specUndoCommand* other) ;
        bool mergeWith(const QUndoCommand* other) ;
};
#endif
```

src/actionlib/commands/specresizesvgcommand.cpp

```
#include "specresizesvgcommand.h"
#include "specsvgitem.h"
#include "specgenealogy.h"
#include "qwt_plot.h"
specResizeSVGcommand::specResizeSVGcommand(specUndoCommand* parent)
        : specSingleItemCommand(parent),
          anchor(specSVGItem::undefined)
{
}
void specResizeSVGcommand::setItem(specModelItem* item, const specSVGItem::bounds& bounds, ►
     specSVGItem::SVGCornerPoint a)
{
        specSingleItemCommand::setItem(item) ;
        other = bounds ;
        anchor = a ;
        doIt() ;
}
void specResizeSVGcommand::writeCommand(QDataStream& out) const
{
        out << other ;
        writeItem(out) ;
        out << (qint8) anchor ;
}
void specResizeSVGcommand::readCommand(QDataStream& in)
{
        qint8 newAnchor ;
        in >> other ;
        readItem(in) ;
        in >> newAnchor ;
        anchor = (specSVGItem::SVGCornerPoint) newAnchor ;
}
void specResizeSVGcommand::doIt()
{
        specSVGItem* pointer = itemPointer() ;
        specSVGItem::bounds oldBounds = pointer->getBounds() ;
        pointer->setBounds(other);
        if(anchor != specSVGItem::undefined)
                anchor = pointer->setAnchor(anchor) ;
        other = oldBounds ;
        if(pointer->plot())
                pointer->plot()->replot();
}
void specResizeSVGcommand::undoIt()
{
        doIt() ;
}
bool specResizeSVGcommand::mergeable(const specUndoCommand* other)
{
        return (itemPointer() == ((specResizeSVGcommand*) other)->itemPointer()) ;
}
bool specResizeSVGcommand::mergeWith(const QUndoCommand* other)
{
        if(!parentObject()) return false ;
        return mergeable((specUndoCommand*) other) ;
}
```

src/actionlib/commands/specsingleitemcommand.h

```
#ifndef SPECITEMCOMMAND_H
#define SPECITEMCOMMAND_H
#include "specundocommand.h"
#include "specgenealogy.h"
template <class itemType>
class specSingleItemCommand : public specUndoCommand
{
private:
        specGenealogy item ;
        void writeCommand(QDataStream& out) const ;
        void readCommand(QDataStream& in) ;
protected:
        void writeItem(QDataStream& out) const ;
        void readItem(QDataStream& in) ;
        void parentAssigned() ;
        itemType* itemPointer() ;
```

```
        itemType* itemPointer() const ;
public:
        specSingleItemCommand(specUndoCommand* parent = 0);
        virtual void setItem(specModelItem*) ;
};
#endif
```

src/actionlib/commands/specsingleitemcommand.cpp

```
#include "specsingleitemcommand.h"
#include "specdataitem.h"
#include "specsvgitem.h"
#include "specmetaitem.h"
template <class itemType>
specSingleItemCommand<itemType>::specSingleItemCommand(specUndoCommand* parent)
        : specUndoCommand(parent)
{
}
template <class itemType>
void specSingleItemCommand<itemType>::setItem(specModelItem* pointer)
{
        if(!model()) return ;
        if(dynamic_cast<itemType*>(pointer))
                item = specGenealogy(pointer, model()) ;
}
template <class itemType>
void specSingleItemCommand<itemType>::readCommand(QDataStream& in)
{
        readItem(in) ;
}
template <class itemType>
void specSingleItemCommand<itemType>::writeCommand(QDataStream& out) const
{
        writeItem(out) ;
}
template <class itemType>
void specSingleItemCommand<itemType>::readItem(QDataStream& in)
{
        in >> item ;
}
template <class itemType>
void specSingleItemCommand<itemType>::writeItem(QDataStream& out) const
{
        out << item ;
}
template <class itemType>
void specSingleItemCommand<itemType>::parentAssigned()
{
        specModel* model = dynamic_cast<specModel*>(parentObject()) ;
        Q_ASSERT(model) ;
        item.setModel(model) ;
}
template <class itemType>
itemType* specSingleItemCommand<itemType>::itemPointer()
{
        if(!item.valid() && !item.seekParent()) return 0 ;
        Q_ASSERT(item.firstItem()) ;
        return const_cast<const specSingleItemCommand<itemType>* >(this)->itemPointer() ;
}
template <class itemType>
itemType* specSingleItemCommand<itemType>::itemPointer() const
{
        return dynamic_cast<itemType*>(item.firstItem()) ;
}
template class specSingleItemCommand<specModelItem> ;
template class specSingleItemCommand<specDataItem> ;
template class specSingleItemCommand<specMetaItem> ;
template class specSingleItemCommand<specSVGItem> ;
```

src/actionlib/commands/specstylecommand.h

```
#ifndef SPECSTYLECOMMAND_H
```

```
#define SPECSTYLECOMMAND_H
#include "specundocommand.h"
#include "model/specgenealogy.h"
class specStyleCommand : public specUndoCommand
{
public:
        specStyleCommand(specUndoCommand* parent = 0) : specUndoCommand(parent) {}
        virtual void setItems(QList<specModelItem*> list) = 0 ;
        virtual void obtainStyle(specCanvasItem*) = 0 ;
};
specStyleCommand* generateStyleCommand(specStreamable::type id, specUndoCommand* parent = 0) ;
#define specStyleCommandImplTemplate template<class property, \
        property (specCanvasItem::*getProperty)() const, \
        void (specCanvasItem::*setProperty)(const property&), \
        int ID>
#define specStyleCommandImplFuncTemplate specStyleCommandImplementation<property, getProperty, ►
    setProperty, ID>
specStyleCommandImplTemplate
class specStyleCommandImplementation : public specStyleCommand
{
public:
        explicit specStyleCommandImplementation(specUndoCommand* parent = 0);
        void setItems(QList<specModelItem*> list) ;
        void obtainStyle(specCanvasItem*) ;
private:
        void applyStyle(specGenealogy&, int) ;
        int styleNo(specCanvasItem*) ;
        void saveStyles(QList<specGenealogy>&) ;
        type typeId() const { return ID ; }
        void parentAssigned();
        void doIt() ;
        void undoIt() ;
        void writeCommand(QDataStream& out) const;
        void readCommand(QDataStream& in);
        property newProperty ;
        QVector<property> oldProperties ;
        QList<specGenealogy> Genealogies ;
};
#endif
```

src/actionlib/commands/specstylecommand.cpp

```
#include "specstylecommand.h"
#include <qwt_symbol.h>
specStyleCommandImplTemplate
specStyleCommandImplFuncTemplate::specStyleCommandImplementation(specUndoCommand* parent)
        : specStyleCommand(parent)
{
}
specStyleCommandImplTemplate
void specStyleCommandImplFuncTemplate::setItems(QList<specModelItem*> items)
{
        specModel* model = qobject_cast<specModel*> (parentObject()) ;
        QMap<int, QList<specModelItem*> > groups ;
        foreach(specModelItem * item, items)
        groups[styleNo(item)] << item ;
        for(QMap<int, QList<specModelItem*> >::iterator i = groups.begin() ; i != groups.end() ; ++i)
        {
                qSort(i.value().begin(), i.value().end(), model->lessThanItemPointer) ;
                while(!i.value().isEmpty())
                        Genealogies << specGenealogy(i.value(), model) ;
        }
        saveStyles(Genealogies) ;
}
specStyleCommandImplTemplate
void specStyleCommandImplFuncTemplate::doIt()
{
        for(int i = 0 ; i < Genealogies.size() ; ++i)
                applyStyle(Genealogies[i], -1) ;
}
specStyleCommandImplTemplate
void specStyleCommandImplFuncTemplate::undoIt()
{
        for(int i = 0 ; i < Genealogies.size() ; ++i)
```

```
                applyStyle(Genealogies[i], i) ;
    }
    specStyleCommandImplTemplate
36  void specStyleCommandImplFuncTemplate::writeCommand(QDataStream& out) const
    {
            out << qint32(Genealogies.size()) << newProperty << oldProperties ;
            for(int i = 0 ; i < Genealogies.size() ; ++i)
                    out << Genealogies[i] ;
41  }
    specStyleCommandImplTemplate
    void specStyleCommandImplFuncTemplate::readCommand(QDataStream& in)
    {
            qint32 size ;
46          in >> size >> newProperty >> oldProperties ;
            for(qint32 i = 0 ; i < size ; ++i)
            {
                    Genealogies << specGenealogy() ;
                    in >> Genealogies.last() ;
51          }
    }
    specStyleCommandImplTemplate
    void specStyleCommandImplFuncTemplate::parentAssigned()
    {
56          specModel* model = qobject_cast<specModel*> (parentObject()) ;
            for(QList<specGenealogy>::iterator i = Genealogies.begin() ; i != Genealogies.end() ; ++i)
                    i->setModel(model) ;
    }
    specStyleCommandImplTemplate
61  void specStyleCommandImplFuncTemplate::applyStyle(specGenealogy& genealogy, int propertyIndex)
    {
            property prop = (propertyIndex < 0 || !(propertyIndex < oldProperties.size())) ? newProperty►
                  : oldProperties[propertyIndex] ;
            genealogy.seekParent() ;
            if(!genealogy.valid()) return ;
66          QVector<specModelItem*> items = genealogy.items() ;
            for(int j = 0 ; j < items.size() ; ++j)
                    (items[j]->*setProperty)(prop) ;
            genealogy.signalChange();
    }
71  specStyleCommandImplTemplate
    int specStyleCommandImplFuncTemplate::styleNo(specCanvasItem* item)
    {
            property prop = (item->*getProperty)() ;
            if(!oldProperties.contains(prop))
76                  oldProperties << prop ;
            return oldProperties.indexOf(prop) ;
    }
    specStyleCommandImplTemplate
    void specStyleCommandImplFuncTemplate::saveStyles(QList<specGenealogy>& list)
81  {
            oldProperties.clear();
            for(int i = 0 ; i < list.size() ; ++i)
                    oldProperties << (list[i].firstItem()->*getProperty)() ;
    }
86  specStyleCommandImplTemplate
    void specStyleCommandImplFuncTemplate::obtainStyle(specCanvasItem* item)
    {
            newProperty = (item->*getProperty)();
    }
91  specStyleCommand* generateStyleCommand(specStreamable::type id, specUndoCommand* parent)
    {
            switch(id)
            {
                    case specStreamable::penColorCommandId :
96                          return new specStyleCommandImplementation<QColor, &specCanvasItem::penColor,►
                                 &specCanvasItem::setPenColor, specStreamable::penColorCommandId> (►
                                parent) ;
                    case specStreamable::symbolStyleCommandId :
                            return new specStyleCommandImplementation<int, &specCanvasItem::symbolStyle,►
                                 &specCanvasItem::setSymbolStyle, specStreamable::symbolStyleCommandId >►
                                 (parent) ;
                    case specStreamable::symbolPenColorCommandId :
                            return new specStyleCommandImplementation<QColor, &specCanvasItem::►
                                symbolPenColor, &specCanvasItem::setSymbolPenColor, specStreamable::►
                                symbolPenColorCommandId > (parent) ;
```

```
                case specStreamable::symbolBrushColorCommandId:
                        return new specStyleCommandImplementation<QColor, &specCanvasItem::►
                            symbolBrushColor, &specCanvasItem::setSymbolBrushColor, specStreamable::►
                            symbolBrushColorCommandId> (parent) ;
                case specStreamable::symbolSizeCommandId:
                        return new specStyleCommandImplementation<QSize, &specCanvasItem::symbolSize►
                            , &specCanvasItem::setSymbolSize, specStreamable::symbolSizeCommandId > ►
                            (parent) ;
                case specStreamable::lineWidthCommandId:
                        return new specStyleCommandImplementation<double, &specCanvasItem::lineWidth►
                            , &specCanvasItem::setLineWidth, specStreamable::lineWidthCommandId> (►
                            parent) ;
                case specStreamable::penStyleCommandId:
                        return new specStyleCommandImplementation<qint8, &specCanvasItem::penStyle, ►
                            &specCanvasItem::setPenStyle, specStreamable::penStyleCommandId> (parent►
                            ) ;
                default:
                        return 0 ;
        }
        return 0 ;
}
```

src/actionlib/commands/spectogglefitstylecommand.h

```
#ifndef SPECTOGGLEFITSTYLECOMMAND_H
#define SPECTOGGLEFITSTYLECOMMAND_H
#include "specsingleitemcommand.h"
#include "specmetaitem.h"
class specToggleFitStyleCommand : public specSingleItemCommand<specMetaItem>
{
private:
        void doIt() ;
        void undoIt() ;
        type typeId() const { return specStreamable::toggleFitStyleCommand ; }
public:
        explicit specToggleFitStyleCommand(specUndoCommand* parent = 0);
};
#endif
```

src/actionlib/commands/spectogglefitstylecommand.cpp

```
#include "spectogglefitstylecommand.h"
specToggleFitStyleCommand::specToggleFitStyleCommand(specUndoCommand* parent)
        : specSingleItemCommand(parent)
{
}
void specToggleFitStyleCommand::undoIt()
{
        doIt() ;
}
void specToggleFitStyleCommand::doIt()
{
        specMetaItem* pointer = itemPointer() ;
        if(!pointer) return ;
        pointer->toggleFitStyle() ;
}
```

src/actionlib/commands/specundocommand.h

```
#ifndef SPECUNDOCOMMAND_H
#define SPECUNDOCOMMAND_H
#include <QUndoCommand>
#include "specstreamable.h"
class specModel ;
class specUndoCommand : public QUndoCommand, public specStreamable
{
private:
        QObject* p0 ;
        virtual void writeToStream(QDataStream& out) const ;
        virtual void readFromStream(QDataStream& in) ;
```

```
protected:
        virtual QString description() const ;
        virtual void doIt() = 0;
        virtual void undoIt() = 0 ;
        virtual void parentAssigned() {}
        virtual void writeCommand(QDataStream& out) const = 0 ;
        virtual void readCommand(QDataStream& in) = 0 ;
        specModel* model() const ;
public:
        explicit specUndoCommand(specUndoCommand* parent = 0);
        void redo() ;
        void undo() ;
        virtual bool mergeable(const specUndoCommand* other) { Q_UNUSED(other) ; return false ; }
        void setParentObject(QObject*) ;
        int id() const { return typeId() ; }
        QObject* parentObject() const ;
signals:
public slots:
};
#endif
```

src/actionlib/commands/specundocommand.cpp

```
#include "specundocommand.h"
#include "specmodel.h"
specUndoCommand::specUndoCommand(specUndoCommand* parent) :
        QUndoCommand(parent),
        pO(parent ? parent->parentObject() : 0)
{
}
void specUndoCommand::setParentObject(QObject* par)
{
        pO = par ;
        parentAssigned();
}
QObject* specUndoCommand::parentObject() const
{
        return pO ;
}
void specUndoCommand::redo()
{
        if(pO) doIt() ;
}
void specUndoCommand::undo()
{
        if(pO) undoIt();
}
void specUndoCommand::writeToStream(QDataStream& out) const
{
        out << description() ;
        writeCommand(out) ;
}
void specUndoCommand::readFromStream(QDataStream& in)
{
        QString d ;
        in >> d ;
        setText(d) ;
        readCommand(in) ;
}
QString specUndoCommand::description() const
{
        return text() ;
}
specModel* specUndoCommand::model() const
{
        return dynamic_cast<specModel*>(parentObject()) ;
}
```

src/actionlib/specactionlibrary.h

```
#ifndef SPECACTIONLIBRARY_H
#define SPECACTIONLIBRARY_H
```

```
class specActionLibrary ;
#include <QObject>
#include <QAction>
#include <QMenuBar>
#include <QToolBar>
#include <QDataStream>
#include <QUndoStack>
#include "specundocommand.h"
#include "specundoaction.h"
#include "specmodel.h"
#include "specview.h"
#include <typeinfo>
#include "specplot.h"
#include "speccommandgenerator.h"
#include "specdockwidget.h"
class specView ;
class specModel ;
class specUndoAction ;
class QUndoView ;
class QProgressDialog ;
class specActionLibrary : public QObject, public specStreamable
{
        Q_OBJECT
public:
        explicit specActionLibrary(QObject* parent = 0);
        ~specActionLibrary() ;
        QToolBar* toolBar(QWidget*) ;
        QMenu* contextMenu(QWidget*) ;
        QObject* parentId(int) ;
        void addDragDropPartner(specModel*) ;
        void setLastRequested(const QModelIndexList&) ;
        int moveInternally(const QModelIndex&, int row, specModel*) ;
        int deleteInternally(specModel*) ;
        void addPlot(specPlot*) ;
        QAction* undoAction(QObject*) ;
        QAction* redoAction(QObject*) ;
        specDockWidget* undoWidget() ;
        specCommandGenerator commandGenerator ;
        void setProgressDialog(QProgressDialog*) ;
public slots:
        void push(specUndoCommand*) ;
signals:
        void stackModified(bool) ;
        void stackIndexChanged() ;
private slots:
        void stackClean(const bool&) ;
        void purgeUndo() ;
private:
        void writeToStream(QDataStream& out) const;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::actionLibrary ; }
        QUndoStack* undoStack ;
        QVector<QObject*> parents ;
        QVector<specModel*> partners;
        QModelIndexList lastRequested ;
        void addParent(QObject*) ;
        void addNewAction(QToolBar*, specUndoAction*) ;
        specStreamable* factory(const type& t) const ;
        QProgressDialog* progress ;
        QAction* purgeUndoAction ;
        specDockWidget* dockWidget ;
};
class specHistoryWidget : public specDockWidget
{
        Q_OBJECT
        QList<QWidget*> mainWidgets() const ;
        QUndoView* undoView ;
public:
        explicit specHistoryWidget(QUndoStack* stack = 0, QWidget* parent = 0) ;
} ;
#endif
```

src/actionlib/specactionlibrary.cpp

```
#include "specactionlibrary.h"
#include "specdeleteaction.h"
#include "specaddfolderaction.h"
#include "specmovecommand.h"
#include "speccopyaction.h"
#include "specpasteaction.h"
#include "speccutaction.h"
#include "changeplotstyleaction.h"
#include "spectreeaction.h"
#include "specflattentreeaction.h"
#include "specmergeaction.h"
#include "specremovedataaction.h"
#include "specaveragedataaction.h"
#include "specaddsvgitem.h"
#include "specnewmetaitemaction.h"
#include "kinetic/specmetaview.h"
#include "specaddconnectionsaction.h"
#include "log/speclogview.h"
#include "specimportspecaction.h"
#include "names.h"
#include "speclabelaction.h"
#include <QUndoView>
#include "utility-functions.h"
#include "genericexportaction.h"
#include "specitempropertiesaction.h"
#include <QClipboard>
#include <QApplication>
#include "specaddfitaction.h"
#include "specconductfitaction.h"
#include "specremovefitaction.h"
#include "spectogglefitstyleaction.h"
#include "specmetaitem.h"
#include "specselectconnectedaction.h"
#include "specsetmultilineaction.h"
#include "specsvgitem.h"
#include "specdescriptoreditaction.h"
#include <QProgressDialog>
#include "spectiltmatrixaction.h"
#include "specspectrumcalculatoraction.h"
#include "specplotwidget.h"
#include <QMessageBox>
#include <specdockwidget.h>
#include "specnormalizeaction.h"
specDockWidget* specActionLibrary::undoWidget()
{
        return dockWidget ;
}
QList<QWidget*> specHistoryWidget::mainWidgets() const
{
        return QList<QWidget*>() << undoView ;
}
specHistoryWidget::specHistoryWidget(QUndoStack* stack, QWidget* parent)
        : specDockWidget(tr("History"), parent),
          undoView(new QUndoView(stack, this))
{
        setWhatsThis("Undo␣history.␣␣Click␣on␣any␣command␣to␣forward/rewind␣to␣that␣particular␣state►
            .");
        toggleViewAction()->setIcon(QIcon::fromTheme("view-history")) ;
        toggleViewAction()->setWhatsThis(tr("Shows␣and␣hides␣the␣undo␣history."));
        setObjectName(tr("History␣window"));
        toggleViewAction()->setText(tr("Toggle␣undo␣window"));
        setupWindow(0);
}
specActionLibrary::specActionLibrary(QObject* parent) :
        QObject(parent),
        progress(0),
        purgeUndoAction(new QAction(QIcon::fromTheme("user-trash"), tr("Clear␣history"), this)),
        dockWidget(0)
{
        undoStack = new QUndoStack(this) ;
        connect(undoStack, SIGNAL(cleanChanged(bool)), this, SLOT(stackClean(bool))) ;
        connect(undoStack, SIGNAL(indexChanged(int)), this, SIGNAL(stackIndexChanged())) ;
        connect(purgeUndoAction, SIGNAL(triggered()), this, SLOT(purgeUndo())) ;
        purgeUndoAction->setWhatsThis(tr("Deletes␣the␣complete␣undo␣history␣--␣use␣with␣care␣and␣don►
```

```
            't␣point␣this␣at␣humans."));
        dockWidget = new specHistoryWidget(undoStack, qobject_cast<QWidget*> (parent ? parent->►
            parent() : 0)) ;
}
void specActionLibrary::stackClean(const bool& b)
{
        emit stackModified(!b) ;
}
void specActionLibrary::push(specUndoCommand* cmd)
{
        if(!cmd) return ;
        undoStack->push(cmd) ;
}
void specActionLibrary::addNewAction(QToolBar* bar, specUndoAction* action)
{
        action->setLibrary(this) ;
        action->setShortcutContext(Qt::WidgetWithChildrenShortcut);
        bar->addAction(action) ;
}
QAction* specActionLibrary::redoAction(QObject* target)
{
        QAction* redoAction = undoStack->createRedoAction(target) ;
        redoAction->setIcon(QIcon::fromTheme("edit-redo")) ;
        redoAction->setToolTip(tr("Redo")) ;
        redoAction->setWhatsThis(tr("Redo␣-␣Redoes␣what␣has␣been␣undone␣by␣clicking␣the␣undo␣button►
            .\nNote␣that␣all␣of␣your␣undo␣history␣(including␣possible␣redos)␣will␣be␣saved␣along␣►
            with␣your␣work␣and␣will␣be␣available␣again␣upon␣loading␣your␣file␣again."));
        return redoAction ;
}
QAction* specActionLibrary::undoAction(QObject* target)
{
        QAction* undoAction = undoStack->createUndoAction(target) ;
        undoAction->setIcon(QIcon::fromTheme("edit-undo"));
        undoAction->setToolTip(tr("Undo")) ;
        undoAction->setWhatsThis(tr("Undo␣-␣By␣clicking␣this␣button␣you␣can␣revert␣changes.␣␣To␣\"►
            undo␣the␣undo\"␣click␣the␣redo␣button␣right␣next␣door.\nNote␣that␣all␣of␣your␣undo␣►
            history␣will␣be␣saved␣along␣with␣your␣work␣and␣will␣be␣available␣again␣upon␣loading␣your►
            ␣file␣again."));
        return undoAction ;
}
QToolBar* specActionLibrary::toolBar(QWidget* target)
{
        QToolBar* bar = new QToolBar(target) ;
        bar->setContentsMargins(0, 0, 0, 0) ;
        bar->setIconSize(QSize(20, 20)) ;
        specView* view = dynamic_cast<specView*>(target) ;
        specDataView* dataView = dynamic_cast<specDataView*>(target) ;
        specMetaView* metaView = dynamic_cast<specMetaView*>(target) ;
        specLogView* logView = dynamic_cast<specLogView*>(target) ;
        specPlot* plot = dynamic_cast<specPlot*>(target) ;
        specPlotWidget* plotWidget = dynamic_cast<specPlotWidget*>(target) ;
        if(view && view->model())
        {
                addParent(view) ;
                addParent(view->model()) ;
                addNewAction(bar, new specAddFolderAction(target)) ;
                if(metaView)
                        addNewAction(bar, new specNewMetaItemAction(target));
                else
                {
                        addNewAction(bar, new specImportSpecAction(target)) ;
                        addNewAction(bar, new specTreeAction(target)) ;
                        addNewAction(bar, new specFlattenTreeAction(target)) ;
                }
                if(dataView || metaView)
                        addNewAction(bar, new specAddSVGItemAction(target)) ;
                addNewAction(bar, new genericExportAction(target)) ;
                bar->addSeparator() ;
                addNewAction(bar, new specCopyAction(target)) ;
                addNewAction(bar, new specCutAction(target)) ;
                addNewAction(bar, new specPasteAction(target)) ;
                addNewAction(bar, new specDeleteAction(target)) ;
                bar->addSeparator() ;
                if(metaView || logView)
                {
```

```
                        bar->addAction(undoAction(view)) ;
                        bar->addAction(redoAction(view)) ;
                        bar->addSeparator() ;
                }
                if(dataView)
                {
                        addNewAction(bar, new specMergeAction(target)) ;
                        addNewAction(bar, new specTiltMatrixAction(target)) ;
                        addNewAction(bar, new specDescriptorEditAction(target)) ;
                        bar->addSeparator() ;
                        addNewAction(bar, new specRemoveDataAction(target)) ;
                        addNewAction(bar, new specAverageDataAction(target)) ;
                        addNewAction(bar, new specSpectrumCalculatorAction(target)) ;
                        addNewAction(bar, new specNormalizeAction(target)) ;
                }
                addNewAction(bar, new specItemPropertiesAction(target)) ;
                addNewAction(bar, new specSetMultilineAction(target)) ;
                if(metaView)
                {
                        addNewAction(bar, new specAddConnectionsAction(target)) ;
                        addNewAction(bar, new specSelectConnectedAction(target)) ;
                        addNewAction(bar, new specAddFitAction(target)) ;
                        addNewAction(bar, new specRemoveFitAction(target)) ;
                        addNewAction(bar, new specToggleFitStyleAction(target));
                        addNewAction(bar, new specConductFitAction(target)) ;
                }
                if(logView)
                        addNewAction(bar, new specDescriptorEditAction(target)) ;
                bar->addSeparator() ;
                if(dataView || metaView)
                        addNewAction(bar, new changePlotStyleAction(target)) ;
                bar->setWindowTitle(tr("Items␣toolbar"));
        }
        if(plot)
        {
                addParent(plot);
                addNewAction(bar, new specTitleAction(target)) ;
                addNewAction(bar, new specXLabelAction(target)) ;
                addNewAction(bar, new specYLabelAction(target)) ;
                bar->addActions(plot->actions());
                bar->setWindowTitle(tr("Plot␣toolbar"));
        }
        if(plotWidget)
        {
                delete bar ;
                bar = plotWidget->createToolbar() ;
                bar-> addSeparator() ;
                bar-> addAction(purgeUndoAction) ;
                bar-> addSeparator() ;
                bar-> addAction(undoAction(this)) ;
                bar-> addAction(redoAction(this)) ;
                bar->setWindowTitle(tr("Main␣toolbar"));
        }
        return bar ;
}
void specActionLibrary::addDragDropPartner(specModel* model)
{
        if(!partners.contains(model))
                partners << model ;
        model->setDropBuddy(this) ;
}
void specActionLibrary::addParent(QObject* pointer)
{
        if(!parents.contains(pointer))
                parents << pointer ;
}
QObject* specActionLibrary::parentId(int num)
{
        return parents[num] ;
}
void specActionLibrary::writeToStream(QDataStream& out) const
{
        out << qint32(undoStack->count()) ;
        out << qint32(undoStack->index()) ;
        int progressToGo = 0, progressStart = 0 ;
```

```
217         if(progress)
            {
                    progressStart = progress->value() ;
                    progressToGo = progress->maximum() - progressStart ;
            }
222         for(int i = 0 ; i < undoStack->count() ; ++i)
            {
                    if(progress) progress->setValue(progressStart + (i * progressToGo) / undoStack->►
                        count());
                    specUndoCommand* command = (specUndoCommand*) undoStack->command(i) ;
                    out << type(command->id()) << qint32(parents.indexOf(command->parentObject()))
227                     << *command ;
            }
            undoStack->setClean();
    }
    specStreamable* specActionLibrary::factory(const type& t) const
232 {
            return commandGenerator.commandById(t) ;
    }
    void specActionLibrary::readFromStream(QDataStream& in)
    {
237         qint32 num, position ;
            in >> num >> position ;
            foreach(QObject * parent, parents)
            qDebug() << "Parent:␣" << parent->objectName() ;
            type t ;
242         QVector<qint32> parentIndex(num, 0) ;
            int progressToGo = 0, progressStart = 0 ;
            if(progress)
            {
                    progressStart = progress->value() ;
247                 progressToGo = progress->maximum() - progressStart ;
            }
            for(int i = 0 ; i < num ; ++i)
            {
                    if(progress) progress->setValue(progressStart + (i * progressToGo) / num);
252                 in >> t >> parentIndex[i] ;
                    specStreamable* streamable = produceItem(in) ;
                    specUndoCommand* command = dynamic_cast<specUndoCommand*>(streamable) ;
                    if(!command)
                    {
257                         qDebug() << "Error␣reading␣command␣no." << i << "of␣type" << t ;
                            undoStack->clear();
                            parentIndex.clear();
                            delete streamable ;
                            continue ;
262                 }
    #ifdef DEBUGCOMMANDREADER
                    qDebug() << "Reading␣item:" << i << "total␣count:" << undoStack->count() << "/" << ►
                        num ;
                    if (QMessageBox::question(0,tr("Really␣read?"),
                                              tr("Really␣read␣command␣no.␣")
267                                           + QString::number(i)
                                              + tr("?␣␣Description␣is:\n")
                                              + command->text(),
                                              QMessageBox::Yes | QMessageBox::No,
                                              QMessageBox::Yes)
272                             == QMessageBox::Yes)
    #endif
                            undoStack->push(command) ;
    #ifdef DEBUGCOMMANDREADER
                    else
277                         delete command ;
    #endif
            }
            undoStack->setIndex(position) ;
            for(int i = 0 ; i < undoStack->count() ; ++i)
282             ((specUndoCommand*) undoStack->command(i))->setParentObject(parents[parentIndex[i]]) ;
            qDebug() << "to␣be␣read:" << num << "actually␣on␣stack:" << undoStack->count() ;
            undoStack->setClean();
    }
    void specActionLibrary::setLastRequested(const QModelIndexList& list)
287 {
            lastRequested = list ;
    }
```

```
int specActionLibrary::moveInternally(const QModelIndex& parent, int row, specModel* target)
{
        int count = lastRequested.size() ;
        specMoveCommand* command = new specMoveCommand ;
        command->setParentObject(target);
        command->setItems(lastRequested, parent, row) ;
        command->setText(tr("Move␣items"));
        push(command) ;
        return count ;
}
int specActionLibrary::deleteInternally(specModel* model)
{
        int count = lastRequested.size() ;
        QList<specModelItem*> pointers = model->pointerList(lastRequested) ;
        specUndoCommand* command = specDeleteAction::command(model, pointers) ;
        command->setText(tr("Move␣items"));
        push(command) ;
        return count ;
}
void specActionLibrary::addPlot(specPlot* plot)
{
        connect(undoStack, SIGNAL(indexChanged(int)), plot, SLOT(replot())) ;
        plot->setUndoPartner(this) ;
}
void specActionLibrary::purgeUndo()
{
        if(QMessageBox::Yes ==
                QMessageBox::question(0,
                                      tr("Really␣Clear␣History?"),
                                      tr("Do␣you␣really␣want␣to␣delete␣all␣undo␣and␣redo␣actions?␣␣►
                                          WARNING:␣␣This␣cannot␣be␣undone."),
                                      QMessageBox::Yes | QMessageBox::No,
                                      QMessageBox::No))
                undoStack->clear();
}
QMenu* specActionLibrary::contextMenu(QWidget* w)
{
        specView* view = dynamic_cast<specView*>(w) ;
        QList<QAction*> actions ;
        actions << view->findChild<specAddConnectionsAction*>()
                << view->findChild<specAddFitAction*>()
                << view->findChild<specRemoveFitAction*>()
                << view->findChild<specToggleFitStyleAction*>()
                << view->findChild<specConductFitAction*>()
                << view->findChild<specTiltMatrixAction*>()
                << view->findChild<specSpectrumCalculatorAction*>()
                << view->findChild<changePlotStyleAction*>()
                << view->findChild<specAddFolderAction*>()
                << view->findChild<specSetMultilineAction*>()
                << view->findChild<specItemPropertiesAction*>()
                << view->findChild<specPasteAction*>()
                << view->findChild<specCopyAction*>()
                << view->findChild<specCutAction*>()
                << view->findChild<specDeleteAction*>() ;
        actions.removeAll(0) ;
        if(actions.isEmpty()) return 0 ;
        QMenu* cMenu = new QMenu(w) ;
        cMenu->addActions(actions) ;
        return cMenu ;
}
specActionLibrary::~specActionLibrary()
{
        delete undoStack ;
}
void specActionLibrary::setProgressDialog(QProgressDialog* p)
{
        progress = p ;
}
```

src/actionlib/speccommandgenerator.h

```
#ifndef SPECCOMMANDGENERATOR_H
#define SPECCOMMANDGENERATOR_H
class specUndoCommand ;
```

```
class specCommandGenerator
{
private:
        specUndoCommand* parent ;
public:
        specCommandGenerator(specUndoCommand* parent = 0) ;
        specUndoCommand* commandById(int id) const ;
};
#endif
```

src/actionlib/speccommandgenerator.cpp

```
#include "speccommandgenerator.h"
#include "specaddfoldercommand.h"
#include "specdeletecommand.h"
#include "specmovecommand.h"
#include "specmulticommand.h"
#include "specexchangedatacommand.h"
#include "specresizesvgcommand.h"
#include "specaddconnectionscommand.h"
#include "specdeleteconnectionscommand.h"
#include "speceditdescriptorcommand.h"
#include "specmetarangecommand.h"
#include "specplotlabelcommand.h"
#include "specexchangefitcurvecommand.h"
#include "spectogglefitstylecommand.h"
#include "specstylecommand.h"
#include "specdescriptorflagscommand.h"
#include "specdeletedescriptorcommand.h"
#include "specrenamedescriptorcommand.h"
#include "specexchangefiltercommand.h"
specCommandGenerator::specCommandGenerator(specUndoCommand* p)
        : parent(p)
{
}
specUndoCommand* specCommandGenerator::commandById(int id) const
{
        switch(id)
        {
                case specStreamable::toggleFitStyleCommand:
                        return new specToggleFitStyleCommand(parent) ;
                case specStreamable::exchangeFitCommand:
                        return new specExchangeFitCurveCommand(parent) ;
                case specStreamable::deleteCommandId :
                        return new specDeleteCommand(parent) ;
                case specStreamable::newFolderCommandId :
                        return new specAddFolderCommand(parent) ;
                case specStreamable::moveItemsCommandId :
                        return new specMoveCommand(parent) ;
                case specStreamable::exchangeFilterCommandId :
                        return new specExchangeFilterCommand(parent) ;
                case specStreamable::movePlotCommandId :
                        return new specExchangeFilterCommand(parent, true) ;
                case specStreamable::multiCommandId :
                        return new specMultiCommand(parent) ;
                case specStreamable::exchangeDataCommandId:
                        return new specExchangeDataCommand(parent) ;
                case specStreamable::resizeSVGCommandId :
                        return new specResizeSVGcommand(parent) ;
                case specStreamable::newConnectionsCommandId :
                        return new specAddConnectionsCommand(parent) ;
                case specStreamable::deleteConnectionsCommandId :
                        return new specDeleteConnectionsCommand(parent) ;
                case specStreamable::editDescriptorCommandId :
                        return new specEditDescriptorCommand(parent) ;
                case specStreamable::penColorCommandId:
                case specStreamable::lineWidthCommandId:
                case specStreamable::symbolStyleCommandId:
                case specStreamable::symbolPenColorCommandId:
                case specStreamable::symbolSizeCommandId:
                case specStreamable::symbolBrushColorCommandId:
                case specStreamable::penStyleCommandId:
                        return generateStyleCommand(id, parent) ;
                case specStreamable::metaRangeCommand :
```

```
63                          return new specMetaRangeCommand(parent) ;
                    case specStreamable::plotTitleCommandId:
                    case specStreamable::plotYLabelCommandId:
                    case specStreamable::plotXLabelCommandId:
                            return generatePlotLabelCommand(id, parent) ;
68                  case specStreamable::descriptorFlagsCommand:
                            return new specDescriptorFlagsCommand(parent) ;
                    case specStreamable::deleteDescriptorCommandId:
                            return new specDeleteDescriptorCommand(parent) ;
                    case specStreamable::renameDescriptorCommandId:
73                          return new specRenameDescriptorCommand(parent) ;
                    default:
                            return 0 ;
            }
    }
```

src/actionlib/specworkerthread.h

```
    #ifndef SPECWORKERTHREAD_H
    #define SPECWORKERTHREAD_H
3   #include <QThread>
    #include <QProgressDialog>
    class specWorkerThread : public QThread
    {
            Q_OBJECT
8   protected:
            bool toTerminate ;
            QProgressDialog dialog ;
    public:
            explicit specWorkerThread(int maxVal, QObject* parent = 0);
13  signals:
            void progressValue(int) ;
    public slots:
            void finish() ;
    };
18  #endif
```

src/actionlib/specworkerthread.cpp

```
    #include "specworkerthread.h"
2   specWorkerThread::specWorkerThread(int maxVal, QObject* parent) :
            QThread(parent),
            toTerminate(false),
            dialog("Progress", "Cancel", 0, maxVal)
    {
7           dialog.setMinimumDuration(10) ;
            connect(this, SIGNAL(progressValue(int)), &dialog, SLOT(setValue(int))) ;
            connect(&dialog, SIGNAL(canceled()), this, SLOT(finish())) ;
    }
    void specWorkerThread::finish()
12  {
            toTerminate = true ;
    }
```

src/asciiexporter.h

```
1   #ifndef ASCIIEXPORTER_H
    #define ASCIIEXPORTER_H
    #include <QVector>
    #include <QString>
    #include "specspectrumplot.h"
6   #include "specmodel.h"
    #include "speclogwidget.h"
    #include "speckineticwidget.h"
    #include "specdataview.h"
    class QMimeData ;
11  class asciiExporter
    {
    public:
            enum modelType { log, data, meta } ;
```

```
        asciiExporter(modelType model);
        QString content(QVector<int>) const ;
        ~asciiExporter() ;
        void readFromStream(QDataStream& in) ;
private:
        specModel* modelPointer ;
        specSpectrumPlot plot ;
        specDataView view ;
        specModel itemModel ;
        specLogWidget logWidget ;
        specKineticWidget kineticWidget ;
};
#endif
```

src/asciiexporter.cpp

```
#include "asciiexporter.h"
#include "specspectrumplot.h"
#include "specdataview.h"
#include "speclogwidget.h"
#include "speckineticwidget.h"
#include <QMimeData>
#include "specmimetextexporter.h"
asciiExporter::asciiExporter(modelType m)
        : modelPointer(0),
          kineticWidget()
{
        view.setModel(&itemModel) ;
        kineticWidget.view()->assignDataView(&view);
        new specMimeTextExporter(view.model()) ;
        switch(m)
        {
                case data: modelPointer = view.model() ; break ;
                case log: modelPointer = logWidget.view()->model() ; break ;
                case meta: modelPointer = kineticWidget.view()->model() ; break ;
        }
}
void asciiExporter::readFromStream(QDataStream& in)
{
        in >> plot
           >> view
           >> logWidget
           >> kineticWidget ;
}
asciiExporter::~asciiExporter()
{
}
QString asciiExporter::content(QVector<int> h) const
{
        if(!modelPointer) return QString() ;
        QVector<int> hierarchy(h.size()) ;
        for(int i = 0 ; i < h.size() ; ++i)
                hierarchy[i] = h[h.size() - 1 - i] ;
        QMimeData* data = modelPointer->mimeData(QModelIndexList() << modelPointer->index(hierarchy)►
            ) ;
        QString result = data->text() ;
        delete data ;
        return result ;
}
```

src/cutbyintensitydialog.h

```
#ifndef CUTBYINTENSITYDIALOG_H
#define CUTBYINTENSITYDIALOG_H
#include <QDialog>
#include <QVBoxLayout>
#include <QPushButton>
#include <QDialogButtonBox>
#include "specplot.h"
#include "specmodelitem.h"
class cutByIntensityDialog : public QDialog
{
```

```
        Q_OBJECT
private:
        QVBoxLayout* layout ;
        specPlot* plot ;
        QPushButton* newRange, *deleteRange ;
        QList<specModelItem*> items ;
        QDialogButtonBox* buttons ;
        CanvasPicker* picker ;
        int huevalue ;
private slots:
        void addRange() ;
        void removeRange() ;
        void rangeModified(specCanvasItem* range, int point, double newX, double newY) ;
public:
        explicit cutByIntensityDialog(QWidget* parent = 0);
        ~cutByIntensityDialog() ;
        void assignSpectra(QList<specModelItem*>) ;
        QList<specRange*> ranges() ;
};
#endif
```

src/cutbyintensitydialog.cpp

```
#include "cutbyintensitydialog.h"
#include <qwt_scale_div.h>
#include "canvaspicker.h"
cutByIntensityDialog::cutByIntensityDialog(QWidget* parent) :
        QDialog(parent), huevalue(0)
{
        setWindowTitle(tr("Delete␣data␣by␣range")) ;
        layout = new QVBoxLayout(this) ;
        plot = new specPlot(this) ;
        newRange = new QPushButton("new␣range", this) ;
        deleteRange = new QPushButton("delete␣range", this) ;
        buttons = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::Cancel, Qt::►
            Horizontal, this) ;
        picker = new CanvasPicker(plot) ;
        layout->addWidget(plot) ;
        layout->addWidget(newRange) ;
        layout->addWidget(deleteRange) ;
        layout->addWidget(buttons) ;
        plot->enableAxis(QwtPlot::yRight, true) ;
        plot->setAxisScale(QwtPlot::yRight, 0, 33e3) ;
        plot->setAutoReplot(false) ;
        plot->setAutoDelete(true) ;
        picker->setOwning() ;
        connect(deleteRange, SIGNAL(clicked()), this, SLOT(removeRange())) ;
        connect(newRange, SIGNAL(clicked()), this, SLOT(addRange())) ;
        connect(buttons, SIGNAL(accepted()), this, SLOT(accept())) ;
        connect(buttons, SIGNAL(rejected()), this, SLOT(reject())) ;
        connect(picker, SIGNAL(pointMoved(specCanvasItem*, int, double, double)), this, SLOT(►
            rangeModified(specCanvasItem*, int, double, double))) ;
}
void cutByIntensityDialog::assignSpectra(QList<specModelItem*> spectra)
{
        foreach(specModelItem * item, spectra)
        {
                if(!items.contains(item))
                {
                        items << item ;
                        QwtPlotCurve* intensity = new QwtPlotCurve ;
                        QwtPlotCurve* dataCurve = new QwtPlotCurve ;
                        QVector<double> xdat, ydat ;
                        for(size_t i = 0 ; i < item->dataSize() ; i++)
                        {
                                xdat << item->sample(i).x() ;
                                ydat << item->sample(i).y() ;
                        }
                        intensity->setSamples(xdat, item->intensityData()) ;
                        dataCurve->setSamples(xdat, ydat);
                        intensity->setYAxis(QwtPlot::yRight);
                        QPen pen = item->pen() ;
                        dataCurve->setPen(pen) ;
                        pen.setStyle(Qt::DashLine) ;
```

```
                        intensity->setPen(pen) ;
                        dataCurve->attach(plot) ;
                        intensity->attach(plot) ;
                }
        }
        QRectF boundaries ;
        foreach(specModelItem * item, items)
        boundaries |= item->boundingRect() ;
        QSizeF size = boundaries.size() ;
        boundaries.translate(-.05 * size.width(), -.05 * size.height()) ;
        boundaries.setSize(1.1 * size);
        plot->setAxisScale(QwtPlot::yLeft, boundaries.top(), boundaries.bottom()) ;
        plot->setAxisScale(QwtPlot::xBottom, boundaries.left(), boundaries.right());
        plot->setAutoScaling(false) ;
        plot->replot() ;
}
QList<specRange*> cutByIntensityDialog::ranges()
{
        QList<specCanvasItem*> selectable = picker->items() ;
        QList<specRange*> allRanges ;
        foreach(specCanvasItem * pointer, selectable)
        allRanges << (specRange*) pointer ;
        return allRanges ;
}
void cutByIntensityDialog::addRange()
{
        double min = plot->axisScaleDiv(QwtPlot::xBottom).lowerBound(), max = plot->axisScaleDiv(►
             QwtPlot::xBottom).upperBound() ;
        specRange* range = new specRange(min + .1 * (max - min), max - .1 * (max - min),
                                         (plot->axisScaleDiv(QwtPlot::yLeft).lowerBound() +
                                          plot->axisScaleDiv(QwtPlot::yLeft).upperBound()) / 2.) ;
        range->attach(plot) ;
        picker->addSelectable(range) ;
}
void cutByIntensityDialog::removeRange()
{
        picker->removeSelected();
}
cutByIntensityDialog::~cutByIntensityDialog()
{
        delete picker ;
}
void cutByIntensityDialog::rangeModified(specCanvasItem* range, int point, double newX, double newY)
{
        range->pointMoved(point, newX, newY);
        plot->replot();
}
```

src/kinetic/metaitemproperties.h

```
#ifndef METAITEMPROPERTIES_H
#define METAITEMPROPERTIES_H
#include <QDialog>
#include "specmetaitem.h"
#include <QMap>
class QListWidgetItem ;
class QTableWidgetItem ;
namespace Ui
{
        class metaItemProperties;
}
class metaItemProperties : public QDialog
{
        Q_OBJECT
public:
        explicit metaItemProperties(specMetaItem*, QWidget* parent = 0);
        ~metaItemProperties();
        specUndoCommand* changedConnections(QObject* parent) ;
private slots:
        void on_connectedItemsList_itemSelectionChanged();
        void on_moveUpButton_clicked();
        void on_moveDownButton_clicked();
        void on_dataTable_itemSelectionChanged();
private:
```

```
        Ui::metaItemProperties* ui;
        specMetaItem* originalItem ;
        void moveSelection(bool) ;
        QwtPlotCurve metaCurve, currentCurve, selectedCurve ;
        struct itemLink
        {
                specModelItem* item ;
                QVector<QTableWidgetItem*> points ;
                QwtPlotCurve* curve ;
        };
        QMap<QListWidgetItem*, itemLink> itemInfo ;
        struct pointLink
        {
                QVector<QListWidgetItem*> items ;
                QPointF value ;
        };
        QMap<QTableWidgetItem*, pointLink> pointInfo ;
        QMap<specModelItem*, QListWidgetItem*> modelItemInfo ;
        void buildAssignments(const QList<specModelItem*>& items) ;
        void buildPoints() ;
        void refreshPlots() ;
        void clearItemInfo() ;
        QTableWidgetItem* firstEntry(QTableWidgetItem*) ;
        bool reselectingDataPoints, reselectingItems ;
        void checkSelection(Qt::CheckState) ;
private slots:
        void on_connectedItemsList_itemChanged(QListWidgetItem* item);
        void on_removeSelectedConnections_clicked();
        void on_addSelectedConnections_clicked();
};
#endif
```

src/kinetic/metaitemproperties.cpp

```
#include "metaitemproperties.h"
#include "ui_metaitemproperties.h"
#include "specaddconnectionscommand.h"
#include "specdeleteconnectionscommand.h"
#include "specmulticommand.h"
#include "spectogglefitstylecommand.h"
#include <QTableWidgetItem>
metaItemProperties::metaItemProperties(specMetaItem* i, QWidget* parent) :
        QDialog(parent),
        ui(new Ui::metaItemProperties),
        originalItem(i),
        reselectingDataPoints(false),
        reselectingItems(false)
{
        ui->setupUi(this);
        ui->moveUpButton->setIcon(QIcon::fromTheme("go-up"));
        ui->moveDownButton->setIcon(QIcon::fromTheme("go-down")) ;
        ui->addSelectedConnections->setIcon(QIcon(":/toKinetic.png")) ;
        ui->removeSelectedConnections->setIcon(QIcon::fromTheme("edit-delete")) ;
        ui->itemPreview->setAutoDelete(false) ;
        ui->styleFit->setDisabled(!i->getFitCurve()) ;
        ui->styleFit->setCheckState(i->styleFitCurve ? Qt::Checked : Qt::Unchecked) ;
        metaCurve.attach(ui->metaPlot) ;
        metaCurve.setData(i->filter->evaluate(i->items.toVector())) ;
        selectedCurve.setStyle(QwtPlotCurve::NoCurve);
        selectedCurve.setSymbol(new QwtSymbol(QwtSymbol::Ellipse, QBrush(Qt::black), QPen(Qt::blue),►
              QSize(5, 5)));
        selectedCurve.attach(ui->metaPlot) ;
        currentCurve.setStyle(QwtPlotCurve::NoCurve);
        currentCurve.setSymbol(new QwtSymbol(QwtSymbol::Ellipse, QBrush(Qt::red), QPen(Qt::red), ►
             QSize(5, 5)));
        currentCurve.attach(ui->metaPlot) ;
        buildAssignments(originalItem->serverList());
}
void metaItemProperties::refreshPlots()
{
        ui->itemPreview->detachItems(QwtPlotItem::Rtti_PlotItem, false) ;
        foreach(QListWidgetItem * item, ui->connectedItemsList->selectedItems())
        itemInfo[item].curve->attach(ui->itemPreview) ;
        QVector<QPointF> selectedPoints ;
```

```
        QSet<QTableWidgetItem*> selectedRows ;
        foreach(QTableWidgetItem * item, ui->dataTable->selectedItems())
        selectedRows += firstEntry(item) ;
        foreach(QTableWidgetItem * item, selectedRows)
        selectedPoints << pointInfo[item].value ;
        selectedCurve.setSamples(selectedPoints) ;
        if(QTableWidgetItem* currentItem = ui->dataTable->currentItem())
                currentCurve.setSamples(QVector<QPointF>() << pointInfo[firstEntry(currentItem)].►
                    value);
        else
                currentCurve.setSamples(QVector<QPointF>());
        ui->itemPreview->replot();
        ui->metaPlot->replot();
}
QTableWidgetItem* metaItemProperties::firstEntry(QTableWidgetItem* item)
{
        return ui->dataTable->item(item->row(), 0) ;
}
metaItemProperties::~metaItemProperties()
{
        clearItemInfo() ;
        delete ui;
}
void metaItemProperties::on_connectedItemsList_itemSelectionChanged()
{
        if(reselectingItems) return ;
        reselectingItems = true ;
        QItemSelection selection ;
        if (!reselectingDataPoints)
        {
                foreach(QListWidgetItem * item, ui->connectedItemsList->selectedItems())
                {
                        foreach(QTableWidgetItem * point, itemInfo[item].points)
                        {
                                QModelIndex index = ui->dataTable->model()->index(ui->dataTable->row►
                                    (point), 0) ;
                                selection.append(QItemSelection(index, ui->dataTable->model()->index►
                                    (index.row(), ui->dataTable->columnCount() - 1)));
                        }
                }
                ui->dataTable->selectionModel()->select(selection, QItemSelectionModel::►
                    ClearAndSelect);
                refreshPlots();
        }
        QModelIndexList selectionList = ui->connectedItemsList->selectionModel()->selectedIndexes() ;
        if(!selectionList.isEmpty())
        {
                ui->moveUpButton->setEnabled(selectionList.first().row()) ;
                ui->moveDownButton->setEnabled(selectionList.last().row() + 1
                                               != ui->connectedItemsList->count());
                ui->addSelectedConnections->setEnabled(true) ;
                ui->removeSelectedConnections->setEnabled(true) ;
        }
        else
        {
                ui->moveDownButton->setDisabled(true) ;
                ui->moveUpButton->setDisabled(true) ;
                ui->addSelectedConnections->setDisabled(true) ;
                ui->removeSelectedConnections->setDisabled(true);
        }
        reselectingItems = false ;
}
void metaItemProperties::on_moveUpButton_clicked()
{
        moveSelection(false);
}
void metaItemProperties::on_moveDownButton_clicked()
{
        moveSelection(true) ;
}
void metaItemProperties::clearItemInfo()
{
        foreach(itemLink link, itemInfo)
        delete link.curve ;
        itemInfo.clear();
```

```
        modelItemInfo.clear();
        ui->connectedItemsList->clear();
}
void metaItemProperties::buildAssignments(const QList<specModelItem*>& items)
{
        QVector<specModelItem*> selectedItems ;
        foreach(QListWidgetItem * listItem, ui->connectedItemsList->selectedItems())
        selectedItems << itemInfo[listItem].item ;
        clearItemInfo();
        for(QList<specModelItem*>::const_iterator i = items.begin() ; i != items.end() ; ++i)
        {
                QListWidgetItem* newItem = new QListWidgetItem((*i)->descriptor("")) ;
                newItem->setCheckState(Qt::Checked);
                ui->connectedItemsList->addItem(newItem) ;
                itemLink info ;
                info.item = *i ;
                info.curve = new QwtPlotCurve ;
                QVector<QPointF> curveData ;
                for(size_t j = 0 ; j < (*i)->dataSize() ; ++j)
                        curveData << (*i)->sample(j) ;
                info.curve->setSamples(curveData) ;
                modelItemInfo[*i] = newItem ;
                itemInfo[newItem] = info ;
        }
        buildPoints();
        QItemSelection selection ;
        foreach(specModelItem * modelItem, selectedItems)
        {
                QModelIndex itemIndex =
                    ui->connectedItemsList->model()->index(
                        ui->connectedItemsList->row(modelItemInfo[modelItem]), 0) ;
                selection.append(QItemSelectionRange());
        }
        ui->connectedItemsList->selectionModel()->select(selection, QItemSelectionModel::►
            ClearAndSelect);
}
void metaItemProperties::buildPoints()
{
        pointInfo.clear();
        ui->dataTable->clear();
        ui->dataTable->setRowCount(0);
        ui->dataTable->setColumnCount(2 + originalItem->filter->evaluators.size());
        ui->dataTable->setHorizontalHeaderLabels(QStringList() << "x" << "y" << originalItem->filter►
            ->symbols);
        for(QMap<QListWidgetItem*, itemLink>::iterator i = itemInfo.begin() ; i != itemInfo.end() ; ►
            ++i)
                i->points.clear() ;
        QVector<specModelItem*> itemVector ;
        for(int i = 0 ; i < ui->connectedItemsList->count() ; ++i)
        {
                QListWidgetItem* listItem = ui->connectedItemsList->item(i) ;
                if(listItem->checkState() == Qt::Checked)
                        itemVector << itemInfo[listItem].item ;
        }
        QVector<QVector<specModelItem*> > itemsPerPoint ;
        originalItem->filter->itemsToQuery(itemVector, itemsPerPoint) ;
        QVector<QPointF> evaluatedData ;
        for(int i = 0 ; i < itemsPerPoint.size() ; ++i)
        {
                QVector<QPointF> dataPoints ;
                QVector<QVector<double> > variableValues ;
                originalItem->filter->getVariableValues(itemsPerPoint[i], variableValues) ;
                originalItem->filter->getPoints(variableValues, dataPoints) ;
                pointLink info ;
                foreach(specModelItem * modelItem, itemsPerPoint[i])
                info.items << modelItemInfo[modelItem] ;
                for(int i = 0 ; i < dataPoints.size() ; ++i)
                {
                        QTableWidgetItem* newTableEntry = new QTableWidgetItem(QString::number(►
                            dataPoints[i].x())) ;
                        foreach(QListWidgetItem * listItem, info.items)
                        itemInfo[listItem].points << newTableEntry ;
                        info.value = dataPoints[i] ;
                        int row = ui->dataTable->rowCount() ;
                        ui->dataTable->setRowCount(row + 1) ;
```

```
                        ui->dataTable->setItem(row, 0, newTableEntry) ;
                        ui->dataTable->setItem(row, 1, new QTableWidgetItem(QString::number(►
                            dataPoints[i].y()))) ;
                        for(int j = 0 ; j < variableValues[i].size() ; ++j)
                                ui->dataTable->setItem(row, 2 + j, new QTableWidgetItem(QString::►
                                    number(variableValues[i][j])));
                        pointInfo[newTableEntry] = info ;
                }
                evaluatedData << dataPoints ;
        }
        metaCurve.setSamples(evaluatedData);
}
void metaItemProperties::moveSelection(bool down)
{
        QList<QListWidgetItem*> selection = ui->connectedItemsList->selectedItems() ;
        if(selection.isEmpty()) return ;
        int lastRow = ui->connectedItemsList->row(selection.last()),
            firstRow = ui->connectedItemsList->row(selection.first()) ;
        ui->connectedItemsList->selectionModel()->clearSelection();
        if(down)
        {
                if(lastRow == ui->connectedItemsList->count() - 1)
                        return ;
                ui->connectedItemsList->insertItem(firstRow,
                                                   ui->connectedItemsList->takeItem(lastRow + 1));
        }
        else
        {
                if(!firstRow)
                        return ;
                ui->connectedItemsList->insertItem(lastRow,
                                                   ui->connectedItemsList->takeItem(firstRow - 1));
        }
        buildPoints();
        QItemSelection s ;
        foreach(QListWidgetItem * item, selection)
        {
                QModelIndex index = ui->connectedItemsList->model()->index(ui->connectedItemsList->►
                    row(item), 0) ;
                s.append(QItemSelection(index, index)) ;
        }
        ui->connectedItemsList->selectionModel()->select(s, QItemSelectionModel::ClearAndSelect) ;
}
specUndoCommand* metaItemProperties::changedConnections(QObject* parent)
{
        specModel* model = qobject_cast<specModel*> (parent) ;
        if(!model) return 0 ;
        specMultiCommand* parentCommand = new specMultiCommand ;
        parentCommand->setParentObject(parent) ;
        parentCommand->setText(tr("Modify␣properties␣of␣meta␣item␣") + originalItem->descriptor(""));
        specDeleteConnectionsCommand* deleteCommand = new specDeleteConnectionsCommand(parentCommand►
            ) ;
        deleteCommand->setItems(originalItem, originalItem->serverList()) ;
        deleteCommand->redo();
        QList<specModelItem*> newConnections ;
        for(int i = 0 ; i < ui->connectedItemsList->count() ; ++i)
        {
                QListWidgetItem* widgetItem = ui->connectedItemsList->item(i) ;
                if(widgetItem->checkState() == Qt::Checked)
                        newConnections << itemInfo[widgetItem].item ;
        }
        if(!newConnections.isEmpty())
                (new specAddConnectionsCommand(parentCommand)) ->
                setItems(originalItem, newConnections) ;
        deleteCommand->undo();
        if(ui->styleFit->checkState() != originalItem->styleFitCurve)
        {
                specToggleFitStyleCommand* fitStyleCommand = new specToggleFitStyleCommand(►
                    parentCommand) ;
                fitStyleCommand->setParentObject(parent) ;
                fitStyleCommand->setItem(originalItem);
        }
        if(!parentCommand->childCount())
        {
                delete parentCommand ;
```

```
                parentCommand = 0 ;
        }
        return parentCommand ;
}
void metaItemProperties::on_dataTable_itemSelectionChanged()
{
        if(reselectingDataPoints) return ;
        reselectingDataPoints = true ;
        QItemSelection selection ;
        foreach(QTableWidgetItem * point, ui->dataTable->selectedItems())
        {
                foreach(QListWidgetItem * item, pointInfo[firstEntry(point)].items)
                {
                        QModelIndex index = ui->connectedItemsList->model()->index(ui->►
                            connectedItemsList->row(item), 0) ;
                selection.append(QItemSelection(index, index)) ;
                }
        }
        ui->connectedItemsList->selectionModel()->select(selection, QItemSelectionModel::►
            ClearAndSelect);
        refreshPlots();
        reselectingDataPoints = false ;
}
void metaItemProperties::on_connectedItemsList_itemChanged(QListWidgetItem* item)
{
        Q_UNUSED(item)
        buildPoints();
        on_connectedItemsList_itemSelectionChanged();
}
void metaItemProperties::on_removeSelectedConnections_clicked()
{
        checkSelection(Qt::Unchecked) ;
}
void metaItemProperties::checkSelection(Qt::CheckState state)
{
        QList<QListWidgetItem*> selectedItems = ui->connectedItemsList->selectedItems() ;
        foreach(QListWidgetItem * item, selectedItems)
        item->setCheckState(state) ;
}
void metaItemProperties::on_addSelectedConnections_clicked()
{
        checkSelection(Qt::Checked);
}
```

src/kinetic/metaitemproperties.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>metaItemProperties</class>
 <widget class="QDialog" name="metaItemProperties">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>505</width>
    <height>560</height>
   </rect>
  </property>
  <property name="sizePolicy">
   <sizepolicy hsizetype="Preferred" vsizetype="Fixed">
    <horstretch>0</horstretch>
    <verstretch>0</verstretch>
   </sizepolicy>
  </property>
  <property name="windowTitle">
   <string>Connected Items</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_4">
   <item>
    <widget class="QLabel" name="label">
     <property name="sizePolicy">
      <sizepolicy hsizetype="Expanding" vsizetype="Fixed">
       <horstretch>0</horstretch>
       <verstretch>0</verstretch>
```

```
      </sizepolicy>
     </property>
     <property name="text">
      <string>Rearrange connected items, unchecking will break the connection.</string>
     </property>
    </widget>
   </item>
   <item>
    <widget class="QSplitter" name="plotListSplitter">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <widget class="QSplitter" name="splitter">
      <property name="orientation">
       <enum>Qt::Vertical</enum>
      </property>
      <widget class="QWidget" name="layoutWidget">
       <layout class="QVBoxLayout" name="verticalLayout_2">
        <item>
         <widget class="QLabel" name="label_2">
          <property name="text">
           <string>Connected items</string>
          </property>
          <property name="buddy">
           <cstring>connectedItemsList</cstring>
          </property>
         </widget>
        </item>
        <item>
         <widget class="QListWidget" name="connectedItemsList">
          <property name="sizePolicy">
           <sizepolicy hsizetype="MinimumExpanding" vsizetype="MinimumExpanding">
            <horstretch>0</horstretch>
            <verstretch>0</verstretch>
           </sizepolicy>
          </property>
          <property name="selectionMode">
           <enum>QAbstractItemView::ContiguousSelection</enum>
          </property>
         </widget>
        </item>
        <item>
         <layout class="QHBoxLayout" name="horizontalLayout">
          <item>
           <widget class="QToolButton" name="moveUpButton">
            <property name="enabled">
             <bool>false</bool>
            </property>
            <property name="text">
             <string>...</string>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QToolButton" name="moveDownButton">
            <property name="enabled">
             <bool>false</bool>
            </property>
            <property name="text">
             <string>...</string>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QToolButton" name="removeSelectedConnections">
            <property name="enabled">
             <bool>false</bool>
            </property>
            <property name="text">
             <string>...</string>
            </property>
           </widget>
          </item>
          <item>
           <widget class="QToolButton" name="addSelectedConnections">
```

```
            <property name="enabled">
             <bool>false</bool>
            </property>
            <property name="text">
             <string>...</string>
            </property>
           </widget>
          </item>
          <item>
           <spacer name="horizontalSpacer">
            <property name="orientation">
             <enum>Qt::Horizontal</enum>
            </property>
            <property name="sizeHint" stdset="0">
             <size>
              <width>40</width>
              <height>20</height>
             </size>
            </property>
           </spacer>
          </item>
         </layout>
        </item>
       </layout>
      </widget>
      <widget class="QWidget" name="layoutWidget1">
       <layout class="QVBoxLayout" name="verticalLayout_3">
        <item>
         <widget class="QLabel" name="dataPointsLabel">
          <property name="text">
           <string>Data points</string>
          </property>
          <property name="buddy">
           <cstring>dataTable</cstring>
          </property>
         </widget>
        </item>
        <item>
         <widget class="QTableWidget" name="dataTable">
          <property name="editTriggers">
           <set>QAbstractItemView::NoEditTriggers</set>
          </property>
          <property name="columnCount">
           <number>2</number>
          </property>
          <attribute name="horizontalHeaderVisible">
           <bool>true</bool>
          </attribute>
          <column>
           <property name="text">
            <string>x</string>
           </property>
          </column>
          <column>
           <property name="text">
            <string>y</string>
           </property>
          </column>
         </widget>
        </item>
       </layout>
      </widget>
     </widget>
     <widget class="QWidget" name="layoutWidget2">
      <layout class="QVBoxLayout" name="verticalLayout">
       <item>
        <widget class="specPlot" name="itemPreview">
         <property name="frameShape">
          <enum>QFrame::StyledPanel</enum>
         </property>
         <property name="frameShadow">
          <enum>QFrame::Raised</enum>
         </property>
        </widget>
       </item>
```

```
179         <item>
             <widget class="specPlot" name="metaPlot">
              <property name="frameShape">
               <enum>QFrame::StyledPanel</enum>
              </property>
184           <property name="frameShadow">
               <enum>QFrame::Raised</enum>
              </property>
             </widget>
            </item>
189         <item>
             <widget class="QCheckBox" name="styleFit">
              <property name="enabled">
               <bool>true</bool>
              </property>
194           <property name="text">
               <string>Styling operations apply to fit</string>
              </property>
             </widget>
            </item>
199        </layout>
       </widget>
      </widget>
     </item>
     <item>
204   <widget class="QDialogButtonBox" name="buttonBox">
       <property name="standardButtons">
        <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
       </property>
      </widget>
209  </item>
 </layout>
</widget>
<customwidgets>
 <customwidget>
214  <class>specPlot</class>
  <extends>QFrame</extends>
  <header>specplot.h</header>
  <container>1</container>
 </customwidget>
219 </customwidgets>
<resources/>
<connections>
 <connection>
  <sender>buttonBox</sender>
224  <signal>accepted()</signal>
  <receiver>metaItemProperties</receiver>
  <slot>accept()</slot>
  <hints>
   <hint type="sourcelabel">
229  <x>482</x>
    <y>307</y>
   </hint>
   <hint type="destinationlabel">
    <x>202</x>
234  <y>301</y>
   </hint>
  </hints>
 </connection>
 <connection>
239  <sender>buttonBox</sender>
  <signal>rejected()</signal>
  <receiver>metaItemProperties</receiver>
  <slot>reject()</slot>
  <hints>
244  <hint type="sourcelabel">
    <x>492</x>
    <y>311</y>
   </hint>
   <hint type="destinationlabel">
249  <x>292</x>
    <y>307</y>
   </hint>
  </hints>
 </connection>
```

```
 </connections>
 <slots>
  <slot>selectionChanged()</slot>
 </slots>
</ui>
```

src/kinetic/specdescriptorvariable.h

```
#ifndef SPECDESCRIPTORVARIABLE_H
#define SPECDESCRIPTORVARIABLE_H
#include "specmetavariable.h"
class specDescriptorVariable : public specMetaVariable
{
public:
        QVector<double> values(specModelItem*, const QVector<double>&) const ;
};
#endif
```

src/kinetic/specdescriptorvariable.cpp

```
#include "specdescriptorvariable.h"
QVector<double> specDescriptorVariable::values(specModelItem* item, const QVector<double>& xvals) ►
    const
{
        return QVector<double> (xvals.size(), item->descriptor(descriptor).toDouble()) ;
}
```

src/kinetic/specfitcurve.h

```
#ifndef SPECFITCURVE_H
#define SPECFITCURVE_H
#include "specstreamable.h"
#include <muParser.h>
#include "speccanvasitem.h"
#include <QRectF>
#include "specdescriptor.h"
class specFitCurve : public specCanvasItem
{
private:
        class fitData : public QwtSeriesData<QPointF>
        {
        private:
                static const int fixedSize = 1000 ;
                mu::Parser* parser ;
                double* x ;
                QVector<QPointF> samples ;
        public:
                QString errorString ;
                fitData(mu::Parser* p = 0) ;
                ~fitData() ;
                QRectF boundingRect() const ;
                QPointF sample(size_t i) const ;
                void setRectOfInterest(const QRectF&) ;
                size_t size() const ;
                void reevaluate() ;
        };
        typedef QPair<QString, double> variablePair;
        QList<variablePair> variables ;
        double confidenceInterval(const QString& v) const ;
        QString plainVariableName(const QString& v) const ;
        QVector<double> numericalErrors ;
        qint16 activeVar ;
        QStringList fitParameters ;
        specDescriptor expression ;
        mu::Parser* parser ;
        QString errorString ;
        static bool acceptableVariable(const QString&) ;
        inline static bool inacceptableVariable(const QString& s) { return !acceptableVariable(s) ; }
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
```

```
        type typeId() const { return specStreamable::fitCurve ; }
        void clearParser() ;
        void generateParser() ;
        void setParserConstants();
        bool variablesMulti, expressionMulti, messagesMulti ;
public:
        specFitCurve();
        ~specFitCurve() ;
        QStringList descriptorKeys() ;
        static QStringList genericDescriptorKeys() ;
        QString descriptor(const QString& key, bool full = false) ;
        QString editDescriptor(const QString& key) ;
        bool changeDescriptor(QString key, QString value) ;
        bool setActiveLine(const QString& key, int n) ;
        int activeLine(const QString& key) const ;
        void refreshPlotData();
        void refit(QwtSeriesData<QPointF>* data) ;
        QVector<double> getFitData(QwtSeriesData<QPointF>* data) ;
        int rtti() const { return specStreamable::fitCurve ; }
        void attach(QwtPlot* plot) ;
        void setDescriptorProperties(QString key, spec::descriptorFlags f) ;
        spec::descriptorFlags descriptorProperties(const QString& key) const ;
};
typedef struct
{
        double* x;
        const double* y;
        mu::Parser* parser ;
        QVector<std::string>* variableNames ;
} lmcurve_data_struct;
void evaluateParser(const double* parameters, int count, const void* data, double* fitResults, int* ►
    info) ;
#endif
```

src/kinetic/specfitcurve.cpp

```
#include "specfitcurve.h"
#include "lmmin.h"
#include <algorithm>
#include <gsl/gsl_cdf.h>
specFitCurve::fitData::fitData(mu::Parser* p)
        : parser(p),
          x(0),
          samples(fixedSize)
{
        if(parser)
                parser->DefineVar("x", x = new double[samples.size()]) ;
}
specFitCurve::fitData::~fitData()
{
        delete[] x ;
}
QRectF specFitCurve::fitData::boundingRect() const
{
        return d_boundingRect ;
}
void specFitCurve::fitData::setRectOfInterest(const QRectF& r)
{
        d_boundingRect = r ;
        reevaluate() ;
}
size_t specFitCurve::fitData::size() const
{
        return samples.size() ;
}
QPointF specFitCurve::fitData::sample(size_t i) const
{
        return samples[i] ;
}
void specFitCurve::fitData::reevaluate()
{
        double* y = new double[samples.size()] ;
        const double step = d_boundingRect.width() / (samples.size() - 1) ;
        double xc = d_boundingRect.left() ;
```

```
        for(int i = 0 ; i < samples.size() ; ++i)
        {
                x[i] = xc ;
                y[i] = NAN ;
                xc += step ;
        }
        if(parser)
        {
                try
                {
                        parser->Eval(y, samples.size()) ;
                }
                catch(mu::Parser::exception_type& p)
                {
                        errorString = QString("Evaluation␣of␣fit␣expression␣\"%1\"␣failed\nReason:␣►
                            %2").arg(parser->GetExpr().c_str()).arg(p.GetMsg().c_str()) ;
                        for(int i = 0 ; i < samples.size() ; ++i)
                                samples[i] = QPointF(NAN, NAN) ;
                        delete[] y ;
                        return ;
                }
        }
        for(int i = 0 ; i < samples.size() ; ++i)
                samples[i] = QPointF(x[i], y[i]) ;
        delete[] y ;
}
specFitCurve::specFitCurve()
        : expression(specDescriptor("", spec::editable)),
          parser(0),
          variablesMulti(false),
          expressionMulti(false),
          messagesMulti(false)
{
        setItemAttribute(Legend, true) ;
        setLegendAttribute(LegendNoAttribute, false) ;
        setLegendAttribute(LegendShowLine, true) ;
        setLegendAttribute(LegendShowSymbol, true) ;
}
QStringList specFitCurve::descriptorKeys()
{
        return genericDescriptorKeys() ;
}
QStringList specFitCurve::genericDescriptorKeys()
{
        return QStringList() << QObject::tr("Fit␣variables") <<
               QObject::tr("Fit␣parameters") <<
               QObject::tr("Fit␣expression") <<
               QObject::tr("Fit␣messages") ;
}
void specFitCurve::refreshPlotData()
{
        fitData* fit = dynamic_cast<fitData*>(data()) ;
        if(!fit) return ;
        fit->reevaluate();
}
QString specFitCurve::editDescriptor(const QString& key)
{
        if(key == QObject::tr("Fit␣variables"))
        {
                QStringList results ;
                foreach(variablePair vpair, variables)
                results << vpair.first + "␣=␣" + QString::number(vpair.second) ;
                return results.join("\n") ;
        }
        return descriptor(key, true) ;
}
QString specFitCurve::descriptor(const QString& key, bool full)
{
        if(QObject::tr("Fit␣variables") == key)
        {
                QStringList variableDescriptors ;
                for(int i = 0 ; i < variables.size() ; ++i)
                        variableDescriptors << variables[i].first + "␣=␣" + QString::number(►
                            variables[i].second)
                                        + ((i < numericalErrors.size() && !std::isnan(►
```

```
                                                numericalErrors[i])) ?
112                                            "␣+/-␣" + QString::number(numericalErrors[i])
                                               : QString(""));
                if(full || variablesMulti)
                        return variableDescriptors.join("\n") ;
                if(activeVar >= 0 && activeVar < variableDescriptors.size())
117                     return variableDescriptors[activeVar] ;
                return QString() ;
        }
        if(QObject::tr("Fit␣parameters") == key)
                return fitParameters.join(",␣") ;
122     if(QObject::tr("Fit␣expression") == key)
                return expression.content(full || expressionMulti) ;
        if(QObject::tr("Fit␣messages") == key)
        {
                if(!errorString.isEmpty())
127             {
                        if(full)
                                return errorString ;
                        return errorString.section("\n", 0, 0) ;
                }
132             if(fitData* fit = dynamic_cast<fitData*>(data()))
                {
                        if(full || messagesMulti)
                                return fit->errorString ;
                        return fit->errorString.section("\n", 0, 0) ;
137             }
        }
        return QString() ;
}
bool specFitCurve::changeDescriptor(QString key, QString value)
142 {
        if(QObject::tr("Fit␣variables") == key)
        {
                variables.clear();
                numericalErrors.clear();
147             foreach(QString line, value.split("\n"))
                {
                        line.remove(QRegExp("\\s")) ;
                        double errorValue = NAN ;
                        bool ok = true ;
152                     QStringList l = line.split("+/-") ;
                        line = l.first() ;
                        errorValue = (l.size() > 1 ? l[1].toDouble(&ok) : NAN) ;
                        if(!ok) errorValue = NAN ;
                        if(! line.contains("=")) continue ;
157                     QString var = line.section("=", 0, 0) ;
                        double value = line.section("=", 1, 1).toDouble() ;
                        if(acceptableVariable(var))
                        {
                                numericalErrors << errorValue ;
162                             variables << qMakePair(var, value) ;
                        }
                }
        }
        if(QObject::tr("Fit␣parameters") == key)
167     {
                fitParameters = value.remove(QRegExp("\\s")).split(",") ;
                fitParameters.erase(std::remove_if(fitParameters.begin(), fitParameters.end(), ►
                    inacceptableVariable), fitParameters.end()) ;
                fitParameters.removeDuplicates() ;
        }
172     if(QObject::tr("Fit␣expression") == key)
        {
                expression.setContent(value.replace("**", "^")) ;
        }
        generateParser();
177     setParserConstants();
        refreshPlotData();
        return true ;
}
void specFitCurve::attach(QwtPlot* plot)
182 {
        qDebug() << "Data␣size:" << dataSize() ;
        for(size_t i = 0 ; i < qMin((size_t) 10, dataSize()) ; ++i)
```

```
                qDebug() << sample(i) ;
        specCanvasItem::attach(plot) ;
}
bool specFitCurve::setActiveLine(const QString& key, int n)
{
        if(key == QObject::tr("Fit␣expression"))
                return expression.setActiveLine(n) ;
        if(key == QObject::tr("Fit␣variables"))
                activeVar = n ;
        return false ;
}
int specFitCurve::activeLine(const QString& key) const
{
        if(key == QObject::tr("Fit␣expression"))
                return expression.activeLine() ;
        if(key == QObject::tr("Fit␣variables"))
                return activeVar ;
        return 0 ;
}
bool specFitCurve::acceptableVariable(const QString& s)
{
        QRegExp re("[A-Za-z][A-Za-z0-9]*(/0?.[0-9]+)?") ;
        return re.exactMatch(s) ;
}
void choleskyInversion(double* matrix, double* target, const int& size)
{
        for(int i = 0 ; i < size ; ++i)
        {
                double sumOfSquares = 0 ;
                for(int j = 0 ; j < i ; ++j)
                {
                        double sumOfProducts = 0 ;
                        for(int k = 0 ; k < j ; ++k)
                                sumOfProducts += target[i * size + k] * target[j * size + k] ;
                        double result = (matrix[i * size + j] - sumOfProducts) / target[j * size + j►
                            ] ;
                        target[i * size + j] = result ;
                        sumOfSquares += result * result ;
                }
                target[i * size + i] = sqrt(matrix[i * size + i] - sumOfSquares) ;
        }
        for(int i = 0 ; i < size ; ++i)
        {
                double diag = target[i * size + i] ;
                target[i * size + i] = 1. / diag ;
                for(int j = 0 ; j < i ; ++j)
                {
                        double sumOfProducts = 0 ;
                        for(int k = j ; k < i ; ++k)
                                sumOfProducts += target[i * size + k] * target[k * size + j] ;
                        target[i * size + j] = -sumOfProducts / diag ;
                }
        }
        for(int i = 0 ; i < size ; ++i)
        {
                for(int j = 0 ; j <= i ; ++j)
                {
                        double d = 0 ;
                        for(int k = i ; k < size ; ++k)
                                d += matrix[k * size + i] * matrix[k * size + j] ;
                        matrix[j * size + i] = d ;
                }
        }
}
void specFitCurve::refit(QwtSeriesData<QPointF>* data)
{
        generateParser();
        if(!parser || fitParameters.isEmpty()) return ;
        QVector<std::string> variableNames(fitParameters.size()) ;
        double x[data->size()], y[data->size()], parameters[fitParameters.size()] ;
        for(size_t i = 0 ; i < data->size() ; ++i)
        {
                x[i] = data->sample(i).x() ;
                y[i] = data->sample(i).y() ;
        }
```

```
        foreach(const variablePair & var, variables)
        {
                QString name = plainVariableName(var.first) ;
                if(fitParameters.contains(name))
                {
                        int index = fitParameters.indexOf(name) ;
                        parameters[index] = var.second ;
                        parser->DefineConst(name.toStdString(), parameters[index]);
                        variableNames[index] = name.toStdString() ;
                }
                else
                        parser->DefineConst(name.toStdString(), var.second);
        }
        lm_status_struct status ;
        lm_control_struct control = lm_control_double ;
        lmcurve_data_struct fitParams = { x, y, parser, &variableNames} ;
        double* covarianceMatrix = new double [fitParameters.size() *fitParameters.size()] ;
        double sumOfSquaredResiduals = 0 ;
        lmmin(fitParameters.size(),
              parameters,
              data->size(),
              (const void*) &fitParams,
              evaluateParser,
              &control,
              &status,
              0,
              covarianceMatrix,
              &sumOfSquaredResiduals) ;
        if (status.info > 3)
                errorString = lm_infmsg[status.info] ;
        for(QList<variablePair>::iterator i = variables.begin() ; i != variables.end() ; ++i)
        {
                QString name = plainVariableName(i->first) ;
                if(fitParameters.contains(name))
                        i->second = parameters[fitParameters.indexOf(plainVariableName(name))] ;
        }
        numericalErrors.clear();
        choleskyInversion(covarianceMatrix, covarianceMatrix, fitParameters.size()) ;
        for(int i = 0 ; i < fitParameters.size() ; ++i)
        {
                for(int j = 0 ; j < fitParameters.size() ; ++j)
                        qDebug() << covarianceMatrix[i * fitParameters.size() + j] ;
                qDebug() << ";" ;
        }
        foreach(const variablePair & p, variables)
        {
                QString name = plainVariableName(p.first) ;
                double confidence = confidenceInterval(p.first) ;
                int j = fitParameters.indexOf(name) ;
                int dof = data->size() - fitParameters.size() ;
                numericalErrors << (j > -1 ? sqrt(sumOfSquaredResiduals * covarianceMatrix[j * ►
                    fitParameters.size() + j] / dof) *
                                ((0 < confidence && 1 > confidence) ? gsl_cdf_tdist_Pinv(►
                                    confidence, dof) : 1)
                                    : NAN) ;
        }
        delete [] covarianceMatrix ;
        generateParser();
        setParserConstants();
        setData(new fitData(parser));
}
QVector<double> specFitCurve::getFitData(QwtSeriesData<QPointF>* data)
{
        double* xvals = new double[dataSize()] ;
        for(size_t i = 0 ; i < dataSize() ; ++i)
                xvals[i] = data->sample(i).x() ;
        double* oldX = (double*)(parser->GetVar().at("x")) ;
        parser->DefineVar("x", xvals);
        QVector<double> yValues(dataSize(), NAN) ;
        try
        {
                parser->Eval(yValues.data(), dataSize()) ;
        }
        catch(...)
        {
```

```
        }
        parser->DefineVar("x", oldX) ;
        delete[] xvals ;
        return yValues ;
}
void evaluateParser(const double* parameters, int count, const void* data, double* fitResults, int* ►
    info)
{
        Q_UNUSED(info) ;
        lmcurve_data_struct* fitData = (lmcurve_data_struct*) data ;
        fitData->parser->DefineVar("x", ((lmcurve_data_struct*) data)->x);
        int j = 0 ;
        foreach(const std::string & variableName, * (fitData->variableNames))
        fitData->parser->DefineConst(variableName, parameters[j++]);
        try
        {
                fitData->parser->Eval(fitResults, count) ;
        }
        catch(...)
        {
                for(int i = 0 ; i < count ; ++i)
                        fitResults[i] = NAN ;
                return ;
        }
        for(int i = 0 ; i < count ; ++i)
                fitResults[i] = fitData->y[i] - fitResults[i] ;
}
void specFitCurve::setParserConstants()
{
        if(!parser) return ;
        foreach(const variablePair & var, variables)
        parser->DefineConst(plainVariableName(var.first).toStdString(), var.second) ;
}
void specFitCurve::writeToStream(QDataStream& out) const
{
        specCanvasItem::writeToStream(out);
        out << variables <<
            activeVar <<
            fitParameters <<
            expression <<
            errorString <<
            numericalErrors ;
        if(variablesMulti || expressionMulti || messagesMulti)
                out << qint8(variablesMulti + 2 * expressionMulti + 4 * messagesMulti) ;
}
void specFitCurve::readFromStream(QDataStream& in)
{
        specCanvasItem::readFromStream(in) ;
        in >> variables >>
           activeVar >>
           fitParameters >>
           expression >>
           errorString >>
           numericalErrors ;
        if(in.atEnd())
                expressionMulti = messagesMulti = variablesMulti = false ;
        else
        {
                qint8 a ;
                in >> a ;
                messagesMulti = a & qint8(4) ;
                expressionMulti = a & qint8(2) ;
                variablesMulti = a & qint8(1) ;
        }
        generateParser();
        setParserConstants();
}
void specFitCurve::clearParser()
{
        errorString.clear();
        delete parser ;
        parser = 0 ;
}
void specFitCurve::generateParser()
{
```

```
        clearParser();
        parser = new mu::Parser ;
        try
        {
                parser->SetExpr(expression.content(true).toStdString()) ;
        }
        catch(mu::Parser::exception_type& p)
        {
                errorString = QString("Evaluation␣of␣fit␣expression␣\"%1\"␣failed\nReason:␣%2").arg(►
                    expression.content(true)).arg(p.GetMsg().c_str()) ;
                delete parser ;
                parser = 0 ;
        }
        setData(new fitData(parser));
        QStringList::iterator i = fitParameters.begin() ;
        while(i != fitParameters.end())
        {
                bool isVariable = false ;
                foreach(variablePair variable, variables)
                {
                        if(*i == plainVariableName(variable.first))
                        {
                                isVariable = true ;
                                break ;
                        }
                }
                if(!isVariable) i = fitParameters.erase(i) ;
                else ++i ;
        }
}
specFitCurve::~specFitCurve()
{
        clearParser();
}
void specFitCurve::setDescriptorProperties(QString key, spec::descriptorFlags f)
{
        if(key == QObject::tr("Fit␣variables")) variablesMulti = f & spec::multiline ;
        if(key == QObject::tr("Fit␣expression")) expressionMulti = f & spec::multiline ;
        if(key == QObject::tr("Fit␣messages")) messagesMulti = f & spec::multiline ;
}
spec::descriptorFlags specFitCurve::descriptorProperties(const QString& key) const
{
        spec::descriptorFlags flags = spec::def ;
        if(key != QObject::tr("Fit␣messages")) flags |= spec::editable ;
        if(key == QObject::tr("Fit␣variables") && variablesMulti) flags |= spec::multiline ;
        if(key == QObject::tr("Fit␣expression") && expressionMulti) flags |= spec::multiline ;
        if(key == QObject::tr("Fit␣messages") && messagesMulti) flags |= spec::multiline ;
        return flags ;
}
double specFitCurve::confidenceInterval(const QString& v) const
{
        return v.section("/", 1, 1).toDouble() ;
}
QString specFitCurve::plainVariableName(const QString& v) const
{
        return v.section("/", 0, 0) ;
}
```

src/kinetic/specintegralvariable.h

```
#ifndef SPECINTEGRALVARIABLE_H
#define SPECINTEGRALVARIABLE_H
#include "specmetavariable.h"
class specIntegralVariable : public specMetaVariable
{
protected:
        double processPoints(QVector<QPointF>& points) const ;
};
#endif
```

src/kinetic/specintegralvariable.cpp

```
#include "specintegralvariable.h"
#include "utility-functions.h"
double specIntegralVariable::processPoints(QVector<QPointF>& points) const
{
        double r = 0 ;
        qSort(points.begin(), points.end(), comparePoints) ;
        QPointF previous = points.first() ;
        foreach(const QPointF point, points)
        {
                QPointF diff = point - previous ;
                r += diff.x() * (diff.y() / 2. + previous.y()) ;
                previous = point ;
        }
        return r ;
}
```

src/kinetic/speckineticwidget.h

```
#ifndef SPECKINETICWIDGET_H
#define SPECKINETICWIDGET_H
#include "specdockwidget.h"
#include "specplot.h"
#include "specmetamodel.h"
#include "specmetaview.h"
class specKineticWidget : public specDockWidget, public specStreamable
{
        Q_OBJECT
private:
        specPlot* plot ;
        specMetaView* items ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::metaWidget ;}
private slots:
        void svgModification(bool mod) ;
protected:
        QList<QWidget*> mainWidgets() const ;
public:
        specKineticWidget(QWidget* parent = 0);
        specPlot* internalPlot() { return plot ; }
        specMetaView* view() ;
};
#endif
```

src/kinetic/speckineticwidget.cpp

```
#include "speckineticwidget.h"
#include <QTextStream>
#include "speccanvasitem.h"
#include "specgenericmimeconverter.h"
#include "specmimetextexporter.h"
#include "specsplitter.h"
#include "canvaspicker.h"
#include "specactionlibrary.h"
#include <QApplication>
specMetaView* specKineticWidget::view()
{ return items ; }
specKineticWidget::specKineticWidget(QWidget* parent)
        : specDockWidget(tr("Meta"), parent)
{
        setWhatsThis(tr("Meta␣dock␣widget␣-␣In␣this␣widget,␣further␣processing␣of␣the␣primary␣data␣►
             can␣be␣done␣(integration,␣max,␣min,␣etc.)"));
        plot = new specPlot ;
        items = new specMetaView(this) ;
        plot->setMinimumHeight(100) ;
        items->setModel(new specMetaModel(items)) ;
        new specGenericMimeConverter(items->model());
        new specMimeTextExporter(items->model()) ;
        connect(plot->svgAction(), SIGNAL(toggled(bool)), this, SLOT(svgModification(bool))) ;
        svgModification(false) ;
        setObjectName(tr("Meta␣window")) ;
        toggleViewAction()->setText(tr("Toggle␣meta␣window"));
```

```
        plot->setObjectName("metaPlot");
}
void specKineticWidget::writeToStream(QDataStream& out) const
{
        out << *plot << *items ;
}
void specKineticWidget::readFromStream(QDataStream& in)
{
        in >> *plot >> *items ;
}
void specKineticWidget::svgModification(bool mod)
{
        if(mod) connect(plot->svgPicker(), SIGNAL(pointMoved(specCanvasItem*, int, double, double)),►
             items->model(), SLOT(svgMoved(specCanvasItem*, int, double, double))) ;
        else disconnect(plot->svgPicker(), SIGNAL(pointMoved(specCanvasItem*, int, double, double)),►
             items->model(), SLOT(svgMoved(specCanvasItem*, int, double, double))) ;
        plot->svgPicker()->highlightSelectable(mod) ;
}
QList<QWidget*> specKineticWidget::mainWidgets() const
{
        return QList<QWidget*>() << items << plot ;
}
```

src/kinetic/specmaxposvariable.h

```
#ifndef SPECMAXPOSVARIABLE_H
#define SPECMAXPOSVARIABLE_H
#include "specmetavariable.h"
class specMaxPosVariable : public specMetaVariable
{
protected:
        double processPoints(QVector<QPointF>& points) const ;
};
#endif
```

src/kinetic/specmaxposvariable.cpp

```
#include "specmaxposvariable.h"
double specMaxPosVariable::processPoints(QVector<QPointF>& points) const
{
        double maxX = NAN, maxY = -INFINITY ;
        foreach(const QPointF & point, points)
        {
                if(point.y() > maxY)
                {
                        maxX = point.x() ;
                        maxY = point.y() ;
                }
        }
        return maxX ;
}
```

src/kinetic/specmaxvariable.h

```
#ifndef SPECMAXVARIABLE_H
#define SPECMAXVARIABLE_H
#include "specmetavariable.h"
class specMaxVariable : public specMetaVariable
{
protected:
        double processPoints(QVector<QPointF>& points) const ;
};
#endif
```

src/kinetic/specmaxvariable.cpp

```
#include "specmaxvariable.h"
double specMaxVariable::processPoints(QVector<QPointF>& points) const
```

```
{
        double r = -INFINITY ;
        foreach(const QPointF point, points)
        r = qMax(r, point.y()) ;
        return r ;
}
```

src/kinetic/specmetadelegate.h

```
#ifndef SPECMETADELEGATE_H
#define SPECMETADELEGATE_H
#include "specdelegate.h"
class specMetaDelegate : public specDelegate
{
public:
        specMetaDelegate(QObject* parent = 0) ;
        void paint(QPainter* painter, const QStyleOptionViewItem& option, const QModelIndex& index) ►
            const ;
};
#endif
```

src/kinetic/specmetadelegate.cpp

```
#include "specmetadelegate.h"
#include "specmodel.h"
specMetaDelegate::specMetaDelegate(QObject* parent)
        : specDelegate(parent)
{
}
void specMetaDelegate::paint(QPainter* painter, const QStyleOptionViewItem& option, const ►
    QModelIndex& index) const
{
        QStyleOptionViewItem style = option ;
        if(!(
                    ((specModel*) index.model())->itemPointer(index)
                    ->descriptor("errors").isEmpty())
          )
        {
                QFont font = style.font ;
                font.setBold(true) ;
                style.font = font ;
        }
        specDelegate::paint(painter, style, index) ;
}
```

src/kinetic/specmetaitem.h

```
#ifndef SPECMETAITEM_H
#define SPECMETAITEM_H
#include "specmodelitem.h"
#include "specmetaparser.h"
#include "model/specdescriptor.h"
#include "specgenealogy.h"
class specPlot ;
class specFitCurve ;
class specMetaItem : public specModelItem
{
        friend class metaItemProperties ;
private:
        QList<specModelItem*> items ;
        specMetaParser* filter ;
        QHash<QString, specDescriptor> variables ;
        specModelItem* currentlyConnectingServer ;
        type typeId() const { return specStreamable::metaItem ; }
        void readFromStream(QDataStream& in) ;
        void writeToStream(QDataStream& out) const ;
        specModel* metaModel, *dataModel ;
        QVector<QPair<specGenealogy, qint8> > oldConnections ;
        specFitCurve* fitCurve ;
        bool styleFitCurve ;
```

```
        bool fitCurveDescriptor(const QString&) const ;
        void syncFitCurveName() const ;
public:
        void refreshOtherPlots() ;
        void setModels(specModel* meta, specModel* data) ;
        bool disconnectServer(specModelItem*) ;
        bool connectServer(specModelItem*) ;
        QList<specModelItem*> serverList() const { return items ; }
        bool isServedBy(specModelItem* item) const { return items.contains(item) ;}
        explicit specMetaItem(specFolderItem* par = 0, QString description = "");
        ~specMetaItem() ;
        QList<specModelItem*> purgeConnections() ;
        void attach(QwtPlot* plot) ;
        void detach();
        void refreshPlotData();
        QStringList descriptorKeys() const ;
        static QStringList genericDescriptorKeys() ;
        QString descriptor(const QString& key, bool full = false) const ;
        QString editDescriptor(const QString& key) const ;
        bool changeDescriptor(QString key, QString value) ;
        spec::descriptorFlags descriptorProperties(const QString& key) const ;
        void setDescriptorProperties(const QString& key, spec::descriptorFlags f);
        QIcon decoration() const ;
        void getRangePoint(int variable, int range, int point, double& x, double& y) const ;
        void setRange(int variableNo, int rangeNo, int pointNo, double newX, double newY) ;
        bool setActiveLine(const QString&, int) ;
        int activeLine(const QString& key) const ;
        int rtti() { return spec::metaItem ; }
        specUndoCommand* itemPropertiesAction(QObject* parentObject) ;
        specFitCurve* setFitCurve(specFitCurve*) ;
        specFitCurve* getFitCurve() const ;
        bool hasFitCurve() const ;
        void conductFit() ;
        void toggleFitStyle() ;
        bool getFitStyleState() const ;
        void setLineWidth(const double&) ;
        double lineWidth() const;
        QColor penColor() const;
        void setPenColor(const QColor&) ;
        int symbolStyle() const ;
        void setSymbolStyle(const int&) ;
        QColor symbolPenColor() const ;
        void setSymbolPenColor(const QColor&) ;
        void setSymbolBrushColor(const QColor&) ;
        QColor symbolBrushColor() const ;
        QSize symbolSize() const;
        void setSymbolSize(int w, int h = -1) ;
        void setSymbolSize(const QSize&) ;
        void setPenStyle(const qint8&);
        qint8 penStyle() const ;
        void connectedItems(QModelIndexList& dataItems, QModelIndexList& metaItems) ;
        void exportData(const QList<QPair<bool, QString> >&, const QList<QPair<spec::value, QString>►
             >&, QTextStream&) ;
};
#endif
```

src/kinetic/specmetaitem.cpp

```
#include "specmetaitem.h"
#include "specplot.h"
#include "metaitemproperties.h"
#include "specfitcurve.h"
#include "QApplication"
bool specMetaItem::fitCurveDescriptor(const QString& s) const
{
        return fitCurve && fitCurve->descriptorKeys().contains(s) ;
}
void specMetaItem::syncFitCurveName() const
{
        if(fitCurve)
                fitCurve->setTitle(QObject::tr("") + descriptor("") + QObject::tr("␣(Fit)"));
}
specMetaItem::specMetaItem(specFolderItem* par, QString description)
        : specModelItem(par, description),
```

```
          filter(0),
          currentlyConnectingServer(0),
          metaModel(0),
          dataModel(0),
          fitCurve(0),
          styleFitCurve(false)
{
        filter = new specMetaParser("", "", "", this) ;
        invalidate() ;
        variables["x"] = specDescriptor("", spec::editable) ;
        variables["y"] = specDescriptor("", spec::editable) ;
        variables["variables"] = specDescriptor("", spec::editable) ;
        variables["errors"] = specDescriptor("No␣data") ;
}
specMetaItem::~specMetaItem()
{
        foreach(specModelItem * item, items)
        disconnectServer(item) ;
        delete filter ;
        delete fitCurve ;
}
void specMetaItem::connectedItems(QModelIndexList& dataItems, QModelIndexList& metaItems)
{
        if(!dataModel || !metaModel) return ;
        foreach(specModelItem * item, items)
        {
                QModelIndex index = dataModel->index(item) ;
                if(index.isValid())
                        dataItems << index ;
                else
                {
                        index = metaModel->index(item) ;
                        if(index.isValid())
                                metaItems << index ;
                }
        }
}
void specMetaItem::setModels(specModel* m, specModel* d)
{
        metaModel = m ;
        dataModel = d ;
        for(int i = 0 ; i < oldConnections.size() ; ++i)
        {
                oldConnections[i].first.setModel(oldConnections[i].second ? m : d) ;
                QVector<specModelItem*> oldItems = oldConnections[i].first.items() ;
                foreach(specModelItem * item, oldItems)
                connectServer(item) ;
        }
        oldConnections.clear();
        invalidate();
}
void specMetaItem::writeToStream(QDataStream& out) const
{
        bool stylingWasToFit = styleFitCurve ;
        const_cast<bool&>(styleFitCurve) = false ;
        specModelItem::writeToStream(out) ;
        const_cast<bool&>(styleFitCurve) = stylingWasToFit ;
        out << ((quint8) styleFitCurve) << variables ;
        if(!metaModel || !dataModel) return ;
        QVector<QPair<specGenealogy, qint8> > currentConnections ;
        int i = 0 ;
        while(i < items.size())
        {
                int j = 1 ;
                bool isMeta = metaModel->contains(items[i]) ;
                while(i + j < items.size() &&
                        metaModel->contains(items[i + j]) == isMeta)
                        ++j ;
                QList<specModelItem*> section = items.mid(i, j) ;
                i += j ;
                while(!section.isEmpty())
                        currentConnections << qMakePair<specGenealogy, qint8> (
                                                specGenealogy(section, isMeta ? metaModel : dataModel),
                                                isMeta) ;
        }
```

```
        out << currentConnections ;
        out << (quint8((bool) fitCurve)) ;
        if(fitCurve) out << *fitCurve ;
}
void specMetaItem::readFromStream(QDataStream& in)
{
        delete fitCurve ;
        fitCurve = 0 ;
        styleFitCurve = false ;
        specModelItem::readFromStream(in) ;
        quint8 stylingWasToFit ;
        in >> stylingWasToFit >> variables >> oldConnections;
        styleFitCurve = stylingWasToFit ;
        filter->setAssignments(variables["variables"].content(true), variables["x"].content(true), ►
            variables["y"].content(true)) ;
        quint8 hasFitCurve ;
        in >> hasFitCurve ;
        if(hasFitCurve)
        {
                fitCurve = new specFitCurve ;
                in >> *fitCurve ;
        }
        invalidate() ;
        syncFitCurveName();
}
QList<specModelItem*> specMetaItem::purgeConnections()
{
        QList<specModelItem*> list = items ;
        foreach(specModelItem * item, items)
        item->disconnectClient(this) ;
        items.clear();
        invalidate() ;
        return list ;
}
bool specMetaItem::disconnectServer(specModelItem* server)
{
        if(server == currentlyConnectingServer)
                return true ;
        if(!items.contains(server))
                return false ;
        currentlyConnectingServer = server ;
        if(!server->disconnectClient(this))
        {
                currentlyConnectingServer = 0 ;
                return false ;
        }
        currentlyConnectingServer = 0 ;
        items.removeOne(server) ;
        invalidate() ;
        return true ;
}
bool specMetaItem::connectServer(specModelItem* server)
{
        if(currentlyConnectingServer == server)
                return true ;
        if(items.contains(server))
                return false ;
        if(shortCircuit(server))
                return false ;
        currentlyConnectingServer = server ;
        if(!server->connectClient(this))
        {
                currentlyConnectingServer = 0 ;
                return false ;
        }
        currentlyConnectingServer = 0 ;
        items << server ;
        invalidate() ;
        return true ;
}
void specMetaItem::refreshOtherPlots()
{
        QSet<specPlot*> otherPlots ;
        foreach(specModelItem * item, items)
        otherPlots << ((specPlot*) item->plot()) ;
```

```
        otherPlots.remove(0) ;
        QColor rangeColor = pen().color() ;
        rangeColor.setAlpha(128);
        if(plot())
                filter->attachRanges(otherPlots, rangeColor) ;
        else
                filter->detachRanges();
        foreach(QwtPlot * otherPlot, otherPlots)
        otherPlot->replot();
}
void specMetaItem::attach(QwtPlot* plot)
{
        specModelItem::attach(plot) ;
        if(fitCurve) fitCurve->attach(plot) ;
        refreshOtherPlots() ;
}
void specMetaItem::detach()
{
        if(fitCurve) fitCurve->detach();
        specModelItem::detach() ;
        refreshOtherPlots();
}
void specMetaItem::refreshPlotData()
{
        foreach(specModelItem * item, items)
        item->revalidate();
        setData(filter->evaluate(items.toVector())) ;
        variables["errors"] = filter->warnings() ;
        refreshOtherPlots() ;
}
QStringList specMetaItem::descriptorKeys() const
{
        QStringList keys = genericDescriptorKeys() ;
        if(fitCurve) keys << fitCurve->descriptorKeys() ;
        return keys ;
}
QStringList specMetaItem::genericDescriptorKeys()
{
        QStringList keys ;
        keys << "" << "variables" << "x" << "y" << "errors" ;
        return keys ;
}
QString specMetaItem::descriptor(const QString& key, bool full) const
{
        if(key == "") return specModelItem::descriptor(key, full) ;
        if(fitCurveDescriptor(key))
                return fitCurve->descriptor(key, full) ;
        return variables[key].content(full) ;
}
QString specMetaItem::editDescriptor(const QString& key) const
{
        if(fitCurveDescriptor(key)) return fitCurve->editDescriptor(key) ;
        return specModelItem::editDescriptor(key) ;
}
bool specMetaItem::changeDescriptor(QString key, QString value)
{
        if(key == "")
        {
                bool result = specModelItem::changeDescriptor(key, value) ;
                syncFitCurveName();
                return result ;
        }
        if(fitCurveDescriptor(key))
                return fitCurve->changeDescriptor(key, value) ;
        if(!variables.contains(key)) return false ;
        variables[key] = value ;
        filter->setAssignments(variables["variables"].content(true), variables["x"].content(true), ►
            variables["y"].content(true)) ;
        invalidate();
        return true ;
}
spec::descriptorFlags specMetaItem::descriptorProperties(const QString& key) const
{
        if(key == "") return specModelItem::descriptorProperties(key) ;
        if(variables.contains(key))
```

```
                return variables[key].flags() ;
        if(fitCurveDescriptor(key)) return fitCurve->descriptorProperties(key) ;
        return spec::def ;
}
void specMetaItem::setDescriptorProperties(const QString& key, spec::descriptorFlags f)
{
        if(key == "") specModelItem::setDescriptorProperties(key, f) ;
        if(variables.contains(key)) variables[key].setFlags(f) ;
        if(fitCurveDescriptor(key) && key != "") fitCurve->setDescriptorProperties(key, f) ;
}
QIcon specMetaItem::decoration() const
{
        if(!variables["errors"].content(true).isEmpty())
                return QIcon::fromTheme("dialog-warning") ;
        return QIcon(":/kinetic.png") ;
}
void specMetaItem::getRangePoint(int variable, int range, int point, double& x, double& y) const
{
        filter->getRangePoint(variable, range, point, x, y) ;
}
void specMetaItem::setRange(int variableNo, int rangeNo, int pointNo, double newX, double newY)
{
        filter->setRange(variableNo, rangeNo, pointNo, newX, newY) ;
}
bool specMetaItem::setActiveLine(const QString& s, int i)
{
        if(fitCurveDescriptor(s))
                return fitCurve->setActiveLine(s, i) ;
        if(variables.contains(s))
        {
                variables[s].setActiveLine(i) ;
                return true ;
        }
        bool result = specModelItem::setActiveLine(s, i) ;
        syncFitCurveName();
        return result ;
}
int specMetaItem::activeLine(const QString& key) const
{
        if(fitCurveDescriptor(key)) return fitCurve->activeLine(key) ;
        if(variables.contains(key)) return variables[key].activeLine() ;
        return specModelItem::activeLine(key) ;
}
specUndoCommand* specMetaItem::itemPropertiesAction(QObject* parentObject)
{
        metaItemProperties propertiesDialog(this) ;
        propertiesDialog.exec() ;
        if(propertiesDialog.result() != QDialog::Accepted) return 0 ;
        return propertiesDialog.changedConnections(parentObject) ;
}
specFitCurve* specMetaItem::setFitCurve(specFitCurve* fc)
{
        specFitCurve* old = fitCurve ;
        fitCurve = fc ;
        if(fitCurve && plot()) fitCurve->attach(plot()) ;
        syncFitCurveName();
        return old ;
}
specFitCurve* specMetaItem::getFitCurve() const
{
        return fitCurve ;
}
bool specMetaItem::hasFitCurve() const
{
        return fitCurve ;
}
void specMetaItem::conductFit()
{
        if(!fitCurve) return ;
        revalidate() ;
        fitCurve->refit(data()) ;
}
bool specMetaItem::getFitStyleState() const
{
        return styleFitCurve ;
```

```
}
void specMetaItem::toggleFitStyle()
{
        styleFitCurve = !styleFitCurve ;
}
#define STYLEROUTINGFUNCTION(TYPE,GETNAME,SETNAME) \
        void specMetaItem::SETNAME(const TYPE& arg) { \
                if (styleFitCurve && fitCurve) fitCurve->SETNAME(arg) ; \
                else specModelItem::SETNAME(arg) ; } \
        \
        TYPE specMetaItem::GETNAME() const { \
                if (styleFitCurve && fitCurve) return fitCurve->GETNAME() ; \
                else return specModelItem::GETNAME() ; }
STYLEROUTINGFUNCTION(double, lineWidth, setLineWidth)
STYLEROUTINGFUNCTION(int, symbolStyle, setSymbolStyle)
STYLEROUTINGFUNCTION(QColor, symbolPenColor, setSymbolPenColor)
STYLEROUTINGFUNCTION(QColor, symbolBrushColor, setSymbolBrushColor)
STYLEROUTINGFUNCTION(QSize, symbolSize, setSymbolSize)
STYLEROUTINGFUNCTION(qint8, penStyle, setPenStyle)
void specMetaItem::setPenColor(const QColor& arg)
{
        if(styleFitCurve && fitCurve) fitCurve->setPenColor(arg) ;
        else
        {
                specModelItem::setPenColor(arg) ;
                refreshOtherPlots();
        }
}
QColor specMetaItem::penColor() const
{
        if(styleFitCurve && fitCurve) return fitCurve->penColor() ;
        else return specModelItem::penColor() ;
}
void specMetaItem::setSymbolSize(int w, int h)
{
        if(styleFitCurve && fitCurve)
                fitCurve->setSymbolSize(w, h) ;
        else
                specCanvasItem::setSymbolSize(w, h) ;
}
void specMetaItem::exportData(const QList<QPair<bool, QString> >& headerFormat, const QList<QPair<►
    spec::value, QString> >& dataFormat, QTextStream& out)
{
        revalidate();
        for(int i = 0 ; i < headerFormat.size() ; i++)
                out << (headerFormat[i].first ? headerFormat[i].second : this->descriptor(►
                    headerFormat[i].second)) ;
        out << endl ;
        QVector<double> fitValues ;
        if(fitCurve) fitValues = fitCurve->getFitData(data()) ;
        for(size_t j = 0 ; j < dataSize() ; j++)
        {
                for(int i = 0 ; i < dataFormat.size() ; i++)
                {
                        switch(dataFormat[i].first)
                        {
                                case spec::wavenumber: out << sample(j).x() ; break ;
                                case spec::signal: out << sample(j).y() ; break ;
                                case spec::maxInt:
                                        if(j < (size_t) fitValues.size())
                                                out << fitValues[j] ; break ;
                        }
                        out << dataFormat[i].second ;
                }
        }
        out << endl ;
}
```

src/kinetic/specmetamodel.h

```
#ifndef SPECMETAMODEL_H
#define SPECMETAMODEL_H
#include "specmodel.h"
class specMetaModel : public specModel
```

```
{
6       Q_OBJECT
private:
        type typeId() const { return specStreamable::metaModel ; }
        specModel* dataModel ;
        void setModels(specModelItem*) ;
11      QStringList dataTypes() const ;
public:
        explicit specMetaModel(QObject* parent = 0);
        void setDataModel(specModel*) ;
        specModel* getDataModel() const ;
16      ~specMetaModel() ;
        QList<specFileImportFunction> acceptableImportFunctions() const ;
        QVariant data(const QModelIndex& index, int role) const;
        bool insertItems(QList<specModelItem*> list, QModelIndex parent, int row) ;
};
21 #endif
```

src/kinetic/specmetamodel.cpp

```
#include "specmetamodel.h"
#include "specfolderitem.h"
#include "specmetaitem.h"
4  specMetaModel::specMetaModel(QObject* parent) :
        specModel(parent),
        dataModel(0)
{
        setMetaModel(this) ;
9       setObjectName("metaModel");
}
specMetaModel::~specMetaModel()
{
}
14 void specMetaModel::setDataModel(specModel* m)
{
        dataModel = m ;
        if(m) m->setMetaModel(this) ;
}
19 specModel* specMetaModel::getDataModel() const
{
        return dataModel ;
}
bool specMetaModel::insertItems(QList<specModelItem*> list, QModelIndex parent, int row)
24 {
        foreach(specModelItem * item, list)
        setModels(item) ;
        return specModel::insertItems(list, parent, row) ;
}
29 void specMetaModel::setModels(specModelItem* item)
{
        for(int i = 0 ; i < item->children() ; ++i)
                setModels(((specFolderItem*) item)->child(i)) ;
        specMetaItem* mitem = dynamic_cast<specMetaItem*>(item) ;
34      if(mitem)
                mitem->setModels(this, dataModel) ;
}
QStringList specMetaModel::dataTypes() const
{
39      return QStringList() << "x" << "y" << "fit" ;
}
QList<specFileImportFunction> specMetaModel::acceptableImportFunctions() const
{
        return QList<specFileImportFunction>() ;
44 }
QVariant specMetaModel::data(const QModelIndex& index, int role) const
{
        specModelItem* pointer = itemPointer(index) ;
        if (pointer
49                      && Qt::BackgroundRole == role
                        && index.column() >= 0
                        && index.column() < descriptors().size()
                        && !pointer-> descriptorKeys().contains(descriptors().at(index.column())))
        {
54              return QBrush(Qt::gray) ;
```

```
        }
        return specModel::data(index, role) ;
}
```

src/kinetic/specmetaparser.h

```
#ifndef SPECMETAPARSER_H
#define SPECMETAPARSER_H
#include <QString>
#include <QMap>
#include <muParser.h>
#include "specmodelitem.h"
#include <QVector>
#include "specmetavariable.h"
class specMetaParser
{
        friend class metaItemProperties ;
private:
        bool valid ;
        QStringList errors ;
        QStringList symbols ;
        QVector<double> valueVector ;
        QList<specMetaVariable*> evaluators ;
        mu::Parser prepare(const QString&) ;
        mu::Parser x, y ;
        QString xExp, yExp ;
        specMetaItem* parent ;
        void clear() ;
        bool containsNan(const QVector<double>&) ;
        bool containsNan(const QVector<QVector<double> >& matrix, int column) ;
        bool changingRange ;
        void itemsToQuery(const QVector<specModelItem*>& items, QVector<QVector<specModelItem*> >& ►
            itemsPerPoint) ;
        void getVariableValues(const QVector<specModelItem*>& itemsToUse, QVector<QVector<double> >&►
             variableValues) ;
        void getPoints(const QVector<QVector<double> >& variableValues, QVector<QPointF>& result) ;
public:
        specMetaParser(const QString& expressionList, const QString& xExpression, const QString& ►
            yExpression, specMetaItem* parent);
        QString warnings() const ;
        ~specMetaParser() { clear() ; }
        bool ok() const ;
        void setAssignments(const QString& expressionList, const QString& xExpression, const QString►
            & yExpression) ;
        QwtSeriesData<QPointF>* evaluate(const QVector<specModelItem*>&);
        void attachRanges(QSet<specPlot*>, QColor) ;
        void detachRanges() ;
        specMetaRange::addressObject addressOf(specMetaVariable*) const ;
        void getRangePoint(int variable, int range, int point, double& x, double& y) const ;
        void setRange(int variableNo, int rangeNo, int pointNo, double newX, double newY) ;
};
#endif
```

src/kinetic/specmetaparser.cpp

```
#include "specmetaparser.h"
#include <QStringList>
#include <QRegExp>
#include <qwt_series_data.h>
#include <iostream>
#include "specmetaitem.h"
specMetaParser::specMetaParser(const QString& expressionList, const QString& xExpression, const ►
    QString& yExpression, specMetaItem* par)
        : parent(par),
          changingRange(false)
{
        setAssignments(expressionList, xExpression, yExpression) ;
}
void specMetaParser::clear()
{
        foreach(specMetaVariable * var, evaluators)
        delete var ;
```

```
        evaluators.clear();
        symbols.clear();
        valueVector.clear();
        errors.clear();
}
void specMetaParser::setAssignments(const QString& expressionList, const QString& xExpression, const QString& yExpression)
{
        if(changingRange) return ;
        valid = true ;
        const QRegExp name("[a-zA-Z][a-zA-Z0-9]*") ;
        clear() ;
        const QRegExp acceptable("(\\[[0-9]*(:[0-9]*(:[0-9]*)?)?\\])?"
                                 "(\"[^\"]*\"|"
                                 "((x|y|i|u|l|p|P)"
                                 "("
                                 "(([+\\-]?([0-9]+|[0-9]*.[0-9]+)([eE][+-]?[0-9]+)?)?"
                                 "(:[+\\-]?([0-9]+|[0-9]*.[0-9]+)([eE][+-]?[0-9]+)?)?)|"
                                 "(([+\\-]?([0-9]+|[0-9]*.[0-9]+)([eE][+-]?[0-9]+)?:)?"
                                 "([+\\-]?([0-9]+|[0-9]*.[0-9]+)([eE][+-]?[0-9]+)?)?))"
                                 "))") ;
        const QRegExp assignment("␣*=␣*") ;
        QStringList expressions = expressionList.split("\n") ;
        foreach(QString expression, expressions)
        {
                if(expression.count("=") != 1)
                {
                        errors << QString("May␣contain␣exactly␣one␣\"=\":␣␣%1").arg(expression) ;
                        continue ;
                }
                QString symbol = expression.section(assignment, 0, 0) ;
                QString value = expression.section(assignment, 1, 1) ;
                if(!name.exactMatch(symbol))
                {
                        errors << QString("Not␣an␣acceptable␣variable␣name:␣␣") + symbol ;
                        valid = false ;
                        continue ;
                }
                if(!acceptable.exactMatch(value))
                {
                        errors << QString("Not␣an␣acceptable␣value:␣␣") + value ;
                        valid = false ;
                        continue ;
                }
                symbols << symbol ;
                evaluators << specMetaVariable::factory(value, this) ;
        }
        evaluators.removeAll(0) ;
        valueVector.resize(symbols.size());
        x = prepare(xExpression) ;
        y = prepare(yExpression) ;
        xExp = xExpression ;
        yExp = yExpression ;
}
void specMetaParser::itemsToQuery(const QVector<specModelItem*>& items,
                                  QVector<QVector<specModelItem*> >& itemsPerPoint)
{
        if(evaluators.isEmpty() || items.isEmpty()) return ;
        typedef specMetaVariable::indexLimit trackingIndex ;
        typedef QVector<trackingIndex> trackingIndexes ;
        trackingIndexes indexes ;
        foreach(specMetaVariable * variable, evaluators)
        indexes << variable->indexRange(items.size()) ;
        while(true)
        {
                QVector<specModelItem*> itemsForThisPoint ;
                for(trackingIndexes::iterator i = indexes.begin() ; i != indexes.end() ; ++i)
                {
                        if(i->begin < i->end)
                                itemsForThisPoint << items[i->begin] ;
                        i->begin += i->increment ;
                }
                if(itemsForThisPoint.size() < evaluators.size()) break ;
                itemsPerPoint << itemsForThisPoint ;
        }
```

```
}
void specMetaParser::getVariableValues(const QVector<specModelItem*>& itemsToUse,
                                       QVector<QVector<double> >& variableValues)
{
        if(itemsToUse.size() != evaluators.size()) return ;
        QVector<double> xValues(1, NAN) ;
        for(int i = 0 ; i < evaluators.size() ; ++i)
                evaluators[i]->xValues(itemsToUse[i], xValues) ;
        QVector<QVector<double> > values ;
        for(int i = 0 ; i < evaluators.size() ; ++i)
                values << evaluators[i]->values(itemsToUse[i], xValues) ;
        for(int i = 0 ; i < xValues.size() ; ++i)
        {
                QVector<double> row ;
                for(int j = 0 ; j < evaluators.size() ; ++j)
                        row << values[j][i] ;
                variableValues << row ;
        }
}
void specMetaParser::getPoints(const QVector<QVector<double> >& variableValues,
                               QVector<QPointF>& result)
{
        if(variableValues.isEmpty()) return ;
        if(variableValues.first().isEmpty()) return ;
        for(int i = 0 ; i < variableValues.size() ; ++i)
        {
                if(containsNan(variableValues[i]))
                {
                        result << QPointF(NAN, NAN) ;
                        continue ;
                }
                for(int j = 0 ; j < valueVector.size() ; ++j)
                        valueVector[j] = variableValues[i][j] ;
                try
                {
                        result << QPointF(x.Eval(), y.Eval()) ;
                }
                catch(mu::Parser::exception_type& p)
                {
                        errors << QObject::tr("Evaluation␣of␣muParser-Expression␣failed\nReason:␣") ►
                            + p.GetMsg().c_str() ;
                        valid = false ;
                }
        }
}
QwtSeriesData<QPointF>* specMetaParser::evaluate(const QVector<specModelItem*>& items)
{
        QVector<QVector<specModelItem*> > itemsPerPoint ;
        QVector<QVector<double> > variableValues ;
        QVector<QPointF> data ;
        itemsToQuery(items, itemsPerPoint) ;
        for(QVector<QVector<specModelItem*> >::Iterator i = itemsPerPoint.begin() ; i != ►
            itemsPerPoint.end() ; ++i)
                getVariableValues(*i, variableValues) ;
        getPoints(variableValues, data) ;
        return new QwtPointSeriesData(data) ;
}
QString specMetaParser::warnings() const
{
        return errors.join("\n") ;
}
bool specMetaParser::ok() const
{
        return valid ;
}
mu::Parser specMetaParser::prepare(const QString& val)
{
        mu::Parser retVal ;
        try
        {
                int i = 0 ;
                foreach(QString symbol, symbols)
                retVal.DefineVar(symbol.toStdString(), & (valueVector[i++]));
                retVal.SetExpr(val.toStdString()) ;
        }
```

```
        catch(mu::Parser::exception_type& p)
        {
                errors << QString("Evaluation␣of␣muParser-Expression␣\"%1\"␣failed\nReason:␣%2").arg►
                    (val).arg(p.GetMsg().c_str()) ;
                valid = false ;
        }
        return retVal ;
}
bool specMetaParser::containsNan(const QVector<double>& vector)
{
        foreach(double zahl, vector)
        if(std::isnan(zahl))
                return true ;
        return false ;
}
bool specMetaParser::containsNan(const QVector<QVector<double> >& matrix, int column)
{
        for(int i = 0 ; i < matrix.size() ; ++i)
                if(matrix[i].size() <= column || std::isnan(matrix[i][column]))
                        return true ;
        return false ;
}
void specMetaParser::attachRanges(QSet<specPlot*> plots, QColor color)
{
        foreach(specMetaVariable * evaluator, evaluators)
        evaluator->produceRanges(plots, color) ;
}
void specMetaParser::detachRanges()
{
        foreach(specMetaVariable * evaluator, evaluators)
        evaluator->detachRanges();
}
specMetaRange::addressObject specMetaParser::addressOf(specMetaVariable* v) const
{
        specMetaRange::addressObject a ;
        a.item = parent ;
        a.range = -1 ;
        a.variable = evaluators.indexOf(v) ;
        return a ;
}
void specMetaParser::getRangePoint(int variable, int range, int point, double& x, double& y) const
{
        if(variable < 0 || variable >= evaluators.size()) return ;
        evaluators[variable]->getRangePoint(range, point, x, y) ;
}
void specMetaParser::setRange(int variableNo, int rangeNo, int pointNo, double newX, double newY)
{
        if(variableNo < 0 || variableNo >= evaluators.size()) return ;
        evaluators[variableNo]->setRange(rangeNo, pointNo, newX, newY) ;
        changingRange = true ;
        if(parent)
        {
                QStringList descriptor;
                QStringList::const_iterator j = symbols.begin() ;
                for(QList<specMetaVariable*>::const_iterator i = evaluators.begin() ;
                        i < evaluators.end() && j != symbols.end() ; ++i, ++j)
                        descriptor += *j + "␣=␣" + (*i)->codeValue() ;
                parent->changeDescriptor("variables", descriptor.join("\n")) ;
                parent->setActiveLine("variables", variableNo) ;
        }
        changingRange = false ;
}
```

src/kinetic/specmetarange.h

```
#ifndef SPECMETARANGE_H
#define SPECMETARANGE_H
#include "specrange.h"
#include <QObject>
class specMetaVariable ;
class specMetaItem ;
class specMetaRange : public specRange
{
private:
```

```
        specMetaVariable* parent ;
        type typeId() const { return specStreamable::metaRange ; }
public:
        struct addressObject
        {
                specMetaItem* item;
                int variable, range ;
        };
        specMetaRange(double x1, double x2, double yinit = 0, specMetaVariable* parent = 0) ;
        addressObject address() ;
        void attach(QwtPlot* plot);
        void detach();
        ~specMetaRange() ;
        int rtti() const { return spec::kineticRange ; }
};
#endif
```

src/kinetic/specmetarange.cpp

```
#include "specmetarange.h"
#include "specmetavariable.h"
#include "specplot.h"
#include <QMouseEvent>
#include "canvaspicker.h"
specMetaRange::specMetaRange(double x1, double x2, double yinit, specMetaVariable* par)
        : specRange(x1, x2, yinit),
          parent(par)
{
}
specMetaRange::addressObject specMetaRange::address()
{
        return parent->address(this) ;
}
void specMetaRange::attach(QwtPlot* newPlot)
{
        if(newPlot == plot()) return ;
        specPlot* sp = qobject_cast<specPlot*> (newPlot) ;
        specPlot* oldPlot = qobject_cast<specPlot*> (plot()) ;
        if(oldPlot && oldPlot->metaPicker())
                oldPlot->metaPicker()->removeSelectable(this) ;
        if(sp && sp->metaPicker())
                sp->metaPicker()->addSelectable(this) ;
        specRange::attach(sp) ;
}
void specMetaRange::detach()
{
        specPlot* oldPlot = qobject_cast<specPlot*> (plot()) ;
        if(oldPlot && oldPlot->metaPicker())
                oldPlot->metaPicker()->removeSelectable(this) ;
        specRange::detach() ;
}
specMetaRange::~specMetaRange()
{
        detach() ;
}
```

src/kinetic/specmetavariable.h

```
#ifndef SPECMETAVARIABLE_H
#define SPECMETAVARIABLE_H
#include <QString>
#include "model/specmodelitem.h"
#include <qwt_interval.h>
#include <specmetarange.h>
class specPlot ;
class specMetaParser ;
class CanvasPicker ;
class specMetaVariable : public QwtInterval
{
public:
        struct indexLimit
        {
```

```
                int begin, end, increment ;
        } ;
private:
        indexLimit limits ;
        bool refreshingRanges ;
        static QString extract(QString&, const QRegExp&) ;
        specMetaParser* parent ;
        QVector<specMetaRange*> ranges ;
        QString code ;
        void clearRanges() ;
protected:
        virtual double processPoints(QVector<QPointF>& points) const { Q_UNUSED(points) return 0 ;}
        QString descriptor ;
        bool extractXs(specModelItem* item, QVector<double>& xvals) const ;
public:
        explicit specMetaVariable(specMetaParser* parent = 0);
        virtual ~specMetaVariable() ;
        virtual bool xValues(specModelItem*, QVector<double>&) const;
        virtual QVector<double> values(specModelItem*, const QVector<double>&) const ;
        indexLimit indexRange(int max) const;
        static specMetaVariable* factory(QString, specMetaParser* parent = 0) ;
        void produceRanges(QSet<specPlot*>, QColor) ;
        void detachRanges();
        QString codeValue() const ;
        specMetaRange::addressObject address(specMetaRange*) ;
        void getRangePoint(int range, int point, double& x, double& y) const ;
        void setRange(int rangeNo, int pointNo, double newX, double newY) ;
};
#endif
```

src/kinetic/specmetavariable.cpp

```
#include "specmetavariable.h"
#include "utility-functions.h"
#include "specdescriptorvariable.h"
#include "specminvariable.h"
#include "specmaxvariable.h"
#include "specxvariable.h"
#include "specyvariable.h"
#include "specintegralvariable.h"
#include "specmetarange.h"
#include "specmetaparser.h"
#include "canvaspicker.h"
#include "specminposvariable.h"
#include "specmaxposvariable.h"
#include <specplot.h>
QString specMetaVariable::extract(QString& source, const QRegExp& expression)
{
        int start = expression.indexIn(source) ;
        if(start == -1) return QString() ;
        start = expression.matchedLength() ;
        QString returnValue = source.left(start) ;
        source.remove(0, start) ;
        return returnValue ;
}
specMetaVariable* specMetaVariable::factory(QString init, specMetaParser* par)
{
        const QRegExp rangeExp("(\\[[0-9]*(:([0-9]*)?(:([0-9]*)?)?)?\\])"),
              descriptorExp("(\\\"[^\\\"]*\\\"|" "(x|y|i|u|l|p|P))"),
              number("[+\\-]?([0-9]+|[0-9]*\\.[0-9]+)([eE][+-]?[0-9]+)?");
        QString range = extract(init, rangeExp) ;
        QString descString = extract(init, descriptorExp) ;
        QString xlower = extract(init, number) ;
        extract(init, QRegExp(":")) ;
        QString xupper = extract(init, number) ;
        specMetaVariable* product = 0;
        if(descString[0] == '"')
        {
                product = new specDescriptorVariable ;
                product->descriptor = descString.mid(1, descString.size() - 2) ;
        }
        if(descString[0] == 'i') product = new specIntegralVariable ;
        if(descString[0] == 'u') product = new specMaxVariable ;
        if(descString[0] == 'l') product = new specMinVariable ;
```

```
        if(descString[0] == 'x') product = new specXVariable ;
        if(descString[0] == 'y') product = new specYVariable;
        if(descString[0] == 'p') product = new specMinPosVariable ;
        if(descString[0] == 'P') product = new specMaxPosVariable ;
47      if(!product) return 0 ;
        if(!range.isEmpty())
        {
                QStringList indexes = range.remove("[").remove("]").split(':') ;
                product->limits.begin = qMax(0, indexes[0] == "" ? 0 : indexes[0].toInt()) ;
52              product->limits.end = qMax(product->limits.begin, indexes.size() == 1 ? product->►
                     limits.begin + 1
                                        : (indexes[1] == "" ? int (INFINITY) : indexes[1].toInt►
                                            ())) ;
                product->limits.increment = qBound(1, indexes.size() > 2 && indexes[2] != "" ? ►
                     indexes[2].toInt() : 1, product->limits.end - product->limits.begin) ;
                range = "[" + range + "]" ;
        }
57      if(!xlower.isEmpty() || !xupper.isEmpty())
        {
                product->setMinValue(xlower.isEmpty() ? -INFINITY : xlower.toDouble()) ;
                product->setMaxValue(xupper.isEmpty() ? INFINITY : xupper.toDouble()) ;
                product->setBorderFlags(QwtInterval::IncludeBorders);
62      }
        product->parent = par ;
        product->code = range + descString ;
        return product ;
}
67  specMetaVariable::specMetaVariable(specMetaParser* p)
        :
        refreshingRanges(false),
        parent(p),
        descriptor("")
72  {
        limits.begin = 0 ;
        limits.end = int (INFINITY) ;
        limits.increment = 1 ;
}
77  specMetaVariable::indexLimit specMetaVariable::indexRange(int max) const
{
        indexLimit l ;
        l.begin = qBound(0, limits.begin, max) ;
        l.end = qBound(l.begin, limits.end, max) ;
82      l.increment = qBound(1, limits.increment, l.end - l.begin) ;
        return l ;
}
bool specMetaVariable::xValues(specModelItem* item, QVector<double>& xvals) const
{
87      Q_UNUSED(item)
        Q_UNUSED(xvals)
        return true ;
}
QVector<double> specMetaVariable::values(specModelItem* item, const QVector<double>& xvals) const
92  {
        QVector<QPointF> points ;
        QPointF point ;
        for(size_t i = 0 ; i < item->dataSize() ; ++i)
                if(!isValid() || contains((point = item->sample(i)).x()))
97                      points << point ;
        if(points.empty()) return QVector<double> (xvals.size(), NAN) ;
        return QVector<double> (xvals.size(), processPoints(points)) ;
}
bool specMetaVariable::extractXs(specModelItem* item, QVector<double>& xvals) const
102 {
        item->revalidate() ;
        if(xvals.size() == 1 && std::isnan(xvals[0]))
        {
                double value ;
107             xvals.clear();
                for(size_t i = 0 ; i < item->dataSize() ; ++i)
                        if(contains(value = item->data()->sample(i).x()))
                                xvals << value ;
                return false ;
112     }
        QVector<double> newXVals ;
        foreach(double value, xvals)
```

```
        {
                if(contains(value))
                {
                        for(size_t i = 0 ; i < item->dataSize() ; ++i)
                        {
                                if(item->data()->sample(i).x() == value)
                                {
                                        newXVals << value ;
                                        continue ;
                                }
                        }
                }
        }
        xvals.swap(newXVals) ;
        return true ;
}
void specMetaVariable::clearRanges()
{
        foreach(specRange * range, ranges)
        delete range ;
        ranges.clear();
}
void specMetaVariable::produceRanges(QSet<specPlot*> plots, QColor color)
{
        if(!QwtInterval::isValid()) return ;
        if(plots.size() != ranges.size())
        {
                clearRanges();
                foreach(specPlot * plot, plots)
                ranges << new specMetaRange(minValue(), maxValue(),
                                        (plot->axisScaleDiv(QwtPlot::yLeft).lowerBound() +
                                         plot->axisScaleDiv(QwtPlot::yLeft).upperBound()) / 2., ►
                                             this) ;
        }
        int i = 0 ;
        foreach(specPlot * plot, plots)
        {
                ranges[i]->setPenColor(color) ;
                ranges[i++]->attach((QwtPlot*) plot) ;
        }
}
void specMetaVariable::detachRanges()
{
        foreach(specRange * range, ranges)
        range->detach();
}
QString specMetaVariable::codeValue() const
{
        if(!QwtInterval::isValid()) return code ;
        return code + QString::number(minValue()) + ":" + QString::number(maxValue()) ;
}
specMetaRange::addressObject specMetaVariable::address(specMetaRange* r)
{
        if(!parent)
        {
                specMetaRange::addressObject ao ;
                ao.item = 0 ;
                ao.range = -1 ;
                ao.variable = -1 ;
                return ao ;
        }
        specMetaRange::addressObject a = parent->addressOf(this) ;
        a.range = ranges.indexOf(r) ;
        return a;
}
void specMetaVariable::getRangePoint(int range, int point, double& x, double& y) const
{
        if(range < 0 || range >= ranges.size()) return ;
        if(point != 0 && point != 1) return ;
        x = point == 0 ? minValue() : maxValue() ;
        y = ranges[range]->sample(0).y() ;
}
void specMetaVariable::setRange(int rangeNo, int pointNo, double newX, double newY)
{
        if(pointNo != 0 && pointNo != 1) return ;
```

```
        if(pointNo == 0) setMinValue(newX) ;
        if(pointNo == 1) setMaxValue(newX) ;
        if(!isValid())
192             setInterval(maxValue(), minValue()) ;
        foreach(specMetaRange * range, ranges)
        {
                range->setInterval(minValue(), maxValue());
                range->refreshPlotData();
197     }
        if(rangeNo < 0 || rangeNo >= ranges.size()) return ;
        ranges[rangeNo]->pointMoved(pointNo, newX, newY);
}
specMetaVariable::~specMetaVariable()
202 {
        clearRanges();
}
```

src/kinetic/specmetaview.h

```
 1  #ifndef SPECMETAVIEW_H
    #define SPECMETAVIEW_H
    #include "specview.h"
    class specCanvasItem ;
    class specMetaModel ;
 6  class specMetaView : public specView
    {
        Q_OBJECT
        specView* dataView ;
        type typeId() const { return specStreamable::metaView ; }
11  protected:
        void readFromStream(QDataStream& in) ;
    public:
        explicit specMetaView(QWidget* parent = 0);
        ~specMetaView() ;
16      void setModel(specMetaModel*) ;
        specMetaModel* model() const ;
        void assignDataView(specView*) ;
        specView* getDataView() ;
    public slots:
21      void rangeModified(specCanvasItem*, int, double, double) ;
    };
    #endif
```

src/kinetic/specmetaview.cpp

```
    #include "specmetaview.h"
 2  #include "specmetadelegate.h"
    #include "specmetarange.h"
    #include "specmetarangecommand.h"
    #include "specmetamodel.h"
    #include "specmetaitem.h"
 7  #include "specactionlibrary.h"
    #include <QQueue>
    #include "speccanvasitem.h"
    specMetaView::specMetaView(QWidget* parent) :
        specView(parent),
12      dataView(0)
    {
        QAbstractItemDelegate* olddel = itemDelegate() ;
        setItemDelegate(new specMetaDelegate) ;
        delete olddel ;
17      setWhatsThis(tr("This␣list␣contains␣meta␣entries␣which␣obtain␣their␣data␣from␣data␣items.␣␣►
             Use␣the␣connect␣button␣to␣connect␣meta␣items␣to␣data␣items.␣␣Manage␣items␣by␣dragging␣►
             and␣dropping.␣␣Add␣a␣new␣item␣by␣using␣the␣appropriate␣button.␣␣Connection␣to␣data␣items►
             ␣may␣also␣be␣established␣using␣the␣appropriate␣button␣and␣changed␣by␣right␣clicking␣on␣►
             the␣respective␣meta␣item.\nThe␣first␣column␣is␣for␣your␣comments/descriptions;␣the␣\"►
             variables\"␣column␣lets␣you␣define␣how␣to␣obtain␣data␣from␣data␣items␣(see␣below);␣the␣►
             \"x\"␣and␣\"y\"␣columns␣may␣contain␣a␣formula␣for␣calculating␣the␣x␣and␣y␣values,␣►
             respectively,␣of␣an␣item␣from␣the␣variables␣defined␣in␣the␣\"variables\"␣column.␣␣Any␣►
             errors␣in␣evaluating␣these␣will␣be␣displayed␣in␣the␣\"errors\"␣column.\nIf␣a␣field␣►
             contains␣multiple␣lines,␣an␣indicator␣will␣appear.␣␣The␣full␣contents␣is␣available␣as␣►
             tool␣tip␣text␣(just␣rest␣the␣mous␣cursor␣on␣the␣field).\n\nSyntax␣for␣defining␣variables►
```

```
            :\n␣name␣=␣[from:to:step]\"column\"\nor:\nname␣=␣[from:to:step]ABeginning:End\n␣where␣\"►
            A\"␣may␣be␣\"x\"␣(meaning␣x␣values),␣\"y\"␣(y␣values),\"i\"␣(integral),␣\"u\"␣(max␣y␣►
            value),␣\"l\"␣(min␣y␣value),␣\"p\"␣(position␣of␣min␣y␣value),␣\"P\"␣(position␣of␣max␣y␣►
            value);\n␣\"column\"␣may␣be␣any␣column␣name;␣\"from\",␣\"to\",␣and␣\"step\"␣define␣the␣►
            range␣of␣connected␣data␣items␣to␣use␣amd␣\"Beginning\"␣and␣\"End\"␣define␣the␣x␣range␣of►
            ␣values␣to␣use.\n\n␣When␣using␣a␣fit,␣the␣field␣\"Fit␣variables\"␣is␣to␣contain␣all␣►
            variables␣used␣in␣the␣fitting␣formula,␣one␣line␣each␣with␣starting␣value,␣as␣in␣\n\ta=1\►
            n\tb=3\nThe␣field\"Fit␣parameters\"␣must␣conatain␣a␣list␣of␣comma-separated␣names␣of␣►
            variables␣which␣the␣fit␣algorithm␣is␣allowed␣to␣use␣for␣fitting.␣␣\"Fit␣expression\"␣is␣►
            what␣it␣says."));
        setObjectName("metaView");
}
specMetaView::~specMetaView()
{
}
void specMetaView::setModel(specMetaModel* mod)
{
        specView::setModel((specModel*) mod) ;
}
specMetaModel* specMetaView::model() const
{
        return (specMetaModel*) specView::model() ;
}
void specMetaView::assignDataView(specView* view)
{
        dataView = view ;
        if(model())
                ((specMetaModel*) model())->setDataModel(dataView ? dataView->model() : 0) ;
}
specView* specMetaView::getDataView()
{
        return dataView ;
}
void specMetaView::rangeModified(specCanvasItem* r, int p, double x, double y)
{
        specMetaRange::addressObject address = ((specMetaRange*) r)->address() ;
        specMetaRangeCommand* command = new specMetaRangeCommand ;
        command->setParentObject(model()) ;
        command->setItem(address.item, address.variable, address.range, p, x, y) ;
        command->setText(tr("Modify␣range")) ;
        if(!actionLibrary)
        {
                command->redo();
                delete command ;
        }
        else
                actionLibrary->push(command) ;
}
void specMetaView::readFromStream(QDataStream& in)
{
        specView::readFromStream(in) ;
        QQueue<specModelItem*> itemQueue ;
        itemQueue << model()->itemPointer(QModelIndex()) ;
        while(!itemQueue.isEmpty())
        {
                specModelItem* item = itemQueue.dequeue();
                if(item->isFolder())
                {
                        specFolderItem* folder = (specFolderItem*) item ;
                        for(int i = 0 ; i < item->children() ; ++i)
                                itemQueue.enqueue(folder->child(i)) ;
                }
                specMetaItem* mitem = dynamic_cast<specMetaItem*>(item) ;
                if(mitem)
                        mitem->setModels(model(), dataView->model()) ;
        }
}
```

src/kinetic/specminposvariable.h

```
#ifndef SPECMINPOSVARIABLE_H
#define SPECMINPOSVARIABLE_H
#include "specmetavariable.h"
class specMinPosVariable : public specMetaVariable
```

```
{
protected:
        double processPoints(QVector<QPointF>& points) const ;
};
#endif
```

src/kinetic/specminposvariable.cpp

```
#include "specminposvariable.h"
double specMinPosVariable::processPoints(QVector<QPointF>& points) const
{
        double minX = NAN, minY = INFINITY ;
        foreach(const QPointF & point, points)
        {
                if(point.y() < minY)
                {
                        minX = point.x() ;
                        minY = point.y() ;
                }
        }
        return minX ;
}
```

src/kinetic/specminvariable.h

```
#ifndef SPECMINVARIABLE_H
#define SPECMINVARIABLE_H
#include "specmetavariable.h"
class specMinVariable : public specMetaVariable
{
protected:
        double processPoints(QVector<QPointF>& points) const ;
};
#endif
```

src/kinetic/specminvariable.cpp

```
#include "specminvariable.h"
double specMinVariable::processPoints(QVector<QPointF>& points) const
{
        double r = INFINITY ;
        foreach(const QPointF & point, points)
        r = qMin(r, point.y()) ;
        return r ;
}
```

src/kinetic/specxvariable.h

```
#ifndef SPECXVARIABLE_H
#define SPECXVARIABLE_H
#include "specmetavariable.h"
class specXVariable : public specMetaVariable
{
public:
        QVector<double> values(specModelItem*, const QVector<double>&) const ;
        bool xValues(specModelItem*, QVector<double>&) const ;
};
#endif
```

src/kinetic/specxvariable.cpp

```
#include "specxvariable.h"
QVector<double> specXVariable::values(specModelItem* item, const QVector<double>& xvals) const
{
        Q_UNUSED(item) ;
        return xvals ;
```

```
}
bool specXVariable::xValues(specModelItem* item, QVector<double>& xvals) const
{
        return extractXs(item, xvals) ;
}
```

src/kinetic/specyvariable.h

```
#ifndef SPECYVARIABLE_H
#define SPECYVARIABLE_H
#include "specmetavariable.h"
class specYVariable : public specMetaVariable
{
public:
        bool xValues(specModelItem*, QVector<double>&) const ;
        QVector<double> values(specModelItem*, const QVector<double>&) const ;
};
#endif
```

src/kinetic/specyvariable.cpp

```
#include "specyvariable.h"
QVector<double> specYVariable::values(specModelItem* item, const QVector<double>& xvals) const
{
        QVector<QPointF> points ;
        QPointF point ;
        for(size_t i = 0 ; i < item->dataSize() ; ++i)
                if(xvals.contains((point = item->sample(i)).x()))
                        points << point ;
        if(points.empty()) return QVector<double> (xvals.size(), NAN) ;
        QVector<double> retVal ;
        foreach(QPointF point, points)
        retVal << point.y() ;
        return retVal ;
}
bool specYVariable::xValues(specModelItem* item, QVector<double>& xvals) const
{
        return extractXs(item, xvals) ;
}
```

src/lmfit/lmmin.h

```
/*
 * Project:  LevenbergMarquardtLeastSquaresFitting
 *
 * File:     lmmin.h
 *
 * Contents: Public interface to the Levenberg-Marquardt core implementation.
 *
 * Author:   Joachim Wuttke 2004-2010
 *
 * Homepage: joachimwuttke.de/lmfit
 */

#ifndef LMMIN_H
#define LMMIN_H

#ifdef __cplusplus
extern "C" {
#endif


/** Compact high-level interface. **/

/* Collection of control (input) parameters. */
typedef struct
{
        double ftol;      /* relative error desired in the sum of squares. */
        double xtol;      /* relative error between last two approximations. */
        double gtol;      /* orthogonality desired between fvec and its derivs. */
```

```
        double epsilon;   /* step used to calculate the jacobian. */
        double stepbound; /* initial bound to steps in the outer loop. */
        int maxcall;      /* maximum number of iterations. */
        int scale_diag;   /* UNDOCUMENTED, TESTWISE automatical diag rescaling? */
        int printflags;   /* OR'ed to produce more noise */
} lm_control_struct;

/* Collection of status (output) parameters. */
typedef struct
{
        double fnorm;     /* norm of the residue vector fvec. */
        int nfev;         /* actual number of iterations. */
        int info;         /* status (index for lm_infmsg and lm_shortmsg). */
} lm_status_struct;

/* Recommended control parameter settings. */
extern const lm_control_struct lm_control_double;
extern const lm_control_struct lm_control_float;

/* Standard monitoring routine. */
void lm_printout_std(int n_par, const double* par, int m_dat,
                     const void* data, const double* fvec,
                     int printflags, int iflag, int iter, int nfev);

/* Refined calculation of Eucledian norm, typically used in printout routine. */
double lm_enorm(int, const double*);

/* The actual minimization. */
void lmmin(int n_par, double* par, int m_dat, const void* data,
           void (*evaluate)(const double* par, int m_dat, const void* data,
                            double* fvec, int* info),
           const lm_control_struct* control, lm_status_struct* status,
           void (*printout)(int n_par, const double* par, int m_dat,
                            const void* data, const double* fvec,
                            int printflags, int iflag, int iter, int nfev) , double* ►
                                covarianceMatrix, double* sumOfSquaredResiduals);


/** Legacy low-level interface. **/

/* Alternative to lm_minimize, allowing full control, and read-out
   of auxiliary arrays. For usage, see implementation of lmmin. */
void lm_lmdif(int m, int n, double* x, double* fvec, double ftol,
              double xtol, double gtol, int maxfev, double epsfcn,
              double* diag, int mode, double factor, int* info, int* nfev,
              double* fjac, int* ipvt, double* qtf, double* wa1,
              double* wa2, double* wa3, double* wa4,
              void (*evaluate)(const double* par, int m_dat, const void* data,
                               double* fvec, int* info),
              void (*printout)(int n_par, const double* par, int m_dat,
                               const void* data, const double* fvec,
                               int printflags, int iflag, int iter, int nfev),
              int printflags, const void* data , double* covarianceMatrix, double* ►
                  sumOfSquaredResiduals);

extern const char* lm_infmsg[];
extern const char* lm_shortmsg[];


#ifdef __cplusplus
}
#endif

#endif /* LMMIN_H */
```

src/lmfit/lmmin.c

```
/*
 * Project:  LevenbergMarquardtLeastSquaresFitting
 *
 * File:     lmmin.c
 *
 * Contents: Levenberg-Marquardt core implementation,
 *           and simplified user interface.
```

```
 *
 * Author:   Joachim Wuttke <j.wuttke@fz-juelich.de>
 *
 * Homepage: joachimwuttke.de/lmfit
 */

/*
 * lmfit is released under the LMFIT-BEER-WARE licence:
 *
 * In writing this software, I borrowed heavily from the public domain,
 * especially from work by Burton S. Garbow, Kenneth E. Hillstrom,
 * Jorge J. Moré, Steve Moshier, and the authors of lapack. To avoid
 * unneccessary complications, I put my additions and amendments also
 * into the public domain. Please retain this notice. Otherwise feel
 * free to do whatever you want with this stuff. If we meet some day,
 * and you think this work is worth it, you can buy me a beer in return.
 */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <float.h>
#include "lmmin.h"


/*****************************************************************************/
/*  set numeric constants                                                    */
/*****************************************************************************/

/* machine-dependent constants from float.h */
#define LM_MACHEP     DBL_EPSILON   /* resolution of arithmetic */
#define LM_DWARF      DBL_MIN       /* smallest nonzero number */
#define LM_SQRT_DWARF sqrt(DBL_MIN) /* square should not underflow */
#define LM_SQRT_GIANT sqrt(DBL_MAX) /* square should not overflow */
#define LM_USERTOL    30*LM_MACHEP  /* users are recommended to require this */

/* If the above values do not work, the following seem good for an x86:
 LM_MACHEP     .555e-16
 LM_DWARF      9.9e-324
 LM_SQRT_DWARF 1.e-160
 LM_SQRT_GIANT 1.e150
 LM_USER_TOL   1.e-14
   The following values should work on any machine:
 LM_MACHEP     1.2e-16
 LM_DWARF      1.0e-38
 LM_SQRT_DWARF 3.834e-20
 LM_SQRT_GIANT 1.304e19
 LM_USER_TOL   1.e-14
*/

const lm_control_struct lm_control_double =
{
        LM_USERTOL, LM_USERTOL, LM_USERTOL, LM_USERTOL, 100., 100, 1, 0
};
const lm_control_struct lm_control_float =
{
        1.e-7, 1.e-7, 1.e-7, 1.e-7, 100., 100, 0, 0
};


/*****************************************************************************/
/*  set message texts (indexed by status.info)                               */
/*****************************************************************************/

const char* lm_infmsg[] =
{
        "success (sum of squares below underflow limit)",
        "success (the relative error in the sum of squares is at most tol)",
        "success (the relative error between x and the solution is at most tol)",
        "success (both errors are at most tol)",
        "trapped by degeneracy (fvec is orthogonal to the columns of the jacobian)",
        "timeout (number of calls to fcn has reached maxcall*(n+1))",
        "failure (ftol<tol: cannot reduce sum of squares any further)",
        "failure (xtol<tol: cannot improve approximate solution any further)",
        "failure (gtol<tol: cannot improve approximate solution any further)",
```

```
        "exception␣(not␣enough␣memory)",
        "fatal␣coding␣error␣(improper␣input␣parameters)",
        "exception␣(break␣requested␣within␣function␣evaluation)"
};

const char* lm_shortmsg[] =
{
        "success␣(0)",
        "success␣(f)",
        "success␣(p)",
        "success␣(f,p)",
        "degenerate",
        "call␣limit",
        "failed␣(f)",
        "failed␣(p)",
        "failed␣(o)",
        "no␣memory",
        "invalid␣input",
        "user␣break"
};


/*****************************************************************************/
/*  lm_printout_std (default monitoring routine)                              */
/*****************************************************************************/

void lm_printout_std(int n_par, const double* par, int m_dat,
                     const void* data, const double* fvec,
                     int printflags, int iflag, int iter, int nfev)
/*
 *       data  : for soft control of printout behaviour, add control
 *               variables to the data struct
 *       iflag : 0 (init) 1 (outer loop) 2(inner loop) -1(terminated)
 *       iter  : outer loop counter
 *       nfev  : number of calls to *evaluate
 */
{
        int i;

        if(!printflags)
                return;

        if(printflags & 1)
        {
                /* location of printout call within lmdif */
                if(iflag == 2)
                {
                        printf("trying␣step␣in␣gradient␣direction␣␣");
                }
                else if(iflag == 1)
                {
                        printf("determining␣gradient␣(iteration␣%2d)", iter);
                }
                else if(iflag == 0)
                {
                        printf("starting␣minimization␣␣␣␣␣␣␣␣␣␣␣␣␣");
                }
                else if(iflag == -1)
                {
                        printf("terminated␣after␣%3d␣evaluations␣␣␣", nfev);
                }
        }

        if(printflags & 2)
        {
                printf("␣␣par:␣");
                for(i = 0; i < n_par; ++i)
                        printf("␣%18.11g", par[i]);
                printf("␣=>␣norm:␣%18.11g", lm_enorm(m_dat, fvec));
        }

        if(printflags & 3)
                printf("\n");

        if((printflags & 8) || ((printflags & 4) && iflag == -1))
```

```
        {
                printf("␣␣residuals:\n");
                for(i = 0; i < m_dat; ++i)
                        printf("␣␣␣␣fvec[%2d]=%12g\n", i, fvec[i]);
        }
}

/*************************************************************************/
/*  lm_minimize (intermediate-level interface)                           */
/*************************************************************************/

void lmmin(int n_par, double* par, int m_dat, const void* data,
           void (*evaluate)(const double* par, int m_dat, const void* data,
                            double* fvec, int* info),
           const lm_control_struct* control, lm_status_struct* status,
           void (*printout)(int n_par, const double* par, int m_dat,
                            const void* data, const double* fvec,
                            int printflags, int iflag, int iter, int nfev),
           double* covarianceMatrix,
           double* sumOfSquaredResiduals)
{

        /*** allocate work space. ***/

        double* fvec, *diag, *fjac, *qtf, *wa1, *wa2, *wa3, *wa4;
        int* ipvt, j;

        int n = n_par;
        int m = m_dat;

        if((fvec = (double*) malloc(m * sizeof(double))) == NULL ||
                (diag = (double*) malloc(n * sizeof(double))) == NULL ||
                (qtf  = (double*) malloc(n * sizeof(double))) == NULL ||
                (fjac = (double*) malloc(n * m * sizeof(double))) == NULL ||
                (wa1  = (double*) malloc(n * sizeof(double))) == NULL ||
                (wa2  = (double*) malloc(n * sizeof(double))) == NULL ||
                (wa3  = (double*) malloc(n * sizeof(double))) == NULL ||
                (wa4  = (double*) malloc(m * sizeof(double))) == NULL ||
                (ipvt = (int*)    malloc(n * sizeof(int))) == NULL)
        {
                status->info = 9;
                return;
        }

        if(! control->scale_diag)
                for(j = 0; j < n_par; ++j)
                        diag[j] = 1;

        /*** perform fit. ***/

        status->info = 0;

        /* this goes through the modified legacy interface: */
        lm_lmdif(m, n, par, fvec, control->ftol, control->xtol, control->gtol,
                 control->maxcall * (n + 1), control->epsilon, diag,
                 (control->scale_diag ? 1 : 2),
                 control->stepbound, & (status->info),
                 & (status->nfev), fjac, ipvt, qtf, wa1, wa2, wa3, wa4,
                 evaluate, printout, control->printflags, data,
                 covarianceMatrix, sumOfSquaredResiduals);

        if(printout)
                (*printout)(n, par, m, data, fvec,
                            control->printflags, -1, 0, status->nfev);
        status->fnorm = lm_enorm(m, fvec);
        if(status->info < 0)
                status->info = 11;

        /*** clean up. ***/

        free(fvec);
        free(diag);
        free(qtf);
        free(fjac);
```

```
        free(wa1);
        free(wa2);
        free(wa3);
        free(wa4);
        free(ipvt);
} /*** lmmin. ***/


/*****************************************************************************/
/*  lm_lmdif (low-level, modified legacy interface for full control)         */
/*****************************************************************************/

void lm_lmpar(int n, double* r, int ldr, int* ipvt, double* diag,
              double* qtb, double delta, double* par, double* x,
              double* sdiag, double* aux, double* xdi);
void lm_qrfac(int m, int n, double* a, int pivot, int* ipvt,
              double* rdiag, double* acnorm, double* wa);
void lm_qrsolv(int n, double* r, int ldr, int* ipvt, double* diag,
               double* qtb, double* x, double* sdiag, double* wa);

#define MIN(a,b) (((a)<=(b)) ? (a) : (b))
#define MAX(a,b) (((a)>=(b)) ? (a) : (b))
#define SQR(x)   (x)*(x)


void lm_lmdif(int m, int n, double* x, double* fvec, double ftol,
              double xtol, double gtol, int maxfev, double epsfcn,
              double* diag, int mode, double factor, int* info, int* nfev,
              double* fjac, int* ipvt, double* qtf, double* wa1,
              double* wa2, double* wa3, double* wa4,
              void (*evaluate)(const double* par, int m_dat, const void* data,
                               double* fvec, int* info),
              void (*printout)(int n_par, const double* par, int m_dat,
                               const void* data, const double* fvec,
                               int printflags, int iflag, int iter, int nfev),
              int printflags, const void* data,
              double* covarianceMatrix, double* sumOfSquaredResiduals)
{
        /*
         *   The purpose of lmdif is to minimize the sum of the squares of
         *   m nonlinear functions in n variables by a modification of
         *   the levenberg-marquardt algorithm. The user must provide a
         *   subroutine evaluate which calculates the functions. The jacobian
         *   is then calculated by a forward-difference approximation.
         *
         *   The multi-parameter interface lm_lmdif is for users who want
         *   full control and flexibility. Most users will be better off using
         *   the simpler interface lmmin provided above.
         *
         *   Parameters:
         *
         *      m is a positive integer input variable set to the number
         *        of functions.
         *
         *      n is a positive integer input variable set to the number
         *        of variables; n must not exceed m.
         *
         *      x is an array of length n. On input x must contain an initial
         *        estimate of the solution vector. On OUTPUT x contains the
         *        final estimate of the solution vector.
         *
         *      fvec is an OUTPUT array of length m which contains
         *        the functions evaluated at the output x.
         *
         *      ftol is a nonnegative input variable. Termination occurs when
         *        both the actual and predicted relative reductions in the sum
         *        of squares are at most ftol. Therefore, ftol measures the
         *        relative error desired in the sum of squares.
         *
         *      xtol is a nonnegative input variable. Termination occurs when
         *        the relative error between two consecutive iterates is at
         *        most xtol. Therefore, xtol measures the relative error desired
         *        in the approximate solution.
         *
         *      gtol is a nonnegative input variable. Termination occurs when
```

```
*         the cosine of the angle between fvec and any column of the
*         jacobian is at most gtol in absolute value. Therefore, gtol
*         measures the orthogonality desired between the function vector
*         and the columns of the jacobian.
*
*       maxfev is a positive integer input variable. Termination
*         occurs when the number of calls to lm_fcn is at least
*         maxfev by the end of an iteration.
*
*       epsfcn is an input variable used in choosing a step length for
*         the forward-difference approximation. The relative errors in
*         the functions are assumed to be of the order of epsfcn.
*
*       diag is an array of length n. If mode = 1 (see below), diag is
*         internally set. If mode = 2, diag must contain positive entries
*         that serve as multiplicative scale factors for the variables.
*
*       mode is an integer input variable. If mode = 1, the
*         variables will be scaled internally. If mode = 2,
*         the scaling is specified by the input diag.
*
*       factor is a positive input variable used in determining the
*         initial step bound. This bound is set to the product of
*         factor and the euclidean norm of diag*x if nonzero, or else
*         to factor itself. In most cases factor should lie in the
*         interval (0.1,100.0). Generally, the value 100.0 is recommended.
*
*       info is an integer OUTPUT variable that indicates the termination
*         status of lm_lmdif as follows:
*
*         info < 0  termination requested by user-supplied routine *evaluate;
*
*         info = 0  fnorm almost vanishing;
*
*         info = 1  both actual and predicted relative reductions
*                   in the sum of squares are at most ftol;
*
*         info = 2  relative error between two consecutive iterates
*                   is at most xtol;
*
*         info = 3  conditions for info = 1 and info = 2 both hold;
*
*         info = 4  the cosine of the angle between fvec and any
*                   column of the jacobian is at most gtol in
*                   absolute value;
*
*         info = 5  number of calls to lm_fcn has reached or
*                   exceeded maxfev;
*
*         info = 6  ftol is too small: no further reduction in
*                   the sum of squares is possible;
*
*         info = 7  xtol is too small: no further improvement in
*                   the approximate solution x is possible;
*
*         info = 8  gtol is too small: fvec is orthogonal to the
*                   columns of the jacobian to machine precision;
*
*         info =10  improper input parameters;
*
*       nfev is an OUTPUT variable set to the number of calls to the
*         user-supplied routine *evaluate.
*
*       fjac is an OUTPUT m by n array. The upper n by n submatrix
*         of fjac contains an upper triangular matrix r with
*         diagonal elements of nonincreasing magnitude such that
*
*               pT*(jacT*jac)*p = rT*r
*
*               (NOTE: T stands for matrix transposition),
*
*         where p is a permutation matrix and jac is the final
*         calculated jacobian. Column j of p is column ipvt(j)
*         (see below) of the identity matrix. The lower trapezoidal
*         part of fjac contains information generated during
```

```
         *        the computation of r.
         *
         *      ipvt is an integer OUTPUT array of length n. It defines a
         *        permutation matrix p such that jac*p = q*r, where jac is
         *        the final calculated jacobian, q is orthogonal (not stored),
         *        and r is upper triangular with diagonal elements of
         *        nonincreasing magnitude. Column j of p is column ipvt(j)
         *        of the identity matrix.
         *
         *      qtf is an OUTPUT array of length n which contains
         *        the first n elements of the vector (q transpose)*fvec.
         *
         *      wa1, wa2, and wa3 are work arrays of length n.
         *
         *      wa4 is a work array of length m, used among others to hold
         *        residuals from evaluate.
         *
         *      evaluate points to the subroutine which calculates the
         *        m nonlinear functions. Implementations should be written as follows:
         *
         *        void evaluate( double* par, int m_dat, void *data,
         *                       double* fvec, int *info )
         *        {
         *           // for ( i=0; i<m_dat; ++i )
         *           //     calculate fvec[i] for given parameters par;
         *           // to stop the minimization,
         *           //     set *info to a negative integer.
         *        }
         *
         *      printout points to the subroutine which informs about fit progress.
         *        Call with printout=0 if no printout is desired.
         *        Call with printout=lm_printout_std to use the default implementation.
         *
         *      printflags is passed to printout.
         *
         *      data is an input pointer to an arbitrary structure that is passed to
         *        evaluate. Typically, it contains experimental data to be fitted.
         *
         */
        int i, iter, j;
        double actred, delta, dirder, eps, fnorm, fnorm1, gnorm, par, pnorm,
               prered, ratio, step, sum, temp, temp1, temp2, temp3, xnorm;
        static double p1 = 0.1;
        static double p0001 = 1.0e-4;

        *nfev = 0;                  /* function evaluation counter */
        iter = 0;                   /* outer loop counter */
        par = 0;                    /* levenberg-marquardt parameter */
        delta = 0;   /* to prevent a warning (initialization within if-clause) */
        xnorm = 0;   /* ditto */
        temp = MAX(epsfcn, LM_MACHEP);
        eps = sqrt(temp);  /* for calculating the Jacobian by forward differences */

        /*** lmdif: check input parameters for errors. ***/

        if((n <= 0) || (m < n) || (ftol < 0.)
                || (xtol < 0.) || (gtol < 0.) || (maxfev <= 0) || (factor <= 0.))
        {
                *info = 10;             // invalid parameter
                return;
        }
        if(mode == 2)               /* scaling by diag[] */
        {
                for(j = 0; j < n; j++)          /* check for nonpositive elements */
                {
                        if(diag[j] <= 0.0)
                        {
                                *info = 10;     // invalid parameter
                                return;
                        }
                }
        }
#ifdef LMFIT_DEBUG_MESSAGES
        printf("lmdif\n");
#endif
```

```
        /*** lmdif: evaluate function at starting point and calculate norm. ***/

        *info = 0;
        (*evaluate)(x, m, data, fvec, info);
        ++ (*nfev);
        if(printout)
                (*printout)(n, x, m, data, fvec, printflags, 0, 0, *nfev);
        if(*info < 0)
                return;
        fnorm = lm_enorm(m, fvec);
        if(fnorm <= LM_DWARF)
        {
                *info = 0;
                return;
        }

        /*** lmdif: the outer loop. ***/

        do
        {
#ifdef LMFIT_DEBUG_MESSAGES
                printf("lmdif␣outer␣loop␣iter=%d␣nfev=%d␣fnorm=%.10e\n",
                       iter, *nfev, fnorm);
#endif

                /*** outer: calculate the Jacobian. ***/

                for(j = 0; j < n; j++)
                {
                        temp = x[j];
                        step = MAX(eps * eps, eps * fabs(temp));
                        x[j] = temp + step; /* replace temporarily */
                        *info = 0;
                        (*evaluate)(x, m, data, wa4, info);
                        ++ (*nfev);
                        if(printout)
                                (*printout)(n, x, m, data, wa4, printflags, 1, iter, *nfev);
                        if(*info < 0)
                                return; /* user requested break */
                        for(i = 0; i < m; i++)
                                fjac[j * m + i] = (wa4[i] - fvec[i]) / step;
                        x[j] = temp; /* restore */
                }
#ifdef LMFIT_DEBUG_MATRIX
                /* print the entire matrix */
                for(i = 0; i < m; i++)
                {
                        for(j = 0; j < n; j++)
                                printf("%.5e␣", fjac[j * m + i]);
                        printf("\n");
                }
#endif

                if(covarianceMatrix)
                {
                        double dummy ;
                        for(i = 0 ; i < n ; ++i)
                        {
                                for(j = 0 ; j <= i ; ++j)
                                {
                                        dummy = 0 ;
                                        int k ;
                                        for(k = 0 ; k < m ; ++k)
                                                dummy += fjac[i * m + k] * fjac[j * m + k] ;
                                        covarianceMatrix[j * n + i] = dummy ;
                                        covarianceMatrix[i * n + j] = dummy ;
                                }
                        }
                }
                if(sumOfSquaredResiduals)
                {
                        *sumOfSquaredResiduals = 0 ;
                        for(i = 0 ; i < m ; ++i)
                                *sumOfSquaredResiduals += SQR(wa4[i]) ;
```

```
        }

535     /*** outer: compute the qr factorization of the Jacobian. ***/

        lm_qrfac(m, n, fjac, 1, ipvt, wa1, wa2, wa3);
        /* return values are ipvt, wa1=rdiag, wa2=acnorm */

540     if(!iter)
        {
                /* first iteration only */
                if(mode != 2)
                {
545                     /* diag := norms of the columns of the initial Jacobian */
                        for(j = 0; j < n; j++)
                        {
                                diag[j] = wa2[j];
                                if(wa2[j] == 0.)
550                                     diag[j] = 1.;
                        }
                }
                /* use diag to scale x, then calculate the norm */
                for(j = 0; j < n; j++)
555                     wa3[j] = diag[j] * x[j];
                xnorm = lm_enorm(n, wa3);
                /* initialize the step bound delta. */
                delta = factor * xnorm;
                if(delta == 0.)
560                     delta = factor;
        }
        else
        {
                if(mode != 2)
565             {
                        for(j = 0; j < n; j++)
                                diag[j] = MAX(diag[j], wa2[j]);
                }
        }
570
        /*** outer: form (q transpose)*fvec and store first n components in qtf. ***/

        for(i = 0; i < m; i++)
                wa4[i] = fvec[i];
575
        for(j = 0; j < n; j++)
        {
                temp3 = fjac[j * m + j];
                if(temp3 != 0.)
580             {
                        sum = 0;
                        for(i = j; i < m; i++)
                                sum += fjac[j * m + i] * wa4[i];
                        temp = -sum / temp3;
585                     for(i = j; i < m; i++)
                                wa4[i] += fjac[j * m + i] * temp;
                }
                fjac[j * m + j] = wa1[j];
                qtf[j] = wa4[j];
590     }

        /*** outer: compute norm of scaled gradient and test for convergence. ***/

        gnorm = 0;
595     for(j = 0; j < n; j++)
        {
                if(wa2[ipvt[j]] == 0)
                        continue;
                sum = 0.;
600             for(i = 0; i <= j; i++)
                        sum += fjac[j * m + i] * qtf[i];
                gnorm = MAX(gnorm, fabs(sum / wa2[ipvt[j]] / fnorm));
        }

605     if(gnorm <= gtol)
        {
                *info = 4;
```

```
                        return;
                }

                /*** the inner loop. ***/
                do
                {
#ifdef LMFIT_DEBUG_MESSAGES
                        printf("lmdif/␣inner␣loop␣iter=%d␣nfev=%d\n", iter, *nfev);
#endif

                        /*** inner: determine the levenberg-marquardt parameter. ***/

                        lm_lmpar(n, fjac, m, ipvt, diag, qtf, delta, &par,
                                 wa1, wa2, wa4, wa3);
                        /* used return values are fjac (partly), par, wa1=x, wa3=diag*x */

                        for(j = 0; j < n; j++)
                                wa2[j] = x[j] - wa1[j]; /* new parameter vector ? */

                        pnorm = lm_enorm(n, wa3);

                        /* at first call, adjust the initial step bound. */

                        if(*nfev <= 1 + n)
                                delta = MIN(delta, pnorm);

                        /*** inner: evaluate the function at x + p and calculate its norm. ***/

                        *info = 0;
                        (*evaluate)(wa2, m, data, wa4, info);
                        ++ (*nfev);
                        if(printout)
                                (*printout)(n, wa2, m, data, wa4, printflags, 2, iter, *nfev);
                        if(*info < 0)
                                return; /* user requested break. */

                        fnorm1 = lm_enorm(m, wa4);
#ifdef LMFIT_DEBUG_MESSAGES
                        printf("lmdif/␣pnorm␣%.10e␣␣fnorm1␣%.10e␣␣fnorm␣%.10e"
                               "␣delta=%.10e␣par=%.10e\n",
                               pnorm, fnorm1, fnorm, delta, par);
#endif

                        /*** inner: compute the scaled actual reduction. ***/

                        if(p1 * fnorm1 < fnorm)
                                actred = 1 - SQR(fnorm1 / fnorm);
                        else
                                actred = -1;

                        /*** inner: compute the scaled predicted reduction and
                             the scaled directional derivative. ***/

                        for(j = 0; j < n; j++)
                        {
                                wa3[j] = 0;
                                for(i = 0; i <= j; i++)
                                        wa3[i] -= fjac[j * m + i] * wa1[ipvt[j]];
                        }
                        temp1 = lm_enorm(n, wa3) / fnorm;
                        temp2 = sqrt(par) * pnorm / fnorm;
                        prered = SQR(temp1) + 2 * SQR(temp2);
                        dirder = - (SQR(temp1) + SQR(temp2));

                        /*** inner: compute the ratio of the actual to the predicted reduction. ***/

                        ratio = prered != 0 ? actred / prered : 0;
#ifdef LMFIT_DEBUG_MESSAGES
                        printf("lmdif/␣actred=%.10e␣prered=%.10e␣ratio=%.10e"
                               "␣sq(1)=%.10e␣sq(2)=%.10e␣dd=%.10e\n",
                               actred, prered, prered != 0 ? ratio : 0.,
                               SQR(temp1), SQR(temp2), dirder);
#endif

                        /*** inner: update the step bound. ***/
```

```
                        if(ratio <= 0.25)
                        {
                                if(actred >= 0.)
                                        temp = 0.5;
                                else
                                        temp = 0.5 * dirder / (dirder + 0.5 * actred);
                                if(p1 * fnorm1 >= fnorm || temp < p1)
                                        temp = p1;
                                delta = temp * MIN(delta, pnorm / p1);
                                par /= temp;
                        }
                        else if(par == 0. || ratio >= 0.75)
                        {
                                delta = pnorm / 0.5;
                                par *= 0.5;
                        }

                        /*** inner: test for successful iteration. ***/

                        if(ratio >= p0001)
                        {
                                /* yes, success: update x, fvec, and their norms. */
                                for(j = 0; j < n; j++)
                                {
                                        x[j] = wa2[j];
                                        wa2[j] = diag[j] * x[j];
                                }
                                for(i = 0; i < m; i++)
                                        fvec[i] = wa4[i];
                                xnorm = lm_enorm(n, wa2);
                                fnorm = fnorm1;
                                iter++;
                        }
#ifdef LMFIT_DEBUG_MESSAGES
                        else
                        {
                                printf("ATTN: iteration considered unsuccessful\n");
                        }
#endif

                        /*** inner: test for convergence. ***/

                        if(fnorm <= LM_DWARF)
                        {
                                *info = 0;
                                return;
                        }

                        *info = 0;
                        if(fabs(actred) <= ftol && prered <= ftol && 0.5 * ratio <= 1)
                                *info = 1;
                        if(delta <= xtol * xnorm)
                                *info += 2;
                        if(*info != 0)
                                return;

                        /*** inner: tests for termination and stringent tolerances. ***/

                        if(*nfev >= maxfev)
                        {
                                *info = 5;
                                return;
                        }
                        if(fabs(actred) <= LM_MACHEP &&
                                prered <= LM_MACHEP && 0.5 * ratio <= 1)
                        {
                                *info = 6;
                                return;
                        }
                        if(delta <= LM_MACHEP * xnorm)
                        {
                                *info = 7;
                                return;
                        }
```

```
                        if(gnorm <= LM_MACHEP)
                        {
                                *info = 8;
                                return;
                        }

                        /*** inner: end of the loop. repeat if iteration unsuccessful. ***/

                }
                while(ratio < p0001);

                /*** outer: end of the loop. ***/

        }
        while(1);

} /*** lm_lmdif. ***/


/*****************************************************************************/
/*  lm_lmpar (determine Levenberg-Marquardt parameter)                       */
/*****************************************************************************/

void lm_lmpar(int n, double* r, int ldr, int* ipvt, double* diag,
              double* qtb, double delta, double* par, double* x,
              double* sdiag, double* aux, double* xdi)
{
        /*      Given an m by n matrix a, an n by n nonsingular diagonal
         *      matrix d, an m-vector b, and a positive number delta,
         *      the problem is to determine a value for the parameter
         *      par such that if x solves the system
         *
         *           a*x = b  and  sqrt(par)*d*x = 0
         *
         *      in the least squares sense, and dxnorm is the euclidean
         *      norm of d*x, then either par=0 and (dxnorm-delta) < 0.1*delta,
         *      or par>0 and abs(dxnorm-delta) < 0.1*delta.
         *
         *      Using lm_qrsolv, this subroutine completes the solution of the problem
         *      if it is provided with the necessary information from the
         *      qr factorization, with column pivoting, of a. That is, if
         *      a*p = q*r, where p is a permutation matrix, q has orthogonal
         *      columns, and r is an upper triangular matrix with diagonal
         *      elements of nonincreasing magnitude, then lmpar expects
         *      the full upper triangle of r, the permutation matrix p,
         *      and the first n components of qT*b. On output
         *      lmpar also provides an upper triangular matrix s such that
         *
         *           pT*(aT*a + par*d*d)*p = sT*s.
         *
         *      s is employed within lmpar and may be of separate interest.
         *
         *      Only a few iterations are generally needed for convergence
         *      of the algorithm. If, however, the limit of 10 iterations
         *      is reached, then the output par will contain the best
         *      value obtained so far.
         *
         *      parameters:
         *
         *       n is a positive integer input variable set to the order of r.
         *
         *       r is an n by n array. on input the full upper triangle
         *         must contain the full upper triangle of the matrix r.
         *         on OUTPUT the full upper triangle is unaltered, and the
         *         strict lower triangle contains the strict upper triangle
         *         (transposed) of the upper triangular matrix s.
         *
         *       ldr is a positive integer input variable not less than n
         *         which specifies the leading dimension of the array r.
         *
         *       ipvt is an integer input array of length n which defines the
         *         permutation matrix p such that a*p = q*r. column j of p
         *         is column ipvt(j) of the identity matrix.
         *
         *       diag is an input array of length n which must contain the
```

```
         *         diagonal elements of the matrix d.
         *
         *       qtb is an input array of length n which must contain the first
         *         n elements of the vector (q transpose)*b.
         *
         *       delta is a positive input variable which specifies an upper
         *         bound on the euclidean norm of d*x.
         *
         *       par is a nonnegative variable. on input par contains an
         *         initial estimate of the levenberg-marquardt parameter.
         *         on OUTPUT par contains the final estimate.
         *
         *       x is an OUTPUT array of length n which contains the least
         *         squares solution of the system a*x = b, sqrt(par)*d*x = 0,
         *         for the output par.
         *
         *       sdiag is an array of length n which contains the
         *         diagonal elements of the upper triangular matrix s.
         *
         *       aux is a multi-purpose work array of length n.
         *
         *       xdi is a work array of length n. On OUTPUT: diag[j] * x[j].
         *
         */
        int i, iter, j, nsing;
        double dxnorm, fp, fp_old, gnorm, parc, parl, paru;
        double sum, temp;
        static double p1 = 0.1;

#ifdef LMFIT_DEBUG_MESSAGES
        printf("lmpar\n");
#endif

        /*** lmpar: compute and store in x the gauss-newton direction. if the
             jacobian is rank-deficient, obtain a least squares solution. ***/

        nsing = n;
        for(j = 0; j < n; j++)
        {
                aux[j] = qtb[j];
                if(r[j * ldr + j] == 0 && nsing == n)
                        nsing = j;
                if(nsing < n)
                        aux[j] = 0;
        }
#ifdef LMFIT_DEBUG_MESSAGES
        printf("nsing␣%d␣", nsing);
#endif
        for(j = nsing - 1; j >= 0; j--)
        {
                aux[j] = aux[j] / r[j + ldr * j];
                temp = aux[j];
                for(i = 0; i < j; i++)
                        aux[i] -= r[j * ldr + i] * temp;
        }

        for(j = 0; j < n; j++)
                x[ipvt[j]] = aux[j];

        /*** lmpar: initialize the iteration counter, evaluate the function at the
             origin, and test for acceptance of the gauss-newton direction. ***/

        iter = 0;
        for(j = 0; j < n; j++)
                xdi[j] = diag[j] * x[j];
        dxnorm = lm_enorm(n, xdi);
        fp = dxnorm - delta;
        if(fp <= p1 * delta)
        {
#ifdef LMFIT_DEBUG_MESSAGES
                printf("lmpar/␣terminate␣(fp<p1*delta)\n");
#endif
                *par = 0;
                return;
        }
```

```
        /*** lmpar: if the jacobian is not rank deficient, the newton
             step provides a lower bound, parl, for the 0. of
             the function. otherwise set this bound to 0.. ***/

        parl = 0;
        if(nsing >= n)
        {
                for(j = 0; j < n; j++)
                        aux[j] = diag[ipvt[j]] * xdi[ipvt[j]] / dxnorm;

                for(j = 0; j < n; j++)
                {
                        sum = 0.;
                        for(i = 0; i < j; i++)
                                sum += r[j * ldr + i] * aux[i];
                        aux[j] = (aux[j] - sum) / r[j + ldr * j];
                }
                temp = lm_enorm(n, aux);
                parl = fp / delta / temp / temp;
        }

        /*** lmpar: calculate an upper bound, paru, for the 0. of the function. ***/

        for(j = 0; j < n; j++)
        {
                sum = 0;
                for(i = 0; i <= j; i++)
                        sum += r[j * ldr + i] * qtb[i];
                aux[j] = sum / diag[ipvt[j]];
        }
        gnorm = lm_enorm(n, aux);
        paru = gnorm / delta;
        if(paru == 0.)
                paru = LM_DWARF / MIN(delta, p1);

        /*** lmpar: if the input par lies outside of the interval (parl,paru),
             set par to the closer endpoint. ***/

        *par = MAX(*par, parl);
        *par = MIN(*par, paru);
        if(*par == 0.)
                *par = gnorm / dxnorm;
#ifdef LMFIT_DEBUG_MESSAGES
        printf("lmpar/␣parl␣%.4e␣␣par␣%.4e␣␣paru␣%.4e\n", parl, *par, paru);
#endif

        /*** lmpar: iterate. ***/

        for(;; iter++)
        {

                /** evaluate the function at the current value of par. **/

                if(*par == 0.)
                        *par = MAX(LM_DWARF, 0.001 * paru);
                temp = sqrt(*par);
                for(j = 0; j < n; j++)
                        aux[j] = temp * diag[j];

                lm_qrsolv(n, r, ldr, ipvt, aux, qtb, x, sdiag, xdi);
                /* return values are r, x, sdiag */

                for(j = 0; j < n; j++)
                        xdi[j] = diag[j] * x[j]; /* used as output */
                dxnorm = lm_enorm(n, xdi);
                fp_old = fp;
                fp = dxnorm - delta;

                /** if the function is small enough, accept the current value
                    of par. Also test for the exceptional cases where parl
                    is zero or the number of iterations has reached 10. **/

                if(fabs(fp) <= p1 * delta
                        || (parl == 0. && fp <= fp_old && fp_old < 0.)
```

```
                        || iter == 10)
                        break; /* the only exit from the iteration. */

                /** compute the Newton correction. **/

                for(j = 0; j < n; j++)
                        aux[j] = diag[ipvt[j]] * xdi[ipvt[j]] / dxnorm;

                for(j = 0; j < n; j++)
                {
                        aux[j] = aux[j] / sdiag[j];
                        for(i = j + 1; i < n; i++)
                                aux[i] -= r[j * ldr + i] * aux[j];
                }
                temp = lm_enorm(n, aux);
                parc = fp / delta / temp / temp;

                /** depending on the sign of the function, update parl or paru. **/

                if(fp > 0)
                        parl = MAX(parl, *par);
                else if(fp < 0)
                        paru = MIN(paru, *par);
                /* the case fp==0 is precluded by the break condition  */

                /** compute an improved estimate for par. **/

                *par = MAX(parl, *par + parc);

        }

} /*** lm_lmpar. ***/


/*****************************************************************************/
/*  lm_qrfac (QR factorisation, from lapack)                                 */
/*****************************************************************************/

void lm_qrfac(int m, int n, double* a, int pivot, int* ipvt,
              double* rdiag, double* acnorm, double* wa)
{
        /*
         *      This subroutine uses householder transformations with column
         *      pivoting (optional) to compute a qr factorization of the
         *      m by n matrix a. That is, qrfac determines an orthogonal
         *      matrix q, a permutation matrix p, and an upper trapezoidal
         *      matrix r with diagonal elements of nonincreasing magnitude,
         *      such that a*p = q*r. The householder transformation for
         *      column k, k = 1,2,...,min(m,n), is of the form
         *
         *           i - (1/u(k))*u*uT
         *
         *      where u has zeroes in the first k-1 positions. The form of
         *      this transformation and the method of pivoting first
         *      appeared in the corresponding linpack subroutine.
         *
         *      Parameters:
         *
         *       m is a positive integer input variable set to the number
         *         of rows of a.
         *
         *       n is a positive integer input variable set to the number
         *         of columns of a.
         *
         *       a is an m by n array. On input a contains the matrix for
         *         which the qr factorization is to be computed. On OUTPUT
         *         the strict upper trapezoidal part of a contains the strict
         *         upper trapezoidal part of r, and the lower trapezoidal
         *         part of a contains a factored form of q (the non-trivial
         *         elements of the u vectors described above).
         *
         *       pivot is a logical input variable. If pivot is set true,
         *         then column pivoting is enforced. If pivot is set false,
         *         then no column pivoting is done.
         *
```

```
         *      ipvt is an integer OUTPUT array of length lipvt. This array
         *        defines the permutation matrix p such that a*p = q*r.
         *        Column j of p is column ipvt(j) of the identity matrix.
         *        If pivot is false, ipvt is not referenced.
         *
         *      rdiag is an OUTPUT array of length n which contains the
         *        diagonal elements of r.
         *
         *      acnorm is an OUTPUT array of length n which contains the
         *        norms of the corresponding columns of the input matrix a.
         *        If this information is not needed, then acnorm can coincide
         *        with rdiag.
         *
         *      wa is a work array of length n. If pivot is false, then wa
         *        can coincide with rdiag.
         *
         */
        int i, j, k, kmax, minmn;
        double ajnorm, sum, temp;

        /*** qrfac: compute initial column norms and initialize several arrays. ***/

        for(j = 0; j < n; j++)
        {
                acnorm[j] = lm_enorm(m, &a[j * m]);
                rdiag[j] = acnorm[j];
                wa[j] = rdiag[j];
                if(pivot)
                        ipvt[j] = j;
        }
#ifdef LMFIT_DEBUG_MESSAGES
        printf("qrfac\n");
#endif

        /*** qrfac: reduce a to r with householder transformations. ***/

        minmn = MIN(m, n);
        for(j = 0; j < minmn; j++)
        {
                if(!pivot)
                        goto pivot_ok;

                /** bring the column of largest norm into the pivot position. **/

                kmax = j;
                for(k = j + 1; k < n; k++)
                        if(rdiag[k] > rdiag[kmax])
                                kmax = k;
                if(kmax == j)
                        goto pivot_ok;

                for(i = 0; i < m; i++)
                {
                        temp = a[j * m + i];
                        a[j * m + i] = a[kmax * m + i];
                        a[kmax * m + i] = temp;
                }
                rdiag[kmax] = rdiag[j];
                wa[kmax] = wa[j];
                k = ipvt[j];
                ipvt[j] = ipvt[kmax];
                ipvt[kmax] = k;

pivot_ok:
                /** compute the Householder transformation to reduce the
                    j-th column of a to a multiple of the j-th unit vector. **/

                ajnorm = lm_enorm(m - j, &a[j * m + j]);
                if(ajnorm == 0.)
                {
                        rdiag[j] = 0;
                        continue;
                }

                if(a[j * m + j] < 0.)
```

```
                        ajnorm = -ajnorm;
                for(i = j; i < m; i++)
                        a[j * m + i] /= ajnorm;
                a[j * m + j] += 1;

                /** apply the transformation to the remaining columns
                    and update the norms. **/

                for(k = j + 1; k < n; k++)
                {
                        sum = 0;

                        for(i = j; i < m; i++)
                                sum += a[j * m + i] * a[k * m + i];

                        temp = sum / a[j + m * j];

                        for(i = j; i < m; i++)
                                a[k * m + i] -= temp * a[j * m + i];

                        if(pivot && rdiag[k] != 0.)
                        {
                                temp = a[m * k + j] / rdiag[k];
                                temp = MAX(0., 1 - temp * temp);
                                rdiag[k] *= sqrt(temp);
                                temp = rdiag[k] / wa[k];
                                if(0.05 * SQR(temp) <= LM_MACHEP)
                                {
                                        rdiag[k] = lm_enorm(m - j - 1, &a[m * k + j + 1]);
                                        wa[k] = rdiag[k];
                                }
                        }
                }

                rdiag[j] = -ajnorm;
        }
}


/**************************************************************************/
/*  lm_qrsolv (linear least-squares)                                      */
/**************************************************************************/

void lm_qrsolv(int n, double* r, int ldr, int* ipvt, double* diag,
               double* qtb, double* x, double* sdiag, double* wa)
{
        /*
         *      Given an m by n matrix a, an n by n diagonal matrix d,
         *      and an m-vector b, the problem is to determine an x which
         *      solves the system
         *
         *           a*x = b  and  d*x = 0
         *
         *      in the least squares sense.
         *
         *      This subroutine completes the solution of the problem
         *      if it is provided with the necessary information from the
         *      qr factorization, with column pivoting, of a. That is, if
         *      a*p = q*r, where p is a permutation matrix, q has orthogonal
         *      columns, and r is an upper triangular matrix with diagonal
         *      elements of nonincreasing magnitude, then qrsolv expects
         *      the full upper triangle of r, the permutation matrix p,
         *      and the first n components of (q transpose)*b. The system
         *      a*x = b, d*x = 0, is then equivalent to
         *
         *           r*z = qT*b,  pT*d*p*z = 0,
         *
         *      where x = p*z. If this system does not have full rank,
         *      then a least squares solution is obtained. On output qrsolv
         *      also provides an upper triangular matrix s such that
         *
         *           pT *(aT *a + d*d)*p = sT *s.
         *
         *      s is computed within qrsolv and may be of separate interest.
         *
```

```
         *     Parameters
         *
         *      n is a positive integer input variable set to the order of r.
         *
         *      r is an n by n array. On input the full upper triangle
         *        must contain the full upper triangle of the matrix r.
         *        On OUTPUT the full upper triangle is unaltered, and the
         *        strict lower triangle contains the strict upper triangle
         *        (transposed) of the upper triangular matrix s.
         *
         *      ldr is a positive integer input variable not less than n
         *        which specifies the leading dimension of the array r.
         *
         *      ipvt is an integer input array of length n which defines the
         *        permutation matrix p such that a*p = q*r. Column j of p
         *        is column ipvt(j) of the identity matrix.
         *
         *      diag is an input array of length n which must contain the
         *        diagonal elements of the matrix d.
         *
         *      qtb is an input array of length n which must contain the first
         *        n elements of the vector (q transpose)*b.
         *
         *      x is an OUTPUT array of length n which contains the least
         *        squares solution of the system a*x = b, d*x = 0.
         *
         *      sdiag is an OUTPUT array of length n which contains the
         *        diagonal elements of the upper triangular matrix s.
         *
         *      wa is a work array of length n.
         *
         */
        int i, kk, j, k, nsing;
        double qtbpj, sum, temp;
        double _sin, _cos, _tan, _cot; /* local variables, not functions */

        /*** qrsolv: copy r and (q transpose)*b to preserve input and initialize s.
             in particular, save the diagonal elements of r in x. ***/

        for(j = 0; j < n; j++)
        {
                for(i = j; i < n; i++)
                        r[j * ldr + i] = r[i * ldr + j];
                x[j] = r[j * ldr + j];
                wa[j] = qtb[j];
        }
#ifdef LMFIT_DEBUG_MESSAGES
        printf("qrsolv\n");
#endif

        /*** qrsolv: eliminate the diagonal matrix d using a Givens rotation. ***/

        for(j = 0; j < n; j++)
        {

                /*** qrsolv: prepare the row of d to be eliminated, locating the
                     diagonal element using p from the qr factorization. ***/

                if(diag[ipvt[j]] == 0.)
                        goto L90;
                for(k = j; k < n; k++)
                        sdiag[k] = 0.;
                sdiag[j] = diag[ipvt[j]];

                /*** qrsolv: the transformations to eliminate the row of d modify only
                     a single element of qT*b beyond the first n, which is initially 0. ***/

                qtbpj = 0.;
                for(k = j; k < n; k++)
                {

                        /** determine a Givens rotation which eliminates the
                            appropriate element in the current row of d. **/

                        if(sdiag[k] == 0.)
```

```
                        continue;
                kk = k + ldr * k;
                if(fabs(r[kk]) < fabs(sdiag[k]))
                {
                        _cot = r[kk] / sdiag[k];
                        _sin = 1 / sqrt(1 + SQR(_cot));
                        _cos = _sin * _cot;
                }
                else
                {
                        _tan = sdiag[k] / r[kk];
                        _cos = 1 / sqrt(1 + SQR(_tan));
                        _sin = _cos * _tan;
                }

                /** compute the modified diagonal element of r and
                    the modified element of ((q transpose)*b,0). **/

                r[kk] = _cos * r[kk] + _sin * sdiag[k];
                temp = _cos * wa[k] + _sin * qtbpj;
                qtbpj = -_sin * wa[k] + _cos * qtbpj;
                wa[k] = temp;

                /** accumulate the tranformation in the row of s. **/

                for(i = k + 1; i < n; i++)
                {
                        temp = _cos * r[k * ldr + i] + _sin * sdiag[i];
                        sdiag[i] = -_sin * r[k * ldr + i] + _cos * sdiag[i];
                        r[k * ldr + i] = temp;
                }
        }

L90:
        /** store the diagonal element of s and restore
            the corresponding diagonal element of r. **/

        sdiag[j] = r[j * ldr + j];
        r[j * ldr + j] = x[j];
}

/*** qrsolv: solve the triangular system for z. if the system is
     singular, then obtain a least squares solution. ***/

nsing = n;
for(j = 0; j < n; j++)
{
        if(sdiag[j] == 0. && nsing == n)
                nsing = j;
        if(nsing < n)
                wa[j] = 0;
}

for(j = nsing - 1; j >= 0; j--)
{
        sum = 0;
        for(i = j + 1; i < nsing; i++)
                sum += r[j * ldr + i] * wa[i];
        wa[j] = (wa[j] - sum) / sdiag[j];
}

/*** qrsolv: permute the components of z back to components of x. ***/

for(j = 0; j < n; j++)
        x[ipvt[j]] = wa[j];

} /*** lm_qrsolv. ***/


/**************************************************************************/
/*  lm_enorm (Euclidean norm)                                             */
/**************************************************************************/

double lm_enorm(int n, const double* x)
{
```

```
        /*      Given an n-vector x, this function calculates the
         *      euclidean norm of x.
         *
         *      The euclidean norm is computed by accumulating the sum of
         *      squares in three different sums. The sums of squares for the
         *      small and large components are scaled so that no overflows
         *      occur. Non-destructive underflows are permitted. Underflows
         *      and overflows do not occur in the computation of the unscaled
         *      sum of squares for the intermediate components.
         *      The definitions of small, intermediate and large components
         *      depend on two constants, LM_SQRT_DWARF and LM_SQRT_GIANT. The main
         *      restrictions on these constants are that LM_SQRT_DWARF**2 not
         *      underflow and LM_SQRT_GIANT**2 not overflow.
         *
         *      Parameters
         *
         *       n is a positive integer input variable.
         *
         *       x is an input array of length n.
         */
        int i;
        double agiant, s1, s2, s3, xabs, x1max, x3max, temp;

        s1 = 0;
        s2 = 0;
        s3 = 0;
        x1max = 0;
        x3max = 0;
        agiant = LM_SQRT_GIANT / n;

        /** sum squares. **/

        for(i = 0; i < n; i++)
        {
                xabs = fabs(x[i]);
                if(xabs > LM_SQRT_DWARF)
                {
                        if(xabs < agiant)
                        {
                                s2 += xabs * xabs;
                        }
                        else if(xabs > x1max)
                        {
                                temp = x1max / xabs;
                                s1 = 1 + s1 * SQR(temp);
                                x1max = xabs;
                        }
                        else
                        {
                                temp = xabs / x1max;
                                s1 += SQR(temp);
                        }
                }
                else if(xabs > x3max)
                {
                        temp = x3max / xabs;
                        s3 = 1 + s3 * SQR(temp);
                        x3max = xabs;
                }
                else if(xabs != 0.)
                {
                        temp = xabs / x3max;
                        s3 += SQR(temp);
                }
        }

        /** calculation of norm. **/

        if(s1 != 0)
                return x1max * sqrt(s1 + (s2 / x1max) / x1max);
        else if(s2 != 0)
                if(s2 >= x3max)
                        return sqrt(s2 * (1 + (x3max / s2) * (x3max * s3)));
                else
                        return sqrt(x3max * ((s2 / x3max) + (x3max * s3)));
```

```
        else
                return x3max * sqrt(s3);

} /*** lm_enorm. ***/
```

src/log/speclogentryitem.h

```
#ifndef SPECLOGENTRYITEM_H
#define SPECLOGENTRYITEM_H
#include "specfolderitem.h"
#include "specdescriptor.h"
#include <QHash>
class specLogEntryItem : public specModelItem
{
protected:
        QHash<QString, specDescriptor> description ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        bool isNumeric(const QString& key) const ;
        double numericalValue(const QString& key) const ;
private:
        type typeId() const { return specStreamable::logEntry ; }
public:
        explicit specLogEntryItem(QHash<QString, specDescriptor> description = QHash<QString, ►
            specDescriptor>(), specFolderItem* par = 0, QString tag = "");
        specLogEntryItem(const specLogEntryItem&) ;
        ~specLogEntryItem();
        bool isEditable(QString key) const;
        bool changeDescriptor(QString key, QString value) ;
        QStringList descriptorKeys() const ;
        QIcon decoration() const ;
        spec::descriptorFlags descriptorProperties(const QString& key) const ;
        void setDescriptorProperties(const QString& key, spec::descriptorFlags f) ;
        QString descriptor(const QString& key, bool full = false) const ;
        double descriptorValue(const QString& key) const ;
        bool setActiveLine(const QString&, int) ;
        int activeLine(const QString& key) const ;
        void renameDescriptors(const QMap<QString, QString>& map);
        void deleteDescriptor(const QString& descriptor);
        void dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const;
        void restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key) ;
};
#endif
```

src/log/speclogentryitem.cpp

```
#include "speclogentryitem.h"
specLogEntryItem::specLogEntryItem(QHash<QString, specDescriptor> desc, specFolderItem* par, QString►
      tag)
        : specModelItem(par, tag),
          description(desc)
{
}
specLogEntryItem::~specLogEntryItem()
{
}
bool specLogEntryItem::isEditable(QString key) const
{
        if(description.contains(key)) return description[key].isEditable() ;
        return specModelItem::isEditable(key);
}
bool specLogEntryItem::isNumeric(const QString& key) const
{
        if(description.contains(key)) return description[key].isNumeric() ;
        return specModelItem::isNumeric(key);
}
bool specLogEntryItem::changeDescriptor(QString key, QString value)
{
        if(description.contains(key))
                return description[key].setContent(value) ;
        if(specModelItem::descriptorKeys().contains(key))
                return specModelItem::changeDescriptor(key, value) ;
```

```
        description[key] = specDescriptor(value, spec::editable) ;
        return true ;
}
QStringList specLogEntryItem::descriptorKeys() const
{
        return (specModelItem::descriptorKeys() << description.keys()) ;
}
QIcon specLogEntryItem::decoration() const { return QIcon(":/logs.png") ; }
spec::descriptorFlags specLogEntryItem::descriptorProperties(const QString& key) const
{
        if(description.contains(key)) return description[key].flags() ;
        return specModelItem::descriptorProperties(key) ;
}
void specLogEntryItem::setDescriptorProperties(const QString& key, spec::descriptorFlags f)
{
        if(description.contains(key)) description[key].setFlags(f) ;
        else specModelItem::setDescriptorProperties(key, f) ;
}
QString specLogEntryItem::descriptor(const QString& key, bool full) const
{
        if(description.contains(key)) return description[key].content(full) ;
        return specModelItem::descriptor(key, full) ;
}
void specLogEntryItem::readFromStream(QDataStream& in)
{
        specModelItem::readFromStream(in) ;
        in >> description ;
}
void specLogEntryItem::writeToStream(QDataStream& out) const
{
        specModelItem::writeToStream(out) ;
        out << description ;
}
bool specLogEntryItem::setActiveLine(const QString& key, int line)
{
        if(description.contains(key)) return description[key].setActiveLine(line) ;
        return specModelItem::setActiveLine(key, line) ;
}
int specLogEntryItem::activeLine(const QString& key) const
{
        if(description.contains(key)) return description[key].activeLine() ;
        return specModelItem::activeLine(key) ;
}
specLogEntryItem::specLogEntryItem(const specLogEntryItem& other)
        : specModelItem(other),
          description(other.description)
{}
double specLogEntryItem::numericalValue(const QString& key) const
{
        if(description.contains(key)) return description[key].numericValue() ;
        return NAN ;
}
void specLogEntryItem::renameDescriptors(const QMap<QString, QString>& map)
{
        QHash<QString, specDescriptor> newDescription ;
        foreach(const QString & key, map.keys())
        newDescription[map[key]] = description[key] ;
        foreach(const QString & key, description.keys())
        if(!map.contains(key))
                newDescription[key] = description[key] ;
        description.swap(newDescription) ;
}
void specLogEntryItem::deleteDescriptor(const QString& key)
{
        description.remove(key) ;
}
void specLogEntryItem::dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const
{
        if(description.contains(key))
                destination << description[key] ;
        else
                specModelItem::dumpDescriptor(destination, key) ;
}
void specLogEntryItem::restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key)
{
```

```
        if(!origin.hasNext()) return ;
        if(key == "") specModelItem::restoreDescriptor(origin, key) ;
        else description[key] = origin.next() ;
}
double specLogEntryItem::descriptorValue(const QString& key) const
{
        if(description.contains(key)) return description[key].numericValue() ;
        return specModelItem::descriptorValue(key) ;
}
```

src/log/speclogmessage.h

```
#ifndef SPECLOGMESSAGE_H
#define SPECLOGMESSAGE_H
#include "speclogentryitem.h"
class specLogMessage : public specLogEntryItem
{
private:
        type typeId() const { return specStreamable::sysEntry ; }
public:
        specLogMessage(QHash<QString, specDescriptor> description = QHash<QString, specDescriptor>(),
                       specFolderItem* par = 0,
                       QString tag = "") ;
        ~specLogMessage();
        bool changeDescriptor(QString key, QString value) ;
        bool isEditable(QString key) const;
        QIcon decoration() const;
};
#endif
```

src/log/speclogmessage.cpp

```
#include "speclogmessage.h"
specLogMessage::specLogMessage(QHash<QString, specDescriptor> description, specFolderItem* par, ►
    QString tag)
        : specLogEntryItem(description, par, tag)
{
}
specLogMessage::~specLogMessage()
{
}
bool specLogMessage::isEditable(QString key) const
{
        Q_UNUSED(key)
        return false ;
}
QIcon specLogMessage::decoration() const { return QIcon::fromTheme("dialog-warning") ; }
bool specLogMessage::changeDescriptor(QString key, QString value)
{
        Q_UNUSED(key)
        Q_UNUSED(value)
        return false ;
}
```

src/log/speclogmodel.h

```
#ifndef SPECLOGMODEL_H
#define SPECLOGMODEL_H
#include "specmodel.h"
class specLogModel : public specModel
{
private:
        Q_OBJECT
        type typeId() const { return specStreamable::logModel ; }
        QStringList dataTypes() const ;
public:
        explicit specLogModel(QObject* parent = 0);
        QList<specFileImportFunction> acceptableImportFunctions() const ;
};
#endif
```

src/log/speclogmodel.cpp

```
#include "speclogmodel.h"
#include <QFile>
#include <QFileDialog>
#include "utility-functions.h"
specLogModel::specLogModel(QObject* parent)
        : specModel(parent)
{
        setObjectName("logModel");
}
QStringList specLogModel::dataTypes() const
{
        return QStringList() ;
}
QList<specFileImportFunction> specLogModel::acceptableImportFunctions() const
{
        return QList<specFileImportFunction>() << readLogFile ;
}
```

src/log/speclogtodataconverter.h

```
#ifndef SPECLOGTODATACONVERTER_H
#define SPECLOGTODATACONVERTER_H
#include "specmimeconverter.h"
#include <QFileDialog>
#include "specstreamable.h"
#include "specmodelitem.h"
class specLogToDataConverter : public specMimeConverter, private specStreamable
{
        Q_OBJECT
private:
        specModelItem* getData(specModelItem*) ;
        QDir currentDirectory ;
        specModelItem* factory(const type& t) const ;
        void writeToStream(QDataStream& out) const { Q_UNUSED(out) }
        void readFromStream(QDataStream& in) { Q_UNUSED(in) }
        specStreamable::type typeId() const { return specStreamable::none ;}
        void toStream(specModelItem*, QDataStream&) ;
public:
        explicit specLogToDataConverter(QObject* parent = 0);
        QList<specModelItem*> importData(const QMimeData*) ;
        void exportData(QList<specModelItem*>&, QMimeData*) ;
        bool canImport(const QStringList&) ;
        QStringList importableTypes() const ;
};
#endif
```

src/log/speclogtodataconverter.cpp

```
#include "speclogtodataconverter.h"
#include "speclogentryitem.h"
#include "utility-functions.h"
#include <QStack>
#include <QMessageBox>
#include <QMimeData>
#include "speclogmodel.h"
#include <QSettings>
#include "speclogmessage.h"
specLogToDataConverter::specLogToDataConverter(QObject* parent)
        : specMimeConverter(parent)
{
}
specModelItem* specLogToDataConverter::factory(const type& t) const
{
        return specModelItem::itemFactory(t) ;
}
bool specLogToDataConverter::canImport(const QStringList& types)
{
        return qobject_cast<specModel*> (parent()) &&
               !qobject_cast<specLogModel*> (parent()) &&
               types.contains("application/spec.logged.files") ;
```

```
}
QStringList specLogToDataConverter::importableTypes() const
{
        return specMimeConverter::importableTypes() << "application/spec.logged.files" ;
}
void specLogToDataConverter::toStream(specModelItem* item, QDataStream& out)
{
        QPair<qint8, QString> entry ;
        if(item->isFolder())
        {
                entry.first = 1 ;
                entry.second = item->descriptor("", true) ;
                out << entry ;
                entry.first = 2 ;
                specFolderItem* folder = (specFolderItem*) item ;
                for(int i = 0 ; i < folder->children() ; ++i)
                        toStream(folder->child(i), out) ;
        }
        else if(!dynamic_cast<specLogMessage*>(item))
        {
                entry.first = 0 ;
                entry.second = item->descriptor("Datei") ;
        }
        else return ;
        out << entry ;
}
void specLogToDataConverter::exportData(QList<specModelItem*>& items, QMimeData* data)
{
        QByteArray ba ;
        QDataStream out(&ba, QIODevice::WriteOnly) ;
        foreach(specModelItem * item, items)
        toStream(item, out) ;
        if(!ba.isEmpty())
                data->setData("application/spec.logged.files", ba) ;
}
QList<specModelItem*> specLogToDataConverter::importData(const QMimeData* data)
{
        QSettings settings ;
        QByteArray ba(data->data("application/spec.logged.files")) ;
        QDataStream in(&ba, QIODevice::ReadOnly) ;
        QPair<qint8, QString> entry ;
        QStack<specFolderItem*> parents ;
        QList<specModelItem*> items ;
        while(!in.atEnd())
        {
                in >> entry ;
                if(entry.first == 1)
                {
                        specFolderItem* newItem = new specFolderItem(0, entry.second) ;
                        if(parents.empty())
                                items << newItem ;
                        else
                                parents.top()->addChild(newItem, parents.top()->children());
                        parents.push(newItem) ;
                }
                else if(entry.first == 2)
                        parents.pop() ;
                else
                {
                        QString fileName = entry.second ;
                        fileName = fileName.section("\\", -1, -1) ;
                        QDir::setCurrent(currentDirectory.absolutePath()) ;
                        QFile file(fileName) ;
                        if(!file.exists())
                        {
                                QStringList directories(settings.value("logConverter/importDirectory►
                                    ").toStringList()) ;
                                foreach(QString directory, directories)
                                {
                                        currentDirectory.setPath(directory) ;
                                        QDir::setCurrent(currentDirectory.absolutePath()) ;
                                        file.setFileName(currentDirectory.absoluteFilePath(fileName)►
                                            ) ;
                                        if(file.exists()) break ;
                                }
```

```
                                if(!file.exists())
                                {
                                        QFileDialog path ;
                                        path.setWindowTitle("Datei␣zum␣Importieren␣angeben") ;
100                                     path.setDirectory(currentDirectory) ;
                                        path.setFileMode(QFileDialog::ExistingFile);
                                        path.selectFile(fileName) ;
                                        path.setNameFilters(QStringList(QString("%1␣(%1)").arg(►
                                             fileName))) ;
                                        if(path.exec() == QDialog::Rejected || path.selectedFiles().►
                                             isEmpty())
105                                     {
                                                if(QMessageBox::question(0,
                                                                         "Import␣abbrechen?",
                                                                         "Soll␣der␣Import␣von␣►
                                                                              Datendateien␣►
                                                                              abgebrochen␣werden?",
                                                                         QMessageBox::Yes | ►
                                                                              QMessageBox::No,
110                                                                      QMessageBox::No) == ►
                                                                              QMessageBox::Yes)
                                                {
                                                        foreach(specModelItem * item, items)
                                                        delete item ;
                                                        items.clear();
115                                                     return items ;
                                                }
                                                continue ;
                                        }
                                        currentDirectory = path.directory() ;
120                                     directories << currentDirectory.absolutePath() ;
                                        settings.setValue("logConverter/importDirectory", ►
                                             directories) ;
                                        file.setFileName(path.selectedFiles().first()) ;
                                }
                        }
125                     if(!file.open(QFile::ReadOnly | QFile::Text))
                        {
                                QMessageBox::warning(0,
                                                     "Fehler",
                                                     "Kann␣Datei␣nicht␣oeffnen:␣" + fileName) ;
130                             continue ;
                        }
                        specFileImportFunction filter = fileFilter(file.fileName()) ;
                        if(!filter)
                        {
135                             QMessageBox::warning(0,
                                                     "Fehler",
                                                     "Kein␣gueltiger␣Dateityp:" + fileName) ;
                                continue ;
                        }
140                     specFolderItem* newItem = new specFolderItem ;
                        newItem->changeDescriptor("", currentDirectory.absolutePath() + "\n" + ►
                             fileName) ;
                        newItem->setActiveLine("", 1) ;
                        newItem->addChildren(filter(file)) ;
                        if(parents.isEmpty())
145                             items << newItem ;
                        else
                                parents.top()->addChild(newItem, parents.top()->children());
                }
        }
150     return items ;
}
```

src/log/speclogview.h

```
#ifndef SPECLOGVIEW_H
#define SPECLOGVIEW_H
#include "specview.h"
4 #include "speclogmodel.h"
class specLogView : public specView
{
private:
```

```
        Q_OBJECT
        type typeId() const { return specStreamable::logView ; }
public:
        explicit specLogView(QWidget* parent = 0);
        void setModel(specLogModel*) ;
};
#endif
```

src/log/speclogview.cpp

```
#include "speclogview.h"
specLogView::specLogView(QWidget* par)
        : specView(par)
{
        setWhatsThis(tr("This␣list␣contains␣log␣entries␣and␣system␣warnings␣obtained␣from␣log␣files.►
            ␣␣Use␣the␣import␣button␣to␣add␣the␣contents␣of␣log␣files␣to␣this␣list.␣␣Manage␣items␣by␣►
            dragging␣and␣dropping.\nItems␣can␣be␣dragged␣from␣this␣list␣to␣the␣data␣list␣to␣insert␣►
            the␣corresponding␣data␣sets␣there.\nSome␣log␣data␣may␣be␣changed␣by␣double␣clicking␣onto►
            ␣the␣corresponding␣entry.␣␣The␣first␣column,␣e.g.␣is␣reserved␣for␣your␣comments/►
            descriptions.␣␣If␣a␣field␣contains␣multiple␣lines,␣an␣indicator␣will␣be␣displayed."));
        setObjectName("logView");
}
void specLogView::setModel(specLogModel* mod)
{
        specView::setModel(mod) ;
}
```

src/log/speclogwidget.h

```
#ifndef SPECLOGWIDGET_H
#define SPECLOGWIDGET_H
#include "specdockwidget.h"
#include "speclogview.h"
class QVBoxLayout ;
class specLogWidget : public specDockWidget, public specStreamable
{
        Q_OBJECT
private:
        specLogView* logView ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::logWidget ; }
protected:
        QList<QWidget*> mainWidgets() const ;
public:
        explicit specLogWidget(QWidget* parent = 0);
        specView* view() { return logView ; }
};
#endif
```

src/log/speclogwidget.cpp

```
#include "speclogwidget.h"
#include <QVBoxLayout>
#include "speclogtodataconverter.h"
#include "specgenericmimeconverter.h"
#include "speclogtodataconverter.h"
#include "specmimetextexporter.h"
#include "specmimefileimporter.h"
#include "specactionlibrary.h"
specLogWidget::specLogWidget(QWidget* parent)
        : specDockWidget(tr("Log"), parent),
          logView(new specLogView(this))
{
        setWhatsThis(tr("Log␣widget␣--␣This␣widget␣contains␣log␣data.␣␣Individual␣log␣items␣may␣be␣►
            dragged␣to␣the␣data␣widget␣in␣order␣to␣import␣the␣corresponding␣data␣sets."));
        logView->setModel(new specLogModel(logView));
        new specGenericMimeConverter(logView->model()) ;
        new specLogToDataConverter(logView->model()) ;
        new specMimeFileImporter(logView->model()) ;
```

```
        new specMimeTextExporter(logView->model()) ;
        setWhatsThis(tr("This␣dock␣window␣contains␣log␣data."));
        setObjectName(tr("Log␣window")) ;
        toggleViewAction()->setText(tr("Toggle␣log␣window")) ;
}
void specLogWidget::writeToStream(QDataStream& out) const
{
        out << *logView ;
}
void specLogWidget::readFromStream(QDataStream& in)
{
        in >> *logView ;
}
QList<QWidget*> specLogWidget::mainWidgets() const
{
        return QList<QWidget*>() << logView ;
}
```

src/main.cpp

```
#include <QApplication>
#include <specappwindow.h>
#include <QVector>
#include "asciiexporter.h"
#include <QFile>
#include "bzipiodevice.h"
#include <QPair>
#include "specshortcutdialog.h"
int main(int argc, char* argv[])
{
#ifdef WIN32BUILD
        QIcon::setThemeName("oxygen") ;
#endif
        Q_INIT_RESOURCE(application);
        QCoreApplication::setOrganizationName("MPIbpC") ;
        QCoreApplication::setOrganizationDomain("mpibpc.mpg.de") ;
        QCoreApplication::setApplicationName("spec-PumpProbe") ;
        QApplication app(argc, argv) ;
        QString parameter(argc > 1 ? QString(argv[1]) : QString("")) ;
        if(parameter.left(2).toLower() == "-e")
        {
                bool withFiles(parameter.left(2) == "-E") ;
                QVector<QPair<QVector<int>, QString> > items ;
                for(int i = 3 ; i < argc ; ++i)
                {
                        QStringList l = QString(argv[i]).split("/") ;
                        QVector<int> item ;
                        foreach(QString n, l)
                        item << n.toInt() ;
                        QString filename ;
                        if(withFiles && i < argc) filename = argv[++i] ;
                        items << qMakePair(item, filename) ;
                }
                QFile* file = new QFile(argv[2]) ;
                if(!file->open(QFile::ReadOnly))
                {
                        qDebug() << "Could␣not␣open␣file␣" << file->fileName() << "␣for␣export.";
                        delete file ;
                        return 1 ;
                }
                QTextStream cout(stdout, QIODevice::WriteOnly) ;
                QDataStream in(file) ;
                asciiExporter exporter(parameter.mid(2) == "d" ? asciiExporter::data :
                                       (parameter.mid(2) == "l" ? asciiExporter::log : asciiExporter►
                                            ::meta)) ;
                quint64 check ;
                in >> check ;
                if(check != FILECHECKRANDOMNUMBER && check != FILECHECKCOMPRESSNUMBER)
                {
                        qDebug() << "File␣" << file->fileName() << "␣is␣not␣valid." ;
                        delete file ;
                        return 2 ;
                }
                QByteArray fileContent = file->readAll() ;
```

```
                file->close();
                QDataStream inStream(fileContent) ;
56              bzipIODevice* zipDevice = 0 ;
                QBuffer buffer(&fileContent) ;
                if(FILECHECKCOMPRESSNUMBER == check)
                {
                        qDebug() << "File " << file->fileName() << " is compressed." ;
61                      zipDevice = new bzipIODevice(&buffer) ;
                        qDebug() << "opening buffer:" << zipDevice->open(bzipIODevice::ReadOnly) ;
                        inStream.setDevice(zipDevice);
                }
                exporter.readFromStream(inStream) ;
66              zipDevice->releaseDevice() ;
                delete zipDevice ;
                typedef QPair<QVector<int>, QString> itemPair ;
                qDebug() << "Exporting from file " << file->fileName() ;
                delete file ;
71              foreach(const itemPair & item, items)
                {
                        QFile outfile(item.second);
                        qDebug() << "Exporting item " << item.first << " to " << item.second ;
                        if(!item.second.isEmpty() && outfile.open(QFile::WriteOnly))
76                      {
                                QTextStream out(&outfile) ;
                                out << exporter.content(item.first) ;
                                out.setDevice(&outfile) ;
                        }
81                      else
                                cout << exporter.content(item.first) ;
                }
        }
        else
86      {
                specAppWindow* mainWindow = new specAppWindow();
                new specShortcutDialog(mainWindow) ;
                mainWindow->show();
                return app.exec();
91      }
}
```

src/model/exportdialog.h

```
#ifndef EXPORTDIALOG_H
#define EXPORTDIALOG_H
3   #include <QDialog>
#include <QDialogButtonBox>
#include <QPushButton>
#include <QHBoxLayout>
#include <QStringList>
8   #include <QScrollArea>
#include "names.h"
class exportFormatItem ;
class exportLayoutItem ;
class exportDialog : public QDialog
13  {
        Q_OBJECT
private:
        QPushButton* addButton, *addDataButton ;
        QBoxLayout* layout, *inLayout, *dataLayout ;
18      QStringList descriptors ;
        QStringList dataTypes ;
        QScrollArea* scrollHeader, *scrollData ;
        void prepareHeader() ;
        void prepareData() ;
23      template<class itemType> QList<itemType*> getItems() const ;
private slots :
        void addHeaderItem() ;
        void addDataItem() ;
public:
28      explicit exportDialog(QWidget* parent = 0);
        void setDataTypes(const QStringList& dataTypes) ;
        void setDescriptors(const QStringList& descriptors) ;
        QList<QPair<bool, QString> > headerFormat() const ;
        QList<QPair<spec::value, QString> > dataFormat() const ;
```

```
33          ~exportDialog();
};
#endif
```

src/model/exportdialog.cpp

```
#include "exportdialog.h"
#include <QVBoxLayout>
#include <QSpacerItem>
#include <QTextStream>
5  #include <QLabel>
#include "exportlayoutitem.h"
#include "exportformatitem.h"
template <class itemType>
QList<itemType*> exportDialog::getItems() const
10 {
        return findChildren<itemType*>() ;
}
exportDialog::exportDialog(QWidget* parent)
        : QDialog(parent),
15        scrollData(0)
{
        setWindowTitle(tr("Output␣format")) ;
        setMinimumWidth(400) ;
        layout = new QVBoxLayout(this) ;
20      setLayout(layout) ;
        QDialogButtonBox* buttonBox = new QDialogButtonBox(QDialogButtonBox::Ok
                | QDialogButtonBox::Cancel);
        connect(buttonBox, SIGNAL(accepted()), this, SLOT(accept()));
        connect(buttonBox, SIGNAL(rejected()), this, SLOT(reject()));
25      prepareHeader() ;
        prepareData() ;
        layout->addWidget(buttonBox) ;
}
void exportDialog::setDataTypes(const QStringList& dTs)
30 {
        bool dataTypesWereEmpty = dataTypes.isEmpty() ;
        dataTypes = dTs ;
        if(dataTypesWereEmpty)
                prepareData();
35      foreach(exportFormatItem * pointer, getItems<exportFormatItem>())
        pointer->setDataTypes(dataTypes) ;
}
void exportDialog::setDescriptors(const QStringList& newDs)
{
40      descriptors = newDs ;
        foreach(exportLayoutItem * pointer, getItems<exportLayoutItem>())
        pointer->setDescriptors(newDs) ;
}
void exportDialog::prepareHeader()
45 {
        scrollHeader = new QScrollArea() ;
        QWidget* scrollWidget = new QWidget() ;
        inLayout = new QVBoxLayout(scrollWidget) ;
        scrollWidget->setLayout(inLayout) ;
50      scrollHeader->setWidget(scrollWidget) ;
        scrollHeader->setWidgetResizable(true) ;
        QWidget* add = new QWidget() ;
        QHBoxLayout* addButtonLayout = new QHBoxLayout() ;
        addButtonLayout->addStretch() ;
55      addButton = new QPushButton(QIcon::fromTheme("list-add"),"");
        addButton->setFixedHeight(16) ;
        addButton->setFixedWidth(16) ;
        addButton->setFlat(true) ;
        addButtonLayout->addWidget(addButton) ;
60      addButtonLayout->setContentsMargins(0, 4, 0, 0) ;
        add->setLayout(addButtonLayout) ;
        inLayout->addWidget(add) ;
        connect(addButton, SIGNAL(clicked()), this, SLOT(addHeaderItem())) ;
        inLayout->addStretch() ;
65      inLayout->setSpacing(0) ;
        inLayout->setContentsMargins(0, 0, 0, 0) ;
        layout->addWidget(new QLabel("Item␣description␣format", this)) ;
        layout->addWidget(scrollHeader) ;
```

```
}
void exportDialog::prepareData()
{
        if(dataTypes.isEmpty()) return ;
        delete scrollData ;
        scrollData = new QScrollArea() ;
        QWidget* scrollWidget = new QWidget() ;
        dataLayout = new QVBoxLayout(scrollWidget) ;
        scrollWidget->setLayout(dataLayout) ;
        scrollData->setWidget(scrollWidget) ;
        scrollData->setWidgetResizable(true) ;
        QWidget* add = new QWidget() ;
        QHBoxLayout* addButtonLayout = new QHBoxLayout() ;
        addButtonLayout->addStretch() ;
        addDataButton = new QPushButton(QIcon::fromTheme("list-add"),"") ;
        addDataButton->setFixedHeight(16) ;
        addDataButton->setFixedWidth(16) ;
        addDataButton->setFlat(true) ;
        addButtonLayout->addWidget(addDataButton) ;
        addButtonLayout->setContentsMargins(0, 4, 0, 0) ;
        add->setLayout(addButtonLayout) ;
        dataLayout->addWidget(add) ;
        connect(addDataButton, SIGNAL(clicked()), this, SLOT(addDataItem())) ;
        dataLayout->addStretch() ;
        dataLayout->setSpacing(0) ;
        dataLayout->setContentsMargins(0, 0, 0, 0) ;
        exportFormatItem* pointer ;
        for(int i = 0 ; i < dataTypes.size() ; ++i)
        {
                addDataItem();
                if(dataLayout->itemAt(i) &&
                        (pointer =
                             dynamic_cast<exportFormatItem*>(dataLayout->itemAt(i)->widget())))
                {
                        pointer->setValue((spec::value) i) ;
                        pointer->setSeparator(spec::space) ;
                }
        }
        if(!dataTypes.isEmpty() &&
                dataLayout->itemAt(dataTypes.size() - 1) &&
                (pointer =
                     dynamic_cast<exportFormatItem*>(dataLayout->itemAt(dataTypes.size() - 1)->►
                         widget())))
                pointer->setSeparator(spec::newline) ;
        if(scrollData)
        {
                layout->insertWidget(layout->count() - 1, new QLabel("Individual␣data␣point␣output␣►
                    format", this)) ;
                layout->insertWidget(layout->count() - 1, scrollData);
        }
}
void exportDialog::addHeaderItem()
{
        inLayout->insertWidget(inLayout->count() - 2, new exportLayoutItem(descriptors, this)) ;
}
void exportDialog::addDataItem()
{
        if(dataTypes.isEmpty()) return ;
        dataLayout->insertWidget(dataLayout->count() - 2, new exportFormatItem(dataTypes, this)) ;
}
QList<QPair<spec::value, QString> > exportDialog::dataFormat() const
{
        QList<QPair<spec::value, QString> > returnData ;
        foreach(exportFormatItem * pointer, getItems<exportFormatItem>())
        returnData << QPair<spec::value, QString> (pointer->value(), pointer->separator()) ;
        return returnData ;
}
QList<QPair<bool, QString> > exportDialog::headerFormat() const
{
        QList<QPair<bool, QString> > returnData ;
        foreach(exportLayoutItem * pointer, getItems<exportLayoutItem>())
        returnData << QPair<bool, QString> (pointer->isFreeText(), pointer->text()) ;
        return returnData ;
}
exportDialog::~exportDialog()
```

```
{
}
```

src/model/exportformatitem.h

```
#ifndef EXPORTFORMATITEM_H
#define EXPORTFORMATITEM_H
#include <QWidget>
#include <QComboBox>
#include <QPushButton>
#include <QHBoxLayout>
#include "names.h"
class exportFormatItem : public QWidget
{
        Q_OBJECT
private:
        QComboBox* Value, *Separator ;
        QPushButton* removeButton ;
        QHBoxLayout* layout ;
private slots:
        void remove() ;
public:
        exportFormatItem(const QStringList& values, QWidget* parent = 0);
        ~exportFormatItem();
        QString separator() ;
        spec::value value() ;
        void setValue(spec::value) ;
        void setSeparator(spec::separator) ;
        void setDataTypes(const QStringList& ds) ;
signals:
        void changed() ;
};
#endif
```

src/model/exportformatitem.cpp

```
#include "exportformatitem.h"
#include <QLabel>
exportFormatItem::exportFormatItem(const QStringList& values, QWidget* parent)
        : QWidget(parent)
{
        Value = new QComboBox() ;
        Separator = new QComboBox() ;
        removeButton = new QPushButton(QIcon::fromTheme("list-remove"), "") ;
        layout = new QHBoxLayout() ;
        QStringList separators ;
        separators << "none" << "space" << "tab" << "new␣line" ;
        Value->addItems(values) ;
        Separator->addItems(separators) ;
        layout->addWidget(new QLabel("value:")) ;
        layout->addWidget(Value) ;
        layout->addWidget(new QLabel("separator:")) ;
        layout->addWidget(Separator) ;
        layout->addStretch() ;
        layout->addWidget(removeButton) ;
        layout->setContentsMargins(0, 0, 0, 0) ;
        setLayout(layout) ;
        removeButton->setFixedHeight(16) ;
        removeButton->setFixedWidth(16) ;
        removeButton->setFlat(true) ;
        connect(removeButton, SIGNAL(clicked()), this, SLOT(remove())) ;
}
void exportFormatItem::remove()
{
        delete this ;
}
exportFormatItem::~exportFormatItem()
{
}
QString exportFormatItem::separator()
{
        QString retString ;
```

```
        switch((spec::separator) Separator->currentIndex())
        {
                case spec::space:
                        retString = "␣" ; break ;
                case spec::tab:
                        retString = '\t' ; break ;
                case spec::newline:
                        retString = '\n'; break ;
                default: ;
        }
        return retString ;
}
spec::value exportFormatItem::value()
{
        return (spec::value) Value->currentIndex() ;
}
void exportFormatItem::setValue(spec::value v)
{
        Value->setCurrentIndex(v) ;
}
void exportFormatItem::setSeparator(spec::separator s)
{
        Separator->setCurrentIndex(s) ;
}
void exportFormatItem::setDataTypes(const QStringList& ds)
{
        QString currentText = Value->currentText() ;
        if(!ds.contains(currentText))
        {
                remove() ;
                return ;
        }
        Value->clear();
        Value->addItems(ds) ;
        Value->setCurrentIndex(ds.indexOf(currentText));
}
```

src/model/exportlayoutitem.h

```
#ifndef EXPORTLAYOUTITEM_H
#define EXPORTLAYOUTITEM_H
#include <QWidget>
#include <QCheckBox>
#include <QComboBox>
#include <QLineEdit>
#include <QStringList>
#include <QHBoxLayout>
#include <QPushButton>
class exportLayoutItem : public QWidget
{
        Q_OBJECT
private:
        QCheckBox* isFreeForm ;
        QComboBox* descriptor ;
        QLineEdit* freeText ;
        QPushButton* removeButton ;
        QHBoxLayout* layout ;
private slots:
        void freeFormMode(int on = Qt::Unchecked) ;
        void remove() ;
public:
        exportLayoutItem(QStringList&, QWidget* parent = 0);
        bool isFreeText() ;
        QString text() ;
        void setDescriptors(const QStringList&) ;
        ~exportLayoutItem();
signals:
        void changed() ;
};
#endif
```

src/model/exportlayoutitem.cpp

```
#include "exportlayoutitem.h"
#include <QTextStream>
exportLayoutItem::exportLayoutItem(QStringList& descriptors, QWidget* parent)
        : QWidget(parent)
{
        isFreeForm = new QCheckBox("Free form string") ;
        descriptor = new QComboBox() ;
        freeText = new QLineEdit() ;
        removeButton = new QPushButton(QIcon::fromTheme("list-remove"), "") ;
        layout = new QHBoxLayout() ;
        connect(isFreeForm, SIGNAL(stateChanged(int)), this, SLOT(freeFormMode(int))) ;
        descriptor->addItems(descriptors) ;
        layout->addWidget(descriptor) ;
        layout->addWidget(freeText) ;
        layout->addWidget(isFreeForm) ;
        layout->addWidget(removeButton) ;
        layout->setContentsMargins(0, 0, 0, 0) ;
        setLayout(layout) ;
        isFreeForm->setFixedSize(isFreeForm->sizeHint()) ;
        descriptor->setFixedHeight(descriptor->sizeHint().height()) ;
        descriptor->setMaximumWidth(QWIDGETSIZE_MAX) ;
        freeText->setFixedHeight(descriptor->sizeHint().height()) ;
        freeText->setMaximumWidth(QWIDGETSIZE_MAX) ;
        removeButton->setFixedHeight(16) ;
        removeButton->setFixedWidth(16) ;
        removeButton->setFlat(true) ;
        freeFormMode(Qt::Unchecked) ;
        connect(removeButton, SIGNAL(clicked()), this, SLOT(remove())) ;
}
void exportLayoutItem::freeFormMode(int on)
{
        (on == Qt::Checked ? (QWidget*) descriptor : (QWidget*) freeText)->hide() ;
        (on == Qt::Checked ? (QWidget*) freeText : (QWidget*) descriptor)->show() ;
}
void exportLayoutItem::setDescriptors(const QStringList& ds)
{
        QString currentText = descriptor->currentText() ;
        if(!ds.contains(currentText) && !isFreeText())
        {
                remove();
                return ;
        }
        descriptor->clear();
        descriptor->addItems(ds) ;
        if(ds.contains(currentText))
                descriptor->setCurrentIndex(ds.indexOf(currentText)) ;
}
void exportLayoutItem::remove()
{
        delete this ;
}
exportLayoutItem::~exportLayoutItem()
{
}
bool exportLayoutItem::isFreeText()
{
        return isFreeForm->checkState() == Qt::Checked ;
}
QString exportLayoutItem::text()
{
        return (isFreeText() ? freeText->text() : descriptor->currentText()) ;
}
```

src/model/specdatapointfilter.h

```
#ifndef SPECDATAPOINTFILTER_H
#define SPECDATAPOINTFILTER_H
#include <QDataStream>
#include "specdatapoint.h"
class specDataPointFilter
{
        friend QDataStream& operator<< (QDataStream& out, const specDataPointFilter&) ;
        friend QDataStream& operator>> (QDataStream& in, specDataPointFilter&) ;
```

```
private:
        double offset, slope, factor, xshift ;
        int zeroMultiplications ;
public:
        explicit specDataPointFilter(double off = 0, double sl = 0, double scale = 1, double x = 0) ;
        void applyCorrection(specDataPoint&) const ;
        void reverseCorrection(specDataPoint&) const;
        double getOffset() const { return offset ;}
        double getSlope() const { return slope ;}
        double getXShift() const { return xshift ;}
        double getFactor() const { return factor ;}
        QString description() const ;
        specDataPointFilter& operator+= (const specDataPointFilter&) ;
        bool operator== (const specDataPointFilter&) const ;
        specDataPointFilter operator-() const ;
        bool valid() const ;
};
#endif
```

src/model/specdatapointfilter.cpp

```
#include "specdatapointfilter.h"
QDataStream& operator << (QDataStream& out, const specDataPointFilter& f)
{
        return out << f.offset
               << f.slope
               << f.factor
               << f.xshift
               << f.zeroMultiplications ;
}
QDataStream& operator >> (QDataStream& in, specDataPointFilter& f)
{
        return in >> f.offset
               >> f.slope
               >> f.factor
               >> f.xshift
               >> f.zeroMultiplications ;
}
specDataPointFilter& specDataPointFilter::operator += (const specDataPointFilter& other)
{
        xshift += other.xshift ;
        if(0 == other.factor)
                zeroMultiplications ++ ;
        else if(INFINITY == other.factor)
                zeroMultiplications = 0 ;
        else if(!isnan(other.factor))
        {
                slope *= other.factor ;
                factor *= other.factor ;
                offset *= other.factor ;
                offset += other.offset ;
                slope += other.slope ;
        }
        return *this;
}
specDataPointFilter specDataPointFilter::operator -() const
{
        if(factor == 0) return specDataPointFilter(offset,
                                        slope,
                                        INFINITY,
                                        -xshift) ;
        if(factor == INFINITY) return specDataPointFilter(offset,
                                                slope,
                                                0,
                                                -xshift) ;
        return specDataPointFilter(-factor / offset,
                                -factor / slope,
                                1 / factor,
                                -xshift) ;
}
void specDataPointFilter::applyCorrection(specDataPoint& point) const
{
        point.nu += xshift ;
        point.sig = zeroMultiplications ? 0 : point.sig * factor + offset + slope * point.nu ;
```

```
}
void specDataPointFilter::reverseCorrection(specDataPoint& point) const
{
        point.sig = (point.sig - offset - slope * point.nu) / factor ;
        point.nu = point.nu - xshift ;
}
specDataPointFilter::specDataPointFilter(double off, double sl, double scale, double x)
        : offset(off),
          slope(sl),
          factor(scale),
          xshift(x),
          zeroMultiplications(0)
{}
QString specDataPointFilter::description() const
{
        return QObject::tr("Offset: ") + QString::number(offset)
               + QObject::tr(", slope: ") + QString::number(slope)
               + QObject::tr(", scaling: ") + QString::number(factor)
               + QObject::tr(", x shift:") + QString::number(xshift) ;
}
bool specDataPointFilter::operator == (const specDataPointFilter& other) const
{
        return (factor == other.factor &&
                xshift == other.xshift &&
                offset == other.offset &&
                slope == other.slope) ;
}
bool specDataPointFilter::valid() const
{
        return isfinite(offset) &&
                        isfinite(slope) &&
                        isfinite(factor) &&
                        isfinite(xshift) ;
}
```

src/model/specdelegate.h

```
#ifndef SPECDELEGATE_H
#define SPECDELEGATE_H
#include <QItemDelegate>
#include <QMap>
class QStringList ;
class specDelegate : public QItemDelegate
{
        Q_OBJECT
        QMap<QString, QStringList*>* completerMap ;
        void syncCompleterMap(const QStringList&) const;
public:
        explicit specDelegate(QObject* parent = 0);
        ~specDelegate() ;
        QWidget* createEditor(QWidget* parent, const QStyleOptionViewItem& option, const QModelIndex►
            & index) const;
        void setEditorData(QWidget* editor, const QModelIndex& index) const;
        void setModelData(QWidget* editor, QAbstractItemModel* model, const QModelIndex& index) const;
        void updateEditorGeometry(QWidget* editor, const QStyleOptionViewItem& option, const ►
            QModelIndex& index) const;
        bool editorEvent(QEvent* event, QAbstractItemModel* model, const QStyleOptionViewItem& ►
            option, const QModelIndex& index) ;
};
#endif
```

src/model/specdelegate.cpp

```
#include "specdelegate.h"
#include <QModelIndex>
#include <QTextStream>
#include "textedit.h"
#include "names.h"
#include <QCompleter>
#include <QSet>
#include "specmodel.h"
specDelegate::specDelegate(QObject* parent)
```

```
        : QItemDelegate(parent),
          completerMap(new QMap<QString, QStringList*>())
{
}
specDelegate::~specDelegate()
{
        foreach(QStringList * list, completerMap->values())
        delete list ;
        delete completerMap ;
}
bool specDelegate::editorEvent(QEvent* event, QAbstractItemModel* model, const QStyleOptionViewItem&►
     option, const QModelIndex& index)
{
        return QItemDelegate::editorEvent(event, model, option, index) ;
}
void specDelegate::syncCompleterMap(const QStringList& descriptors) const
{
        QSet<QString> newDescriptors(descriptors.toSet()),
             currentDescriptors(completerMap->keys().toSet()) ;
        foreach(const QString & descriptor, currentDescriptors - newDescriptors)
        delete completerMap->take(descriptor) ;
        foreach(const QString & descriptor, newDescriptors - currentDescriptors)
        completerMap->insert(descriptor, new QStringList()) ;
}
QWidget* specDelegate::createEditor(QWidget* parent, const QStyleOptionViewItem& option, const ►
    QModelIndex& index) const
{
        Q_UNUSED(option)
        TextEdit* editor = new TextEdit(parent) ;
        const specModel* model = qobject_cast<const specModel*> (index.model()) ;
        if(model)
        {
                syncCompleterMap(model->descriptors()) ;
                QCompleter* completer = new QCompleter(* (completerMap->value(model->descriptors() [►
                     index.column()])), editor) ;
                completer->setModelSorting(QCompleter::CaseSensitivelySortedModel);
                editor->setCompleter(completer);
        }
        return editor ;
}
void specDelegate::setEditorData(QWidget* e, const QModelIndex& index) const
{
        QTextEdit* editor = qobject_cast<QTextEdit*> (e) ;
        if(!editor) return ;
        editor->setText(index.model()->data(index, Qt::EditRole).toString()) ;
}
void specDelegate::setModelData(QWidget* editor, QAbstractItemModel* model, const QModelIndex& index►
    ) const
{
        QTextEdit* ed = (QTextEdit*) editor ;
        QString text = ed->toPlainText() ;
        model->setData(index,
                       QList<QVariant>() << text << ed->textCursor().blockNumber(),
                       Qt::EditRole) ;
        specModel* sm = qobject_cast<specModel*> (model) ;
        if(sm)
        {
                QString descriptor = sm->descriptors() [index.column()] ;
                if(completerMap->contains(descriptor))
                {
                        QSet<QString> newWords = completerMap->value(descriptor)->toSet() ;
                        foreach(const QString & word, text.split(QRegExp("\\s+")))
                        newWords << word ;
                        QStringList newList = newWords.toList() ;
                        qSort(newList) ;
                        completerMap->value(descriptor)->swap(newList) ;
                }
        }
}
void specDelegate::updateEditorGeometry(QWidget* editor, const QStyleOptionViewItem& option, const ►
    QModelIndex& index) const
{
        QRect geom = option.rect ;
        if(!index.column())
                geom.setX(geom.x() + option.decorationSize.width() + 3) ;
```

```
        editor->setGeometry(geom);
}
```

src/model/specdescriptorcomparisoncriterion.h

```
#ifndef SPECDESCRIPTORCOMPARISONCRITERION_H
#define SPECDESCRIPTORCOMPARISONCRITERION_H
#include <QString>
#include <QList>
class specModelItem ;
class specDescriptorComparisonCriterion
{
private:
        double toleranceValue ;
        QString descriptorName ;
public:
        specDescriptorComparisonCriterion();
        explicit specDescriptorComparisonCriterion(const QString& descriptor, bool numeric = false) ;
        specDescriptorComparisonCriterion(const QString& descriptor, double tolerance) ;
        QString descriptor() const ;
        void setDescriptor(const QString& s) ;
        bool isNumeric() const ;
        void setNumeric(bool a = true) ;
        double tolerance() const ;
        void setTolerance(double d) ;
        bool itemsEqual(const specModelItem* a, const specModelItem* b) const ;
        typedef QList<specDescriptorComparisonCriterion> container ;
        static bool itemsEqual(specModelItem *a, specModelItem *b, const container& criteria) ;
};
#endif
```

src/model/specdescriptorcomparisoncriterion.cpp

```
#include "specdescriptorcomparisoncriterion.h"
#include <cmath>
#include "specmodelitem.h"
specDescriptorComparisonCriterion::specDescriptorComparisonCriterion()
{
        setNumeric(false) ;
}
specDescriptorComparisonCriterion::specDescriptorComparisonCriterion(const QString &descriptor, bool►
     numeric)
{
        setDescriptor(descriptor);
        setNumeric(numeric) ;
}
specDescriptorComparisonCriterion::specDescriptorComparisonCriterion(const QString &descriptor, ►
    double tol)
        : descriptorName(descriptor)
{
        setDescriptor(descriptor);
        setTolerance(tol);
}
QString specDescriptorComparisonCriterion::descriptor() const
{
        return descriptorName ;
}
void specDescriptorComparisonCriterion::setDescriptor(const QString &s)
{
        descriptorName = s ;
}
bool specDescriptorComparisonCriterion::isNumeric() const
{
        return !isnan(toleranceValue) ;
}
void specDescriptorComparisonCriterion::setNumeric(bool a)
{
        if (a != isNumeric())
                toleranceValue = a ? 0 : NAN ;
}
double specDescriptorComparisonCriterion::tolerance() const
{
```

```
        return toleranceValue ;
}
void specDescriptorComparisonCriterion::setTolerance(double d)
{
        toleranceValue = fabs(d) ;
}
bool specDescriptorComparisonCriterion::itemsEqual(const specModelItem *first, const specModelItem *►
    second) const
{
        if(!first || !second)
                return false ;
        if (first == second) return true ;
        if(isNumeric())
        {
                double a = first->descriptorValue(descriptorName),
                       b = second->descriptorValue(descriptorName) ;
                if(b - toleranceValue <= a && a <= b + toleranceValue)
                        return true ;
                if (isnan(a) && isnan(b)) return true ;
                return false ;
        }
        return first->descriptor(descriptorName, true)
                        == second->descriptor(descriptorName, true) ;
}
bool specDescriptorComparisonCriterion::itemsEqual(specModelItem *a, specModelItem *b, const ►
    container &criteria)
{
        foreach(const specDescriptorComparisonCriterion& criterion, criteria)
                if (!criterion.itemsEqual(a,b))
                        return false ;
        return true ;
}
```

src/model/specdescriptor.h

```
#ifndef SPECDESCRIPTOR_H
#define SPECDESCRIPTOR_H
#include <QDataStream>
#include <QString>
#include <names.h>
#include "specstreamable.h"
class specDescriptor : public specStreamable
{
private:
        QString contentValue ;
        int currentLine ;
        spec::descriptorFlags properties ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::descriptor ; }
public:
        specDescriptor(QString cont = "", spec::descriptorFlags prop = spec::def) ;
        specDescriptor(double d) ;
        double numericValue() const ;
        QString content(bool full = false) const;
        bool setContent(const QString&) ;
        bool setActiveLine(int) ;
        bool setContent(const double&) ;
        bool isNumeric() const;
        bool isEditable() const;
        spec::descriptorFlags flags() const ;
        void setFlags(spec::descriptorFlags) ;
        int activeLine() const ;
        specDescriptor& operator= (const double&) ;
        specDescriptor& operator= (const QString&) ;
};
#endif
```

src/model/specdescriptor.cpp

```
#include "specdescriptor.h"
#include <QRegExp>
```

```
#include <QStringList>
void specDescriptor::writeToStream(QDataStream& out) const
{
        out << contentValue << contentValue.split("\n") [currentLine] << (qint8) properties ;
}
void specDescriptor::readFromStream(QDataStream& in)
{
        qint8 prop ;
        QString cline ;
        in >> contentValue >> cline >> prop ;
        currentLine = contentValue.left(contentValue.indexOf(cline)).count("\n") ;
        properties = (spec::descriptorFlags) prop ;
}
specDescriptor::specDescriptor(QString cont, spec::descriptorFlags prop)
        : currentLine(0),
          properties(prop)
{
        (*this) = cont ;
}
specDescriptor::specDescriptor(double d)
        : currentLine(0),
          properties(spec::numeric)
{
        (*this) = d ;
}
double specDescriptor::numericValue() const
{ return contentValue.toDouble() ; }
QString specDescriptor::content(bool full) const
{
        if(full || (properties & spec::multiline))
                return contentValue ;
        else
                return contentValue.split("\n") [currentLine] ;
}
bool specDescriptor::setContent(const QString& string)
{
        if(isEditable())
        {
                (*this) = string ;
                return true ;
        }
        return false ;
}
bool specDescriptor::setActiveLine(int line)
{
        currentLine = qBound(0, line, contentValue.count("\n")) ;
        return true ;
}
int specDescriptor::activeLine() const
{
        return currentLine ;
}
bool specDescriptor::setContent(const double& number)
{
        if(isNumeric() && isEditable())
        {
                (*this) = number ;
                return true ;
        }
        return false ;
}
bool specDescriptor::isNumeric() const
{
        return properties & spec::numeric ;
}
bool specDescriptor::isEditable() const
{
        return properties & spec::editable ;
}
spec::descriptorFlags specDescriptor::flags() const
{
        return properties ;
}
specDescriptor& specDescriptor::operator= (const double& val)
{
```

```
        (*this) = QString::number(val) ;
        properties |= spec::numeric ;
        return *this ;
}
specDescriptor& specDescriptor::operator= (const QString& val)
{
        contentValue = val ;
        setActiveLine(currentLine) ;
        return *this ;
}
void specDescriptor::setFlags(spec::descriptorFlags f)
{
        properties = f ;
}
```

src/model/specfolderitem.h

```
#ifndef SPECFOLDERITEM_H
#define SPECFOLDERITEM_H
#include "specmodelitem.h"
#include <QList>
class specFolderItem : public specModelItem
{
private:
        QList<specModelItem*> ChildrenList ;
        void readFromStream(QDataStream& in) ;
        void writeToStream(QDataStream& out) const ;
        type typeId() const { return specStreamable::folder ; }
public:
        specFolderItem(specFolderItem* par = 0, QString description = "");
        ~specFolderItem();
        template <typename T>
        QVector<T> findDescendants() ;
        bool addChild(specModelItem*, QList<specModelItem*>::size_type position) ;
        bool addChildren(QList<specModelItem*> list, QList<specModelItem*>::size_type position = 0) ;
        void removeChild(specModelItem*) ;
        QList<specModelItem*>::size_type children() const;
        bool isEditable(QString key) const;
        void refreshPlotData() ;
        bool isFolder() const ;
        QStringList descriptorKeys() const ;
        QIcon decoration() const ;
        void buildTree(QStringList descriptors) ;
        specModelItem* child(QList<specModelItem*>::size_type i) const ;
        int childNo(specModelItem* child) const ;
        spec::descriptorFlags descriptorProperties(const QString& key) const ;
        void addDataFilter(const specDataPointFilter&) ;
        void exportData(const QList<QPair<bool, QString> >&, const QList<QPair<spec::value, QString>►
             >&, QTextStream&) ;
        void deleteDescriptor(const QString& key) ;
        void renameDescriptors(const QMap<QString, QString>& map);
        void dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const ;
        void restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key) ;
        QList<specModelItem*> childrenList() const ;
};
#endif
```

src/model/specfolderitem.cpp

```
#include "specfolderitem.h"
#include <QTextStream>
#include <QTime>
specFolderItem::specFolderItem(specFolderItem* par, QString description)
        : specModelItem(par, description)
{
}
specFolderItem::~specFolderItem()
{
        foreach(specModelItem * childPointer, ChildrenList)
        delete childPointer ;
        ChildrenList.clear() ;
}
```

```
bool specFolderItem::addChild(specModelItem* item, QList<specModelItem*>::size_type position)
{
        addChildren(QList<specModelItem*>() << item, position) ;
        return true ;
}
bool specFolderItem::addChildren(QList<specModelItem*> list, QList<specModelItem*>::size_type ►
    position)
{
        list.removeAll(0) ;
        for(int i = 0 ; i < list.size() ; i++)
        {
                list[i]->setParent(this) ;
                ChildrenList.insert(position + i, list[i]) ;
        }
        invalidate();
        return true ;
}
QList<specModelItem*>::size_type specFolderItem::children() const
{
        return ChildrenList.size() ;
}
bool specFolderItem::isFolder() const { return true ;}
QIcon specFolderItem::decoration() const { return QIcon::fromTheme("folder") ; }
bool specFolderItem::isEditable(QString key) const
{
        if(key == "")
                return specModelItem::isEditable(key) ;
        foreach(specModelItem * item, ChildrenList)
        if(item->isEditable(key))
                return true ;
        return false ;
}
void specFolderItem::refreshPlotData()
{
        QVector<double> x, y ;
        foreach(specModelItem * item, ChildrenList)
        {
                item->revalidate();
                for(size_t i = 0 ; i < item->dataSize() ; i++)
                {
                        x << item->sample(i).x() ;
                        y << item->sample(i).y() ;
                }
        }
        setSamples(x, y) ;
}
void specFolderItem::removeChild(specModelItem* child)
{
        ChildrenList.removeOne(child) ;
        invalidate();
}
void specFolderItem::readFromStream(QDataStream& in)
{
        foreach(specModelItem * child, ChildrenList)
        delete child ;
        quint64 numChildren ;
        specModelItem::readFromStream(in) ;
        in >> numChildren ;
        QList<specModelItem*> newChildren ;
        for(quint64 i = 0 ; i < numChildren ; ++i)
                newChildren << (specModelItem*) produceItem(in) ;
        newChildren.removeAll(0) ;
        addChildren(newChildren) ;
}
void specFolderItem::writeToStream(QDataStream& out) const
{
        specModelItem::writeToStream(out) ;
        out << (quint64) children() ;
        foreach(specModelItem * child, ChildrenList)
        out << *child ;
}
QStringList specFolderItem::descriptorKeys() const
{
        QStringList keys;
        keys << specModelItem::descriptorKeys() ;
```

```
        foreach(specModelItem * child, ChildrenList)
        {
                QStringList childKeys = child->descriptorKeys() ;
                for(QStringList::size_type i = 0 ; i < childKeys.size() ; i++)
92                      if(!keys.contains(childKeys[i]))
                                keys << childKeys[i] ;
        }
        return keys ;
}
97 specModelItem* specFolderItem::child(QList<specModelItem*>::size_type i) const
{ return 0 <= i && i < ChildrenList.size() ? ChildrenList[i] : NULL ; }
int specFolderItem::childNo(specModelItem* child) const
{ return ChildrenList.indexOf(child) ; }
spec::descriptorFlags specFolderItem::descriptorProperties(const QString& key) const
102 {
        if(key == "") return specModelItem::descriptorProperties(key) ;
        spec::descriptorFlags flags = spec::numeric | spec::editable ;
        foreach(specModelItem * child, ChildrenList)
        flags = (spec::descriptorFlags)(flags & child->descriptorProperties(key)) ;
107     return flags ;
}
void specFolderItem::addDataFilter(const specDataPointFilter& filter)
{
        foreach(specModelItem * child, ChildrenList)
112     child->addDataFilter(filter);
        invalidate();
}
void specFolderItem::exportData(const QList<QPair<bool, QString> >& headerFormat, const QList<QPair<►
    spec::value, QString> >& dataFormat, QTextStream& out)
{
117     revalidate();
        for(int i = 0 ; i < headerFormat.size() ; i++)
                out << (headerFormat[i].first ? headerFormat[i].second : this->descriptor(►
                    headerFormat[i].second)) ;
        out << endl ;
        foreach(specModelItem * child, ChildrenList)
122     child->exportData(headerFormat, dataFormat, out) ;
}
void specFolderItem::deleteDescriptor(const QString& descriptorName)
{
        foreach(specModelItem * child, ChildrenList)
127     child->deleteDescriptor(descriptorName) ;
}
void specFolderItem::renameDescriptors(const QMap<QString, QString>& map)
{
        foreach(specModelItem * child, ChildrenList)
132     child->renameDescriptors(map) ;
}
void specFolderItem::dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const
{
        specModelItem::dumpDescriptor(destination, key) ;
137     foreach(specModelItem * child, ChildrenList)
        child->dumpDescriptor(destination, key) ;
}
void specFolderItem::restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key)
{
142     specModelItem::restoreDescriptor(origin, key) ;
        foreach(specModelItem * child, ChildrenList)
        child->restoreDescriptor(origin, key) ;
}
QList<specModelItem*> specFolderItem::childrenList() const
147 {
        return ChildrenList ;
}
template<typename T>
QVector<T> specFolderItem::findDescendants()
152 {
        QVector<T> items = specModelItem::findDescendants<T>();
        for(int i = 0 ; i < children() ; ++i)
        {
                specModelItem* item = child(i) ;
157             if(T value = dynamic_cast<T>(item))
                        items << value ;
                items << item->findDescendants<T>() ;
        }
```

```
        return items ;
}
```

src/model/specgenealogy.h

```
#ifndef SPECGENEALOGY_H
#define SPECGENEALOGY_H
#include "specmodel.h"
#include <QVector>
#include "specmodelitem.h"
class specGenealogy : public specStreamable
{
        friend QTextStream& operator<< (QTextStream& out, const specGenealogy& g) ;
private:
        specModel* Model ;
        QVector<int> indexes ;
        specFolderItem* Parent ;
        QVector<specModelItem*> Items ;
        bool owning ;
        void getItemPointers() ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::genealogyId ; }
        specModelItem* factory(const type& t) const ;
        void clear() ;
public:
        specGenealogy(specModelItem* p, specModel* m) ;
        specGenealogy(QList<specModelItem*>& l, specModel* m) ;
        virtual ~specGenealogy() ;
        specGenealogy() ;
        void setModel(specModel* model) ;
        void takeItems() ;
        void returnItems() ;
        bool valid() const ;
        bool seekParent() ;
        int size() const { return Items.size() ; }
        void signalChange() const ;
        specModel* model() const ;
        specFolderItem* parent() const ;
        QVector<specModelItem*> items() ;
        QVector<specModelItem*> items() const ;
        specModelItem* firstItem() ;
        specModelItem* firstItem() const ;
        QModelIndex firstIndex() ;
        bool operator== (const specGenealogy& other) const ;
        bool operator< (const specGenealogy& other) const ;
        bool operator!= (const specGenealogy& other) const ;
        specGenealogy& operator= (specGenealogy& other) ;
        specGenealogy& operator= (const specGenealogy& other) ;
};
QTextStream& operator<< (QTextStream& out, const specGenealogy& g) ;
#endif
```

src/model/specgenealogy.cpp

```
#include "specgenealogy.h"
#include "specfolderitem.h"
specGenealogy::specGenealogy(specModelItem* p, specModel* m)
        : Model(m),
          owning(false)
{
        Items << p ;
        indexes = Model->hierarchy(p) ;
        Parent = p->parent() ;
}
specGenealogy::specGenealogy(QList<specModelItem*>& l, specModel* m)
        : Model(m),
          owning(false)
{
        if(l.empty()) return ;
        Parent = l.first()->parent() ;
        indexes = Model->hierarchy(l.first()) ;
```

```
        int row = Parent->childNo(l.first()) ;
        QList<specModelItem*>::iterator it = l.begin() ;
        while(it != l.end()
                && (*it)->parent() == Parent
                && Parent->childNo(*it) == row)
        {
                Items << *it ;
                ++it ;
                ++row ;
        }
        l.erase(l.begin(), it) ;
}
bool specGenealogy::valid() const
{
        return Model && Parent && !indexes.isEmpty() && !Items.isEmpty() && Items.first() ;
}
void specGenealogy::takeItems()
{
        if(owning) return ;
        if(!Parent)
                seekParent() ;
        if(!valid()) return ;
        foreach(specModelItem * item, Items)
        if(item) item->setParent(0) ;
        owning = true ;
}
void specGenealogy::returnItems()
{
        if(!owning) return ;
        if(!Parent)
                seekParent() ;
        if(!valid()) return ;
        Parent->addChildren(Items.toList(), indexes.first()) ;
        owning = false ;
}
specModel* specGenealogy::model() const
{
        return Model ;
}
specFolderItem* specGenealogy::parent() const
{
        return Parent ;
}
void specGenealogy::writeToStream(QDataStream& out) const
{
        out << indexes
            << qint8(owning)
            << qint32(Items.size())
            << quint64(Model) ;
        if(owning)
        {
                for(int i = 0 ; i < Items.size() ; ++i)
                        out << * (Items[i]) ;
        }
}
void specGenealogy::readFromStream(QDataStream& in)
{
        clear();
        qint8 own ;
        qint32 itemCount ;
        quint64 m ;
        in >> indexes >> own >> itemCount >> m;
        owning = own ;
        Model = 0 ;
        Items.fill(0, itemCount);
        if(owning)
                for(int i = 0 ; i < itemCount ; ++i)
                        Items[i] = (specModelItem*) produceItem(in) ;
}
specModelItem* specGenealogy::factory(const type& t) const
{
        return specModelItem::itemFactory(t) ;
}
specGenealogy::specGenealogy()
        : Model(0),
```

```
93            Parent(0),
              owning(false)
{
}
bool specGenealogy::seekParent()
98 {
        if(!Model) return false ;
        specModelItem* item = Model->itemPointer(indexes) ;
        if(!item) return false ;
        Parent = item->parent() ;
103     if(Parent && !owning)
                getItemPointers();
        return Parent ;
}
void specGenealogy::setModel(specModel* model)
108 {
        Model = model ;
}
void specGenealogy::getItemPointers()
{
113     for(int i = 0 ; i < Items.size() ; ++i)
                Items[i] = Parent->child(i + indexes.first()) ;
}
void specGenealogy::clear()
{
118     indexes.clear();
        if(owning)
                foreach(specModelItem * item, Items)
                delete item ;
}
123 specGenealogy::~specGenealogy()
{
        clear() ;
}
QVector<specModelItem*> specGenealogy::items()
128 {
        if(!Parent) seekParent() ;
        if(!valid()) return QVector<specModelItem*>() ;
        return Items ;
}
133 QVector<specModelItem*> specGenealogy::items() const
{
        if(!valid()) return QVector<specModelItem*>() ;
        return Items ;
}
138 specModelItem* specGenealogy::firstItem()
{
        if(!Parent)
                seekParent() ;
        if(!valid()) return 0 ;
143     if(Items.isEmpty()) return 0 ;
        return Items.first() ;
}
specModelItem* specGenealogy::firstItem() const
{
148     if(!valid()) return 0 ;
        if(Items.isEmpty()) return 0 ;
        return Items.first() ;
}
QModelIndex specGenealogy::firstIndex()
153 {
        return model()->index(firstItem()) ;
}
bool specGenealogy::operator == (const specGenealogy& other) const
{
158     bool returnValue = true ;
        if(other.Items.size() != Items.size()) return false ;
        for(int i = 0 ; i < Items.size() ; ++i)
                returnValue = returnValue && (Items[i] == other.Items[i]) ;
        return returnValue ;
163 }
bool specGenealogy::operator < (const specGenealogy& other) const
{
        return specModel::lessThanHierarchies(indexes, other.indexes) ;
}
```

```
bool specGenealogy::operator != (const specGenealogy& other) const
{
        return !(*this == other) ;
}
specGenealogy& specGenealogy::operator = (const specGenealogy& other)
{
        clear() ;
        Model = other.Model ;
        indexes = other.indexes ;
        Parent = other.Parent ;
        Items = other.Items ;
        owning = false;
        return *this ;
}
specGenealogy& specGenealogy::operator = (specGenealogy& other)
{
        *this = * (const_cast<const specGenealogy*>(&other)) ;
        owning = other.owning ;
        other.owning = false ;
        return *this ;
}
void specGenealogy::signalChange() const
{
        if(!Model) return ;
        if(Items.isEmpty()) return ;
        if(owning) return ;
        Model->signalChanged(Model->index(Items.first()), Model->index(Items.last(), Model->►
            descriptors().size() - 1));
}
QTextStream& operator<< (QTextStream& out, const specGenealogy& g)
{
        QString result = "Genealogy␣";
        foreach(const int & i, g.indexes)
        result += QString::number(i) + "/" ;
        result += "␣size:␣" + QString::number(g.Items.size()) ;
        return out << result ;
}
```

src/model/specgenericmimeconverter.h

```
#ifndef SPECGENERICMIMECONVERTER_H
#define SPECGENERICMIMECONVERTER_H
#include "specmimeconverter.h"
#include "specstreamable.h"
#include "specmodelitem.h"
class specGenericMimeConverter : public specMimeConverter, private specStreamable
{
        Q_OBJECT
private:
        specModelItem* factory(const type& t) const ;
        void writeToStream(QDataStream& out) const { Q_UNUSED(out) }
        void readFromStream(QDataStream& in) { Q_UNUSED(in) }
        specStreamable::type typeId() const { return specStreamable::none ;}
        QString ownType() const ;
public:
        explicit specGenericMimeConverter(QObject* parent = 0);
        QList<specModelItem*> importData(const QMimeData*);
        void exportData(QList<specModelItem*>&, QMimeData*) ;
        QStringList importableTypes() const ;
};
#endif
```

src/model/specgenericmimeconverter.cpp

```
#include "specgenericmimeconverter.h"
#include "specmetamodel.h"
#include "speclogmodel.h"
#include "specmodel.h"
#include <QMimeData>
specGenericMimeConverter::specGenericMimeConverter(QObject* parent)
        : specMimeConverter(parent)
{
```

```
}
specModelItem* specGenericMimeConverter::factory(const type& t) const
{
        return specModelItem::itemFactory(t) ;
}
QList<specModelItem*> specGenericMimeConverter::importData(const QMimeData* data)
{
        QList<specModelItem*> items ;
        QByteArray ba(data->data(ownType())) ;
        QDataStream in(&ba, QIODevice::ReadOnly) ;
        while(!in.atEnd())
                items << (specModelItem*) produceItem(in) ;
        items.removeAll(0) ;
        return items ;
}
void specGenericMimeConverter::exportData(QList<specModelItem*>& items, QMimeData* data)
{
        QByteArray ba ;
        QDataStream out(&ba, QIODevice::WriteOnly) ;
        foreach(specModelItem * item, items)
        out << *item ;
        data->setData(ownType(), ba) ;
}
QString specGenericMimeConverter::ownType() const
{
        specModel* model = 0 ;
        if((model = qobject_cast<specLogModel*> (parent())))
                return "application/spec.log.items" ;
        if((model = qobject_cast<specMetaModel*> (parent())))
                return "application/spec.meta.items" ;
        if((model = qobject_cast<specModel*> (parent())))
                return "application/spec.spectral.items" ;
        return QString() ;
}
QStringList specGenericMimeConverter::importableTypes() const
{
        return specMimeConverter::importableTypes() << ownType() ;
}
```

src/model/specmimeconverter.h

```
#ifndef SPECMIMECONVERTER_H
#define SPECMIMECONVERTER_H
#include <QObject>
class QMimeData ;
class specModelItem ;
class specMimeConverter : public QObject
{
        Q_OBJECT
public:
        explicit specMimeConverter(QObject* parent = 0) ;
        virtual QList<specModelItem*> importData(const QMimeData*) = 0 ;
        virtual void exportData(QList<specModelItem*>&, QMimeData*) = 0 ;
        virtual bool canImport(const QStringList&) ;
        virtual bool canImport(const QMimeData*) ;
        virtual QStringList importableTypes() const ;
};
#endif
```

src/model/specmimeconverter.cpp

```
#include "specmimeconverter.h"
#include <QMimeData>
#include <QStringList>
specMimeConverter::specMimeConverter(QObject* parent)
        : QObject(parent)
{
}
bool specMimeConverter::canImport(const QMimeData* data)
{
        return canImport(data->formats()) ;
}
```

```
bool specMimeConverter::canImport(const QStringList& types)
{
        QStringList importable = importableTypes() ;
        foreach(const QString & type, types)
        if(importable.contains(type)) return true ;
        return false ;
}
QStringList specMimeConverter::importableTypes() const
{
        return QStringList() ;
}
```

src/model/specmimefileimporter.h

```
#ifndef SPECMIMEFILEIMPORTER_H
#define SPECMIMEFILEIMPORTER_H
#include "specmimeconverter.h"
#include "names.h"
class specMimeFileImporter : public specMimeConverter
{
        Q_OBJECT
private:
public:
        explicit specMimeFileImporter(QObject* parent = 0);
        QList<specModelItem*> importData(const QMimeData*) ;
        void exportData(QList<specModelItem*>&, QMimeData*) ;
        QStringList importableTypes() const ;
signals:
public slots:
};
#endif
```

src/model/specmimefileimporter.cpp

```
#include "specmimefileimporter.h"
#include <QFile>
#include <QMimeData>
#include "utility-functions.h"
#include "specmodel.h"
#include <QMessageBox>
#include <QUrl>
specMimeFileImporter::specMimeFileImporter(QObject* parent) :
        specMimeConverter(parent)
{
}
void specMimeFileImporter::exportData(QList<specModelItem*>& l, QMimeData* d)
{
        Q_UNUSED(l)
        Q_UNUSED(d)
        return ;
}
QStringList specMimeFileImporter::importableTypes() const
{
        return QStringList() << "text/uri-list" ;
}
QList<specModelItem*> specMimeFileImporter::importData(const QMimeData* mime)
{
        specModel* model = qobject_cast<specModel*> (parent()) ;
        QStringList failed ;
        QList<specModelItem*> importedItems ;
        if(!model) return importedItems ;
        foreach(const QUrl & fileUrl, mime->urls())
        {
                QString filename = fileUrl.toLocalFile() ;
                QFile file(filename) ;
                specFileImportFunction fileImportFunction = fileFilter(filename) ;
                if(!model->acceptableImportFunctions().contains(fileImportFunction))
                {
                        failed << filename ;
                        break;
                }
                file.open(QFile::ReadOnly | QFile::Text) ;
```

```
                importedItems << fileImportFunction(file) ;
        }
        if(!failed.isEmpty())
                QMessageBox::warning(0, tr("Failed␣to␣import"),
                                     tr("These␣files␣could␣not␣be␣imported␣here:\n")
                                     + failed.join("\n")) ;
        return importedItems ;
}
```

src/model/specmimetextexporter.h

```
#ifndef SPECMIMETEXTEXPORTER_H
#define SPECMIMETEXTEXPORTER_H
#include "specmimeconverter.h"
class QTextStream ;
class specMimeTextExporter : public specMimeConverter
{
        Q_OBJECT
private:
        void writeItem(specModelItem*, QTextStream& out) ;
public:
        explicit specMimeTextExporter(QObject* parent = 0);
        virtual QList<specModelItem*> importData(const QMimeData* data) ;
        void exportData(QList<specModelItem*>&, QMimeData*) ;
};
#endif
```

src/model/specmimetextexporter.cpp

```
#include "specmimetextexporter.h"
#include <QTextStream>
#include "specmodelitem.h"
#include "specfolderitem.h"
#include <QDoubleValidator>
#include <QMimeData>
specMimeTextExporter::specMimeTextExporter(QObject* parent) :
        specMimeConverter(parent)
{
}
void specMimeTextExporter::exportData(QList<specModelItem*>& items, QMimeData* data)
{
        QString text ;
        QTextStream out(&text, QIODevice::WriteOnly) ;
        foreach(specModelItem * item, items)
        writeItem(item, out) ;
        data->setText(text) ;
}
void specMimeTextExporter::writeItem(specModelItem* item, QTextStream& out)
{
        QList<QPair<spec::value, QString> > dataTypeList ;
        dataTypeList << QPair<spec::value, QString> (spec::wavenumber, "\t")
                     << QPair<spec::value, QString> (spec::signal, "\t")
                     << QPair<spec::value, QString> (spec::maxInt, "\n") ;
        foreach(QString descriptor, item->descriptorKeys())
        {
                QString padding(descriptor.length(), '␣') ;
                QStringList content = (QString("#␣") + descriptor + (descriptor == "" ? "" : ":␣") +
                                       item->descriptor(descriptor, true)).split("\n") ;
                for(QStringList::iterator i = content.begin() + 1 ; i != content.end() ; ++i)
                        i->prepend("#␣␣␣" + padding) ;
                out << content.join("\n") << endl ;
        }
        if(item->isFolder())
        {
                for(int i = 0 ; i < item->children() ; ++i)
                {
                        out << "#␣Child␣item" << endl ;
                        writeItem(((specFolderItem*) item)->child(i), out) ;
                }
        }
        else
                item->exportData(QList<QPair<bool, QString> >(), dataTypeList , out);
```

```
        out << endl ;
}
QList<specModelItem*> specMimeTextExporter::importData(const QMimeData* data)
{
        Q_UNUSED(data) ;
        return QList<specModelItem*>() ;
}
```

src/model/specmodel.h

```
#ifndef SPECMODEL_H
#define SPECMODEL_H
#include <QAbstractItemModel>
#include <QList>
#include <QStringList>
#include "specstreamable.h"
#include "names.h"
#include "specfolderitem.h"
class specModel ;
class specMetaModel ;
class specModelItem ;
class specActionLibrary ;
class specMimeConverter ;
class specCanvasItem ;
class QValidator ;
QDataStream& operator<< (QDataStream&, specModel&);
QDataStream& operator>> (QDataStream& in, specModel& model) ;
class specModel : public QAbstractItemModel, public specStreamable
{
        Q_OBJECT
private:
        specModelItem* root ;
        QStringList Descriptors ;
        QList<spec::descriptorFlags> DescriptorProperties ;
        void insertFromStream(QDataStream& stream, const QModelIndex& parent, int row) ;
        bool itemsAreEqual(QModelIndex& first, QModelIndex& second, const QList<QPair<QStringList::►
             size_type, double> >& criteria) ;
        QModelIndexList merge(QModelIndexList& list, const QList<QPair<QStringList::size_type, ►
             double> >& criteria) ;
        QModelIndexList allChildren(const QModelIndex&) const ;
        QMap<double, double> subMap ;
        bool internalDrop ;
        specActionLibrary* dropBuddy ;
        int dontDelete ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::model ; }
        QList<specMimeConverter*> mimeConverters() const ;
        specMetaModel* metaModel ;
        void checkForNewDescriptors(const QList<specModelItem*>& list, const QModelIndex& parent) ;
        bool resetPending ;
public:
        virtual QStringList dataTypes() const ;
        specModel(QObject* par = 0) ;
        ~specModel() ;
        QStringList mimeTypes() const ;
        void setMetaModel(specMetaModel*) ;
        specMetaModel* getMetaModel() const ;
        specModelItem* itemPointer(const QModelIndex&) const;
        specModelItem* itemPointer(const QVector<int>&) const;
        QModelIndex index(const QVector<int>&, int column = 0) const ;
        QModelIndex index(specModelItem*, int column = 0) const ;
        static QVector<int> hierarchy(specModelItem*) ;
        static QVector<int> hierarchy(const QModelIndex&) ;
        static bool lessThanItemPointer(specModelItem*, specModelItem*) ;
        static bool lessThanHierarchies(const QVector<int>& a, const QVector<int>& b) ;
        static bool lessThanIndex(const QModelIndex&, const QModelIndex&) ;
        QList<specModelItem*> pointerList(const QModelIndexList&) const ;
        QModelIndexList indexList(const QList<specModelItem*>&) const ;
        bool contains(specModelItem*) const ;
        template<class T>
        QModelIndexList indexList(const QList<T*>& pointers) const
        {
                QModelIndexList returnList ;
```

```
                foreach(T * tpointer, pointers)
                {
                        specModelItem* pointer = dynamic_cast<specModelItem*>(tpointer) ;
                        if(pointer)
                                returnList << index(pointer) ;
                }
                return returnList ;
        }
        bool isFolder(const QModelIndex&) const ;
        static void eliminateChildren(QModelIndexList&);
        static void eliminateChildren(QList<specModelItem*>&);
        static QList<specModelItem*> expandFolders(const QList<specModelItem*>&, bool retain = false) ;
        virtual bool insertItems(QList<specModelItem*> list, QModelIndex parent, int row = 0) ;
        void fillSubMap(const QModelIndexList&) ;
        void applySubMap(const QModelIndexList&) ;
        const QStringList& descriptors() const ;
        const QList<spec::descriptorFlags>& descriptorProperties() const;
        void setDescriptorProperties(spec::descriptorFlags flags, const QString& key) ;
        void renameDescriptors(const QMap<QString, QString>&) ;
        void deleteDescriptor(const QString&) ;
        void dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const ;
        void restoreDescriptor(qint16 position, spec::descriptorFlags flags, QListIterator<specDescriptor>& origin, const QString& key) ;
        int columnCount(const QModelIndex& parent) const ;
        int rowCount(const QModelIndex&) const ;
        bool removeRows(int position, int rows, const QModelIndex& parent = QModelIndex());
        bool setHeaderData(int section, Qt::Orientation orientation, const QVariant& value, int role = Qt::EditRole);
        bool insertColumns(int column, int count, const QModelIndex& parent = QModelIndex());
        bool removeColumns(int column, int count, const QModelIndex& parent = QModelIndex());
        QModelIndex parent(const QModelIndex& index) const ;
        QModelIndex index(int row, int column, const QModelIndex& parent = QModelIndex()) const ;
        Qt::ItemFlags flags(const QModelIndex& index) const;
        QVariant data(const QModelIndex& index, int role) const;
        bool setData(const QModelIndex& index, const QVariant& value, int role = Qt::EditRole);
        QVariant headerData(int section, Qt::Orientation orientation,
                            int role = Qt::DisplayRole) const;
        void setInternalDrop(bool) ;
        void setDropBuddy(specActionLibrary*) ;
        Qt::DropActions supportedDropActions() const ;
        QMimeData* mimeData(const QModelIndexList&) const;
        bool dropMimeData(const QMimeData*, Qt::DropAction, int row, int column, const QModelIndex& parent) ;
        bool mimeAcceptable(const QMimeData*) const ;
        friend QDataStream& operator<< (QDataStream&, specModel&);
        friend QDataStream& operator>> (QDataStream&, specModel&);
        void signalBeginReset() ;
        void signalChanged(const QModelIndex& index) ;
        void signalChanged(QModelIndex originalBegin, QModelIndex originalEnd) ;
        virtual QList<specFileImportFunction> acceptableImportFunctions() const ;
        virtual QValidator* createValidator(const QModelIndex&) const ;
private slots:
        void signalEndReset() ;
public slots:
        void svgMoved(specCanvasItem*, int, double, double) ;
};
#endif
```

src/model/specmodel.cpp

```
#include "specmodel.h"
#include <QString>
#include <QMimeData>
#include "exportdialog.h"
#include "specmovecommand.h"
#include "specaddfoldercommand.h"
#include "speceditdescriptorcommand.h"
#include "specresizesvgcommand.h"
#include "specfolderitem.h"
#include "specactionlibrary.h"
#include "specmimeconverter.h"
#include "specfolderitem.h"
#include "utility-functions.h"
```

```
#include <QtAlgorithms>
bool specModel::lessThanItemPointer(specModelItem* a, specModelItem* b)
{
        return lessThanHierarchies(hierarchy(a), hierarchy(b)) ;
}
bool specModel::lessThanHierarchies(const QVector<int> &aHierarchy, const QVector<int> &bHierarchy)
{
        int i = 0, n = qMin(aHierarchy.size(), bHierarchy.size()) ;
        int haSize = aHierarchy.size() - 1, hbSize = bHierarchy.size() - 1 ;
        while(i < n && aHierarchy[haSize] == bHierarchy[hbSize])
        {
                ++i ;
                --haSize ;
                --hbSize ;
        }
        if(i == n) return aHierarchy.size() < bHierarchy.size() ;
        return aHierarchy[haSize] < bHierarchy[hbSize] ;
}
bool specModel::itemsAreEqual(QModelIndex& first, QModelIndex& second, const QList<QPair<QStringList►
    ::size_type, double> >& criteria)
{
        if(!first.isValid() || !second.isValid())
                return false ;
        bool equal = true ;
        for(QList<QPair<QStringList::size_type, double> >::size_type i = 0 ; i < criteria.size() ; i►
            ++)
        {
                QStringList::size_type descriptor = criteria[i].first ;
                double tolerance = criteria[i].second ;
                if(DescriptorProperties[descriptor] & spec::numeric)
                {
                        double a = itemPointer(first)->descriptor(Descriptors[descriptor]).toDouble(),
                               b = itemPointer(second)->descriptor(Descriptors[descriptor]).toDouble►
                                   () ;
                        equal &= b - tolerance <= a && a <= b + tolerance ;
                }
                else
                        equal &= itemPointer(first)->descriptor(Descriptors[descriptor]) ==
                                  itemPointer(second)->descriptor(Descriptors[descriptor]) ;
        }
        return equal ;
}
QStringList specModel::dataTypes() const
{
        return QStringList() << "wavenumber" << "signal" << "maximum␣intensity" ;
}
QModelIndexList ancestry(const QModelIndex& index)
{ return index.parent().isValid() ? (QModelIndexList() << ancestry(index.parent()) << index.parent()►
    ) : QModelIndexList() ; }
bool lessThanIndex(const QModelIndex& one, const QModelIndex& two)
{
        if(one.parent() == two.parent())
                return one.row() < two.row() ;
        int levelOne = ancestry(one).size(), levelTwo = ancestry(two).size() ;
        if(levelOne == levelTwo)
                return lessThanIndex(one.parent(), two.parent()) ;
        return levelOne < levelTwo ? lessThanIndex(one, two.parent()) :
               lessThanIndex(one.parent(), two) ;
}
QModelIndexList specModel::allChildren(const QModelIndex& parent) const
{
        if(!parent.isValid() || !isFolder(parent))
                return QModelIndexList() ;
        QModelIndexList list ;
        for(int i = 0 ; i < rowCount(parent) ; i++)
                list << this->index(i, 0, parent) ;
        return list ;
}
const QStringList& specModel::descriptors() const
{
        return Descriptors ;
}
const QList<spec::descriptorFlags>& specModel::descriptorProperties() const
{
        return DescriptorProperties ;
```

```
}
bool specModel::isFolder(const QModelIndex& index) const
{ return itemPointer(index)->isFolder() ;}
void specModel::writeToStream(QDataStream& out) const
{
        out << Descriptors ;
        foreach(spec::descriptorFlags flag, DescriptorProperties)
        out << quint8(flag) ;
        out << *root ;
}
void specModel::readFromStream(QDataStream& in)
{
        Descriptors.clear();
        DescriptorProperties.clear();
        delete root ;
        root = new specFolderItem ;
        in >> Descriptors ;
        quint8 flag ;
        for(int i = 0 ; i < Descriptors.size() ; ++i)
        {
                in >> flag ;
                DescriptorProperties << QFlag(flag) ;
        }
        in >> *root ;
}
specModelItem* specModel::itemPointer(const QModelIndex& index) const
{ return (index.isValid() && index.internalPointer() != 0 ? (specModelItem*) index.internalPointer()►
     : root) ; }
specModel::specModel(QObject* par)
        : QAbstractItemModel(par),
          internalDrop(false),
          dropBuddy(0),
          dontDelete(0),
          metaModel(0),
          resetPending(false)
{
        root = new specFolderItem ;
        Descriptors += "" ;
        DescriptorProperties += spec::editable ;
        setObjectName("mainModel");
}
specModel::~specModel()
{
        delete root ;
}
bool specModel::setHeaderData(int section, Qt::Orientation orientation, const QVariant& value, int ►
    role)
{
        Q_UNUSED(orientation)
        if(role == Qt::EditRole)
        {
                Descriptors.replace(section, value.toString()) ;
                emit headerDataChanged(Qt::Horizontal, section, section) ;
                emit headerDataChanged(Qt::Vertical , section, section) ;
                return true ;
        }
        if(role == spec::descriptorPropertyRole)
        {
                DescriptorProperties[section] = (spec::descriptorFlags) value.toInt() ;
                return true ;
        }
        return false;
}
bool specModel::insertColumns(int column, int count, const QModelIndex& parent)
{
        if (!count) return true ;
        emit beginInsertColumns(parent, column, column + count - 1) ;
        for(QStringList::size_type i = 0 ; i < count ; i++)
        {
                Descriptors.insert(column + i, "") ;
                DescriptorProperties.insert(column + i, spec::def) ;
        }
        emit endInsertColumns() ;
        return true ;
}
```

```
bool specModel::removeColumns(int column, int count, const QModelIndex& parent)
{
        emit beginRemoveColumns(parent, column, column + count - 1) ;
        for(QStringList::size_type i = 0 ; i < count ; i++)
        {
                Descriptors.removeAt(column) ;
                DescriptorProperties.removeAt(column) ;
        }
        emit endRemoveColumns() ;
        return true ;
}
void specModel::checkForNewDescriptors(const QList<specModelItem*>& list, const QModelIndex& parent)
{
        QMap<QString, spec::descriptorFlags> newDescriptors ;
        foreach(specModelItem * pointer, list)
                foreach(const QString & descriptor, pointer->descriptorKeys())
                        if(!Descriptors.contains(descriptor))
                                newDescriptors[descriptor] = pointer->descriptorProperties(►
                                    descriptor) ;
        int col = columnCount(parent) ;
        insertColumns(columnCount(parent), newDescriptors.size(), parent) ;
        foreach (const QString& newDescriptor, newDescriptors.keys())
        {
                setHeaderData(col, Qt::Horizontal, newDescriptor) ;
                setHeaderData(col, Qt::Horizontal, (int) newDescriptors[newDescriptor], spec::►
                    descriptorPropertyRole) ;
                ++ col ;
        }
}
bool specModel::insertItems(QList<specModelItem*> list, QModelIndex parent, int row)
{
        row = row < rowCount(parent) ? row : rowCount(parent) ;
        emit layoutAboutToBeChanged() ;
        checkForNewDescriptors(list, parent) ;
        beginInsertRows(parent, row, row + list.size() - 1);
        bool retVal = itemPointer(parent)->addChildren(list, row) ;
        endInsertRows();
        emit layoutChanged() ;
        return retVal ;
}
int specModel::columnCount(const QModelIndex& parent) const
{
        Q_UNUSED(parent)
        return Descriptors.size() ;
}
int specModel::rowCount(const QModelIndex& parent) const
{
        return itemPointer(parent)->children() ;
}
QModelIndex specModel::parent(const QModelIndex& index) const
{
        specFolderItem* pointer = itemPointer(index)->parent() ;
        if(pointer && pointer != root)
                return createIndex(pointer->parent()->childNo(pointer), 0, pointer);
        return QModelIndex() ;
}
QModelIndex specModel::index(int row, int column, const QModelIndex& parent) const
{
        specModelItem* pointer = itemPointer(parent)->isFolder() ?
                                 ((specFolderItem*) itemPointer(parent))->child(row) : 0 ;
        return pointer ? createIndex(row, column, pointer) : QModelIndex() ;
}
Qt::ItemFlags specModel::flags(const QModelIndex& index) const
{
        if(!index.isValid())
                return Qt::ItemIsEnabled | Qt::ItemIsDropEnabled ;
        Qt::ItemFlags defaultFlags = QAbstractItemModel::flags(index) | (itemPointer(index)->►
            isFolder() ? Qt::ItemIsDragEnabled | Qt::ItemIsDropEnabled : Qt::ItemIsDragEnabled);
        if(itemPointer(index)->isEditable(Descriptors[index.column()]))
                return defaultFlags | Qt::ItemIsEditable ;
        return defaultFlags ;
}
QVariant specModel::data(const QModelIndex& index, int role) const
{
        if(!index.isValid() || index.row() >= rowCount(parent(index)))
```

```
                return QVariant();
        specModelItem* pointer = itemPointer(index) ;
        switch(role)
        {
                case Qt::DecorationRole :
                        return pointer->indicator(Descriptors[index.column()]) ;
                case Qt::DisplayRole :
                        if(index.column() < columnCount(index))
                                return pointer->descriptor(Descriptors[index.column()]) ;
                        return "" ;
                case Qt::ForegroundRole :
                        return pointer->pen().color() ;
                case Qt::EditRole :
                        return pointer->editDescriptor(Descriptors[index.column()]) ;
                case Qt::ToolTipRole :
                        return pointer->toolTip(Descriptors[index.column()]) ;
        }
        return QVariant() ;
}
QList<specMimeConverter*> specModel::mimeConverters() const
{
        return findChildren<specMimeConverter*>() ;
}
bool specModel::setData(const QModelIndex& index, const QVariant& value, int role)
{
        bool changed = false ;
        if(index.isValid())
        {
                QString desc = Descriptors[index.column()] ;
                specModelItem* pointer = itemPointer(index) ;
                if(role == Qt::EditRole)
                {
                        if(!(pointer->descriptorProperties(desc) & spec::editable)) return false ;
                        changed = true ;
                        specEditDescriptorCommand* command = new specEditDescriptorCommand ;
                        command->setParentObject(this) ;
                        if(value.canConvert(QVariant::List))
                        {
                                QList<QVariant> list = value.toList() ;
                                if(list.isEmpty())
                                {
                                        delete command ;
                                        return false ;
                                }
                                command->setItem(pointer, desc, list[0].toString(), list[1].toInt()) ;
                        }
                        else
                                command->setItem(pointer, desc, value.toString()) ;
                        dropBuddy->push(command) ;
                }
                else if(role == Qt::ForegroundRole)
                {
                        itemPointer(index)->setPen(value.value<QPen>()) ;
                        changed = true ;
                }
                else if(role == 33)
                {
                        changed = true ;
                        itemPointer(index)->mergePlotData = value.toBool() ;
                }
        }
        if(changed) emit dataChanged(index, index) ;
        return changed ;
}
QVariant specModel::headerData(int section, Qt::Orientation orientation,
                               int role) const
{
        Q_UNUSED(orientation)
        if(role == Qt::DisplayRole)
                return Descriptors[section] ;
        if(role == spec::descriptorPropertyRole)
                return (int) DescriptorProperties[section] ;
        return QVariant();
}
bool specModel::removeRows(int position, int rows, const QModelIndex& parent)
```

```
{
        Q_UNUSED(position)
        Q_UNUSED(rows)
        Q_UNUSED(parent)
        if(position < 0 || rows < 1) return false ;
        if(dontDelete)
        {
                dontDelete -- ;
                return true ;
        }
        if(dropBuddy) dropBuddy->deleteInternally(this) ;
        return true ;
}
void specModel::setInternalDrop(bool val)
{
        internalDrop = val ;
}
Qt::DropActions specModel::supportedDropActions() const
{ return Qt::CopyAction | Qt::MoveAction | Qt::LinkAction ; }
void specModel::eliminateChildren(QModelIndexList& list)
{
        for(QModelIndexList::size_type j = 0 ; j < list.size() ; j++)
                if(list[j].column())
                        list.removeAt(j--) ;
        for(QModelIndexList::size_type j = 0 ; j < list.size() ; j++)
        {
                QModelIndexList parents = ancestry(list[j]) ;
                for(QModelIndexList::size_type i = 0 ; i < list.size() ; i++)
                {
                        if(parents.contains(list[i]))
                        {
                                list.removeAt(j--) ;
                                break ;
                        }
                }
        }
}
void specModel::eliminateChildren(QList<specModelItem*>& l)
{
        QSet<specModelItem*> allChildren ;
        for(QList<specModelItem*>::iterator it = l.begin() ; it != l.end() ; ++it)
        {
                specFolderItem* folder = dynamic_cast<specFolderItem*>(*it) ;
                if(folder)
                        foreach(specModelItem * pointer, folder->childrenList())
                        allChildren << pointer ;
        }
        QList<specModelItem*> filtered ;
        for(QList<specModelItem*>::iterator it = l.begin() ; it != l.end() ; ++it)
                if(!allChildren.contains(*it))
                        filtered << *it ;
        l.swap(filtered);
}
QMimeData* specModel::mimeData(const QModelIndexList& indices) const
{
        QMimeData* mimeData = new QMimeData() ;
        QModelIndexList list = indices ;
        eliminateChildren(list) ;
        if(dropBuddy)
                dropBuddy->setLastRequested(list) ;
        QList<specModelItem*> pointers = pointerList(list) ;
        foreach(specMimeConverter * converter, mimeConverters())
        converter->exportData(pointers, mimeData) ;
        return mimeData ;
}
bool specModel::mimeAcceptable(const QMimeData* data) const
{
        foreach(specMimeConverter * converter, mimeConverters())
        if(converter->canImport(data))
                return true ;
        return false ;
}
bool specModel::dropMimeData(const QMimeData* data,
                             Qt::DropAction action, int row, int column, const QModelIndex& parent)
{
```

```
        Q_UNUSED(column)
        if(action == Qt::IgnoreAction)
                return true;
        row = (row != -1 ? row : rowCount(parent));
        if(internalDrop && dropBuddy)
        {
                dontDelete = dropBuddy->moveInternally(parent, row, this) ;
                internalDrop = false ;
                return true ;
        }
        QList<specModelItem*> newItems ;
        foreach(specMimeConverter * converter, mimeConverters())
        {
                if(converter->canImport(data))
                {
                        newItems = converter->importData(data) ;
                        break ;
                }
        }
        if(!newItems.isEmpty())
        {
                insertItems(newItems, parent, row) ;
                if(dropBuddy)
                {
                        specAddFolderCommand* command = new specAddFolderCommand ;
                        command->setParentObject(this) ;
                        command->setItems(newItems) ;
                        command->setText(tr("Insert␣items")) ;
                        dropBuddy->push(command);
                }
        }
        return true ;
}
#ifdef QT_DEBUG
void printModelContentSummary(specFolderItem* folder, QString indent)
{
        int n = 0 ;
        if(folder->parent()) n = folder->parent()->childNo(folder) ;
        qDebug() << indent + QString::number(n) + "␣contains␣" + QString::number(folder->children()) ;
        indent += "␣␣" ;
        for(int i = 0 ; i < folder->children() ; ++i)
        {
                if(folder->child(i)->isFolder())
                        printModelContentSummary((specFolderItem*)(folder->child(i)), indent);
        }
}
#endif
specModelItem* specModel::itemPointer(const QVector<int>& indexes) const
{
        specModelItem* pointer = root ;
#ifdef MODEL_DEBUG
        qDebug() << "seeking␣pointer␣for␣item" << indexes ;
        qDebug() << "============␣Model␣summary:␣============" ;
        printModelContentSummary((specFolderItem*) root, "");
        qDebug() << "========================================" ;
#endif
        for(int i = indexes.size() - 1 ; i >= 0 && pointer && pointer->isFolder() ; --i)
                pointer = ((specFolderItem*) pointer)->child(indexes[i]) ;
        return pointer ;
}
QVector<int> specModel::hierarchy(specModelItem* item)
{
        QVector<int> retVal ;
        specFolderItem* parent ;
        while((parent = item->parent()))
        {
                retVal << parent->childNo(item) ;
                item = parent ;
        }
        return retVal ;
}
QVector<int> specModel::hierarchy(const QModelIndex& index)
{
        QVector<int> retVal ;
        QModelIndex item = index ;
```

```
        while(item.isValid())
        {
                retVal << item.row();
                item = item.parent() ;
        }
        return retVal ;
}
void specModel::setDropBuddy(specActionLibrary* buddy)
{
        dropBuddy = buddy ;
        connect(dropBuddy, SIGNAL(stackIndexChanged()), this, SLOT(signalEndReset())) ;
}
QModelIndex specModel::index(const QVector<int>& ancestry, int column) const
{
        QModelIndex returnIndex = QModelIndex() ;
        for(int i = ancestry.size() - 1 ; i >= 0 ; --i)
                returnIndex = index(ancestry[i], column, returnIndex) ;
        return returnIndex ;
}
bool specModel::contains(specModelItem* parent) const
{
        while(parent->parent()) parent = parent->parent() ;
        return parent == root ;
}
QModelIndex specModel::index(specModelItem* pointer, int column) const
{
        if(!pointer) return QModelIndex() ;
        if(!contains(pointer)) return QModelIndex() ;
        return index(hierarchy(pointer), column) ;
}
QList<specModelItem*> specModel::pointerList(const QModelIndexList& indexes) const
{
        QList<specModelItem*> returnList ;
        for(int i = 0 ; i < indexes.size() ; ++i)
                returnList << itemPointer(indexes[i]) ;
        return returnList ;
}
QModelIndexList specModel::indexList(const QList<specModelItem*>& pointers) const
{
        QModelIndexList returnList ;
        for(int i = 0 ; i < pointers.size() ; ++i)
                returnList << index(pointers[i]) ;
        return returnList ;
}
void specModel::svgMoved(specCanvasItem* i, int point, double x, double y)
{
        specSVGItem* item = dynamic_cast<specSVGItem*>(i) ;
        if(!item || !dropBuddy) return ;
        specResizeSVGcommand* command = new specResizeSVGcommand ;
        command->setParentObject(this) ;
        item->pointMoved(point, x, y) ;
        command->setItem(item, item->getBounds()) ;
        command->undo();
        command->setText(tr("Resize/move␣SVG␣item"));
        dropBuddy->push(command) ;
}
void specModel::setMetaModel(specMetaModel* m)
{
        metaModel = m ;
}
specMetaModel* specModel::getMetaModel() const
{
        return metaModel ;
}
void specModel::signalChanged(const QModelIndex& i)
{
        QModelIndex begin = index(i.row(), 0, i.parent()),
                    end = index(i.row(), columnCount(i) - 1, i.parent()) ;
        signalChanged(begin, end) ;
        specModelItem* item = itemPointer(i) ;
        checkForNewDescriptors(QList<specModelItem*>() << item, i.parent());
        emit dataChanged(begin, end) ;
}
void specModel::signalChanged(QModelIndex begin, QModelIndex end)
{
```

```
        while(begin.parent() != end.parent())
        {
                if(begin.isValid())
                        begin = begin.parent() ;
534             else
                        end = QModelIndex() ;
        }
        QModelIndex i(begin) ;
        QList<specModelItem*> items ;
539     do
        {
                specModelItem* pointer = itemPointer(i) ;
                items << pointer;
                if(pointer->children())
544             {
                        QModelIndex firstChild = index(0, 0, i),
                                    lastChild = index(pointer->children() - 1, 0, i) ;
                        signalChanged(firstChild, lastChild) ;
                }
549             i = i.sibling(i.row() + 1, 0) ;
        }
        while(i != end && i.isValid()) ;
        checkForNewDescriptors(items, begin.parent()) ;
        emit dataChanged(begin, index(end.row(), Descriptors.count(), end.parent())) ;
554 }
    QList<specModelItem*> specModel::expandFolders(const QList<specModelItem*>& list, bool retain)
    {
        QList<specModelItem*> result ;
        foreach(specModelItem * item, list)
559     {
                if(item->isFolder())
                {
                        if(retain) result << item ;
                        result << expandFolders(((specFolderItem*) item)->childrenList(), retain) ;
564             }
                else
                        result << item ;
        }
        return result ;
569 }
    QStringList specModel::mimeTypes() const
    {
        QStringList types ;
        foreach(specMimeConverter * converter, mimeConverters())
574     types << converter->importableTypes() ;
        return types ;
    }
    void specModel::setDescriptorProperties(spec::descriptorFlags flags, const QString& key)
    {
579     if(Descriptors.contains(key))
                DescriptorProperties[Descriptors.indexOf(key)] = flags ;
        else
        {
                Descriptors << key ;
584             DescriptorProperties << flags ;
        }
    }
    void specModel::deleteDescriptor(const QString& key)
    {
589     int i = Descriptors.indexOf(key) ;
        if(i < Descriptors.size() && i < DescriptorProperties.size())
        {
                Descriptors.takeAt(i) ;
                DescriptorProperties.takeAt(i) ;
594     }
        root->deleteDescriptor(key) ;
    }
    void specModel::renameDescriptors(const QMap<QString, QString>& map)
    {
599     for(QStringList::iterator i = Descriptors.begin() ; i != Descriptors.end() ; ++i)
                if(map.contains(*i))
                        *i = map[*i] ;
        root->renameDescriptors(map) ;
    }
604 void specModel::dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const
```

```
{
        root->dumpDescriptor(destination, key) ;
}
void specModel::restoreDescriptor(qint16 position, spec::descriptorFlags flags, QListIterator<►
    specDescriptor>& origin, const QString& key)
{
        if(!Descriptors.contains(key))
        {
                Descriptors.insert(position, key) ;
                DescriptorProperties.insert(position, flags) ;
        }
        root->restoreDescriptor(origin, key) ;
}
QList<specFileImportFunction> specModel::acceptableImportFunctions() const
{
        return QList<specFileImportFunction>()
               << readHVFile
               << readPEFile
               << readPEBinary
               << readJCAMPFile
               << readSKHIFile
               << readXYFILE ;
}
QValidator* specModel::createValidator(const QModelIndex& i) const
{
        if(itemPointer(i)->isNumeric(Descriptors[i.column()]))
                return new QDoubleValidator ;
        return 0 ;
}
void specModel::signalBeginReset()
{
        if(resetPending) return ;
        resetPending = true ;
        beginResetModel();
}
void specModel::signalEndReset()
{
        if(resetPending)
                endResetModel();
        resetPending = false ;
}
```

src/model/specmodelitem.h

```
#ifndef SPECMODELITEM_H
#define SPECMODELITEM_H
#include <QString>
#include <QMultiMap>
#include "plot/speccanvasitem.h"
#include <QIcon>
#include <QStringList>
#include "names.h"
#include <QDataStream>
#include <QTextStream>
#include "specrange.h"
#include "specdescriptor.h"
class specModelItem ;
class specFolderItem ;
class specMetaItem ;
class specUndoCommand ;
class specModelItem : public specCanvasItem
{
private:
        specFolderItem* iparent ;
        void detachChild(specModelItem* child) ;
        bool dataValid ;
        QSet<specMetaItem*> clients ;
        specDescriptor description ;
        void processData() ;
        void initializeData();
protected:
        virtual bool shortCircuit(specModelItem* server) ;
        void readFromStream(QDataStream&) ;
        void writeToStream(QDataStream&) const ;
```

```
        specModelItem* factory(const type&) const ;
public:
        void revalidate() ;
        void invalidate() ;
        virtual bool connectServer(specModelItem*) ;
        virtual bool disconnectServer(specModelItem*) ;
        bool connectClient(specMetaItem* clnt) ;
        bool disconnectClient(specMetaItem* clnt) ;
        virtual QList<specModelItem*> serverList() const { return QList<specModelItem*>() ; }
        QSet<specMetaItem*> clientList() const { return clients ; }
        bool mergePlotData, sortPlotData ;
        specModelItem(specFolderItem* par = 0, QString description = "");
        specModelItem(const specModelItem&) ;
        virtual ~specModelItem();
        void setParent(specFolderItem*) ;
        specFolderItem* parent() const;
        virtual QList<specModelItem*>::size_type children() const;
        virtual bool isFolder() const ;
        virtual QString descriptor(const QString& key, bool full = false) const ;
        virtual double descriptorValue(const QString& key) const { Q_UNUSED(key) ; return NAN ; }
        virtual bool isEditable(QString key) const ;
        virtual bool isNumeric(const QString& key) const ;
        virtual bool changeDescriptor(QString key, QString value) ;
        virtual bool setActiveLine(const QString&, int) ;
        virtual int activeLine(const QString& key) const ;
        virtual void refreshPlotData() ;
        virtual QIcon decoration() const ;
        QIcon indicator(const QString&) const ;
        virtual bool addChild(specModelItem* child, QList<specModelItem*>::size_type position) ;
        virtual bool addChildren(QList<specModelItem*> list, QList<specModelItem*>::size_type ►
            position) ;
        virtual QStringList descriptorKeys() const ;
        virtual spec::descriptorFlags descriptorProperties(const QString& key) const ;
        virtual void setDescriptorProperties(const QString& key, spec::descriptorFlags f) ;
        virtual void exportData(const QList<QPair<bool, QString> >&, const QList<QPair<spec::value, ►
            QString> >&, QTextStream&) ;
        virtual QVector<double> intensityData() const ;
        virtual void movingAverage(int) {}
        virtual void average(int) {}
        virtual QString toolTip(const QString& column) const ;
        virtual void renameDescriptors(const QMap<QString, QString>& map) ;
        virtual void deleteDescriptor(const QString& key) ;
        virtual void dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const ;
        virtual void restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key) ;
        virtual QString editDescriptor(const QString& key) const ;
        int rtti() const { return spec::spectrum ; }
        void attach(QwtPlot* plot) ;
        void detach();
        static specModelItem* itemFactory(specStreamable::type) ;
        virtual specUndoCommand* itemPropertiesAction(QObject* parentObject) { Q_UNUSED(parentObject►
            ) ; return 0 ; }
        template <typename T>
        QVector<T> findDescendants() ;
        class descriptorComparison
        {
        private:
                const QStringList* description ;
        public:
                descriptorComparison(const QStringList* description) ;
                bool operator()(specModelItem*&, specModelItem*&) ;
        };
};
#endif
```

src/model/specmodelitem.cpp

```
#include "specmodelitem.h"
#include "specfolderitem.h"
#include "specdataitem.h"
#include "speclogentryitem.h"
#include "speclogmessage.h"
#include <QTextStream>
#include <QTime>
#include "kinetic/specmetaitem.h"
```

```
#include "utility-functions.h"
#include <QPainter>
#include "plot/specplotstyle.h"
#include "specsvgitem.h"
specModelItem::specModelItem(specFolderItem* par, QString desc)
        : specCanvasItem(),
          iparent(0),
          dataValid(false),
          description(desc, spec::editable),
          mergePlotData(true),
          sortPlotData(true)
{
        setParent(par) ;
        setItemAttribute(Legend, true) ;
        setLegendAttribute(LegendNoAttribute, false) ;
        setLegendAttribute(LegendShowLine, true) ;
        setLegendAttribute(LegendShowSymbol, true) ;
}
specModelItem::~specModelItem()
{
        detach();
        foreach(specMetaItem * client, clients)
        disconnectClient(client) ;
        setParent(0) ;
}
void specModelItem::setParent(specFolderItem* par)
{
        if(iparent)
                iparent->removeChild(this) ;
        if(!par)
                detachChild(this) ;
        iparent = par ;
}
void specModelItem::detachChild(specModelItem* child)
{
        child->detach();
        for(int i = 0 ; i < child->children() ; ++i)
                detachChild(((specFolderItem*) child)->child(i)) ;
}
specFolderItem* specModelItem::parent() const
{ return iparent ; }
QList<specModelItem*>::size_type specModelItem::children() const
{ return 0 ; }
bool specModelItem::isEditable(QString key) const
{
        if(key == "")
                return true ;
        return descriptorProperties(key) & spec::editable ;
}
bool specModelItem::changeDescriptor(QString key, QString value)
{
        if(key == "" && description.isEditable())
        {
                description.setContent(value) ;
                setTitle(value) ;
                return true ;
        }
        return false ;
}
bool specModelItem::setActiveLine(const QString& key, int line)
{
        if(key == "")
                return description.setActiveLine(line) ;
        return false ;
}
int specModelItem::activeLine(const QString& key) const
{
        if(key == "")
                return description.activeLine() ;
        return -1 ;
}
void specModelItem::refreshPlotData()
{ }
template<typename T>
QVector<T> specModelItem::findDescendants()
```

```
{
        QVector<T> result ;
        T self = dynamic_cast<T>(this) ;
        if(self) result << self ;
        return result ;
}
void specModelItem::processData()
{
        if(!sortPlotData && !mergePlotData)
                return ;
        QVector<QPointF> newData(dataSize()) ;
        for(size_t i = 0 ; i < dataSize() ; ++i)
                newData[i] = sample(i) ;
        if(sortPlotData)
                qSort(newData.begin(), newData.end(), comparePoints) ;
        typedef QVector<QPointF>::const_iterator vecit ;
        if(mergePlotData)
        {
                QVector<QPointF> realData ;
                for(vecit i = newData.begin() ; i != newData.end() ; ++i)
                {
                        double x = i->x(), y = i->y() ;
                        vecit j ;
                        for(j = i + 1 ; j != newData.end() && j->x() == x; ++j)
                                y += j->y() ;
                        realData << QPointF(x, y / (j - i)) ;
                }
                setData(new QwtPointSeriesData(realData)) ;
        }
        else
                setData(new QwtPointSeriesData(newData)) ;
}
QString specModelItem::descriptor(const QString& key, bool full) const
{
        if(key == "") return description.content(full) ;
        return QString() ;
}
bool specModelItem::isFolder() const { return false ;}
QIcon specModelItem::indicator(const QString& Descriptor) const
{
        if(descriptor(Descriptor, true).contains(QRegExp("\n")))
        {
                if(Descriptor == "")
                {
                        QIcon def = decoration(), arrow = QIcon::fromTheme("go-down") ;
                        const QSize size = def.availableSizes().first() ;
                        const QSize small = size / 2. ;
                        QPixmap map = def.pixmap(size) ;
                        QPainter painter(&map) ;
                        painter.drawPixmap(small.width(), small.height(), small.width(), small.►
                             height(), arrow.pixmap(small)) ;
                        return QIcon(map) ;
                }
                else return QIcon::fromTheme("go-down") ;
        }
        if(Descriptor == "")
                return decoration() ;
        return QIcon() ;
}
QIcon specModelItem::decoration() const { return QIcon() ; }
bool specModelItem::addChild(specModelItem* child, QList<specModelItem*>::size_type position)
{
        Q_UNUSED(child) ;
        Q_UNUSED(position) ;
        return false ;
}
bool specModelItem::addChildren(QList<specModelItem*> list, QList<specModelItem*>::size_type position)
{
        Q_UNUSED(list)
        Q_UNUSED(position)
        return false ;
}
QStringList specModelItem::descriptorKeys() const
{ return QStringList(QString("")) ;}
void specModelItem::writeToStream(QDataStream& out) const
```

```
{
        specCanvasItem::writeToStream(out) ;
        out << mergePlotData << sortPlotData
            << description ;
}
void specModelItem::readFromStream(QDataStream& in)
{
        specCanvasItem::readFromStream(in) ;
        in >> mergePlotData >> sortPlotData
           >> description ;
        setTitle(descriptor("", true));
        invalidate() ;
}
spec::descriptorFlags specModelItem::descriptorProperties(const QString& key) const
{
        if(key == "") return description.flags() ;
        return spec::def ;
}
void specModelItem::setDescriptorProperties(const QString& key, spec::descriptorFlags f)
{
        if(key == "") description.setFlags(f) ;
}
void specModelItem::exportData(const QList<QPair<bool, QString> >& headerFormat, const QList<QPair<►
    spec::value, QString> >& dataFormat, QTextStream& out)
{
        for(int i = 0 ; i < headerFormat.size() ; i++)
                out << (headerFormat[i].first ? headerFormat[i].second : this->descriptor(►
                    headerFormat[i].second)) ;
        out << endl ;
        for(size_t j = 0 ; j < dataSize() ; j++)
                for(int i = 0 ; i < dataFormat.size() ; i++)
                        out << (dataFormat[i].first ? sample(j).x() : sample(j).y()) << dataFormat[i►
                            ].second ;
        out << endl ;
}
QVector<double> specModelItem::intensityData() const
{
        return QVector<double>() ;
}
bool specModelItem::connectClient(specMetaItem* clnt)
{
        if(clients.contains(clnt))
                return false ;
        if(!clnt->connectServer(this))
                return false ;
        clients << clnt ;
        return true ;
}
bool specModelItem::disconnectClient(specMetaItem* clnt)
{
        if(!clients.contains(clnt))
                return false ;
        if(!clnt->disconnectServer(this))
                return false ;
        return clients.remove(clnt) ;
}
void specModelItem::invalidate()
{
        dataValid = false ;
        foreach(specMetaItem * client, clients)
        client->invalidate();
        if(iparent) iparent->invalidate();
}
void specModelItem::revalidate()
{
        if(dataValid) return ;
        refreshPlotData();
        processData() ;
        dataValid = true ;
}
bool specModelItem::shortCircuit(specModelItem* server)
{
        if(server == this) return true ;
        foreach(specMetaItem * client, clients)
        if(client->shortCircuit(server)) return true ;
```

```
        return false;
}
bool specModelItem::connectServer(specModelItem* itm)
{
        Q_UNUSED(itm)
        return false ;
}
bool specModelItem::disconnectServer(specModelItem* itm)
{
        Q_UNUSED(itm)
        return false ;
}
specModelItem* specModelItem::itemFactory(specStreamable::type t)
{
        switch(t)
        {
                case specStreamable::dataItem : return new specDataItem ;
                case specStreamable::folder : return new specFolderItem ;
                case specStreamable::logEntry : return new specLogEntryItem ;
                case specStreamable::sysEntry : return new specLogMessage ;
                case specStreamable::svgItem : return new specSVGItem ;
                case specStreamable::metaItem : return new specMetaItem ;
                case specStreamable::legacyDataItem : return new specLegacyDataItem ;
                default: return 0 ;
        }
}
specModelItem* specModelItem::factory(const type& t) const
{
        return itemFactory(t) ;
}
specModelItem::descriptorComparison::descriptorComparison(const QStringList* d)
        : description(d)
{
}
bool specModelItem::descriptorComparison::operator()(specModelItem*& a, specModelItem*& b)
{
        if(!description) return false ;
        foreach(QString criterion, *description)
        {
                QString aDescriptor = a->descriptor(criterion, true),
                        bDescriptor = b->descriptor(criterion, true) ;
                if(aDescriptor != bDescriptor)
                        return aDescriptor < bDescriptor ;
        }
        return false ;
}
void specModelItem::attach(QwtPlot* plot)
{
        specCanvasItem::attach(plot) ;
        foreach(specMetaItem * client, clients)
        client->refreshOtherPlots();
}
void specModelItem::detach()
{
        specCanvasItem::detach() ;
        foreach(specMetaItem * client, clients)
        client->refreshOtherPlots();
}
QString specModelItem::toolTip(const QString& column) const
{
        return descriptor(column, true) ;
}
specModelItem::specModelItem(const specModelItem& other)
        : specCanvasItem(other.description.content()),
          iparent(0),
          dataValid(false),
          description(other.description)
{}
bool specModelItem::isNumeric(const QString& key) const
{
        Q_UNUSED(key)
        return false ;
}
void specModelItem::renameDescriptors(const QMap<QString, QString>& map)
{
```

```
305         Q_UNUSED(map) ;
    }
    void specModelItem::deleteDescriptor(const QString& descriptors)
    {
            Q_UNUSED(descriptors) ;
310 }
    void specModelItem::dumpDescriptor(QList<specDescriptor>& destination, const QString& key) const
    {
            if(key == "")
                    destination << description ;
315         else
                    destination << specDescriptor() ;
    }
    void specModelItem::restoreDescriptor(QListIterator<specDescriptor>& origin, const QString& key)
    {
320         if(!origin.hasNext()) return ;
            if(key == "")
                    description = origin.next() ;
            else
                    origin.next() ;
325 }
    QString specModelItem::editDescriptor(const QString& key) const
    {
            return descriptor(key, true) ;
    }
330 void specModelItem::initializeData()
    {
            revalidate();
    }
    template QVector<specDataItem*> specModelItem::findDescendants() ;
```

src/model/specsvgitem.h

```
 1  #ifndef SPECSVGITEM_H
    #define SPECSVGITEM_H
    #include "specmodelitem.h"
    #include <qwt_plot_svgitem.h>
    #include <QRectF>
 6  #include <QPointF>
    class specSVGItem : public specModelItem
    {
            friend class svgItemProperties ;
    public:
11          typedef QPair<qint8, qreal> dimension ;
            enum SVGCornerPoint { undefined = -1,
                                  center = 0,
                                  left = 1,
                                  right = 2,
16                                top = 3,
                                  bottom = 4,
                                  topLeft = 5,
                                  bottomLeft = 6,
                                  topRight = 7,
21                                bottomRight = 8,
                                  size = 9
                                } ;
    private:
            QwtPlotSvgItem image ;
26          QByteArray data ;
            bool highlighting ;
            SVGCornerPoint anchor ;
            double transform(const dimension&) const ;
            void setDimension(dimension&, double) ;
31          type typeId() const { return specStreamable::svgItem ; }
            void readFromStream(QDataStream& in) ;
            void writeToStream(QDataStream& out) const;
            QRectF boundRect() const ;
            QPointF getPoint(SVGCornerPoint, const QRectF&) const ;
36          void setBoundRect(const QRectF&) ;
    public:
            specSVGItem(specFolderItem* par = 0, QString description = "") ;
            int rtti() const { return spec::SVGItem ; }
            void attach(QwtPlot* plot) ;
41          void detach() ;
```

```
        void setImage(const QByteArray&) ;
        void highlight(bool highlight) ;
        void refreshSVG();
        void pointMoved(const int&, const double&, const double&);
        struct bounds
        {
                dimension x, y, width, height ;
                bool operator== (const bounds&) const ;
        };
        bounds getBounds() const ;
        void setBounds(const bounds&) ;
        SVGCornerPoint setAnchor(const SVGCornerPoint&) ;
        QIcon decoration() const ;
        specUndoCommand* itemPropertiesAction(QObject* parentObject) ;
        QString toolTip(const QString& column) const ;
private:
        bounds ownBounds ;
};
QDataStream& operator<< (QDataStream& out, const specSVGItem::bounds& b) ;
QDataStream& operator>> (QDataStream& in, specSVGItem::bounds& b) ;
#endif
```

src/model/specsvgitem.cpp

```
#include "specsvgitem.h"
#include "specplot.h"
#include "canvaspicker.h"
#include <QSvgRenderer>
#include "svgitemproperties.h"
#include <QBuffer>
#include <QPainter>
void specSVGItem::attach(QwtPlot* newPlot)
{
        detach() ;
        image.attach(newPlot) ;
        specCanvasItem::attach(newPlot) ;
        specPlot* nPlot = qobject_cast<specPlot*> (newPlot) ;
        if(nPlot && nPlot->svgPicker())
                nPlot->svgPicker()->addSelectable(this) ;
}
void specSVGItem::detach()
{
        if(!plot()) return ;
        specPlot* oldPlot = qobject_cast<specPlot*> (plot()) ;
        if(oldPlot && oldPlot->svgPicker())
                oldPlot->svgPicker()->removeSelectable(this);
        image.detach();
        specModelItem::detach() ;
}
void specSVGItem::highlight(bool highlight)
{
        highlighting = highlight ;
        QVector<QPointF> points(specSVGItem::size) ;
        QRectF br = boundRect() ;
        for(int i = 0 ; i < specSVGItem::size ; ++i)
                points[i] = getPoint((SVGCornerPoint) i, br) ;
        setSamples(points) ;
        if(!highlight)
                setSymbol(0) ;
        else
                setSymbol(new QwtSymbol(QwtSymbol::Ellipse, QBrush(Qt::black), QPen(Qt::white), ►
                    QSize(5, 5))) ;
}
specSVGItem::specSVGItem(specFolderItem* par, QString description)
        : specModelItem(par, description),
          highlighting(false),
          anchor(topLeft)
{
        ownBounds.width = qMakePair<qint8, qreal> (QwtPlot::axisCnt, 0) ;
        ownBounds.height = qMakePair<qint8, qreal> (QwtPlot::axisCnt, 0) ;
        ownBounds.x = qMakePair<qint8, qreal> (QwtPlot::axisCnt, 0) ;
        ownBounds.y = qMakePair<qint8, qreal> (QwtPlot::axisCnt, 0) ;
        setStyle(QwtPlotCurve::NoCurve);
        setItemAttribute(Legend, false) ;
```

```
}
void specSVGItem::setImage(const QByteArray& newData)
{
        QSize defaultSize(QSvgRenderer(newData).defaultSize()) ;
        ownBounds.width = qMakePair<qint8, qreal> (QwtPlot::xBottom, defaultSize.width()) ;
        ownBounds.height = qMakePair<qint8, qreal> (QwtPlot::yLeft, defaultSize.height()) ;
        ownBounds.x = qMakePair<qint8, qreal> (QwtPlot::xBottom, 0) ;
        ownBounds.y = qMakePair<qint8, qreal> (QwtPlot::yLeft, 0) ;
        anchor = topLeft ;
        data = newData ;
}
QRectF specSVGItem::boundRect() const
{
        if(!plot()) return QRectF() ;
        qreal px = (ownBounds.x.first == QwtPlot::axisCnt) ? ownBounds.x.second :
                   plot()->invTransform(ownBounds.x.first, ownBounds.x.second),
                   py = (ownBounds.y.first == QwtPlot::axisCnt) ? ownBounds.y.second :
                        plot()->invTransform(ownBounds.y.first, ownBounds.y.second),
                        w = (ownBounds.width.first == QwtPlot::axisCnt) ? ownBounds.width.second :
                            (plot()->invTransform(ownBounds.width.first, ownBounds.width.second)
                             - plot()->invTransform(ownBounds.width.first, 0)),
                            h = (ownBounds.height.first == QwtPlot::axisCnt) ? ownBounds.height.►
                                 second :
                                (plot()->invTransform(ownBounds.height.first, 0)
                                 - plot()->invTransform(ownBounds.height.first, ownBounds.height.►
                                      second)) ;
        switch(anchor)
        {
                case topRight:
                case right:
                case bottomRight:
                        px -= w / 2 ;
                case center:
                case top:
                case bottom:
                        px -= w / 2 ;
                default: ;
        }
        switch(anchor)
        {
                case top:
                case topLeft:
                case topRight:
                        py -= h / 2 ;
                case center:
                case left:
                case right:
                        py -= h / 2 ;
                default: ;
        }
        return QRectF(px, py, w, h) ;
}
void specSVGItem::setBoundRect(const QRectF& br)
{
        if(!plot()) return ;
        QPointF anchorPoint = getPoint(anchor, br) ;
        ownBounds.width.second = (ownBounds.width.first == QwtPlot::axisCnt) ? br.width() :
                                 (plot()->transform(ownBounds.width.first, br.width())
                                  - plot()->transform(ownBounds.width.first, 0));
        ownBounds.height.second = (ownBounds.height.first == QwtPlot::axisCnt) ? br.height() :
                                  (plot()->transform(ownBounds.height.first, 0)
                                   - plot()->transform(ownBounds.height.first, br.height()));
        ownBounds.x.second = (ownBounds.x.first == QwtPlot::axisCnt) ? anchorPoint.x() :
                             plot()->transform(ownBounds.x.first, anchorPoint.x()) ;
        ownBounds.y.second = (ownBounds.y.first == QwtPlot::axisCnt) ? anchorPoint.y() :
                             plot()->transform(ownBounds.y.first, anchorPoint.y()) ;
        highlight(highlighting) ;
}
QPointF specSVGItem::getPoint(SVGCornerPoint point, const QRectF& br) const
{
        switch(point)
        {
                case top:
                        return QPointF(br.center().x(), br.bottom()) ;
                case bottom:
```

```
123                         return QPointF(br.center().x(), br.top()) ;
                    case left:
                            return QPointF(br.left(), br.center().y()) ;
                    case right:
                            return QPointF(br.right(), br.center().y()) ;
128                 case topLeft:
                            return br.bottomLeft() ;
                    case topRight:
                            return br.bottomRight() ;
                    case bottomLeft:
133                         return br.topLeft() ;
                    case bottomRight:
                            return br.topRight() ;
                    case center:
                            return br.center() ;
138                 default: ;
            }
            return QPointF() ;
    }
    void specSVGItem::refreshSVG()
143 {
            if(!plot()) return ;
            highlight(highlighting) ;
            image.loadData(boundRect(), data) ;
    }
148 void specSVGItem::writeToStream(QDataStream& out) const
    {
            specModelItem::writeToStream(out) ;
            out << data << ownBounds << qint8(anchor) ;
    }
153 void specSVGItem::readFromStream(QDataStream& in)
    {
            qint8 anc ;
            specModelItem::readFromStream(in) ;
            in >> data >> ownBounds >> anc ;
158         anchor = (SVGCornerPoint) anc ;
    }
    void specSVGItem::pointMoved(const int& p, const double& x, const double& y)
    {
            SVGCornerPoint point((SVGCornerPoint) p) ;
163         if(point == undefined) return ;
            if(!plot()) return ;
            QRectF br = boundRect() ;
            QPointF newPoint(x, y) ;
            if((point == topLeft ||
168             point == topRight ||
                point == bottomLeft ||
                point == bottomRight) &&
                br.width() && br.height())
            {
173             QPointF oldPoint(getPoint(point, br)) ;
                QPointF diff(newPoint - oldPoint) ;
                if(point == bottomRight || point == topLeft)
                        diff.setY(-diff.y()) ;
                if(br.height() *diff.x() < br.width() *diff.y())
178                     diff.setX(diff.y() *br.width() / br.height());
                else
                        diff.setY(diff.x() *br.height() / br.width());
                if(point == bottomRight || point == topLeft)
                        diff.setY(-diff.y()) ;
183             newPoint = oldPoint + diff ;
            }
            switch(point)
            {
                    case center: br.moveCenter(newPoint); break ;
188                 case left: br.setLeft(x); break ;
                    case right: br.setRight(x); break ;
                    case top: br.setBottom(y) ; break ;
                    case bottom: br.setTop(y) ; break ;
                    case topLeft: br.setBottomLeft(newPoint) ; break ;
193                 case topRight: br.setBottomRight(newPoint) ; break ;
                    case bottomLeft: br.setTopLeft(newPoint) ; break ;
                    case bottomRight: br.setTopRight(newPoint) ; break ;
                    default: ;
            }
```

```
        setBoundRect(br.normalized()) ;
}
specSVGItem::bounds specSVGItem::getBounds() const
{
        return ownBounds ;
}
void specSVGItem::setBounds(const bounds& b)
{
        ownBounds = b ;
}
QDataStream& operator<< (QDataStream& out, const specSVGItem::bounds& b)
{
        return out << b.x << b.y << b.width << b.height ;
}
QDataStream& operator>> (QDataStream& in, specSVGItem::bounds& b)
{
        return in >> b.x >> b.y >> b.width >> b.height ;
}
QIcon specSVGItem::decoration() const
{
        return QIcon::fromTheme("image-x-generic") ;
}
specUndoCommand* specSVGItem::itemPropertiesAction(QObject* parentObject)
{
        svgItemProperties propertiesDialog(this) ;
        if(propertiesDialog.exec() != QDialog::Accepted) return 0 ;
        return propertiesDialog.generateCommand(parentObject) ;
}
specSVGItem::SVGCornerPoint specSVGItem::setAnchor(const SVGCornerPoint& p)
{
        SVGCornerPoint old = anchor ;
        anchor = p ;
        return old ;
}
bool specSVGItem::bounds::operator == (const specSVGItem::bounds& other) const
{
        return x == other.x &&
               y == other.y &&
               width == other.width &&
               height == other.height ;
}
QString specSVGItem::toolTip(const QString& column) const
{
        QByteArray byteArray ;
        QBuffer buffer(&byteArray) ;
        QSvgRenderer renderer(data) ;
        QSize defaultSize = renderer.defaultSize() ;
        defaultSize.scale(150, 150, Qt::KeepAspectRatio);
        QImage image(defaultSize.width(), defaultSize.height(), QImage::Format_ARGB32) ;
        image.fill(Qt::transparent);
        QPainter painter(&image) ;
        renderer.render(&painter);
        buffer.open(QIODevice::WriteOnly) ;
        image.save(&buffer, "PNG") ;
        return QString("%1<br><img␣src=\"%2\"></img>")
               .arg(specModelItem::toolTip(column))
               .arg(QString(buffer.data().toBase64())) ;
}
```

src/model/specsvgunitbutton.h

```
#ifndef SPECSVGUNITBUTTON_H
#define SPECSVGUNITBUTTON_H
#include <QPushButton>
class specSvgUnitButton : public QPushButton
{
        Q_OBJECT
private:
private slots:
        void stateChanged() ;
public:
        explicit specSvgUnitButton(QWidget* parent = 0);
};
#endif
```

src/model/specsvgunitbutton.cpp

```
#include "specsvgunitbutton.h"
#include <qwt_plot.h>
specSvgUnitButton::specSvgUnitButton(QWidget* parent) :
        QPushButton(parent)
{
        setText(tr(""));
        setCheckable(true) ;
        connect(this, SIGNAL(toggled(bool)), this, SLOT(stateChanged())) ;
}
void specSvgUnitButton::stateChanged()
{
        if(isChecked()) setText(tr("Pixels"));
        else setText(tr("Axis Units")) ;
}
```

src/model/spectextmimeconverter.h

```
#ifndef SPECTEXTMIMECONVERTER_H
#define SPECTEXTMIMECONVERTER_H
#include "specmimetextexporter.h"
class QTextStream ;
class specTextMimeConverter : public specMimeTextExporter
{
        Q_OBJECT
public:
        explicit specTextMimeConverter(QObject* parent = 0);
        virtual QList<specModelItem*> importData(const QMimeData* data) ;
        bool canImport(const QStringList& types) ;
};
#endif
```

src/model/spectextmimeconverter.cpp

```
#include "spectextmimeconverter.h"
#include <QTextStream>
#include "specmodelitem.h"
#include <QMimeData>
#include "specfolderitem.h"
#include "specdataitem.h"
#include <QDoubleValidator>
specTextMimeConverter::specTextMimeConverter(QObject* parent)
        : specMimeTextExporter(parent)
{
}
QList<specModelItem*> specTextMimeConverter::importData(const QMimeData* data)
{
        QStringList lines = data->text().split("\n") ;
        QDoubleValidator validator ;
        QVector<specDataPoint> dataPoints ;
        foreach(QString line, lines)
        {
                QStringList values = line.split(QRegExp("\\s+"), QString::SkipEmptyParts) ;
                if(values.size() < 2) continue ;
                int pos = 0 ;
                if(validator.validate(values[0], pos) != QValidator::Acceptable) continue ;
                pos = 0 ;
                if(validator.validate(values[1], pos) != QValidator::Acceptable) continue ;
                dataPoints << specDataPoint(values[0].toDouble(), values[1].toDouble(), 0) ;
        }
        if(dataPoints.isEmpty()) return QList<specModelItem*>() ;
        return QList<specModelItem*>() << new specDataItem(dataPoints, QHash<QString, specDescriptor►
            >()) ;
}
bool specTextMimeConverter::canImport(const QStringList& types)
{
        return types.contains("text/plain") ;
}
```

src/model/specview.h

```
#ifndef SPECVIEW_H
#define SPECVIEW_H
#include <QTreeView>
#include "specstreamable.h"
class specViewState ;
class specModel ;
class specActionLibrary ;
class QContextMenuEvent ;
class specUndoCommand ;
class specView : public QTreeView, public specStreamable
{
        Q_OBJECT
private:
        specViewState* state ;
        void contextMenuEvent(QContextMenuEvent*) ;
        void triggerReselect() ;
        bool setAllSelected(const QVariant&, int role = Qt::EditRole) ;
        type typeId() const { return specStreamable::mainView ; }
        bool acceptData(const QMimeData*) ;
        void dragMoveEvent(QDragMoveEvent* event) ;
        void dragEnterEvent(QDragEnterEvent* event) ;
        void handleDropEventAction(QDropEvent* event) ;
private slots:
        void columnsInserted(const QModelIndex& parent, int start, int end) ;
protected:
        void keyPressEvent(QKeyEvent*) ;
        void dropEvent(QDropEvent* event) ;
        void readFromStream(QDataStream& in) ;
        void writeToStream(QDataStream& out) const ;
        specActionLibrary* actionLibrary ;
protected slots:
        void columnMoved(int, int, int) ;
public:
        specView(QWidget* parent = 0);
        virtual ~specView();
        specModel* model() const ;
        void setModel(specModel*) ;
        QModelIndexList getSelection() const ;
        void setActionLibrary(specActionLibrary*) ;
signals:
        void changed() ;
        void newUndoCommand(specUndoCommand*) ;
public slots:
        void prepareReset() ;
        void resetDone() ;
};
#endif
```

src/model/specview.cpp

```
#include "specview.h"
#include <specdelegate.h>
#include "specviewstate.h"
#include "specmodel.h"
#include "specactionlibrary.h"
#include <QContextMenuEvent>
#include <QHeaderView>
QModelIndexList specView::getSelection() const
{
        QModelIndexList l = selectedIndexes() ;
        QModelIndexList list ;
        foreach(QModelIndex index, l)
                if (!index.column())
                        list << index ;
        qSort(list) ;
        return list ;
}
void specView::columnsInserted(const QModelIndex& parent, int start, int end)
{
        if(!model()) return ;
        if(!start) return ;
        if(end != model()->columnCount(parent) - 1) return ;
```

```
        int column = start - 1 ;
        resizeColumnToContents(column) ;
        if(!column) setColumnWidth(column, qMax(columnWidth(column), 50));
}
void specView::columnMoved(int i, int j, int k)
{
        QTreeView::columnMoved() ;
        if(k != i)
        {
                header()->moveSection(k, i) ;
                QList<QPair<QVariant, QVariant> > descriptors ;
                for(int a = qMin(j, k) ; a < qMax(j, k) + 1 ; a++)
                        descriptors += qMakePair(model()->headerData(a, Qt::Horizontal),
                                                 model()->headerData(a, Qt::Horizontal, spec::►
                                                     descriptorPropertyRole)) ;
                if(k < j) descriptors.prepend(descriptors.takeLast()) ;
                else descriptors.append(descriptors.takeFirst()) ;
                for(QList<QVariant>::size_type a = 0 ; a < descriptors.size() ; a++)
                {
                        model()->setHeaderData(qMin(j, k) + a, Qt::Horizontal, descriptors[a].first) ;
                        model()->setHeaderData(qMin(j, k) + a, Qt::Horizontal, descriptors[a].second►
                             , spec::descriptorPropertyRole) ;
                }
        }
}
void specView::triggerReselect()
{
        QItemSelection selected = selectionModel()->selection() ;
        selectionModel()->clearSelection() ;
        selectionModel()->select(selected, QItemSelectionModel::ClearAndSelect) ;
}
specModel* specView::model() const
{return (specModel*) QAbstractItemView::model() ;}
void specView::keyPressEvent(QKeyEvent* event)
{
        if(event->key() == Qt::Key_Escape)
        {
                clearSelection() ;
                setCurrentIndex(QModelIndex()) ;
        }
        else if(event->key() == Qt::Key_A && event->modifiers() == Qt::ControlModifier)
        {
                QModelIndex index = currentIndex() ;
                if(!model()->isFolder(currentIndex()))
                        index = index.parent() ;
                else
                        selectionModel()->select(QItemSelection(model()->index(index.row(), 0, index►
                             .parent()),
                                                                model()->index(index.row(), model()►
                                                                    ->columnCount(index) - 1, index.►
                                                                    parent())),
                                                 QItemSelectionModel::Deselect);
                QItemSelection newSelection ;
                int columnCount = model()->columnCount(index) ;
                for(int i = 0 ; i < model()->rowCount(index) ; i++)
                        newSelection.select(model()->index(i, 0, index), model()->index(i, ►
                             columnCount - 1, index)) ;
                selectionModel()->select(newSelection, QItemSelectionModel::Select) ;
        }
        else QTreeView::keyPressEvent(event) ;
}
specView::specView(QWidget* parent)
        : QTreeView(parent),
          state(0),
          actionLibrary(0)
{
        setAutoExpandDelay(500);
        setVerticalScrollMode(QAbstractItemView::ScrollPerItem);
        setAlternatingRowColors(true) ;
        setSelectionBehavior(QTreeView::SelectRows);
        setSelectionMode(QAbstractItemView::ExtendedSelection) ;
        setDragEnabled(true) ;
        setAcceptDrops(true) ;
        setDropIndicatorShown(true) ;
        QAbstractItemDelegate* olddel = itemDelegate() ;
```

```
        setItemDelegate(new specDelegate(this)) ;
        delete olddel ;
        setAllColumnsShowFocus(true) ;
        connect(header(), SIGNAL(sectionMoved(int, int, int)), this, SLOT(columnMoved(int, int, int)►
            )) ;
}
void specView::contextMenuEvent(QContextMenuEvent* event)
{
        if(!indexAt(event->pos()).isValid())
        {
                clearSelection() ;
                setCurrentIndex(QModelIndex()) ;
        }
        if(!actionLibrary) return ;
        QMenu* cMenu = actionLibrary->contextMenu(this) ;
        cMenu->exec(event->globalPos()) ;
        delete cMenu ;
        event->accept() ;
}
void specView::setModel(specModel* model)
{
        QTreeView::setModel(model) ;
        connect(model, SIGNAL(modelAboutToBeReset()), this, SLOT(prepareReset())) ;
        connect(model, SIGNAL(modelReset()), this, SLOT(resetDone())) ;
        connect(model, SIGNAL(columnsInserted(const QModelIndex&, int, int)), this, SLOT(►
            columnsInserted(QModelIndex, int, int))) ;
}
specView::~specView()
{
        delete state ;
}
void specView::dropEvent(QDropEvent* event)
{
        bool internalMove = event->source() == this && event->proposedAction() == Qt::MoveAction ;
        model()->setInternalDrop(internalMove) ;
        if(internalMove)
                state = new specViewState(this) ;
        QTreeView::dropEvent(event) ;
}
void specView::writeToStream(QDataStream& out) const
{
        if(model())
                out << * ((specStreamable*) model()) ;
        specViewState state(this) ;
        out << state ;
}
void specView::readFromStream(QDataStream& in)
{
        if(model())
                in >> * ((specStreamable*) model()) ;
        specViewState state(this) ;
        in >> state ;
        state.restoreState();
}
void specView::prepareReset()
{
        if(!state)
                state = new specViewState(this) ;
}
void specView::resetDone()
{
        if(!state) return ;
        state->restoreState();
        delete state ;
        state = 0 ;
}
void specView::dragMoveEvent(QDragMoveEvent* event)
{
        QTreeView::dragMoveEvent(event) ;
        handleDropEventAction(event) ;
}
void specView::handleDropEventAction(QDropEvent* event)
{
        if(!acceptData(event->mimeData()))
        {
```

```
                event->ignore();
                return ;
        }
        if(qobject_cast<specView*> (event->source()) || event->proposedAction() == Qt::LinkAction)
        {
                event->acceptProposedAction();
                event->accept();
                return ;
        }
        if(event->possibleActions() & Qt::CopyAction)
        {
                event->setDropAction(Qt::CopyAction) ;
                event->accept();
                return ;
        }
        if(event->possibleActions() & Qt::LinkAction)
        {
                event->setDropAction(Qt::LinkAction) ;
                event->accept();
                return ;
        }
        event->ignore();
        return ;
}
void specView::dragEnterEvent(QDragEnterEvent* event)
{
        QTreeView::dragEnterEvent(event) ;
        handleDropEventAction(event) ;
}
bool specView::acceptData(const QMimeData* data)
{
        if(!model()) return false ;
        return model()->mimeAcceptable(data) ;
}
void specView::setActionLibrary(specActionLibrary* a)
{
        actionLibrary = a ;
}
```

src/model/specviewstate.h

```
#ifndef SPECVIEWSTATE_H
#define SPECVIEWSTATE_H
#include <QVector>
#include "specmodel.h"
#include <QByteArray>
#include "specview.h"
class specView ;
class specModel ;
class specViewState : public specStreamable
{
private:
        specView* parent ;
        QVector<specFolderItem*> openFolders ;
        QVector<specModelItem*> selectedItems ;
        specModelItem* currentTopItem ;
        specModelItem* currentItem ;
        QVector<int> hierarchyOfTopItem ;
        QVector<int> hierarchyOfCurrentItem ;
        QVector<qint32> widths ;
        void purgeLists() ;
        inline specModel* model() const { return parent ? parent->model() : 0 ; }
        specModelItem* hierarchyPointer(const QVector<int>&) ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::viewState ; }
public:
        explicit specViewState(specView* Parent) ;
        explicit specViewState(const specView* Parent) ;
        void setParent(specView* Parent) ;
        void getState(const specView* view) ;
        void getState() ;
        void restoreState() ;
```

```
        static QItemSelection indexesToSelection(const QModelIndexList& selectedItems, const ►
            specModel* model) ;
        static QItemSelection pointersToSelection(const QVector<specModelItem*>& selectedItems, ►
            const specModel* model) ;
};
#endif
```

src/model/specviewstate.cpp

```
#include "specviewstate.h"
#include <QHeaderView>
specViewState::specViewState(specView* Parent)
{
        setParent(Parent) ;
}
specViewState::specViewState(const specView* temp)
        : parent(0)
{
        getState(temp) ;
}
void specViewState::setParent(specView* Parent)
{
        parent = Parent ;
        getState() ;
}
void specViewState::purgeLists()
{
        openFolders.clear();
        selectedItems.clear();
        currentTopItem = 0 ;
        currentItem = 0 ;
        hierarchyOfTopItem.clear();
        widths.clear();
}
void specViewState::getState()
{
        getState(parent) ;
}
void specViewState::getState(const specView* view)
{
        if(!view) return ;
        purgeLists();
        specModel* model = view->model() ;
        int columnCount = view->header()->count() ;
        for(int i = 0 ; i < columnCount ; ++i)
                widths << view->columnWidth(i) ;
        QModelIndexList selectionList = view->getSelection() ;
        for(int i = 0 ; i < selectionList.size() ; ++i)
                selectedItems << model->itemPointer(selectionList[i]) ;
        QModelIndex item = model->index(0, 0, QModelIndex()) ;
        while(item.isValid())
        {
                if(view->isExpanded(item))
                        openFolders << (specFolderItem*) model->itemPointer(item) ;
                if(model->rowCount(item))
                        item = model->index(0, 0, item) ;
                else
                {
                        while(model->rowCount(item.parent()) == item.row() + 1 && item.isValid())
                                item = item.parent() ;
                        item = item.sibling(item.row() + 1, 0);
                }
        }
        currentItem = model->itemPointer(view->currentIndex()) ;
        hierarchyOfCurrentItem = model->hierarchy(currentItem) ;
        currentTopItem = model->itemPointer(view->indexAt(QPoint(0, 0))) ;
        qDebug() << view->visualRect(view->indexAt(QPoint(0, 0))) ;
        hierarchyOfTopItem = model->hierarchy(currentTopItem) ;
}
specModelItem* specViewState::hierarchyPointer(const QVector<int>& hierarchy)
{
        specModelItem* pointer = 0 ;
        int start = 0 ;
        while(start < hierarchy.size() && !(pointer = model()->itemPointer(hierarchy.mid(start++)))) ;
```

```
        return pointer ;
}
QItemSelection specViewState::pointersToSelection(const QVector<specModelItem*>& selectedItems, ►
    const specModel* model)
{
        QModelIndexList indexes ;
        foreach(specModelItem * item, selectedItems)
        indexes << model->index(item) ;
        return indexesToSelection(indexes, model) ;
}
QItemSelection specViewState::indexesToSelection(const QModelIndexList& selectedItems, const ►
    specModel* model)
{
        QItemSelection selectedIndexes ;
        foreach(const QModelIndex index, selectedItems)
        selectedIndexes << QItemSelectionRange(index,
                                               model->index(index.row(),
                                                       model->columnCount(index) - 1,
                                                       index.parent())) ;
        return selectedIndexes ;
}
void specViewState::restoreState()
{
        if(!parent) return ;
        currentTopItem = hierarchyPointer(hierarchyOfTopItem) ;
        currentItem = hierarchyPointer(hierarchyOfCurrentItem) ;
        for(int i = 0 ; i < widths.size() ; ++i)
                parent->setColumnWidth(i, widths[i]) ;
        if(currentItem)
                parent->selectionModel()->setCurrentIndex(model()->index(currentItem), ►
                    QItemSelectionModel::Current) ;
        parent->selectionModel()->select(pointersToSelection(selectedItems, model()), ►
            QItemSelectionModel::Select) ;
        parent->collapseAll();
        for(int i = 0 ; i < openFolders.size() ; ++i)
                parent->expand(model()->index(openFolders[i])) ;
        if(currentTopItem)
                parent->scrollTo(model()->index(currentTopItem), specView::PositionAtTop) ;
}
void specViewState::writeToStream(QDataStream& out) const
{
        out << quint32(openFolders.size())
            << quint32(selectedItems.size())
            << hierarchyOfTopItem
            << hierarchyOfCurrentItem
            << widths ;
        for(int i = 0 ; i < openFolders.size() ; ++i)
                out << model()->hierarchy(openFolders[i]) ;
        for(int i = 0 ; i < selectedItems.size() ; ++i)
                out << model()->hierarchy(selectedItems[i]) ;
}
void specViewState::readFromStream(QDataStream& in)
{
        openFolders.clear();
        selectedItems.clear();
        quint32 openFoldersSize ;
        quint32 selectedItemsSize ;
        in >> openFoldersSize
           >> selectedItemsSize
           >> hierarchyOfTopItem
           >> hierarchyOfCurrentItem
           >> widths ;
        QVector<int> hierarchy ;
        if(!model())
        {
                purgeLists();
                return ;
        }
        for(quint32 i = 0 ; i < openFoldersSize ; ++i)
        {
                in >> hierarchy ;
                specFolderItem* p = dynamic_cast<specFolderItem*>(model()->itemPointer(hierarchy)) ;
                if(p) openFolders << p ;
        }
        for(quint32 i = 0 ; i < selectedItemsSize ; ++i)
```

```
        {
                in >> hierarchy ;
                specModelItem* p = model()->itemPointer(hierarchy) ;
                if(p) selectedItems << p ;
        }
}
```

src/model/svgitemproperties.h

```
#ifndef SVGITEMPROPERTIES_H
#define SVGITEMPROPERTIES_H
#include <QDialog>
class specSVGItem ;
class specUndoCommand ;
namespace Ui
{
        class svgItemProperties;
}
class svgItemProperties : public QDialog
{
        Q_OBJECT
public:
        explicit svgItemProperties(specSVGItem* item, QWidget* parent = 0);
        ~svgItemProperties();
        specUndoCommand* generateCommand(QObject* parent) ;
private slots:
        void on_widthOriginalAspect_clicked();
        void on_heightOriginalAspect_clicked();
private:
        Ui::svgItemProperties* ui;
        specSVGItem* item ;
        double originalAspectRatio ;
};
#endif
```

src/model/svgitemproperties.cpp

```
#include "svgitemproperties.h"
#include "ui_svgitemproperties.h"
#include "specsvgitem.h"
#include "qwt_plot.h"
#include <QSvgRenderer>
#include "specresizesvgcommand.h"
#include "specmodel.h"
svgItemProperties::svgItemProperties(specSVGItem* i, QWidget* parent) :
        QDialog(parent),
        ui(new Ui::svgItemProperties),
        item(i)
{
        ui->setupUi(this);
        ui->itemPreview->load(i->data);
        QSize svgSize = QSvgRenderer(item->data).defaultSize() ;
        originalAspectRatio = (double) svgSize.width() / svgSize.height() ;
        svgSize.scale(300, 300, Qt::KeepAspectRatio);
        ui->itemPreview->setFixedSize(svgSize) ;
        ui->anchorRadioButtons->setId(ui->topLeftRadio, specSVGItem::topLeft);
        ui->anchorRadioButtons->setId(ui->topRadio, specSVGItem::top);
        ui->anchorRadioButtons->setId(ui->topRightRadio, specSVGItem::topRight);
        ui->anchorRadioButtons->setId(ui->leftRadio, specSVGItem::left);
        ui->anchorRadioButtons->setId(ui->centerRadio, specSVGItem::center);
        ui->anchorRadioButtons->setId(ui->rightRadio, specSVGItem::right);
        ui->anchorRadioButtons->setId(ui->bottomLeftRadio, specSVGItem::bottomLeft);
        ui->anchorRadioButtons->setId(ui->bottomRadio, specSVGItem::bottom);
        ui->anchorRadioButtons->setId(ui->bottomRightRadio, specSVGItem::bottomRight);
        ui->anchorRadioButtons->button(item->anchor)->setChecked(true) ;
        ui->xValue->setRange(-INFINITY, INFINITY);
        ui->yValue->setRange(-INFINITY, INFINITY);
        ui->widthValue->setRange(0, INFINITY);
        ui->heightValue->setRange(0, INFINITY);
        ui->xValue->setValue(item->ownBounds.x.second) ;
        ui->yValue->setValue(item->ownBounds.y.second) ;
        ui->widthValue->setValue(item->ownBounds.width.second) ;
```

```
        ui->heightValue->setValue(item->ownBounds.height.second) ;
        ui->xPixelsButton->setChecked(item->ownBounds.x.first != QwtPlot::axisCnt) ;
        ui->yPixelsButton->setChecked(item->ownBounds.y.first != QwtPlot::axisCnt) ;
        ui->widthPixelsButton->setChecked(item->ownBounds.width.first != QwtPlot::axisCnt) ;
        ui->heightPixelsButton->setChecked(item->ownBounds.height.first != QwtPlot::axisCnt) ;
}
svgItemProperties::~svgItemProperties()
{
        delete ui;
}
specUndoCommand* svgItemProperties::generateCommand(QObject* parent)
{
        specSVGItem::SVGCornerPoint newAnchor =
            (specSVGItem::SVGCornerPoint) ui->anchorRadioButtons->checkedId() ;
        specSVGItem::SVGCornerPoint oldAnchor = item->setAnchor(newAnchor) ;
        specSVGItem::bounds newBounds =
        {
                specSVGItem::dimension(ui->xPixelsButton->isChecked() ?
                QwtPlot::xBottom : QwtPlot::axisCnt,
                ui->xValue->value()),
                specSVGItem::dimension(ui->yPixelsButton->isChecked() ?
                QwtPlot::yLeft : QwtPlot::axisCnt,
                ui->yValue->value()),
                specSVGItem::dimension(ui->widthPixelsButton->isChecked() ?
                QwtPlot::xBottom : QwtPlot::axisCnt,
                ui->widthValue->value()),
                specSVGItem::dimension(ui->heightPixelsButton->isChecked() ?
                QwtPlot::yLeft : QwtPlot::axisCnt,
                ui->heightValue->value())
        },
        oldBounds = item->getBounds() ;
        if(newAnchor == oldAnchor && newBounds == oldBounds) return 0 ;
        specResizeSVGcommand* command = new specResizeSVGcommand ;
        command->setParentObject(parent) ;
        item->setBounds(newBounds) ;
        command->setItem(item, oldBounds, oldAnchor) ;
        command->setText(tr("Modify␣SVG␣item"));
        return command ;
}
void svgItemProperties::on_widthOriginalAspect_clicked()
{
        ui->widthValue->setValue(ui->heightValue->value() *originalAspectRatio) ;
}
void svgItemProperties::on_heightOriginalAspect_clicked()
{
        ui->heightValue->setValue(ui->widthValue->value() / originalAspectRatio);
}
```

src/model/svgitemproperties.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>svgItemProperties</class>
 <widget class="QDialog" name="svgItemProperties">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>637</width>
    <height>430</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Image Item Properties</string>
  </property>
  <layout class="QGridLayout" name="gridLayout_3">
   <item row="0" column="0" rowspan="2">
    <widget class="QSvgWidget" name="itemPreview" native="true">
     <property name="sizePolicy">
      <sizepolicy hsizetype="MinimumExpanding" vsizetype="MinimumExpanding">
       <horstretch>0</horstretch>
       <verstretch>0</verstretch>
      </sizepolicy>
     </property>
```

```
    <property name="maximumSize">
     <size>
      <width>300</width>
      <height>300</height>
     </size>
    </property>
   </widget>
  </item>
  <item row="0" column="1">
   <widget class="QGroupBox" name="groupBox">
    <property name="title">
     <string>Anchor Point</string>
    </property>
    <property name="flat">
     <bool>false</bool>
    </property>
    <property name="checkable">
     <bool>false</bool>
    </property>
    <layout class="QGridLayout" name="gridLayout_2">
     <item row="0" column="0">
      <widget class="QRadioButton" name="topLeftRadio">
       <property name="text">
        <string>Top Left</string>
       </property>
       <attribute name="buttonGroup">
        <string notr="true">anchorRadioButtons</string>
       </attribute>
      </widget>
     </item>
     <item row="0" column="1">
      <widget class="QRadioButton" name="topRadio">
       <property name="text">
        <string>Top</string>
       </property>
       <attribute name="buttonGroup">
        <string notr="true">anchorRadioButtons</string>
       </attribute>
      </widget>
     </item>
     <item row="0" column="2">
      <widget class="QRadioButton" name="topRightRadio">
       <property name="text">
        <string>TopRight</string>
       </property>
       <attribute name="buttonGroup">
        <string notr="true">anchorRadioButtons</string>
       </attribute>
      </widget>
     </item>
     <item row="1" column="0">
      <widget class="QRadioButton" name="leftRadio">
       <property name="text">
        <string>Left</string>
       </property>
       <attribute name="buttonGroup">
        <string notr="true">anchorRadioButtons</string>
       </attribute>
      </widget>
     </item>
     <item row="1" column="1">
      <widget class="QRadioButton" name="centerRadio">
       <property name="text">
        <string>Center</string>
       </property>
       <attribute name="buttonGroup">
        <string notr="true">anchorRadioButtons</string>
       </attribute>
      </widget>
     </item>
     <item row="1" column="2">
      <widget class="QRadioButton" name="rightRadio">
       <property name="text">
        <string>Right</string>
       </property>
```

```
        <attribute name="buttonGroup">
         <string notr="true">anchorRadioButtons</string>
        </attribute>
       </widget>
      </item>
      <item row="2" column="0">
       <widget class="QRadioButton" name="bottomLeftRadio">
        <property name="text">
         <string>Bottom Left</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">anchorRadioButtons</string>
        </attribute>
       </widget>
      </item>
      <item row="2" column="1">
       <widget class="QRadioButton" name="bottomRadio">
        <property name="text">
         <string>Bottom</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">anchorRadioButtons</string>
        </attribute>
       </widget>
      </item>
      <item row="2" column="2">
       <widget class="QRadioButton" name="bottomRightRadio">
        <property name="text">
         <string>Bottom Right</string>
        </property>
        <attribute name="buttonGroup">
         <string notr="true">anchorRadioButtons</string>
        </attribute>
       </widget>
      </item>
     </layout>
    </widget>
   </item>
   <item row="1" column="1">
    <widget class="QGroupBox" name="groupBox_2">
     <property name="title">
      <string>Dimensions</string>
     </property>
     <layout class="QGridLayout" name="gridLayout">
      <item row="1" column="1">
       <widget class="QDoubleSpinBox" name="yValue">
        <property name="sizePolicy">
         <sizepolicy hsizetype="MinimumExpanding" vsizetype="Fixed">
          <horstretch>0</horstretch>
          <verstretch>0</verstretch>
         </sizepolicy>
        </property>
       </widget>
      </item>
      <item row="0" column="0">
       <widget class="QLabel" name="xLabel">
        <property name="text">
         <string>x</string>
        </property>
        <property name="buddy">
         <cstring>xValue</cstring>
        </property>
       </widget>
      </item>
      <item row="1" column="0">
       <widget class="QLabel" name="yLabel">
        <property name="text">
         <string>y</string>
        </property>
        <property name="buddy">
         <cstring>yValue</cstring>
        </property>
       </widget>
      </item>
      <item row="2" column="0">
```

```
        <widget class="QLabel" name="widthLabel">
         <property name="text">
          <string>Width</string>
         </property>
         <property name="buddy">
          <cstring>widthValue</cstring>
         </property>
        </widget>
       </item>
       <item row="3" column="0">
        <widget class="QLabel" name="heightLabel">
         <property name="text">
          <string>Height</string>
         </property>
         <property name="buddy">
          <cstring>heightValue</cstring>
         </property>
        </widget>
       </item>
       <item row="0" column="1">
        <widget class="QDoubleSpinBox" name="xValue">
         <property name="sizePolicy">
          <sizepolicy hsizetype="MinimumExpanding" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
        </widget>
       </item>
       <item row="2" column="1">
        <widget class="QDoubleSpinBox" name="widthValue">
         <property name="sizePolicy">
          <sizepolicy hsizetype="MinimumExpanding" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
        </widget>
       </item>
       <item row="3" column="1">
        <widget class="QDoubleSpinBox" name="heightValue">
         <property name="sizePolicy">
          <sizepolicy hsizetype="MinimumExpanding" vsizetype="Fixed">
           <horstretch>0</horstretch>
           <verstretch>0</verstretch>
          </sizepolicy>
         </property>
        </widget>
       </item>
       <item row="0" column="2">
        <widget class="specSvgUnitButton" name="xPixelsButton">
         <property name="text">
          <string>Axis Units</string>
         </property>
         <property name="checkable">
          <bool>true</bool>
         </property>
         <property name="checked">
          <bool>false</bool>
         </property>
        </widget>
       </item>
       <item row="1" column="2">
        <widget class="specSvgUnitButton" name="yPixelsButton">
         <property name="text">
          <string>Axis Units</string>
         </property>
         <property name="checkable">
          <bool>true</bool>
         </property>
         <property name="checked">
          <bool>false</bool>
         </property>
        </widget>
       </item>
```

```
      <item row="2" column="2">
       <widget class="specSvgUnitButton" name="widthPixelsButton">
        <property name="text">
253       <string>Axis Units</string>
        </property>
        <property name="checkable">
         <bool>true</bool>
        </property>
258     <property name="checked">
         <bool>false</bool>
        </property>
       </widget>
      </item>
263   <item row="3" column="2">
       <widget class="specSvgUnitButton" name="heightPixelsButton">
        <property name="text">
         <string>Axis Units</string>
        </property>
268     <property name="checkable">
         <bool>true</bool>
        </property>
        <property name="checked">
         <bool>false</bool>
273     </property>
       </widget>
      </item>
      <item row="2" column="3">
       <widget class="QPushButton" name="widthOriginalAspect">
278     <property name="text">
         <string>Original Aspect</string>
        </property>
       </widget>
      </item>
283   <item row="3" column="3">
       <widget class="QPushButton" name="heightOriginalAspect">
        <property name="text">
         <string>Original Aspect</string>
        </property>
288     </widget>
      </item>
     </layout>
    </widget>
   </item>
293  <item row="2" column="1">
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
298  <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
303 </layout>
</widget>
<customwidgets>
 <customwidget>
  <class>QSvgWidget</class>
308  <extends>QWidget</extends>
  <header>QSvgWidget</header>
  <container>1</container>
 </customwidget>
 <customwidget>
313  <class>specSvgUnitButton</class>
  <extends>QPushButton</extends>
  <header>specsvgunitbutton.h</header>
 </customwidget>
</customwidgets>
318 <tabstops>
 <tabstop>buttonBox</tabstop>
 <tabstop>topLeftRadio</tabstop>
 <tabstop>topRadio</tabstop>
 <tabstop>topRightRadio</tabstop>
323  <tabstop>leftRadio</tabstop>
 <tabstop>centerRadio</tabstop>
```

```
   <tabstop>rightRadio</tabstop>
   <tabstop>bottomLeftRadio</tabstop>
   <tabstop>bottomRadio</tabstop>
   <tabstop>bottomRightRadio</tabstop>
   <tabstop>xValue</tabstop>
   <tabstop>xPixelsButton</tabstop>
   <tabstop>yValue</tabstop>
   <tabstop>yPixelsButton</tabstop>
   <tabstop>widthValue</tabstop>
   <tabstop>widthPixelsButton</tabstop>
   <tabstop>heightValue</tabstop>
   <tabstop>heightPixelsButton</tabstop>
  </tabstops>
  <resources/>
  <connections>
   <connection>
    <sender>buttonBox</sender>
    <signal>accepted()</signal>
    <receiver>svgItemProperties</receiver>
    <slot>accept()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>332</x>
      <y>401</y>
     </hint>
     <hint type="destinationlabel">
      <x>157</x>
      <y>274</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>buttonBox</sender>
    <signal>rejected()</signal>
    <receiver>svgItemProperties</receiver>
    <slot>reject()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>400</x>
      <y>401</y>
     </hint>
     <hint type="destinationlabel">
      <x>286</x>
      <y>274</y>
     </hint>
    </hints>
   </connection>
  </connections>
  <buttongroups>
   <buttongroup name="anchorRadioButtons"/>
  </buttongroups>
</ui>
```

src/model/textedit.h

```
#ifndef TEXTEDIT_H
#define TEXTEDIT_H
#include <QTextEdit>
QT_BEGIN_NAMESPACE
class QCompleter;
QT_END_NAMESPACE
class TextEdit : public QTextEdit
{
        Q_OBJECT
public:
        TextEdit(QWidget* parent = 0);
        ~TextEdit();
        void setCompleter(QCompleter* c);
        QCompleter* completer() const;
protected:
        void keyPressEvent(QKeyEvent* e);
        void focusInEvent(QFocusEvent* e);
private slots:
        void insertCompletion(const QString& completion);
```

```
private:
        QString textUnderCursor() const;
private:
        QCompleter* c;
};
#endif
```

src/model/textedit.cpp

```
#include "textedit.h"
#include <QCompleter>
#include <QKeyEvent>
#include <QAbstractItemView>
#include <QtDebug>
#include <QApplication>
#include <QModelIndex>
#include <QAbstractItemModel>
#include <QScrollBar>
#include <QSettings>
TextEdit::TextEdit(QWidget* parent)
        : QTextEdit(parent), c(0)
{
        QPalette palette = this->palette() ;
        palette.setColor(QPalette::Base, palette.toolTipBase().color());
        palette.setColor(QPalette::Text, palette.toolTipText().color());
        setPalette(palette) ;
        setFrameShadow(QFrame::Plain);
        setFrameShape(QFrame::NoFrame) ;
        setAcceptRichText(false) ;
        setBackgroundRole(QPalette::Text);
        setLineWrapMode(QTextEdit::NoWrap) ;
        setHorizontalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
        setVerticalScrollBarPolicy(Qt::ScrollBarAlwaysOff);
}
TextEdit::~TextEdit()
{
}
void TextEdit::setCompleter(QCompleter* completer)
{
        if(c)
                QObject::disconnect(c, 0, this, 0);
        c = completer;
        if(!c)
                return;
        c->setWidget(this);
        c->setCompletionMode(QCompleter::PopupCompletion);
        c->setCaseSensitivity(Qt::CaseInsensitive);
        QObject::connect(c, SIGNAL(activated(QString)),
                         this, SLOT(insertCompletion(QString)));
}
QCompleter* TextEdit::completer() const
{
        return c;
}
void TextEdit::insertCompletion(const QString& completion)
{
        if(c->widget() != this)
                return;
        QTextCursor tc = textCursor();
        int extra = completion.length() - c->completionPrefix().length();
        tc.movePosition(QTextCursor::Left);
        tc.movePosition(QTextCursor::EndOfWord);
        tc.insertText(completion.right(extra));
        setTextCursor(tc);
}
QString TextEdit::textUnderCursor() const
{
        QTextCursor tc = textCursor();
        tc.select(QTextCursor::WordUnderCursor);
        return tc.selectedText();
}
void TextEdit::focusInEvent(QFocusEvent* e)
{
        if(c)
```

```
                c->setWidget(this);
        QTextEdit::focusInEvent(e);
}
void TextEdit::keyPressEvent(QKeyEvent* e)
{
        if(c && c->popup()->isVisible())
        {
                switch(e->key())
                {
                        case Qt::Key_Enter:
                        case Qt::Key_Return:
                        case Qt::Key_Escape:
                        case Qt::Key_Tab:
                        case Qt::Key_Backtab:
                                e->ignore();
                                return;
                        default:
                                break;
                }
        }
        bool isShortcut = ((e->modifiers() & Qt::ControlModifier) && e->key() == Qt::Key_E);
        if(!c || !isShortcut)
                QTextEdit::keyPressEvent(e);
        const bool ctrlOrShift = e->modifiers() & (Qt::ControlModifier | Qt::ShiftModifier);
        if(!c || (ctrlOrShift && e->text().isEmpty()))
                return;
        static QString eow("~!@#$%^&*()_+{}|:\"<>?,./;'[]\\-=");
        bool hasModifier = (e->modifiers() != Qt::NoModifier) && !ctrlOrShift;
        QString completionPrefix = textUnderCursor();
        QSettings settings ;
        if(!isShortcut && (hasModifier || e->text().isEmpty() || completionPrefix.length() < ►
             settings.value("autoCompleter/Limit").toInt()
                        || eow.contains(e->text().right(1))))
        {
                c->popup()->hide();
                return;
        }
        if(completionPrefix != c->completionPrefix())
        {
                c->setCompletionPrefix(completionPrefix);
                c->popup()->setCurrentIndex(c->completionModel()->index(0, 0));
        }
        QRect cr = cursorRect();
        cr.setWidth(c->popup()->sizeHintForColumn(0)
                    + c->popup()->verticalScrollBar()->sizeHint().width());
        c->complete(cr);
}
```

src/names.h

```
#ifndef NAMES_H
#define NAMES_H
#define STRINGIFYMACRO(x) MAKESTRINGMACRO(x)
#define MAKESTRINGMACRO(x) #x
class specModelItem ;
class QFile ;
namespace spec
{
        enum desc {def = 0, numeric = 1, editable = 2, multiline = 4} ;
        Q_DECLARE_FLAGS(descriptorFlags, desc)
        Q_DECLARE_OPERATORS_FOR_FLAGS(descriptorFlags)
        enum separator { nosep = 0, space = 1, tab = 2, newline = 3 } ;
        enum value { wavenumber = 0, signal = 1, maxInt = 2 } ;
        enum undoActionIds { } ;
        enum itemRoles { descriptorPropertyRole = 34 } ;
        enum rtti {canvasItem = 1001, spectrum = 1010, zeroRange = 1020, metaItem = 1030, ►
             kineticRange = 1040, SVGItem = 1050, metaRange = 1060 } ;
        enum correctionMode { noCorrection = -1, offset = 0, offsetAndSlope = 1 } ;
}
typedef QList<specModelItem*> (* specFileImportFunction)(QFile&) ;
#endif
```

src/plot/canvaspicker.h

```
#ifndef CANVASPICKER_H
#define CANVASPICKER_H
#include <QObject>
#include <QHash>
#include <QAbstractItemModel>
#include <specmodelitem.h>
#include <names.h>
#include <specrange.h>
#include <QSet>
class QPoint;
class QCustomEvent;
class specPlot;
class specCanvasItem;
class CanvasPicker: public QObject
{
        Q_OBJECT
private:
        void select(const QPoint&);
        void move(const QPoint&);
        void moveBy(int dx, int dy);
        void release();
        void showCursor(bool enable);
        void shiftPointCursor(bool up);
        void shiftCurveCursor(bool up);
        void movePointExplicitly() ;
        bool owning ;
        specPlot* plot() { return (specPlot*) parent(); }
        QSet<specCanvasItem*> selectable ;
        void highlightSelectable() ;
        void switchHighlighting(bool) ;
        specCanvasItem* d_selectedCurve, *lastSelected;
        int d_selectedPoint;
        bool highlighting ;
public:
        explicit CanvasPicker(specPlot* plot);
        ~CanvasPicker() ;
        virtual bool eventFilter(QObject*, QEvent*);
        virtual bool event(QEvent*);
        void addSelectable(const QSet<specCanvasItem*>&) ;
        void addSelectable(specCanvasItem*) ;
        void removeSelectable(QSet<specCanvasItem*>&) ;
        void removeSelectable(specCanvasItem*) ;
        void removeSelectable();
        void setSelectable(const QSet<specCanvasItem*>&) ;
        void removeSelected() ;
        int countSelectable() const ;
        void highlightSelectable(bool) ;
        QList<specCanvasItem*> items() const ;
        inline void setOwning(bool Owning = true) { owning = Owning ; }
        void purgeSelectable() ;
signals:
        void pointMoved(specCanvasItem*, int, double, double) ;
};
#endif
```

src/plot/canvaspicker.cpp

```
#include <qapplication.h>
#include <qevent.h>
#include <qwhatsthis.h>
#include <qpainter.h>
#include "specplot.h"
#include <qwt_symbol.h>
#include <qwt_scale_map.h>
#include <qwt_plot_canvas.h>
#include <qwt_plot_curve.h>
#include <qwt_scale_div.h>
#include "canvaspicker.h"
#include <QTextStream>
#include <QBrush>
#include <QColor>
#include <QDoubleValidator>
```

```
#include <qwt_plot_canvas.h>
#include "speccanvasitem.h"
#include <QGridLayout>
#include <QDialog>
#include <QLineEdit>
#include <QDialogButtonBox>
#include <QLabel>
#include <qwt_plot_directpainter.h>
CanvasPicker::CanvasPicker(specPlot* plot)
        : QObject(plot),
          owning(false),
          d_selectedCurve(NULL),
          d_selectedPoint(-1),
          highlighting(true)
{
        QwtPlotCanvas* canvas = qobject_cast<QwtPlotCanvas*> (plot->canvas());
        canvas->installEventFilter(this);
#if QT_VERSION >= 0x040000
        canvas->setFocusPolicy(Qt::StrongFocus);
#ifndef QT_NO_CURSOR
        canvas->setCursor(Qt::CrossCursor);
#endif
#else
        canvas->setFocusPolicy(QWidget::StrongFocus);
#ifndef QT_NO_CURSOR
        canvas->setCursor(Qt::CrossCursor);
#endif
#endif
        canvas->setFocusIndicator(QwtPlotCanvas::ItemFocusIndicator);
        canvas->setFocus();
        const char* text =
            "Points in this plot may be moved by dragging (if indicated) or positioned explicitly by "
            "double clicking on them.  If zooming is enabled, step-wise zoom may be performed using "
            "the middle mouse button (click and drag the zoom region).  Clicking the right mouse "
            "button will go back one zoom step, while clicking the middle mouse button while holding "
            "the Ctrl key will go back to the unzoomed view.";
#if QT_VERSION >= 0x040000
        canvas->setWhatsThis(text);
#else
        QWhatsThis::add(canvas, text);
#endif
}
void CanvasPicker::setSelectable(const QSet<specCanvasItem*>& toSet)
{
        QSet<specCanvasItem*> toRemove(selectable - toSet),
             toAdd(toSet - selectable) ;
        removeSelectable(toRemove) ;
        addSelectable(toAdd) ;
}
bool CanvasPicker::event(QEvent* e)
{
        if(e->type() == QEvent::User)
        {
                showCursor(true);
                return true;
        }
        return QObject::event(e);
}
bool CanvasPicker::eventFilter(QObject* object, QEvent* e)
{
        if(object != (QObject*) plot()->canvas())
                return false;
        QMenu* contextMenu ;
        switch(e->type())
        {
                case QEvent::FocusIn:
                        showCursor(true);
                case QEvent::FocusOut:
                        showCursor(false);
                case QEvent::Paint:
                {
                        QApplication::postEvent(this, new QEvent(QEvent::User));
                        break;
                }
                case QEvent::MouseButtonDblClick:
```

```
91                  {
                            select(((QMouseEvent*) e)->pos());
                            if(d_selectedCurve)
                            {
                                    movePointExplicitly() ;
96                                  return true;
                            }
                            return false ;
                    }
                    case QEvent::MouseButtonPress:
101                 {
                            select(((QMouseEvent*) e)->pos());
                            if(((QMouseEvent*) e)->button() == Qt::RightButton && d_selectedCurve)
                            {
                                    contextMenu = d_selectedCurve->contextMenu() ;
106                                 if(contextMenu)
                                    {
                                            contextMenu->exec(((QMouseEvent*) e)->globalPos()) ;
                                            delete contextMenu ;
                                            contextMenu = 0 ;
111                                 }
                                    d_selectedCurve = NULL ;
                                    d_selectedPoint = -1 ;
                                    return true ;
                            }
116                         return d_selectedCurve ;
                    }
                    case QEvent::MouseMove:
                    {
                            move(((QMouseEvent*) e)->pos());
121                         return d_selectedCurve ;
                    }
                    case QEvent::MouseButtonRelease:
                    {
                            showCursor(false);
126                         d_selectedCurve = NULL;
                            d_selectedPoint = -1;
                    }
                    case QEvent::KeyPress:
                    {
131                         const int delta = 1;
                            switch(((const QKeyEvent*) e)->key())
                            {
                                    case Qt::Key_Up:
                                            shiftCurveCursor(true);
136                                 return true;
                                    case Qt::Key_Down:
                                            shiftCurveCursor(false);
                                            return true;
                                    case Qt::Key_Right:
141                                 case Qt::Key_Plus:
                                            if(d_selectedCurve)
                                                    shiftPointCursor(true);
                                            else
                                                    shiftCurveCursor(true);
146                                         return true;
                                    case Qt::Key_Left:
                                    case Qt::Key_Minus:
                                            if(d_selectedCurve)
                                                    shiftPointCursor(false);
151                                         else
                                                    shiftCurveCursor(true);
                                            return true;
                                    case Qt::Key_1:
                                            moveBy(-delta, delta);
156                                         break;
                                    case Qt::Key_2:
                                            moveBy(0, delta);
                                            break;
                                    case Qt::Key_3:
161                                         moveBy(delta, delta);
                                            break;
                                    case Qt::Key_4:
                                            moveBy(-delta, 0);
                                            break;
```

```
                                case Qt::Key_6:
                                        moveBy(delta, 0);
                                        break;
                                case Qt::Key_7:
                                        moveBy(-delta, -delta);
                                        break;
                                case Qt::Key_8:
                                        moveBy(0, -delta);
                                        break;
                                case Qt::Key_9:
                                        moveBy(delta, -delta);
                                        break;
                                default:
                                        break;
                        }
                }
                default:
                        break;
        }
        return QObject::eventFilter(object, e);
}
void CanvasPicker::select(const QPoint& pos)
{
        specCanvasItem* curve = NULL;
        double dist = 10e10;
        int index = -1;
        d_selectedCurve = NULL;
        d_selectedPoint = -1;
        for(QSet<specCanvasItem*>::iterator it = selectable.begin(); it != selectable.end(); ++it)
        {
                specCanvasItem* c = (specCanvasItem*)(*it);
                double d;
                int idx = c->closestPoint(pos, &d);
                if(d < dist)
                {
                        curve = c;
                        index = idx;
                        dist = d;
                }
        }
        showCursor(false);
        if(curve && dist < 10)
        {
                d_selectedCurve = curve;
                d_selectedPoint = index;
                showCursor(true);
        }
        lastSelected = d_selectedCurve ;
}
void CanvasPicker::moveBy(int dx, int dy)
{
        if(dx == 0 && dy == 0)
                return;
        if(!d_selectedCurve)
                return;
        const int x = plot()->transform(d_selectedCurve->xAxis(),
                                        d_selectedCurve->sample(d_selectedPoint).x()) + dx;
        const int y = plot()->transform(d_selectedCurve->yAxis(),
                                        d_selectedCurve->sample(d_selectedPoint).y()) + dy;
        move(QPoint(x, y));
}
void CanvasPicker::move(const QPoint& pos)
{
        if(!d_selectedCurve)
                return;
        emit pointMoved(d_selectedCurve, d_selectedPoint,
                        plot()->invTransform(d_selectedCurve->xAxis(), pos.x()),
                        plot()->invTransform(d_selectedCurve->yAxis(), pos.y())) ;
        showCursor(true);
}
void CanvasPicker::showCursor(bool showIt)
{
        Q_UNUSED(showIt)
}
void CanvasPicker::shiftCurveCursor(bool up)
```

```
{
        QSet<specCanvasItem*>::iterator it;
        if(selectable.isEmpty())
                return;
        it = selectable.begin();
        if(d_selectedCurve)
        {
                for(it = selectable.begin(); it != selectable.end(); ++it)
                {
                        if(d_selectedCurve == *it)
                                break;
                }
                if(it == selectable.end())
                        it = selectable.begin();
                if(up)
                {
                        ++it;
                        if(it == selectable.end())
                                it = selectable.begin();
                }
                else
                {
                        if(it == selectable.begin())
                                it = selectable.end();
                        --it;
                }
        }
        showCursor(false);
        d_selectedPoint = 0;
        d_selectedCurve = (specCanvasItem*) *it;
        showCursor(true);
}
void CanvasPicker::shiftPointCursor(bool up)
{
        if(!d_selectedCurve)
                return;
        int index = d_selectedPoint + (up ? 1 : -1);
        index = (index + d_selectedCurve->dataSize()) % d_selectedCurve->dataSize();
        if(index != d_selectedPoint)
        {
                showCursor(false);
                d_selectedPoint = index;
                showCursor(true);
        }
}
void CanvasPicker::movePointExplicitly()
{
        specCanvasItem* curve = d_selectedCurve ;
        int point = d_selectedPoint ;
        QDialog* query = new QDialog ;
        query->setWindowTitle("Move␣point␣to␣value") ;
        QGridLayout* layout = new QGridLayout ;
        QLineEdit* xBox = new QLineEdit(QString("%1").arg(curve->sample(point).x())),
        *yBox = new QLineEdit(QString("%1").arg(curve->sample(point).y())) ;
        xBox->setValidator(new QDoubleValidator(xBox)) ;
        yBox->setValidator(new QDoubleValidator(yBox)) ;
        QLabel* xLabel = new QLabel("x:"), *yLabel = new QLabel("y:") ;
        QDialogButtonBox* buttons = new QDialogButtonBox(QDialogButtonBox::Ok | QDialogButtonBox::►
            Cancel) ;
        connect(buttons, SIGNAL(accepted()), query, SLOT(accept())) ;
        connect(buttons, SIGNAL(rejected()), query, SLOT(reject())) ;
        layout->addWidget(xLabel, 0, 0) ;
        layout->addWidget(yLabel, 1, 0) ;
        layout->addWidget(xBox, 0, 1) ;
        layout->addWidget(yBox, 1, 1) ;
        layout->addWidget(buttons, 2, 0, 1, 2) ;
        query->setLayout(layout) ;
        if(query->exec() == QDialog::Accepted)
                emit pointMoved(curve, point, xBox->text().toDouble(), yBox->text().toDouble());
}
void CanvasPicker::highlightSelectable(bool highlight)
{
        highlighting = highlight ;
        highlightSelectable();
}
```

```
void CanvasPicker::highlightSelectable()
{
        switchHighlighting(highlighting);
}
void CanvasPicker::switchHighlighting(bool on)
{
        foreach(specCanvasItem * item, selectable)
        item->highlight(on) ;
        plot()->replot();
}
void CanvasPicker::addSelectable(specCanvasItem* item)
{
        QSet<specCanvasItem*> list ;
        list << item ;
        addSelectable(list) ;
}
void CanvasPicker::removeSelectable(specCanvasItem* item)
{
        QSet<specCanvasItem*> list ;
        list << item ;
        removeSelectable(list) ;
}
void CanvasPicker::addSelectable(const QSet<specCanvasItem*>& list)
{
        selectable += list ;
        highlightSelectable() ;
}
void CanvasPicker::removeSelectable(QSet<specCanvasItem*>& list)
{
        if(highlighting) switchHighlighting(false) ;
        if(list.contains(d_selectedCurve))
        {
                d_selectedCurve = 0 ;
                d_selectedPoint = -1 ;
        }
        if(owning)
        {
                foreach(specCanvasItem * item, list)
                {
                        item->detach();
                        delete item ;
                }
        }
        foreach(specCanvasItem * item, list)
        selectable.remove(item) ;
        if(highlighting) highlightSelectable() ;
        plot()->replot() ;
}
void CanvasPicker::removeSelected()
{
        if(selectable.contains(lastSelected))
                removeSelectable(lastSelected);
}
int CanvasPicker::countSelectable() const
{
        return selectable.size() ;
}
void CanvasPicker::removeSelectable()
{
        removeSelectable(selectable) ;
}
CanvasPicker::~CanvasPicker()
{
        highlightSelectable(false) ;
        if(owning) removeSelectable(selectable);
}
QList<specCanvasItem*> CanvasPicker::items() const
{
        return selectable.toList() ;
}
void CanvasPicker::purgeSelectable()
{
        selectable.clear() ;
}
```

src/plot/speccanvasitem.h

```
#ifndef SPECCANVASITEM_H
#define SPECCANVASITEM_H
#include <qwt_plot_curve.h>
#include <QList>
#include <QPair>
#include <qwt_interval.h>
#include "names.h"
#include <QMenu>
#include <qwt_symbol.h>
#include "specstreamable.h"
#include "specdatapointfilter.h"
class specCanvasItem : public QwtPlotCurve, public specStreamable
{
private:
        QList<int> selectedPoints ;
        QwtSymbol* oldSymbol ;
        class moveIndicator : public QwtSymbol
        {
        private:
                QSize boundingSize() const ;
                void renderSymbols(QPainter*, const QPointF*, int numPoints) const ;
                QRect boundingRect() const ;
        };
        virtual void initializeData() ;
protected:
        void writeToStream(QDataStream& out) const;
        void readFromStream(QDataStream& in) ;
public:
        specCanvasItem(QString description = "");
        virtual void pointMoved(const int&, const double&, const double&) {}
        virtual void refreshPlotData() = 0;
        virtual void addDataFilter(const specDataPointFilter&) {}
        virtual void attach(QwtPlot* plot) { QwtPlotCurve::attach(plot) ; }
        virtual void detach() { QwtPlotCurve::detach() ; }
        virtual QMenu* contextMenu() ;
        ~specCanvasItem();
        virtual QMap<double,double> dataMap() ;
        virtual QVector<double> xVector() ;
        virtual QVector<double> yVector() ;
        virtual QVector<QPointF> dataVector() ;
        int rtti() const { return spec::canvasItem ; }
        virtual void highlight(bool highlight) ;
        virtual void setLineWidth(const double&) ;
        virtual double lineWidth() const;
        virtual QColor penColor() const;
        virtual void setPenColor(const QColor&) ;
        virtual int symbolStyle() const ;
        virtual void setSymbolStyle(const int&) ;
        virtual QColor symbolPenColor() const ;
        virtual void setSymbolPenColor(const QColor&) ;
        virtual void setSymbolBrushColor(const QColor&) ;
        virtual QColor symbolBrushColor() const ;
        virtual QSize symbolSize() const;
        virtual void setSymbolSize(int w, int h = -1) ;
        virtual void setSymbolSize(const QSize&) ;
        virtual qint8 penStyle() const;
        virtual void setPenStyle(const qint8&) ;
};
#endif
```

src/plot/speccanvasitem.cpp

```
#include "speccanvasitem.h"
#include "utility-functions.h"
#include "specplot.h"
#include <QTextStream>
#include <QInputDialog>
#include <qwt_symbol.h>
#include "specplotstyle.h"
#include <QPainter>
specCanvasItem::specCanvasItem(QString description)
        : QwtPlotCurve(description),
```

```
          oldSymbol(0)
{
        setItemAttribute(Legend, false) ;
}
specCanvasItem::~specCanvasItem()
{
        detach() ;
        delete oldSymbol ;
}
void specCanvasItem::setLineWidth(const double& w)
{
        QPen newPen(pen()) ;
        newPen.setWidthF(w) ;
        setPen(newPen) ;
}
double specCanvasItem::lineWidth() const
{
        return pen().widthF() ;
}
QMenu* specCanvasItem::contextMenu()
{ return 0 ;}
void specCanvasItem::highlight(bool highlight)
{
        if(highlight)
        {
                if(!oldSymbol && symbol())
                        oldSymbol = cloneSymbol(symbol()) ;
                QwtSymbol* indicator = new QwtSymbol ;
                indicator->setPixmap(QPixmap(":/moveIndicator.png", "PNG"));
                indicator->setPinPoint(QPointF(5.5, 5.5));
                setSymbol(indicator) ;
        }
        else
        {
                setSymbol(oldSymbol) ;
                oldSymbol = 0 ;
        }
}
QColor specCanvasItem::penColor() const
{
        return pen().color() ;
}
void specCanvasItem::writeToStream(QDataStream& out) const
{
        out << specPlotStyle(this) ;
}
void specCanvasItem::readFromStream(QDataStream& in)
{
        specPlotStyle style(this) ;
        in >> style ;
        style.apply(this) ;
}
void specCanvasItem::setPenColor(const QColor& newColor)
{
        QPen newPen(pen());
        newPen.setColor(newColor);
        setPen(newPen) ;
}
int specCanvasItem::symbolStyle() const
{
        if(oldSymbol)
                return oldSymbol->style() ;
        if(symbol())
                return symbol()->style() ;
        return specPlotStyle::noSymbol ;
}
void specCanvasItem::setSymbolStyle(const int& newStyle)
{
        if(oldSymbol)
        {
                if(specPlotStyle::noSymbol == newStyle)
                {
                        delete oldSymbol ;
                        oldSymbol = 0 ;
                }
```

```
86                  else
                            oldSymbol->setStyle(QwtSymbol::Style(newStyle)) ;
            }
            else
            {
91                  if(specPlotStyle::noSymbol == newStyle)
                            setSymbol(0) ;
                    else
                    {
                            QwtSymbol* newSymbol = symbol() ? cloneSymbol(symbol()) : (new QwtSymbol) ;
96                          newSymbol->setStyle(QwtSymbol::Style(newStyle)) ;
                            setSymbol(newSymbol) ;
                    }
            }
    }
101 QColor specCanvasItem::symbolPenColor() const
    {
            if(oldSymbol) return oldSymbol->pen().color() ;
            if(!symbol()) return QColor() ;
            return symbol()->pen().color() ;
106 }
    void specCanvasItem::setSymbolPenColor(const QColor& newColor)
    {
            if(oldSymbol)
            {
111                 QPen newPen(oldSymbol->pen()) ;
                    newPen.setColor(newColor) ;
                    oldSymbol->setPen(newPen) ;
            }
            else
116         {
                    QwtSymbol* newSymbol = symbol() ? cloneSymbol(symbol()) : (new QwtSymbol) ;
                    QPen newPen = newSymbol->pen() ;
                    newPen.setColor(newColor) ;
                    newSymbol->setPen(newPen) ;
121                 setSymbol(newSymbol) ;
            }
    }
    void specCanvasItem::setSymbolBrushColor(const QColor& newColor)
    {
126         if(oldSymbol)
            {
                    QBrush brush(oldSymbol->brush()) ;
                    brush.setColor(newColor) ;
                    oldSymbol->setBrush(brush) ;
131         }
            else
            {
                    QwtSymbol* newSymbol = symbol() ? cloneSymbol(symbol()) : (new QwtSymbol) ;
                    newSymbol->setBrush(newColor) ;
136                 setSymbol(newSymbol) ;
            }
    }
    QColor specCanvasItem::symbolBrushColor() const
    {
141         if(oldSymbol) return oldSymbol->brush().color() ;
            if(!symbol()) return QColor() ;
            return symbol()->brush().color() ;
    }
    QSize specCanvasItem::symbolSize() const
146 {
            if(oldSymbol) return oldSymbol->size() ;
            if(!symbol()) return QSize() ;
            return symbol()->size() ;
    }
151 void specCanvasItem::setSymbolSize(int w, int h)
    {
            if(h < 0) setSymbolSize(QSize(w, w)) ;
            else setSymbolSize(QSize(w, h)) ;
    }
156 void specCanvasItem::setSymbolSize(const QSize& s)
    {
            if(oldSymbol)
            {
                    oldSymbol->setSize(s) ;
```

```
        }
        else
        {
                QwtSymbol* newSymbol = symbol() ? cloneSymbol(symbol()) : (new QwtSymbol) ;
                newSymbol->setSize(s) ;
                setSymbol(newSymbol) ;
        }
}
void specCanvasItem::moveIndicator::renderSymbols(QPainter* painter, const QPointF* points, int ►
    numPoints) const
{
        Q_UNUSED(points)
        QPixmap pixmap(":/moveIndicator.png", "PNG") ;
        for(int i = 0 ; i < numPoints ; ++i)
                painter->drawPixmap(boundingRect(), pixmap, pixmap.rect());
}
QRect specCanvasItem::moveIndicator::boundingRect() const
{
        return QRect(-5, -5, 11, 11) ;
}
QSize specCanvasItem::moveIndicator::boundingSize() const
{
        return QSize(11, 11) ;
}
qint8 specCanvasItem::penStyle() const
{
        return pen().style() ;
}
void specCanvasItem::setPenStyle(const qint8& s)
{
        QPen newPen(pen()) ;
        newPen.setStyle((Qt::PenStyle) s) ;
        setPen(newPen) ;
}
#define GETDATAFUNCTIONMACRO(DATATYPE,FUNCTIONNAME,INNERLOOP) \
        DATATYPE specCanvasItem::FUNCTIONNAME() \
        { initializeData() ; \
        DATATYPE result ; \
        for (size_t i = 0 ; i < dataSize() ; ++i) { INNERLOOP } \
        return result ; }
GETDATAFUNCTIONMACRO(QVector<double>,
                     xVector,
                     result << sample(i).x() ; )
GETDATAFUNCTIONMACRO(QVector<double>,
                     yVector,
                     result << sample(i).y() ; )
typedef QMap<double,double> doubleDoubleMap ;
GETDATAFUNCTIONMACRO(doubleDoubleMap,
                     dataMap,
                     const QPointF point = sample(i) ; result[point.x()] = point.y() ;)
GETDATAFUNCTIONMACRO(QVector<QPointF>,
                     dataVector,
                     result << sample(i) ; )
void specCanvasItem::initializeData()
{}
```

src/plot/specfiltergenerator.h

```
#ifndef SPECFILTERGENERATOR_H
#define SPECFILTERGENERATOR_H
#include <QMap>
#include <QPointF>
#include <QVector>
#include "specdatapointfilter.h"
#include <qwt_interval.h>
#include <qwt_plot_dict.h>
class QwtPlotItem ;
class specCanvasItem ;
class specFilterGenerator
{
private:
        QMap<double, double> reference ;
        QVector<QwtInterval> ranges, originalRanges ;
        QwtInterval referenceInterval ;
```

```
        bool CalcOffset, CalcSlope, CalcScale ;
        void correctionSpectrum(QVector<QPointF>&) ;
        double referenceValue(const double& x) ;
        bool isContainedInRanges(const double& x) const ;
        void refreshRanges() ;
public:
        specFilterGenerator();
        void setReference(const QMap<double, double>&) ;
        specDataPointFilter generateFilter(specCanvasItem *) ;
        void setRanges(const QwtPlotItemList& newRanges) ;
        void calcOffset(bool doIt = true) ;
        void calcSlope(bool doIt = true) ;
        void calcScale(bool doIt = true) ;
};
#endif
```

src/plot/specfiltergenerator.cpp

```
#include "specfiltergenerator.h"
#include "speccanvasitem.h"
#include "utility-functions.h"
#include <qwt_interval.h>
specFilterGenerator::specFilterGenerator() :
        CalcOffset(false),
        CalcSlope(false),
        CalcScale(false)
{
}
void specFilterGenerator::setReference(const QMap<double, double> &ref)
{
        reference = ref ;
        if (ref.isEmpty())
                referenceInterval = QwtInterval() ;
        else
                referenceInterval.setInterval(ref.begin().key(),
                                              (ref.end() -1).key());
        refreshRanges();
}
void specFilterGenerator::calcOffset(bool doIt)
{
        CalcOffset = doIt ;
}
void specFilterGenerator::calcSlope(bool doIt)
{
        CalcSlope = doIt ;
}
void specFilterGenerator::calcScale(bool doIt)
{
        CalcScale = doIt ;
}
bool specFilterGenerator::isContainedInRanges(const double &x) const
{
        if (!ranges.empty())
        {
                foreach(const QwtInterval& range, ranges)
                        if (range.contains(x)) return true ;
                return false ;
        }
        if (referenceInterval.isValid())
        {
                return referenceInterval.contains(x) ;
        }
        return true ;
}
void specFilterGenerator::correctionSpectrum(QVector<QPointF> &spectrum)
{
        if (reference.empty() && ranges.empty()) return ;
        QVector<QPointF> newSpec ;
        foreach(const QPointF& p, spectrum)
                if (isContainedInRanges(p.x()))
                        newSpec << p ;
        spectrum.swap(newSpec);
}
double specFilterGenerator::referenceValue(const double &x)
```

```
{
        if (reference.empty()) return 0 ;
        if (!reference.contains(x))
        {
                if (!referenceInterval.contains(x)) return NAN ;
                QMap<double, double>::const_iterator after = reference.upperBound(x) ;
                QMap<double, double>::const_iterator before = after - 1 ;
                reference[x] = before.value()
                                + (after.value() - before.value()) /
                                (after.key() - before.key()) *
                                (x - before.key()) ;
        }
        return reference[x] ;
}
void specFilterGenerator::setRanges(const QwtPlotItemList &newRanges)
{
        originalRanges.clear() ;
        foreach(const QwtPlotItem* item, newRanges)
        {
                const QwtInterval* range = dynamic_cast<const QwtInterval*>(item) ;
                if (range) originalRanges << *range ;
        }
        refreshRanges() ;
}
void specFilterGenerator::refreshRanges()
{
        ranges.clear();
        foreach(const QwtInterval& range, originalRanges)
        {
                if (referenceInterval.isValid())
                        ranges << (referenceInterval & range) ;
                else
                        ranges << range ;
        }
}
specDataPointFilter specFilterGenerator::generateFilter(specCanvasItem *item)
{
        if (!CalcOffset && !CalcSlope) return specDataPointFilter(NAN,NAN,NAN) ;
        QVector<QPointF> spectrum = item->dataVector() ;
        correctionSpectrum(spectrum) ;
        if (spectrum.size() < CalcOffset + CalcScale + CalcSlope)
                return specDataPointFilter(NAN,NAN,NAN) ;
#define ACCUMULATE_X x += p.x() ;
#define ACCUMULATE_XX xx += p.x()*p.x() ;
#define ACCUMULATE_XY xy += p.x()*p.y() ;
#define ACCUMULATE_XZ xz += p.x()*zv ;
#define ACCUMULATE_Y y += p.y() ;
#define ACCUMULATE_YZ yz += p.y()*zv ;
#define ACCUMULATE_YY yy += p.y()*p.y() ;
#define ACCUMULATE_Z z += zv ;
#define LOOPPOINTS(LOOPCORE) \
foreach(const QPointF& p, spectrum) { \
        double zv = (referenceValue(p.x()) - p.y()) ; \
        LOOPCORE \
                }
#define MATRIX_VECTOR_ASSIGNMENT_TWO(MATRIX_A,MATRIX_B,MATRIX_C,VECTOR_A,VECTOR_B) \
        QVector<double> vec(2) ; \
        QVector<QVector<double> > matrix(2, vec) ; \
        matrix[0][0] = MATRIX_A ; \
        matrix[1][0] = matrix[0][1] = MATRIX_B ; \
        matrix[1][1] = MATRIX_C ; \
        vec[0] = VECTOR_A ; \
        vec[1] = VECTOR_B ; \
        QVector<double> correction = gaussjinv(matrix, vec) ;
        double count = spectrum.size(),
                        x = 0,
                        xx = 0,
                        y = 0,
                        yy = 0,
                        z = 0,
                        xy = 0,
                        xz = 0,
                        yz = 0 ;
        double offset = 0, slope = 0 ;
        if(CalcOffset && !CalcSlope && !CalcScale)
```

```
        {
                LOOPPOINTS(ACCUMULATE_Z) ;
                offset = z / count ;
        }
        else if(!CalcOffset && CalcSlope && !CalcScale)
        {
                LOOPPOINTS(ACCUMULATE_XX ACCUMULATE_XZ) ;
                slope = xz / xx;
        }
        else if(CalcOffset && CalcSlope && !CalcScale)
        {
                LOOPPOINTS(ACCUMULATE_X ACCUMULATE_XX ACCUMULATE_XZ ACCUMULATE_Z) ;
                MATRIX_VECTOR_ASSIGNMENT_TWO(count, x, xx, z, xz) ;
                offset = correction[0] ;
                slope = correction[1] ;
        }
        else if(CalcOffset && !CalcSlope && CalcScale)
        {
                LOOPPOINTS(ACCUMULATE_Y ACCUMULATE_YY ACCUMULATE_YZ ACCUMULATE_Z) ;
                MATRIX_VECTOR_ASSIGNMENT_TWO(count, y, yy, z, yz) ;
                if(correction[1] != -1.)
                        offset = correction[0] / (1. + correction[1]) ;
        }
        else if(!CalcOffset && CalcSlope && CalcScale)
        {
                LOOPPOINTS(ACCUMULATE_XX ACCUMULATE_XY ACCUMULATE_YY ACCUMULATE_XZ ACCUMULATE_YZ)
                MATRIX_VECTOR_ASSIGNMENT_TWO(xx, xy, yy, xz, yz) ;
                if(correction[1] != -1.)
                        slope = correction[0] / (1. + correction[1]) ;
        }
        else if(CalcOffset && CalcSlope && CalcScale)
        {
                LOOPPOINTS(ACCUMULATE_X ACCUMULATE_Y ACCUMULATE_Z
                           ACCUMULATE_XX ACCUMULATE_XY ACCUMULATE_YY
                           ACCUMULATE_XZ ACCUMULATE_YZ) ;
                QVector<double> vec(3) ;
                QVector<QVector<double> > matrix(3, vec) ;
                matrix[0][0] = count ;
                matrix[1][1] = xx ;
                matrix[2][2] = yy ;
                matrix[0][1] = matrix[1][0] = x ;
                matrix[0][2] = matrix[2][0] = y ;
                matrix[1][2] = matrix[2][1] = xy;
                vec[0] = z ;
                vec[1] = xz ;
                vec[2] = yz ;
                QVector<double> correction = gaussjinv(matrix, vec) ;
                if(correction[2] != -1.)
                {
                        offset = correction[0] / (1. + correction[2]) ;
                        slope = correction[1] / (1. + correction[2]) ;
                }
        }
        return specDataPointFilter(offset, slope) ;
}
```

src/plot/specplot.h

```
#ifndef SPECPLOT_H
#define SPECPLOT_H
#include <qwt_plot.h>
#include "specstreamable.h"
#include <QLineEdit>
class specCanvasItem ;
class specZoomer ;
class QAction ;
class CanvasPicker ;
class specActionLibrary ;
class specView ;
class QwtPlotMarker ;
class plotAxisEdit : public QLineEdit
{
        Q_OBJECT
public:
```

```
        plotAxisEdit(QWidget* parent = 0) : QLineEdit(parent) { }
public slots:
        void hideAfterEditing(QWidget* old, QWidget* newW)
        {
                Q_UNUSED(old)
                if(newW != (QWidget*) this)
                {
                        hide() ;
                        disconnect(0, 0, this, SLOT(hideAfterEditing(QWidget*, QWidget*))) ;
                }
        }
};
class specPlot : public QwtPlot, public specStreamable
{
        Q_OBJECT
private:
        bool replotting ;
        specZoomer* zoom ;
        bool scaleX, scaleY ;
        QAction* fixXAxisAction,
                 *fixYAxisAction,
                 *modifySVGs,
                 *printAction,
                 *legendAction ;
        CanvasPicker* MetaPicker, *SVGpicker ;
        QwtPlotMarker* zeroYLine, *zeroXLine ;
        bool autoScaling ;
        void readFromStream(QDataStream& in);
        void writeToStream(QDataStream& out) const ;
        type typeId() const {return specStreamable::mainPlot ;}
        specActionLibrary* undoP ;
        void resizeEvent(QResizeEvent* e) ;
        void autoScale(const QwtPlotItemList& allItems) ;
        void mouseDoubleClickEvent(QMouseEvent* e) ;
        plotAxisEdit xminEdit, xmaxEdit, yminEdit, ymaxEdit ;
        QwtPlotMarker* initializeZeroLine() ;
private slots:
        void setPlotAxis() ;
        void showLegend(bool) ;
        void toggleItem(QwtPlotItem*, bool) ;
        void toggleItem(const QVariant&, bool) ;
protected:
        specView* view ;
        specActionLibrary* undoPartner() const ;
public:
        explicit specPlot(QWidget* parent = NULL);
        ~specPlot();
        QAction* svgAction() const ;
        CanvasPicker* metaPicker() ;
        CanvasPicker* svgPicker() ;
        virtual QList<QAction*> actions() ;
        void setView(specView* mod) ;
        void setUndoPartner(specActionLibrary* lib) ;
        virtual void attachToPicker(specCanvasItem*) ;
        virtual void detachFromPicker(specCanvasItem*) ;
        void setAutoScaling(bool) ;
signals:
        void startingReplot() ;
        void replotted() ;
        void metaRangeModified(specCanvasItem*, int, double, double) ;
public slots :
        void replot() ;
        void updateLegend() ;
};
#endif
```

src/plot/specplot.cpp

```
#include "specplot.h"
#include "speccanvasitem.h"
#include "speczoomer.h"
#include <QAction>
#include "canvaspicker.h"
#include "specactionlibrary.h"
```

```
#include "specview.h"
#include "specprintplotaction.h"
#include "specsvgitem.h"
#include "specmetaitem.h"
#include <qwt_scale_widget.h>
#include <QApplication>
#include <qwt_legend.h>
#include <QMouseEvent>
#include <qwt_plot_marker.h>
#include "specfitcurve.h"
#include <qwt_plot_layout.h>
QwtPlotMarker* specPlot::initializeZeroLine()
{
        QwtPlotMarker* zeroLine = new QwtPlotMarker ;
        zeroLine->setLinePen(QPen(Qt::DotLine)) ;
        zeroLine->attach(this) ;
        return zeroLine ;
}
specPlot::specPlot(QWidget* parent)
        : QwtPlot(parent),
          replotting(false),
          MetaPicker(0),
          SVGpicker(0),
          zeroYLine(0),
          zeroXLine(0),
          autoScaling(true),
          undoP(0),
          xminEdit(this),
          xmaxEdit(this),
          yminEdit(this),
          ymaxEdit(this),
          view(0)
{
        setAutoDelete(false);
        zeroYLine = initializeZeroLine() ;
        zeroYLine->setLineStyle(QwtPlotMarker::HLine);
        zeroXLine = initializeZeroLine() ;
        zeroXLine->setLineStyle(QwtPlotMarker::VLine);
        MetaPicker = new CanvasPicker(this) ;
        SVGpicker = new CanvasPicker(this) ;
        modifySVGs = new QAction(QIcon(":/resizeImage.png"), "Modify␣SVGs", this) ;
        modifySVGs->setToolTip(tr("Resize␣image␣items")) ;
        modifySVGs->setWhatsThis(tr("Modify␣the␣size␣of␣images␣displayed␣on␣the␣plot's␣canvas.")) ;
        modifySVGs->setCheckable(true) ;
        connect(MetaPicker, SIGNAL(pointMoved(specCanvasItem*, int, double, double)), this, SIGNAL(►
            metaRangeModified(specCanvasItem*, int, double, double))) ;
        setAutoReplot(false) ;
        zoom = new specZoomer(this->canvas()) ;
        fixYAxisAction = new QAction(QIcon(":/fixyaxis.png"), tr("Fixate␣&y␣axis"), this);
        fixYAxisAction->setShortcut(tr("Ctrl+Shift+y"));
        fixYAxisAction->setWhatsThis(tr("Disables␣auto␣scaling␣for␣the␣y␣axis␣and␣fixates␣the␣►
            current␣axis␣range."));
        fixYAxisAction->setIcon(QIcon(":/fixYAxis.png")) ;
        fixYAxisAction->setCheckable(true) ;
        fixYAxisAction->setChecked(false) ;
        fixXAxisAction = new QAction(QIcon(":/fixxaxis.png"), tr("Fixate␣&x␣axis"), this);
        fixXAxisAction->setShortcut(tr("Ctrl+Shift+x"));
        fixXAxisAction->setWhatsThis(tr("Disables␣auto␣scaling␣for␣the␣y␣axis␣and␣fixates␣the␣►
            current␣axis␣range.")) ;
        fixXAxisAction->setIcon(QIcon(":/fixXAxis.png")) ;
        fixXAxisAction->setCheckable(true) ;
        fixXAxisAction->setChecked(false) ;
        printAction = new specPrintPlotAction(this) ;
        legendAction = new QAction(QIcon(":/legend.png"), tr("Legend"), this) ;
        legendAction->setToolTip(tr("Legend")) ;
        legendAction->setWhatsThis(tr("Turn␣legend␣on/off")) ;
        legendAction->setCheckable(true) ;
        legendAction->setChecked(false) ;
        connect(legendAction, SIGNAL(toggled(bool)), this, SLOT(showLegend(bool))) ;
        QList<QLineEdit*> lineEdits ;
        lineEdits << &yminEdit << &ymaxEdit << &xminEdit << &xmaxEdit ;
        foreach(QLineEdit * lineEdit, lineEdits)
        {
                connect(lineEdit, SIGNAL(returnPressed()), this, SLOT(setPlotAxis())) ;
                connect(lineEdit, SIGNAL(editingFinished()), lineEdit, SLOT(hide())) ;
```

```
                lineEdit->hide();
                lineEdit->setFrame(false);
                lineEdit->adjustSize();
        }
        zoom->changeZoomBase(QRectF(-10, -10, 20, 20));
        showLegend(legendAction->isChecked()) ;
}
void specPlot::showLegend(bool l)
{
        if(!plotLayout()) return ;
        if(l)
        {
                QwtLegend* newLegend = new QwtLegend(this) ;
                newLegend->setDefaultItemMode(QwtLegendData::Checkable) ;
                insertLegend(newLegend) ;
                connect(newLegend, SIGNAL(checked(QVariant, bool, int)), this, SLOT(toggleItem(►
                    QVariant, bool))) ;
        }
        else
                insertLegend(0);
}
void specPlot::updateLegend()
{
        if (!legend()) return ;
        showLegend(true) ;
}
void specPlot::toggleItem(const QVariant& v, bool on)
{
        QwtPlotItem* item = infoToItem(v) ;
        if(item) item->setVisible(!on) ;
        replot() ;
}
void specPlot::toggleItem(QwtPlotItem* item, bool invisible)
{
        if(!item) return ;
        item->setVisible(!invisible) ;
        replot() ;
}
void specPlot::setAutoScaling(bool on)
{
        autoScaling = on ;
}
void specPlot::replot()
{
        if(replotting) return ;
        replotting = true ;
        emit startingReplot();
        QwtPlotItemList allItems = itemList();
        QSet<specCanvasItem*> newMetaRanges;
        QVector<specSVGItem*> svgitems ;
        foreach(QwtPlotItem * item, allItems)
        {
                if(dynamic_cast<specMetaRange*>(item))
                        newMetaRanges << ((specCanvasItem*) item) ;
                else if(dynamic_cast<specSVGItem*>(item))
                        svgitems << (specSVGItem*) item ;
        }
        if(allItems.isEmpty())
        {
                QwtPlot::replot() ;
                emit replotted() ;
                replotting = false ;
                return ;
        }
        autoScale(allItems) ;
        QwtPlot::replot() ;
        foreach(specSVGItem * svgitem, svgitems)
        svgitem->refreshSVG() ;
        QwtPlot::replot() ;
        emit replotted();
        replotting = false ;
}
void specPlot::autoScale(const QwtPlotItemList& allItems)
{
        if(!autoScaling) return ;
```

```
        QRectF boundaries, zoomBase = zoom->zoomBase() ;
        specModelItem* pointer = 0 ;
        bool firstItem = true ;
        foreach(QwtPlotItem * item, allItems)
        {
                if(!(dynamic_cast<specSVGItem*>(item))
                        && !(dynamic_cast<QwtPlotSvgItem*>(item))
                        && !(dynamic_cast<specFitCurve*>(item))
                        && item != zeroYLine
                        && item != zeroXLine)
                {
                        if((pointer = dynamic_cast<specModelItem*>(item)))
                                pointer->revalidate() ;
                        QRectF br = item->boundingRect() ;
                        if(br.isValid() || br.width() || br.height())
                                boundaries |= br ;
                        else
                        {
                                QPointF p = br.topLeft() ;
                                if(!boundaries.contains(br))
                                {
                                        if(firstItem)
                                                boundaries.moveTopLeft(p) ;
                                        else
                                        {
                                                if(p.y() > boundaries.bottom()) boundaries.setBottom►
                                                    (p.y()) ;
                                                else if(p.y() < boundaries.top()) boundaries.setTop(►
                                                    p.y()) ;
                                                if(p.x() < boundaries.left()) boundaries.setLeft(p.x►
                                                    ()) ;
                                                else if(p.x() > boundaries.right()) boundaries.►
                                                    setRight(p.x()) ;
                                        }
                                }
                        }
                        firstItem = false ;
                }
        }
        boundaries = boundaries.normalized() ;
        if(!boundaries.width())
        {
                boundaries.moveLeft(boundaries.left() - 5. / 1.1);
                boundaries.setWidth(10) ;
        }
        else
                boundaries.setWidth(1.1 * boundaries.width());
        if(!boundaries.height())
        {
                boundaries.moveTop(boundaries.top() - 5. / 1.1) ;
                boundaries.setHeight(10) ;
        }
        else
                boundaries.setHeight(1.1 * boundaries.height()) ;
        boundaries.translate(-.05 / 1.1 * boundaries.width(), -.05 / 1.1 * boundaries.height()) ;
        if(fixXAxisAction->isChecked())
        {
                boundaries.setLeft(zoomBase.left()) ;
                boundaries.setRight(zoomBase.right());
        }
        if(fixYAxisAction->isChecked())
        {
                boundaries.setTop(zoomBase.top()) ;
                boundaries.setBottom(zoomBase.bottom()) ;
        }
        zoom->changeZoomBase(boundaries) ;
}
specPlot::~specPlot()
{
        delete zeroYLine ;
        delete zeroXLine ;
        MetaPicker->purgeSelectable();
        SVGpicker->purgeSelectable();
        delete MetaPicker ;
        delete SVGpicker ;
```

```
}
QList<QAction*> specPlot::actions()
{
        return (QList<QAction*>() << modifySVGs << printAction << fixXAxisAction << fixYAxisAction ►
            << legendAction) ;
}
void specPlot::writeToStream(QDataStream& out) const
{
        out << title().text()
            << axisTitle(QwtPlot::xBottom).text()
            << axisTitle(QwtPlot::yLeft).text()
            << axisTitle(QwtPlot::xTop).text()
            << axisTitle(QwtPlot::yRight).text() ;
}
void specPlot::readFromStream(QDataStream& in)
{
        QString Title, xlabel, ylabel, x2label, y2label ;
        in >> Title >> xlabel >> ylabel >> x2label >> y2label ;
        setTitle(Title) ;
        setAxisTitle(QwtPlot::xBottom, xlabel) ;
        setAxisTitle(QwtPlot::yLeft, ylabel) ;
        setAxisTitle(QwtPlot::xTop, x2label) ;
        setAxisTitle(QwtPlot::yRight, y2label) ;
}
void specPlot::setUndoPartner(specActionLibrary* lib)
{
        undoP = lib ;
        ((specUndoAction*) printAction)->setLibrary(lib);
}
specActionLibrary* specPlot::undoPartner() const
{
        return undoP ;
}
void specPlot::setView(specView* mod)
{
        if (view && view->model())
                disconnect(view->model(), 0, this, 0) ;
        view = mod ;
        if (view && view->model())
                connect(view->model(), SIGNAL(dataChanged(QModelIndex,QModelIndex)), this, SLOT(►
                    updateLegend())) ;
}
CanvasPicker* specPlot::metaPicker()
{
        return MetaPicker ;
}
CanvasPicker* specPlot::svgPicker()
{
        return SVGpicker ;
}
void specPlot::resizeEvent(QResizeEvent* e)
{
        QwtPlot::resizeEvent(e) ;
        replot() ;
}
void specPlot::attachToPicker(specCanvasItem* i)
{
        Q_UNUSED(i)
}
void specPlot::detachFromPicker(specCanvasItem* i)
{
        Q_UNUSED(i)
}
QAction* specPlot::svgAction() const
{
        return modifySVGs ;
}
void specPlot::mouseDoubleClickEvent(QMouseEvent* e)
{
        QWidget* child = childAt(e->pos()) ;
        QwtScaleWidget* xAxisWidget = axisWidget(QwtPlot::xBottom) ;
        QwtScaleWidget* yAxisWidget = axisWidget(QwtPlot::yLeft) ;
        xmaxEdit.setText(QString::number(zoom->zoomBase().right())) ;
        xminEdit.setText(QString::number(zoom->zoomBase().left()));
        ymaxEdit.setText(QString::number(zoom->zoomBase().bottom())) ;
```

```
        yminEdit.setText(QString::number(zoom->zoomBase().top()));
        ymaxEdit.selectAll();
        xmaxEdit.selectAll();
        yminEdit.selectAll();
        xminEdit.selectAll();
        if(child == (QWidget*) xAxisWidget && fixXAxisAction->isChecked())
        {
                QRect geom = xAxisWidget->geometry() ;
                if((e->pos() - xAxisWidget->pos()).x() < xAxisWidget->width() / 2)
                {
                        geom.setWidth(qMin(xminEdit.width(), geom.width()));
                        geom.moveLeft(xAxisWidget->geometry().left());
                        xminEdit.setGeometry(geom) ;
                        xminEdit.show();
                        connect(qApp, SIGNAL(focusChanged(QWidget*, QWidget*)), &xminEdit, SLOT(►
                             hideAfterEditing(QWidget*, QWidget*))) ;
                        xminEdit.setFocus();
                }
                else
                {
                        geom.setWidth(qMin(xmaxEdit.geometry().width(), geom.width()));
                        geom.moveRight(xAxisWidget->geometry().right());
                        xmaxEdit.setGeometry(geom);
                        xmaxEdit.show();
                        connect(qApp, SIGNAL(focusChanged(QWidget*, QWidget*)), &xmaxEdit, SLOT(►
                             hideAfterEditing(QWidget*, QWidget*))) ;
                        xmaxEdit.setFocus();
                }
        }
        if(child == (QWidget*) yAxisWidget && fixYAxisAction->isChecked())
        {
                QRect geom = yAxisWidget->geometry() ;
                if((e->pos() - yAxisWidget->pos()).y() < yAxisWidget->height() / 2)
                {
                        geom.setHeight(qMin(ymaxEdit.height(), geom.height()));
                        geom.moveTop(yAxisWidget->geometry().top());
                        ymaxEdit.setGeometry(geom) ;
                        ymaxEdit.show();
                        connect(qApp, SIGNAL(focusChanged(QWidget*, QWidget*)), &ymaxEdit, SLOT(►
                             hideAfterEditing(QWidget*, QWidget*))) ;
                        ymaxEdit.setFocus();
                }
                else
                {
                        geom.setHeight(qMin(yminEdit.geometry().height(), geom.height()));
                        geom.moveBottom(yAxisWidget->geometry().bottom());
                        yminEdit.setGeometry(geom);
                        yminEdit.show();
                        connect(qApp, SIGNAL(focusChanged(QWidget*, QWidget*)), &yminEdit, SLOT(►
                             hideAfterEditing(QWidget*, QWidget*))) ;
                        yminEdit.setFocus();
                }
        }
}
void specPlot::setPlotAxis()
{
        plotAxisEdit* edit = qobject_cast<plotAxisEdit*> (sender()) ;
        if(!edit) return ;
        QDoubleValidator dv ;
        QString text = edit->text() ;
        int pos = 0 ;
        if(dv.validate(text, pos) != QValidator::Acceptable) return ;
        double value = text.toDouble() ;
        QRectF zoomBase = zoom->zoomBase() ;
        if(edit == &yminEdit && value < zoomBase.bottom())
                zoomBase.setTop(value) ;
        if(edit == &ymaxEdit && value > zoomBase.top())
                zoomBase.setBottom(value) ;
        if(edit == &xminEdit && value < zoomBase.right())
                zoomBase.setLeft(value);
        if(edit == &xmaxEdit && value > zoomBase.left())
                zoomBase.setRight(value) ;
        zoom->changeZoomBase(zoomBase);
}
```

src/plot/specplotstyle.h

```
#ifndef SPECPLOTSTYLE_H
#define SPECPLOTSTYLE_H
#include <QPen>
#include <qwt_symbol.h>
#include "speccanvasitem.h"
#include "specstreamable.h"
class specPlotStyle : public specStreamable
{
        qreal lineWidth ;
        QColor penColor, symbolPenColor, symbolBrushColor ;
        qint16 symbolStyle ;
        QSize symbolSize ;
        qint8 penStyle ;
        void initialize(const specCanvasItem*) ;
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::plotStyle ; }
public:
        enum ownSymbolTypes
        { noSymbol = -2 };
        explicit specPlotStyle(QDataStream&);
        explicit specPlotStyle(const specCanvasItem*);
        specPlotStyle() ;
        void apply(specCanvasItem*) const;
        void retrieve(specCanvasItem*) ;
};
#endif
```

src/plot/specplotstyle.cpp

```
#include "specplotstyle.h"
#include "names.h"
#define PLOTSTYLEINITIALIZER lineWidth(0),\
        penColor(Qt::black),\
        symbolPenColor(Qt::black),\
        symbolBrushColor(Qt::transparent),\
        symbolStyle(QwtSymbol::NoSymbol),\
        penStyle(Qt::SolidLine)
specPlotStyle::specPlotStyle()
        : PLOTSTYLEINITIALIZER
{
}
specPlotStyle::specPlotStyle(QDataStream& in)
        : PLOTSTYLEINITIALIZER
{
        readFromStream(in) ;
}
specPlotStyle::specPlotStyle(const specCanvasItem* c)
        : PLOTSTYLEINITIALIZER
{
        initialize(c) ;
}
void specPlotStyle::initialize(const specCanvasItem* curve)
{
        lineWidth = curve->lineWidth() ;
        penColor = curve->penColor() ;
        symbolStyle = curve->symbolStyle() ;
        penStyle = curve->penStyle() ;
        if(symbolStyle != noSymbol)
        {
                symbolSize = curve->symbolSize() ;
                symbolPenColor = curve->symbolPenColor() ;
                symbolBrushColor = curve->symbolBrushColor() ;
        }
}
void specPlotStyle::writeToStream(QDataStream& out) const
{
        out << lineWidth
            << penColor
            << symbolStyle
            << penStyle ;
        if(symbolStyle != noSymbol)
```

```
                out << symbolSize
                    << symbolPenColor
                    << symbolBrushColor ;
}
void specPlotStyle::readFromStream(QDataStream& in)
{
        in >> lineWidth
           >> penColor
           >> symbolStyle
           >> penStyle ;
        if(symbolStyle != noSymbol)
                in >> symbolSize
                   >> symbolPenColor
                   >> symbolBrushColor ;
}
void specPlotStyle::apply(specCanvasItem* c) const
{
        c->setLineWidth(lineWidth) ;
        c->setPenColor(penColor) ;
        c->setPenStyle(penStyle) ;
        c->setSymbolStyle(symbolStyle) ;
        c->setSymbolPenColor(symbolPenColor) ;
        c->setSymbolBrushColor(symbolBrushColor) ;
        c->setSymbolSize(symbolSize) ;
}
void specPlotStyle::retrieve(specCanvasItem* c)
{
        initialize(c) ;
}
```

src/plot/specrange.h

```
#ifndef SPECRANGE_H
#define SPECRANGE_H
#include "speccanvasitem.h"
#include <qwt_interval.h>
#include <names.h>
class specRange : public specCanvasItem, public QwtInterval
{
private:
        void writeToStream(QDataStream& out) const ;
        void readFromStream(QDataStream& in) ;
        type typeId() const { return specStreamable::range ; }
protected:
        double yVal ;
public:
        specRange(double, double, double y = 0);
        void pointMoved(const int&, const double&, const double&) ;
        virtual void refreshPlotData() ;
        int rtti() const { return spec::zeroRange ; }
        ~specRange();
};
#endif
```

src/plot/specrange.cpp

```
#include "specrange.h"
#include <qwt_symbol.h>
#include <QTextStream>
#include "specplot.h"
specRange::specRange(double min, double max, double yinit)
        : QwtInterval(min, max, QwtInterval::IncludeBorders)
{
        QVector<double> x, y ;
        x << min << max ;
        y << yinit << yinit ;
        QwtPlotCurve::setSamples(x, y) ;
        QPen pen(QColor(255, 139, 15, 100)) ;
        pen.setWidth(5) ;
        pen.setCapStyle(Qt::RoundCap);
        setPen(pen) ;
        pen.setColor(QColor(255, 139, 15)) ;
```

```
        setSymbol(new QwtSymbol(QwtSymbol::Ellipse, pen.brush(), (QPen) pen.color(), QSize(5, 5))) ;
}
void specRange::pointMoved(const int& point, const double& x, const double& y)
{
        if(point == 0)
                setMinValue(x) ;
        else
                setMaxValue(x) ;
        if(!isValid())
                setInterval(maxValue(), minValue()) ;
        yVal = y ;
        refreshPlotData() ;
}
void specRange::writeToStream(QDataStream& out) const
{
        specCanvasItem::writeToStream(out) ;
        out << qreal(yVal) << qreal(minValue()) << qreal(maxValue()) ;
}
void specRange::readFromStream(QDataStream& in)
{
        qreal y, min, max ;
        in >> y >> min >> max ;
        yVal = y ;
        setInterval(min, max, QwtInterval::IncludeBorders) ;
        refreshPlotData();
}
void specRange::refreshPlotData()
{
        QVector<double> xarr, yarr ;
        xarr << minValue() << maxValue() ;
        yarr << yVal << yVal ;
        setSamples(xarr, yarr) ;
}
specRange::~specRange()
{
        detach();
}
```

src/plot/specspectrumplot.h

```
#ifndef SPECSPECTRUMPLOT_H
#define SPECSPECTRUMPLOT_H
#include "specplot.h"
#include <QHash>
#include <QList>
class QActionGroup ;
class specMultiCommand ;
class specRange ;
class specDataItem ;
class specModelItem ;
class specModel ;
class specSpectrumPlot : public specPlot
{
        Q_OBJECT
private:
        QAction* offsetAction,
                 *offlineAction,
                 *scaleAction,
                 *shiftAction,
                 *setReferenceAction,
                 *alignWithReferenceAction,
                 *addRangeAction,
                 *removeRangeAction,
                 *subInterpolatedAction,
                 *applyRangesAction ;
        QActionGroup* correctionActions, *alignmentActions ;
        CanvasPicker* correctionPicker, *alignmentPicker ;
public:
        enum move { NoMoveMode = 0,
                    Offset = 1,
                    Scale = 2,
                    Slope = 4
                  } ;
        Q_DECLARE_FLAGS(moveMode, move)
```

```
        moveMode manualAlignment, rangeAlignment ;
private:
        QHash<specCanvasItem*, QList<int> > pointHash ;
        specDataItem* reference ;
        void invalidateReference() ;
        bool correctionChecked() ;
        QList<specDataItem*> folderContent(specModelItem*) ;
        moveMode correctionsStatus() const ;
        void setCorrectionsStatus(moveMode) ;
public:
        explicit specSpectrumPlot(QWidget* parent = 0);
        ~specSpectrumPlot() ;
        QList<QAction*> actions() ;
        void attachToPicker(specCanvasItem*) ;
        void detachFromPicker(specCanvasItem*) ;
signals:
private slots:
        void correctionsChanged(QAction* action = 0) ;
        void alignmentChanged(QAction*) ;
        void pointMoved(specCanvasItem*, int point, double x, double y) ;
        void applyZeroRanges(specCanvasItem* range, int point, double x, double y) ;
        void applyZeroRanges() ;
        void multipleSubtraction() ;
        void setReference() ;
        void toggleAligning(bool on = true) ;
        void checkReferenceForScaling() ;
};
Q_DECLARE_OPERATORS_FOR_FLAGS(specSpectrumPlot::moveMode)
#endif
```

src/plot/specspectrumplot.cpp

```
unsigned int qHash(const double& d)
{
        return * ((unsigned int*)(&d)) ;
}
#include "specspectrumplot.h"
#include <QActionGroup>
#include <QBuffer>
#include <qwt_scale_draw.h>
#include "specmulticommand.h"
#include "specrange.h"
#include "specdataitem.h"
#include "canvaspicker.h"
#include "specfolderitem.h"
#include "specview.h"
#include "qwt_plot_renderer.h"
#include "utility-functions.h"
#include "specexchangefiltercommand.h"
#include "specactionlibrary.h"
#include "specexchangedatacommand.h"
#include <QMessageBox>
#include "specfiltergenerator.h"
specSpectrumPlot::moveMode specSpectrumPlot::correctionsStatus() const
{
        qDebug() << ((offsetAction->isChecked() ? Offset : NoMoveMode)
                     | (offlineAction->isChecked() ? Slope : NoMoveMode)
                     | (scaleAction->isChecked() ? Scale : NoMoveMode)) ;
        qDebug() << (offsetAction->isChecked() ? Offset : NoMoveMode)
                 << (offlineAction->isChecked() ? Slope : NoMoveMode)
                 << (scaleAction->isChecked() ? Scale : NoMoveMode) ;
        return (offsetAction->isChecked() ? Offset : NoMoveMode)
               | (offlineAction->isChecked() ? Slope : NoMoveMode)
               | (scaleAction->isChecked() ? Scale : NoMoveMode) ;
}
void specSpectrumPlot::setCorrectionsStatus(moveMode m)
{
        offsetAction->setChecked(m & Offset) ;
        offlineAction->setChecked(m & Slope) ;
        scaleAction->setChecked(m & Scale) ;
}
void specSpectrumPlot::checkReferenceForScaling()
{
        if(alignWithReferenceAction->isChecked() && !reference)
```

```
        {
                scaleAction->setChecked(false) ;
                scaleAction->setEnabled(false) ;
        }
        else
                scaleAction->setEnabled(true) ;
}
void specSpectrumPlot::toggleAligning(bool on)
{
        alignmentActions->setEnabled(on) ;
        if(on)
        {
                alignmentPicker = new CanvasPicker(this) ;
                alignmentPicker->setOwning() ;
                connect(alignmentPicker, SIGNAL(pointMoved(specCanvasItem*, int, double, double)), ►
                     this, SLOT(applyZeroRanges(specCanvasItem*, int, double, double))) ;
                manualAlignment = correctionsStatus() ;
                setCorrectionsStatus(rangeAlignment) ;
                removeRangeAction->setEnabled(false) ;
                applyRangesAction->setEnabled(false) ;
        }
        else
        {
                delete alignmentPicker ;
                alignmentPicker = 0 ;
                rangeAlignment = correctionsStatus() ;
                setCorrectionsStatus(manualAlignment);
        }
        alignWithReferenceAction->setEnabled(true) ;
}
void specSpectrumPlot::invalidateReference()
{
        if(reference) delete reference ;
        reference = 0 ;
        setReferenceAction->setToolTip(QString("Currently␣no␣reference␣to␣show."));
        subInterpolatedAction->setDisabled(true);
}
specSpectrumPlot::specSpectrumPlot(QWidget* parent) :
        specPlot(parent),
        correctionPicker(0),
        alignmentPicker(0),
        manualAlignment(NoMoveMode),
        rangeAlignment(NoMoveMode),
        reference(0)
{
        correctionActions = new QActionGroup(this) ;
        alignmentActions = new QActionGroup(this) ;
        alignWithReferenceAction = new QAction(QIcon(":/zeroCorrection.png"), tr("Align"), this) ;
        offsetAction = new QAction(QIcon(":/offset.png"), tr("Offset"), correctionActions) ;
        offlineAction = new QAction(QIcon(":/offline.png"), tr("Slope"), correctionActions) ;
        scaleAction = new QAction(QIcon(":/scale.png"), tr("Scale"), correctionActions) ;
        shiftAction = new QAction(QIcon(":/xshift.png"), tr("Shift␣x"), correctionActions) ;
        alignWithReferenceAction->setCheckable(true) ;
        alignWithReferenceAction->setChecked(false) ;
        addRangeAction = new QAction(QIcon(":/addZeroRange.png"), tr("Add␣aligning␣range"), ►
             alignmentActions) ;
        removeRangeAction = new QAction(QIcon(":/deleteZeroRange.png"), tr("Delete␣aligning␣range"),►
              alignmentActions) ;
        applyRangesAction = new QAction(QIcon(":/zeroRange.png"), tr("Apply␣current␣zero␣ranges"), ►
             alignmentActions) ;
        applyRangesAction->setShortcut(tr("a"));
        setReferenceAction = new QAction(QIcon(":/data.png"), tr("Set␣reference"), this) ;
        setReferenceAction->setShortcut(tr("r"));
        subInterpolatedAction = new QAction(QIcon(":/multiminus.png"), tr("Subtract␣Reference"), ►
             this) ;
        subInterpolatedAction->setShortcut(tr("s"));
        correctionActions->setExclusive(false);
        correctionActions->addAction(offsetAction) ;
        correctionActions->addAction(offlineAction) ;
        correctionActions->addAction(scaleAction) ;
        correctionActions->addAction(shiftAction) ;
        foreach(QAction * action, correctionActions->actions())
        action->setCheckable(true) ;
        alignmentActions->setExclusive(false) ;
        alignmentActions->addAction(addRangeAction) ;
```

```
        alignmentActions->addAction(removeRangeAction) ;
        alignmentActions->addAction(applyRangesAction) ;
        connect(correctionActions, SIGNAL(triggered(QAction*)), this, SLOT(correctionsChanged(►
            QAction*))) ;
        connect(alignWithReferenceAction, SIGNAL(toggled(bool)), shiftAction, SLOT(setDisabled(bool)►
            )) ;
        connect(alignWithReferenceAction, SIGNAL(toggled(bool)), this, SLOT(toggleAligning(bool))) ;
        connect(alignmentActions, SIGNAL(triggered(QAction*)), this, SLOT(alignmentChanged(QAction*)►
            )) ;
        connect(subInterpolatedAction, SIGNAL(triggered()), this, SLOT(multipleSubtraction())) ;
        connect(setReferenceAction, SIGNAL(triggered()), this, SLOT(setReference())) ;
        connect(alignWithReferenceAction, SIGNAL(toggled(bool)), this, SLOT(checkReferenceForScaling►
            ())) ;
        connect(setReferenceAction, SIGNAL(triggered()), this, SLOT(checkReferenceForScaling())) ;
        connect(applyRangesAction, SIGNAL(triggered()), this, SLOT(applyZeroRanges())) ;
        toggleAligning(false) ;
        rangeAlignment = (Offset | Slope) ;
        invalidateReference();
        offsetAction->setWhatsThis(tr("When␣this␣button␣is␣enabled,␣you␣can␣correct␣your␣data␣for␣►
            any␣offset␣by␣individually␣moving␣any␣point␣of␣a␣data␣curve␣to␣where␣it␣ought␣to␣be."));
        offlineAction->setWhatsThis(tr("By␣enabling␣this␣button,␣you␣enable␣detrending␣correction.␣␣►
            Pick␣any␣point␣of␣a␣data␣curve␣and␣move␣it␣to␣subtract␣a␣trend."));
        scaleAction->setWhatsThis(tr("This␣button␣lets␣you␣scale␣data␣by␣moving␣a␣data␣point."));
        shiftAction->setWhatsThis(tr("This␣button␣enables␣shifting␣data␣along␣the␣x␣axis␣by␣moving␣a►
            ␣data␣point."));
        subInterpolatedAction->setWhatsThis(tr("Subtract␣the␣reference␣data␣from␣selected␣data␣sets␣►
            (applying␣interpolation␣if␣necessary)."));
        setReferenceAction->setWhatsThis(tr("Sets␣the␣reference␣for␣interpolated␣subtractions␣and/or►
            ␣for␣aligning␣other␣data␣sets."));
        alignWithReferenceAction->setWhatsThis(tr("Enables␣aligning␣selected␣data␣sets␣with␣the␣►
            reference␣defined␣(or␣with␣the␣x␣axis␣if␣no␣reference␣has␣been␣set).\nFor␣aligning,␣a␣►
            linear␣function␣will␣be␣subtracted,␣unless␣the␣correction␣has␣been␣limited␣to␣►
            subtracting␣an␣offset."));
        addRangeAction->setWhatsThis(tr("Add␣a␣range␣for␣alignment.␣␣The␣alignment␣correction␣of␣►
            data␣sets␣selected␣will␣be␣calculated␣based␣on␣the␣points␣that␣lie␣within␣at␣least␣one␣►
            of␣those␣ranges.")) ;
        removeRangeAction->setWhatsThis(tr("Delete␣a␣range␣for␣alignment.␣␣The␣alignment␣correction␣►
            of␣data␣sets␣selected␣will␣be␣calculated␣based␣on␣the␣points␣that␣lie␣within␣at␣least␣►
            one␣of␣those␣ranges."));
        applyRangesAction->setWhatsThis(tr("Applies␣the␣currently␣active␣zero␣ranges")) ;
        setObjectName("mainPlot");
}
QList<specDataItem*> specSpectrumPlot::folderContent(specModelItem* folder)
{
        QList<specDataItem*> content ;
        if(dynamic_cast<specDataItem*>(folder))
                content << (specDataItem*) folder ;
        for(int i = 0 ; i < folder->children() ; ++i)
        {
                specModelItem* item = ((specFolderItem*) folder)->child(i) ;
                if(dynamic_cast<specDataItem*>(item))
                        content << (specDataItem*) item ;
                else if(item->isFolder())
                        content << folderContent((specFolderItem*) item) ;
        }
        return content ;
}
void specSpectrumPlot::setReference()
{
        invalidateReference();
        QModelIndexList referenceItems = view->getSelection() ;
        QList<specDataItem*> referenceDataItems ;
        for(int i = 0 ; i < referenceItems.size() ; ++i)
                referenceDataItems << folderContent(view->model()->itemPointer(referenceItems[i])) ;
        if(referenceDataItems.isEmpty())
                return ;
        reference = new specDataItem(QVector<specDataPoint>(), QHash<QString, specDescriptor>()) ;
        for(int i = 0 ; i < referenceDataItems.size() ; ++i)
                reference->operator += (* (referenceDataItems[i])) ;
        reference->flatten();
        reference->revalidate();
        reference->setPen(QPen(Qt::red));
        QwtPlot toolTipPlot ;
        toolTipPlot.setAutoDelete(false) ;
        reference->attach(&toolTipPlot) ;
```

```
172         toolTipPlot.replot();
            QFont font = axisFont(QwtPlot::xBottom) ;
            font.setPixelSize(8) ;
            toolTipPlot.setAxisFont(QwtPlot::xBottom, font);
            toolTipPlot.axisScaleDraw(QwtPlot::xBottom)->setSpacing(2) ;
177         toolTipPlot.axisScaleDraw(QwtPlot::xBottom)->setTickLength(QwtScaleDiv::MajorTick, 4) ;
            toolTipPlot.axisScaleDraw(QwtPlot::xBottom)->setTickLength(QwtScaleDiv::MediumTick, 3) ;
            toolTipPlot.axisScaleDraw(QwtPlot::xBottom)->setTickLength(QwtScaleDiv::MinorTick, 2) ;
            font = axisFont(QwtPlot::yLeft) ;
            font.setPixelSize(8) ;
182         toolTipPlot.setAxisFont(QwtPlot::yLeft, font) ;
            toolTipPlot.axisScaleDraw(QwtPlot::yLeft)->setSpacing(2) ;
            toolTipPlot.axisScaleDraw(QwtPlot::yLeft)->setTickLength(QwtScaleDiv::MajorTick, 4) ;
            toolTipPlot.axisScaleDraw(QwtPlot::yLeft)->setTickLength(QwtScaleDiv::MediumTick, 3) ;
            toolTipPlot.axisScaleDraw(QwtPlot::yLeft)->setTickLength(QwtScaleDiv::MinorTick, 2) ;
187         toolTipPlot.replot();
            QImage plotImage(200, 100, QImage::Format_ARGB32) ;
            plotImage.fill(0);
            QwtPlotRenderer renderer ;
            renderer.setDiscardFlag(QwtPlotRenderer::DiscardBackground, true);
192         renderer.setDiscardFlag(QwtPlotRenderer::DiscardCanvasBackground, true) ;
            renderer.setLayoutFlags(QwtPlotRenderer::DefaultLayout);
            renderer.renderTo(&toolTipPlot, plotImage) ;
            QByteArray byteArray ;
            QBuffer buffer(&byteArray) ;
197         buffer.open(QIODevice::WriteOnly) ;
            plotImage.save(&buffer, "PNG") ;
            setReferenceAction->setToolTip(QString("Momentane␣Referenz:<br><img␣src=\"data:image/png;►
                base64,%1\"></img>").arg(QString(buffer.data().toBase64())));
            subInterpolatedAction->setEnabled(true) ;
        }
202     void specSpectrumPlot::alignmentChanged(QAction* action)
        {
            if(action == addRangeAction)
            {
                    double min = axisScaleDiv(QwtPlot::xBottom).lowerBound(), max = axisScaleDiv(QwtPlot►
                        ::xBottom).upperBound() ;
207                 specRange* newRange = new specRange(min + .1 * (max - min), max - .1 * (max - min),
                                                        (axisScaleDiv(QwtPlot::yLeft).lowerBound() +
                                                         axisScaleDiv(QwtPlot::yLeft).upperBound()) / ►
                                                             2.) ;
                    newRange->attach(this) ;
                    alignmentPicker->addSelectable(newRange) ;
212         }
            else if(action == removeRangeAction)
                    alignmentPicker->removeSelected();
            bool enableRemove = alignmentPicker && alignmentPicker->countSelectable() ;
            removeRangeAction->setEnabled(enableRemove);
217         applyRangesAction->setEnabled(enableRemove) ;
        }
        bool specSpectrumPlot::correctionChecked()
        {
            bool checked = false ;
222         foreach(QAction * action, correctionActions->actions())
            checked = checked || action->isChecked() ;
            return checked ;
        }
        void specSpectrumPlot::correctionsChanged(QAction* action)
227     {
            if(alignWithReferenceAction->isChecked() && !offsetAction->isChecked() && !offlineAction->►
                isChecked())
            {
                    if(action == offlineAction) offlineAction->setChecked(true) ;
                    else offsetAction->setChecked(true) ;
232         }
            if(correctionChecked() && !alignWithReferenceAction->isChecked())
            {
                    if(!correctionPicker)
                    {
237                         correctionPicker = new CanvasPicker(this) ;
                            connect(correctionPicker, SIGNAL(pointMoved(specCanvasItem*, int, double, ►
                                double)), this, SLOT(pointMoved(specCanvasItem*, int, double, double))) ►
                                ;
                            QList<specCanvasItem*> items ;
                            QwtPlotItemList onPlot = itemList(spec::spectrum) ;
```

```
                        foreach(QwtPlotItem * item, onPlot)
242                     items << dynamic_cast<specDataItem*>(item) ;
                        items.removeAll(0) ;
                        correctionPicker->addSelectable(items.toSet());
                }
        }
247     else
        {
                delete correctionPicker ;
                correctionPicker = 0 ;
                pointHash.clear();
252     }
}
void specSpectrumPlot::attachToPicker(specCanvasItem* item)
{
        if(correctionPicker)
257             correctionPicker->addSelectable(item) ;
}
void specSpectrumPlot::detachFromPicker(specCanvasItem* item)
{
        if(correctionPicker)
262             correctionPicker->removeSelectable(item) ;
}
void specSpectrumPlot::pointMoved(specCanvasItem* item, int no, double x, double y)
{
        if(!view || !view->model()) return ;
267     specModelItem* modelItem = dynamic_cast<specModelItem*>(item) ;
        if(!modelItem) return ;
        QList<int>& selectedPoints = pointHash[item] ;
        selectedPoints.prepend(selectedPoints.contains(no) ? selectedPoints.takeAt(selectedPoints.►
            indexOf(no)) : no) ;
        if(!undoPartner()) return ;
272     QString failedCorrections ;
#define CHECKCORRECTIONPARAMETER(PARAMETER,SPECIALCONDITION,PHRASE,DEFAULT,ACTION) \
        if(!isfinite(PARAMETER) SPECIALCONDITION) { \
                failedCorrections += tr(PHRASE) + ":\t" + QString::number(PARAMETER) + "\n"; \
                PARAMETER = DEFAULT ; \
277             ACTION->setChecked(false) ; }
        double shift = shiftAction->isChecked() ? x - item->sample(no).x() : 0 ;
        CHECKCORRECTIONPARAMETER(shift, , "x␣shift", 0., shiftAction)
        double scale = 1, offset = 0, offline = 0 ;
        if(scaleAction->isChecked() || offsetAction->isChecked() || offlineAction->isChecked())
282     {
                QVector<QVector<double> > matrix ;
                for(int i = 0 ; i < selectedPoints.size() ; i++) matrix << QVector<double>() ;
                if(scaleAction->isChecked())
                        for(int i = 0 ; i < selectedPoints.size(); i++)
287                             matrix[i] << item->sample(selectedPoints[i]).y() ;
                if(offsetAction->isChecked())
                        for(int i = 0 ; i < selectedPoints.size(); i++)
                                matrix[i] << 1. ;
                if(offlineAction->isChecked())
292                     for(int i = 0 ; i < selectedPoints.size(); i++)
                                matrix[i] << item->sample(selectedPoints[i]).x() + shift ;
                for(int i = 0 ; i < matrix.size() ; i++)
                        while(matrix.size() < matrix[i].size())
                                matrix[i].resize(matrix[i].size() - 1) ;
297             while(matrix[0].size() < matrix.size())
                        matrix.resize(matrix.size() - 1) ;
                QVector<double> coeffs ;
                QVector<double> yVals ;
                yVals << y - (scaleAction->isChecked() ? 0. : item->sample(no).y()) ;
302             for(int i = 1 ; i < selectedPoints.size() && i < matrix.size() ; i++)
                        yVals << (scaleAction->isChecked() ? item->sample(selectedPoints[i]).y() : ►
                            0) ;
                coeffs = gaussjinv(matrix, yVals) ;
                int i = 0 ;
                scale = scaleAction->isChecked() && i < coeffs.size() ? coeffs[i++] : 1. ;
307             offset = offsetAction->isChecked() && i < coeffs.size() ? coeffs[i++] : 0. ;
                offline = offlineAction->isChecked() && i < coeffs.size() ? coeffs[i++] : 0. ;
                CHECKCORRECTIONPARAMETER(scale, || !isnormal(scale), "scaling", 1., scaleAction)
                CHECKCORRECTIONPARAMETER(offset, , "offset", 0, offsetAction)
                CHECKCORRECTIONPARAMETER(offline, , "slope", 0, offlineAction)
312     }
        if(!failedCorrections.isEmpty())
```

```
                QMessageBox::critical(0, tr("Aligning␣error"),
                                      tr("The␣following␣required␣alignment␣parameters␣were␣illegal:\►
                                          n")
                                      + failedCorrections
                                      + tr("The␣parameters␣were␣disabled.")) ;
        specExchangeFilterCommand* command = new specExchangeFilterCommand ;
        command->setParentObject(view->model()) ;
        command->setItem(modelItem) ;
        command->setRelativeFilter(specDataPointFilter(offset, offline, scale, shift)) ;
        undoPartner()->push(command) ;
}
void specSpectrumPlot::applyZeroRanges(specCanvasItem* range, int point, double newX, double newY)
{
        ((specRange*) range)->pointMoved(point, newX, newY) ;
        applyZeroRanges();
}
void specSpectrumPlot::applyZeroRanges()
{
        QwtPlotItemList zeroRanges = itemList(spec::zeroRange) ;
        if(zeroRanges.isEmpty()) return ;
        QwtPlotItemList spectra = itemList(spec::spectrum) ;
        if(spectra.isEmpty()) return ;
        specFilterGenerator filterGenerator ;
        if (reference)
                filterGenerator.setReference(reference->dataMap()) ;
        filterGenerator.calcOffset(offsetAction->isChecked());
        filterGenerator.calcScale(scaleAction->isChecked()) ;
        filterGenerator.calcSlope(offlineAction->isChecked()) ;
        filterGenerator.setRanges(zeroRanges);
        specMultiCommand* zeroCommand = new specMultiCommand ;
        zeroCommand->setParentObject(view->model()) ;
        foreach(QwtPlotItem* s, spectra)
        {
                specModelItem* spectrum = dynamic_cast<specModelItem*>(s) ;
                if (!spectrum) continue ;
                specDataPointFilter filter = filterGenerator.generateFilter(spectrum) ;
                if (!filter.valid()) continue ;
                specExchangeFilterCommand* command = new specExchangeFilterCommand(zeroCommand) ;
                command->setParentObject(view->model()) ;
                command->setItem(spectrum) ;
                command->setRelativeFilter(filter) ;
        }
        QStringList rangeStrings ;
        foreach(QwtPlotItem* i, zeroRanges)
        {
                specRange* range = (specRange*) i ;
                rangeStrings << QString::number(range->minValue()) + "--" + QString::number(range->►
                    maxValue()) ;
        }
        zeroCommand->setText(tr("Apply␣range␣correction.␣Ranges:␣") + rangeStrings.join(",␣"));
        undoPartner()->push(zeroCommand) ;
        replot() ;
}
void specSpectrumPlot::multipleSubtraction()
{
        if(!reference) return ;
        QwtPlotItemList spectra = itemList(spec::spectrum) ;
        QMap<double, double> referenceSpectrum ;
        for(size_t i = 0 ; i < reference->dataSize() ; ++i)
                referenceSpectrum[reference->sample(i).x()] = reference->sample(i).y() ;
        specMultiCommand* subCommand = new specMultiCommand ;
        if(view && view->model())
                subCommand->setParentObject(view->model());
        for(int i = 0 ; i < spectra.size() ; ++i)
        {
                specDataItem* spectrum = dynamic_cast<specDataItem*>(spectra[i]) ;
                if(!spectrum) continue ;
                QVector<specDataPoint> data = spectrum->allData() ;
                spectrum->applyCorrection(data) ;
                QMap<double, double>::iterator lower, upper ;
                for(int i = 0 ; i < data.size() ; ++i)
                {
                        lower = referenceSpectrum.lowerBound(data[i].nu) ;
                        if(lower == referenceSpectrum.end()) continue ;
                        double toSub = 0 ;
```

```
                        if(lower.key() == data[i].nu)
                                toSub = lower.value() ;
                        else
                        {
                                if(lower == referenceSpectrum.begin()) continue ;
                                upper = lower-- ;
                                toSub = lower.value() + (upper.value() - lower.value())
                                        * (data[i].nu - lower.key())
                                        / (upper.key() - lower.key());
                        }
                        data[i].sig -= toSub ;
                }
                spectrum->reverseCorrection(data) ;
                specExchangeDataCommand* command = new specExchangeDataCommand(subCommand) ;
                if(view && view->model())
                {
                        command->setParentObject(view->model());
                        command->setItem(spectrum, data);
                }
        }
        subCommand->setText(tr("Subtract reference"));
        undoPartner()->push(subCommand) ;
        replot() ;
}
QList<QAction*> specSpectrumPlot::actions()
{
        return specPlot::actions() << correctionActions->actions()
               << setReferenceAction
               << alignWithReferenceAction
               << alignmentActions->actions()
               << subInterpolatedAction ;
}
specSpectrumPlot::~specSpectrumPlot()
{
        if(correctionPicker) correctionPicker->purgeSelectable();
        delete correctionPicker ;
        delete alignmentPicker ;
}
```

src/plot/speczoomer.h

```
#ifndef SPECZOOMER_H
#define SPECZOOMER_H
#include <qwt_plot_zoomer.h>
class specZoomer : public QwtPlotZoomer
{
public:
        specZoomer(QWidget*);
        virtual QwtText trackerText(const QPointF& pos) const ;
        void changeZoomBase(const QRectF&) ;
        ~specZoomer();
};
#endif
```

src/plot/speczoomer.cpp

```
#include "speczoomer.h"
#include <QTextStream>
specZoomer::specZoomer(QWidget* canvas)
        : QwtPlotZoomer(canvas, false)
{
        setTrackerMode(AlwaysOn);
        setMousePattern(QwtEventPattern::MouseSelect1, Qt::MidButton) ;
        setMousePattern(QwtEventPattern::MouseSelect4, Qt::MidButton) ;
        setMousePattern(QwtEventPattern::MouseSelect2, Qt::MidButton, Qt::ControlModifier);
        setMousePattern(QwtEventPattern::MouseSelect3, Qt::RightButton);
        setRubberBandPen(QColor(Qt::darkBlue)) ;
        setTrackerPen(QColor(Qt::darkBlue)) ;
}
void specZoomer::changeZoomBase(const QRectF& rect)
{
        if(zoomBase() == rect) return ;
```

```
        QStack<QRectF> stack = zoomStack() ;
        stack.remove(0) ;
        for(QStack<QRectF>::size_type i = 0 ; i < stack.size() ; i++)
                if(! rect.contains(stack[i]))
                        stack.remove(i--) ;
        stack.prepend(rect) ;
        setZoomStack(stack, stack.indexOf(zoomRect())) ;
}
QwtText specZoomer::trackerText(const QPointF& pos) const
{
        QColor bg(Qt::white);
#if QT_VERSION >= 0x040300
        bg.setAlpha(200);
#endif
        QwtText text ;
        text.setText(QString("%1, %2").arg(pos.x(), 0, 'f', 2).arg(pos.y(), 0, 'f', 3)) ;
        text.setBackgroundBrush(QBrush(bg));
        return text;
}
specZoomer::~specZoomer()
{
}
```

src/specappwindow.h

```
#ifndef SPECAPPWINDOW_H
#define SPECAPPWINDOW_H
#include <QMainWindow>
#include <QtGui>
#include <QList>
#include "specplotwidget.h"
#include <QSettings>
class specShortcutDialog ;
class specAppWindow : public QMainWindow
{
        Q_OBJECT
public:
        specAppWindow();
        ~specAppWindow();
private slots:
        void newFile();
        void openFile();
        void openFile(const QString&) ;
protected:
        void closeEvent(QCloseEvent*) ;
private:
        QSettings settings ;
        void createActions();
        void createMenus();
        void createToolBars() ;
        void addDock(specPlotWidget*) ;
        QMenu* fileMenu;
        QMenu* helpMenu;
        QToolBar* fileToolBar;
        QAction* newAction;
        QAction* openAction;
        QAction* whatsThisAction ;
        QAction* restoreSessionAction ;
        QAction* shortCutAction ;
private slots:
        void about() ;
        void whatsThisMode() ;
        void editShortcuts() ;
};
#endif
```

src/specappwindow.cpp

```
#include "specappwindow.h"
#include <QFile>
#include "specshortcutdialog.h"
#include "names.h"
```

```
specAppWindow::specAppWindow()
        : QMainWindow(),
          settings()
{
        setAnimated(false) ;
        setDockOptions(QMainWindow::AllowTabbedDocks) ;
        setDockNestingEnabled(true) ;
        createActions();
        createMenus();
        createToolBars();
        restoreGeometry(settings.value("mainWindow/geometry").toByteArray()) ;
        setCentralWidget(0);
        setObjectName("Application␣window") ;
        setWindowTitle("SpecDataElement");
        setWindowFlags(windowFlags() | Qt::WindowContextHelpButtonHint);
        setWhatsThis("This␣is␣the␣main␣application␣window.␣␣It␣can␣be␣used␣for␣docking␣data␣windows,►
            ␣log␣windows,␣and␣kinetic␣windows␣to␣it.␣to␣it.\nStart␣by␣creating␣a␣new␣file␣or␣by␣►
            opening␣a␣saved␣file.␣Use␣the␣\"What's␣this?\"␣help␣from␣the␣title␣bar␣for␣further␣hints►
            .");
        setMinimumSize(300, 300);
        if(restoreSessionAction->isChecked())
                foreach(QString fileName, settings.value("mainWindow/previousSessionFiles").►
                    toStringList())
                openFile(fileName) ;
}
specAppWindow::~specAppWindow()
{
}
void specAppWindow::closeEvent(QCloseEvent* event)
{
        event->ignore() ;
        QStringList openFileNames ;
        foreach(specPlotWidget * plotWidget, findChildren<specPlotWidget*>())
        {
                openFileNames << plotWidget->windowFilePath() ;
                if(!plotWidget->close()) return ;
        }
        settings.setValue("mainWindow/previousSessionFiles",
                          restoreSessionAction->isChecked() ?
                          openFileNames : QVariant());
        settings.setValue("mainWindow/geometry", saveGeometry()) ;
        settings.setValue("mainWindow/sessionRestoration", restoreSessionAction->isChecked()) ;
        event->accept();
}
void specAppWindow::newFile()
{
        addDock(new specPlotWidget(this)) ;
}
void specAppWindow::addDock(specPlotWidget* newDock)
{
        specPlotWidget* inAreaWidget = 0 ;
        foreach(specPlotWidget * widget, findChildren<specPlotWidget*>())
        if(dockWidgetArea(widget) == Qt::LeftDockWidgetArea)
                inAreaWidget = widget ;
        addDockWidget(Qt::LeftDockWidgetArea, newDock) ;
        if(inAreaWidget)
                tabifyDockWidget(inAreaWidget, newDock);
        specShortcutDialog *shortcutDialog = findChild<specShortcutDialog*>() ;
        if (shortcutDialog)
                shortcutDialog->assignShortcuts();
}
void specAppWindow::openFile()
{
        openFile(QFileDialog::getOpenFileName(this, "Name?", "", "spec-Dateien␣(*.spec)"));
}
void specAppWindow::openFile(const QString& fileName)
{
        if(fileName != "" && QFile::exists(fileName))
        {
                specPlotWidget* newWidget = new specPlotWidget(this) ;
                newWidget->read(fileName) ;
                addDock(newWidget) ;
        }
}
void specAppWindow::createActions()
```

```
{
        newAction = new QAction(QIcon::fromTheme("document-new"), tr("&New"), this);
        newAction->setShortcut(tr("Ctrl+N"));
        newAction->setStatusTip(tr("Create a new file"));
        connect(newAction, SIGNAL(triggered()), this, SLOT(newFile()));
        openAction = new QAction(QIcon::fromTheme("document-open"), tr("&Open"), this);
        openAction->setShortcut(tr("Ctrl+O"));
        openAction->setStatusTip(tr("Open an existing file"));
        connect(openAction, SIGNAL(triggered()), this, SLOT(openFile()));
        restoreSessionAction = new QAction(QIcon::fromTheme("document-revert"), tr("Session &►
            restoration"), this) ;
        restoreSessionAction->setStatusTip(tr("Toggle saving of session information")) ;
        restoreSessionAction->setCheckable(true) ;
        restoreSessionAction->setChecked(settings.value("mainWindow/sessionRestoration", false).►
            toBool()) ;
        shortCutAction = new QAction(QIcon::fromTheme("input-keyboard"), tr("Shortcuts..."), this) ;
        connect(shortCutAction, SIGNAL(triggered()), this, SLOT(editShortcuts())) ;
        whatsThisAction = new QAction(QIcon::fromTheme("help-contextual"), tr("&What's this"), this) ;
        whatsThisAction->setStatusTip(tr("What's this?")) ;
        connect(whatsThisAction, SIGNAL(triggered()), this, SLOT(whatsThisMode())) ;
}
void specAppWindow::editShortcuts()
{
        specShortcutDialog *shortcutDialog = findChild<specShortcutDialog*>() ;
        if (shortcutDialog) shortcutDialog->exec();
}
void specAppWindow::whatsThisMode()
{
        QWhatsThis::enterWhatsThisMode() ;
}
void specAppWindow::createToolBars()
{
        fileToolBar = addToolBar(tr("File toolbar"));
        fileToolBar->addAction(newAction);
        fileToolBar->addAction(openAction);
        fileToolBar->addAction(shortCutAction) ;
        fileToolBar->addAction(restoreSessionAction) ;
        fileToolBar->addAction(whatsThisAction) ;
}
void specAppWindow::createMenus()
{
        fileMenu = menuBar()->addMenu(tr("&File"));
        fileMenu->addAction(newAction);
        fileMenu->addAction(openAction);
        fileMenu->addAction(shortCutAction) ;
        fileMenu->addAction(restoreSessionAction) ;
        helpMenu = menuBar()->addMenu(tr("&Help")) ;
        helpMenu->addAction(whatsThisAction) ;
        QAction* aboutQtAction = new QAction(tr("About &Qt..."), helpMenu) ;
        helpMenu->addAction(aboutQtAction) ;
        QAction* aboutAction = new QAction(tr("&About..."), helpMenu) ;
        helpMenu->addAction(aboutAction) ;
        connect(aboutQtAction, SIGNAL(triggered()), qApp, SLOT(aboutQt())) ;
        connect(aboutAction, SIGNAL(triggered()), this, SLOT(about())) ;
}
void specAppWindow::about()
{
        QMessageBox::about(this, tr("About SpecDataElement"),
                           tr("This is a simple program for efficiently managing two dimensional ►
                               data and keeping track of experimental logs.\n\n"
                              "It makes use of the following libraries:\n"
                              "- Qt 4.8 (qt.digia.com)\n"
                              "- Qwt 6 (qwt.sourceforge.net)\n"
                              "- muParser 2.2 (muparser.sourceforge.net)\n"
                              "- lmfit 3.3 (joachimwuttke.de/lmfit)")
                           + tr("\n\nVersion ID is: ") + QString(STRINGIFYMACRO(GITSHA1HASH))) ;
}
```

src/specdockwidget.h

```
#ifndef SPECDOCKWIDGET_H
#define SPECDOCKWIDGET_H
#include <QDockWidget>
#include <QString>
```

```
#include <QDataStream>
#include "specstreamable.h"
#include <QMap>
#include <QItemSelection>
class specActionLibrary ;
class specPlot ;
class specDockWidget : public QDockWidget
{
        Q_OBJECT
private:
        QString widgetTypeName ;
        QMap<specStreamable::type, int> selectedTypes ;
        specPlot* Plot ;
        bool changingTitle ;
private slots:
        void selectionChanged(const QItemSelection& selected, const QItemSelection& deselected) ;
protected:
        void changeEvent(QEvent* event) ;
        virtual QList<QWidget*> mainWidgets() const = 0 ;
public:
        void setupWindow(specActionLibrary* actions) ;
        explicit specDockWidget(QString type, QWidget* parent = 0, bool floating = true);
};
#endif
```

src/specdockwidget.cpp

```
#include "specdockwidget.h"
#include <QEvent>
#include <QFileInfo>
#include <QMainWindow>
#include <QToolBar>
#include "specactionlibrary.h"
#include "specsplitter.h"
#include "specview.h"
#include "specitemaction.h"
#define APPLYTOSUBDOCKS foreach(specDockWidget* sub, subDocks)
specDockWidget::specDockWidget(QString type, QWidget* parent, bool floating) :
        QDockWidget(parent),
        widgetTypeName(type),
        Plot(0),
        changingTitle(false)
{
        setFloating(floating);
        setVisible(false) ;
}
void specDockWidget::setupWindow(specActionLibrary* actions)
{
        QMainWindow* widget = new QMainWindow(this) ;
        widget->setWindowFlags(Qt::Widget);
        setWidget(widget) ;
        QList<QWidget*> innerWidgets = mainWidgets() ;
        if(innerWidgets.size() == 1)
                widget->setCentralWidget(innerWidgets.first()) ;
        else
        {
                specSplitter* splitter = new specSplitter(Qt::Vertical, this) ;
                foreach(QWidget * innerWidget, innerWidgets)
                splitter->insertWidget(0, innerWidget);
                widget->setCentralWidget(splitter);
        }
        if(!actions)
                return ;
        QList<QWidget*> allWidgets ;
        allWidgets << this << innerWidgets ;
        foreach(QWidget * w, allWidgets)
        {
                specView* view = dynamic_cast<specView*>(w);
                specPlot* plot = dynamic_cast<specPlot*>(w) ;
                if(view && view->model())
                {
                        view->setActionLibrary(actions) ;
                        actions->addDragDropPartner(view->model());
```

```
47                          connect(view->selectionModel(), SIGNAL(selectionChanged(QItemSelection, ►
                                QItemSelection)),
                                    this, SLOT(selectionChanged(QItemSelection, QItemSelection))) ;
                    }
                    if(plot)
                    {
52                          Plot = plot ;
                            actions->addPlot(plot) ;
                    }
                    QToolBar* toolbar = actions->toolBar(w) ;
                    if(toolbar->actions().isEmpty())
57                          delete toolbar ;
                    else
                            widget->addToolBar(toolbar);
            }
            specView* view = findChild<specView*>() ;
62          if(!view) return ;
            selectionChanged(view->selectionModel()->selection(), QItemSelection());
    }
    void specDockWidget::changeEvent(QEvent* event)
    {
67          if(changingTitle) return ;
            if(event->type() == QEvent::WindowTitleChange)
            {
                    changingTitle = true ;
                    setWindowTitle(QFileInfo(windowFilePath()).fileName()
72                                  + QLatin1String("[*]")
                                    + QLatin1Char('␣') + QChar(0x2014) + QLatin1Char('␣')
                                    + widgetTypeName) ;
                    changingTitle = false ;
                    event->accept();
77          }
            QDockWidget::changeEvent(event) ;
            toggleViewAction()->setText(tr("Show␣\"") + widgetTypeName + tr("\"␣window"));
    }
    void specDockWidget::selectionChanged(const QItemSelection& selected, const QItemSelection& ►
        deselected)
82  {
            foreach(QModelIndex index, deselected.indexes())
            {
                    if(!index.isValid()) continue ;
                    if(index.column()) continue ;
87                  specModelItem* pointer = (specModelItem*) index.internalPointer() ;
                    pointer->detach();
                    -- selectedTypes[pointer->typeId()] ;
            }
            foreach(QModelIndex index, selected.indexes())
92          {
                    if(!index.isValid()) continue ;
                    if(index.column()) continue ;
                    specModelItem* pointer = (specModelItem*) index.internalPointer() ;
                    pointer->attach(Plot);
97                  ++ selectedTypes[pointer->typeId()] ;
            }
            if(Plot) Plot->replot();
            foreach(specItemAction * action, findChildren<specItemAction*>())
            {
102                 QList<specStreamable::type> ts = action->requiredTypes() ;
                    bool enable = ts.isEmpty();
                    foreach(specStreamable::type t, ts)
                    enable = enable || selectedTypes[t] ;
                    action->setEnabled(enable && action->requirements()) ;
107         }
    }
```

src/specplotwidget.h

```
    #ifndef SPECPLOTWIDGET_H
2   #define SPECPLOTWIDGET_H
    #include "specdockwidget.h"
    #include <QString>
    class specSpectrumPlot ;
    class specDataView ;
7   class QAction ;
```

```
class specKineticWidget ;
class specActionLibrary ;
class specLogWidget ;
class QFile ;
class specView ;
class QItemSelection ;
class QToolBar ;
class specPlotWidget : public specDockWidget
{
        Q_OBJECT
private:
        specActionLibrary* actions ;
        specDataView* items ;
        specKineticWidget* kineticWidget ;
        specLogWidget* logWidget ;
        specSpectrumPlot* plot ;
        QAction* saveAction,
                 *kineticsAction,
                 *saveAsAction,
                 *logAction,
                 *undoViewAction,
                 *purgeUndoAction;
        specDockWidget* undoViewWidget ;
        void createWhatsThis() ;
        void setConnections() ;
        void changeFileName(const QString&) ;
        void changeEvent(QEvent* event) ;
        QList<specDockWidget*> subDocks ;
        QString versionString() const ;
private slots:
        void svgModification(bool) ;
protected :
        void closeEvent(QCloseEvent*) ;
        QList<QWidget*> mainWidgets() const ;
public:
        specPlotWidget(QWidget* parent = 0);
        ~specPlotWidget();
        void read(QString fileName) ;
        specView* mainView() ;
        QToolBar* createToolbar();
public slots :
        bool saveFile() ;
};
#endif
```

src/specplotwidget.cpp

```
#include "specplotwidget.h"
#include <QToolBar>
#include <QAction>
#include <QSplitter>
#include <QFile>
#include <QFileInfo>
#include <QMessageBox>
#include <QMainWindow>
#include "specsplitter.h"
#include "specgenericmimeconverter.h"
#include "speclogtodataconverter.h"
#include "spectextmimeconverter.h"
#include "specmimefileimporter.h"
#include "speckineticwidget.h"
#include "specactionlibrary.h"
#include "specdataview.h"
#include "specspectrumplot.h"
#include "speclogwidget.h"
#include "canvaspicker.h"
#include <QUndoView>
#include <QCloseEvent>
#include "bzipiodevice.h"
#include <QBuffer>
#include <QProgressDialog>
#include <QLabel>
#include "names.h"
specPlotWidget::specPlotWidget(QWidget* parent)
```

```
        : specDockWidget(tr("Data"), parent, false),
          actions(new specActionLibrary(this)),
          items(new specDataView(this)),
          kineticWidget(new specKineticWidget(parent)),
          logWidget(new specLogWidget(parent)),
          plot(new specSpectrumPlot(this)),
          saveAction(new QAction(QIcon::fromTheme("document-save"), tr("Save"), this)),
          kineticsAction(kineticWidget->toggleViewAction()),
          saveAsAction(new QAction(QIcon::fromTheme("document-save-as"), tr("Save as..."), this)),
          logAction(logWidget->toggleViewAction()),
          undoViewWidget(actions->undoWidget())
{
        setVisible(true) ;
        items->setModel(new specModel(items));
        plot->setView(items) ;
        new specGenericMimeConverter(items->model()) ;
        new specLogToDataConverter(items->model()) ;
        new specMimeFileImporter(items->model()) ;
        new specTextMimeConverter(items->model()) ;
        kineticWidget->view()->assignDataView(items) ;
        kineticsAction->setIcon(QIcon(":/kineticwindow.png"));
        logAction->setIcon(QIcon(":/logs.png"));
        setConnections() ;
        createWhatsThis();
        changeFileName(tr("untitled")) ;
        svgModification(false) ;
        kineticWidget->setupWindow(actions);
        logWidget->setupWindow(actions);
        setupWindow(actions);
        subDocks << logWidget << kineticWidget << undoViewWidget ;
        changeEvent(new QEvent(QEvent::WindowTitleChange));
        setObjectName(tr("Main window"));
        toggleViewAction()->setText(tr("Close file"));
}
void specPlotWidget::read(QString fileName)
{
        QFile file(fileName) ;
        if(!file.open(QFile::ReadOnly))
        {
                QMessageBox::critical(0, tr("Error opening"),
                                      tr("Cannot open file ")
                                      + fileName
                                      + tr(" for reading.")) ;
                return ;
        }
        quint64 check ;
        QDataStream inStream(&file) ;
        inStream >> check ;
        if(check != FILECHECKRANDOMNUMBER && check != FILECHECKCOMPRESSNUMBER)
        {
                QString version(file.readAll().right(versionString().size())) ;
                version = version.left(version.size()-1) ;
                QString refVersion = versionString().mid(1) ;
                QMessageBox::critical(0, tr("File error"),
                                      tr("File ")
                                      + fileName
                                      + tr(" does not seem to have the right format. or might be ►
                                          damaged.")
                                      + tr("\nIf it is indeed a spec-file, it might have been ►
                                          created by a different version of this program.  The file'►
                                          s version ID is: ")
                                      + version
                                      + "\n"
                                      + (version == refVersion ?
                                                tr("Since your program version is the same, it is ►
                                                     likely the file is broken.  You can still send ►
                                                     the complete file in for recovery, if possible.►
                                                     ")
                                              : tr("Your program's version ID is: ")
                                                + refVersion
                                      + tr("\nPlease try to obtain the appropriate version of the ►
                                          program."))
                                      + tr("\nFor support e-mail HVennekate@gmx.de \n\n")
                                      ) ;
                return ;
```

```
        }
        QByteArray fileContent = file.readAll() ;
        file.close();
        bzipIODevice* zipDevice = 0 ;
        QBuffer buffer(&fileContent) ;
        if(FILECHECKCOMPRESSNUMBER == check)
        {
                zipDevice = new bzipIODevice(&buffer) ;
                zipDevice->open(bzipIODevice::ReadOnly) ;
                inStream.setDevice(zipDevice);
        }
        else
        {
                buffer.open(QBuffer::ReadOnly) ;
                inStream.setDevice(&buffer) ;
        }
        QProgressDialog progress ;
        progress.setMaximum(30) ;
        progress.setMinimumDuration(300);
        progress.setWindowModality(Qt::WindowModal);
        progress.setCancelButton(0);
        progress.setWindowTitle(tr("Opening␣") + file.fileName());
        progress.setLabel(new QLabel(tr("Reading␣plot"))) ;
        progress.setValue(0);
        inStream >> *plot ;
        progress.setLabel(new QLabel(tr("Reading␣data␣items"))) ;
        progress.setValue(1);
        inStream >> *items ;
        progress.setLabel(new QLabel(tr("Reading␣log␣data"))) ;
        progress.setValue(2);
        inStream >> *logWidget ;
        progress.setLabel(new QLabel(tr("Reading␣meta␣items"))) ;
        progress.setValue(3) ;
        inStream >> *kineticWidget ;
        progress.setLabel(new QLabel(tr("Reading␣undo␣history")));
        progress.setValue(4);
        actions->setProgressDialog(&progress);
        inStream >> *actions ;
        progress.setValue(5);
        qint8 visibility = 0 ;
        inStream >> visibility ;
        if(zipDevice) zipDevice->releaseDevice();
        delete zipDevice ;
        changeFileName(fileName);
        qint8 i = 1 ;
        foreach(specDockWidget * subDock, subDocks)
        {
                subDock->setVisible(visibility & i);
                i *= 2 ;
        }
}
QToolBar* specPlotWidget::createToolbar()
{
        QToolBar* toolbar = new QToolBar(tr("Main"), this) ;
        toolbar-> addAction(saveAction) ;
        toolbar-> addAction(saveAsAction) ;
        toolbar-> addSeparator() ;
        toolbar-> addAction(logAction) ;
        toolbar-> addAction(kineticsAction) ;
        toolbar-> addAction(undoViewWidget->toggleViewAction()) ;
        return toolbar ;
}
void specPlotWidget::closeEvent(QCloseEvent* event)
{
        if(!isWindowModified())
        {
                event->accept() ;
                deleteLater();
                return ;
        }
        QMessageBox wantToSave ;
        wantToSave.setStandardButtons(QMessageBox::Yes | QMessageBox::No | QMessageBox::Cancel) ;
        wantToSave.setDefaultButton(QMessageBox::Cancel) ;
        wantToSave.setEscapeButton(QMessageBox::Cancel) ;
        wantToSave.setWindowTitle("Save␣data?") ;
```

```
        wantToSave.setText(QString("Do␣you␣want␣to␣save␣changes␣made␣to␣")
                           + windowFilePath() + tr("?")) ;
        wantToSave.setIcon(QMessageBox::Warning) ;
        int wantToSaveResult = wantToSave.exec() ;
        event->ignore();
        if(QMessageBox::No == wantToSaveResult ||
                (QMessageBox::Yes == wantToSaveResult
                 && saveFile()))
        {
                event->accept();
                deleteLater();
        }
}
bool specPlotWidget::saveFile()
{
        QFile file(windowFilePath() == tr("untitled") || sender() == saveAsAction ?
                   QFileDialog::getSaveFileName(this, "Name?", "", "spec-Dateien␣(*.spec)") :
                   windowFilePath()) ;
        if(file.fileName().isEmpty()) return false ;
        QBuffer buffer ;
        buffer.open(QBuffer::WriteOnly) ;
        QDataStream out(&buffer) ;
        out << quint64(FILECHECKCOMPRESSNUMBER) ;
        out.setDevice(0);
        bzipIODevice zipDevice(&buffer) ;
        zipDevice.open(bzipIODevice::WriteOnly) ;
        QDataStream zipOut(&zipDevice) ;
        QProgressDialog progress ;
        progress.setCancelButton(0);
        progress.setMinimumDuration(300);
        progress.setWindowModality(Qt::WindowModal);
        progress.setWindowTitle(tr("Saving␣") + file.fileName());
        progress.setMaximum(30);
        progress.setLabel(new QLabel(tr("Saving␣plot"))) ;
        progress.setValue(0);
        zipOut << *plot ;
        progress.setLabel(new QLabel(tr("Saving␣data␣items"))) ;
        progress.setValue(1);
        zipOut << *items ;
        progress.setLabel(new QLabel(tr("Saving␣log␣data"))) ;
        progress.setValue(2);
        zipOut << *logWidget ;
        progress.setLabel(new QLabel(tr("Saving␣meta␣data"))) ;
        progress.setValue(3) ;
        zipOut << *kineticWidget ;
        progress.setLabel(new QLabel(tr("Saving␣undo␣history"))) ;
        progress.setValue(4);
        actions->setProgressDialog(&progress) ;
        zipOut << *actions ;
        progress.setValue(progress.maximum());
        qint8 visibility = 0, i = 1 ;
        foreach(specDockWidget * subDock, subDocks)
        {
                visibility += i * subDock->isVisible() ;
                i *= 2 ;
        }
        zipOut << visibility ;
        zipDevice.close() ;
        zipDevice.releaseDevice() ;
        buffer.buffer().append(versionString()) ;
        qDebug() << QString("GITSHA1HASH") ;
        buffer.close();
        if(!file.open(QFile::WriteOnly))
        {
                QMessageBox::critical(0, tr("Write␣error"), tr("Could␣not␣open␣file␣")
                                      + file.fileName()
                                      + tr("for␣writing.")) ;
                return false ;
        }
        if(file.write(buffer.data()) == -1)
        {
                QMessageBox::critical(0, tr("Severe␣error!"), tr("Error␣writing␣file!␣␣Data␣could␣►
                    not␣be␣saved!")) ;
                return false ;
        }
```

```
        file.close();
        changeFileName(file.fileName());
        return true ;
}
QString specPlotWidget::versionString() const
{
        return QString("/") + STRINGIFYMACRO(GITSHA1HASH) ;
}
specPlotWidget::~specPlotWidget()
{
        foreach(specDockWidget * subDock, subDocks)
        delete subDock ;
        QMainWindow* p = qobject_cast<QMainWindow*> (parentWidget()) ;
        if(p) p->removeDockWidget(this) ;
}
void specPlotWidget::setConnections()
{
        connect(saveAction, SIGNAL(triggered()), this, SLOT(saveFile()));
        connect(saveAsAction, SIGNAL(triggered()), this, SLOT(saveFile()));
        connect(kineticWidget->internalPlot(), SIGNAL(replotted()), plot, SLOT(replot())) ;
        connect(kineticWidget->internalPlot(), SIGNAL(metaRangeModified(specCanvasItem*, int, double►
            , double)), kineticWidget->view(), SLOT(rangeModified(specCanvasItem*, int, double, ►
            double))) ;
        connect(plot, SIGNAL(metaRangeModified(specCanvasItem*, int, double, double)), kineticWidget►
            ->view(), SLOT(rangeModified(specCanvasItem*, int, double, double))) ;
        connect(plot->svgAction(), SIGNAL(toggled(bool)), this, SLOT(svgModification(bool))) ;
        connect(actions, SIGNAL(stackModified(bool)), this, SLOT(setWindowModified(bool))) ;
}
void specPlotWidget::svgModification(bool mod)
{
        if(mod) connect(plot->svgPicker(), SIGNAL(pointMoved(specCanvasItem*, int, double, double)),►
             items->model(), SLOT(svgMoved(specCanvasItem*, int, double, double))) ;
        else disconnect(plot->svgPicker(), SIGNAL(pointMoved(specCanvasItem*, int, double, double)),►
             items->model(), SLOT(svgMoved(specCanvasItem*, int, double, double))) ;
        plot->svgPicker()->highlightSelectable(mod) ;
}
specView* specPlotWidget::mainView()
{
        return items ;
}
void specPlotWidget::changeFileName(const QString& name)
{
        setWindowFilePath(name);
        event(new QEvent(QEvent::WindowTitleChange)) ;
}
void specPlotWidget::changeEvent(QEvent* event)
{
        if(event->type() == QEvent::ModifiedChange)
                foreach(specDockWidget * subDock, subDocks)
                subDock->setWindowModified(isWindowModified());
        if(event->type() == QEvent::WindowTitleChange)
        {
                foreach(specDockWidget * subDock, subDocks)
                {
                        subDock->setWindowFilePath(windowFilePath()) ;
                        subDock->setWindowTitle(QString()) ;
                }
        }
        specDockWidget::changeEvent(event) ;
}
QList<QWidget*> specPlotWidget::mainWidgets() const
{
        return QList<QWidget*>() << items << plot ;
}
void specPlotWidget::createWhatsThis()
{
        setWhatsThis(tr("Data␣Dock␣Window␣-␣This␣is␣the␣main␣window␣for␣managing␣data.␣␣Right-►
            clicking␣on␣it␣will␣change␣the␣arrangement␣of␣its␣contents␣from␣vertical␣to␣horizontal␣►
            and␣back.\nSince␣this␣is␣a␣dock␣window,␣you␣may␣remove␣it␣from␣the␣application's␣main␣►
            window␣and␣move␣it␣around␣within␣the␣main␣window.␣␣To␣do␣so,␣\"grab\"␣it␣by␣its␣title␣►
            bar␣and␣drag␣it␣to␣where␣you␣need␣it␣to␣be.\nThere␣are␣also␣a␣dock␣window␣for␣managing␣►
            logs␣and␣\"meta-data\"␣(i.e.␣data␣based␣on␣processing␣the␣primary␣data)␣associated␣with␣►
            this␣data␣dock␣window.␣␣To␣show␣these,␣click␣the␣log␣or␣meta␣buttons␣(to␣the␣right␣of␣►
            the␣save␣as␣button).\nTo␣start␣working,␣check␣out␣the␣help␣of␣the␣data␣list␣at␣the␣►
            bottom␣(or␣right)␣of␣this␣window␣or␣of␣the␣list␣in␣the␣log␣dock␣window."));
```

```
        saveAsAction->setWhatsThis(tr("Save As... - This button will allow you to save your work ►
            asking you for a file name explicitly."));
        saveAction->setWhatsThis(tr("Save - Clicking this button will save your work and only ask ►
            for a file name if none has been specified so far."));
        kineticsAction->setWhatsThis(tr("Shows and hides the meta widget."));
        logAction->setWhatsThis(tr("Shows and hides the log widget."));
}
```

src/specprofiler.h

```
#ifndef SPECPROFILER_H
#define SPECPROFILER_H
#include <QTime>
#ifdef QT_DEBUG
#define SPECPROFILEMACRO(STRING) profiler DUMMYPROFILER(STRING) ;
#else
#define SPECPROFILEMACRO(STRING)
#endif
class specProfiler : public QTime
{
private:
        QString description ;
public:
        specProfiler(const QString&) ;
        ~specProfiler() ;
};
#endif
```

src/specprofiler.cpp

```
#include "specprofiler.h"
#include <QDebug>
specProfiler::specProfiler(const QString& s)
        : description(s)
{
        start();
}
specProfiler::~specProfiler()
{
        qDebug() << description << elapsed() ;
}
```

src/specshortcutdialog.h

```
#ifndef SPECSHORTCUTDIALOG_H
#define SPECSHORTCUTDIALOG_H
#include <QDialog>
#include <QAbstractItemModel>
#include <QIcon>
#include <QSettings>
class shortcutModel : public QAbstractItemModel
{
        Q_OBJECT
private:
        struct actionItem {
                QIcon icon ;
                QString title ;
                QList<QKeySequence> shortCuts ;
                void init(const QAction&) ;
                actionItem(const QAction&) ;
                actionItem(const QAction*) ;
                QString shortCutString() const ;
                QStringList shortCutList() const ;
                void setShortcuts(const QStringList&) ;
                void setShortcuts(const QString&) ;
                bool operator<(const actionItem&) const;
                bool operator==(const actionItem&) const;
        };
        typedef QList<actionItem> actionItemContainer ;
        typedef QMap<QString, actionItemContainer> modelDataType ;
```

```
        modelDataType modelContent ;
        const actionItem *fromIndex(const QModelIndex&) const ;
        actionItem *fromIndex(const QModelIndex&) ;
public:
        static QString settingsKey(const QString& category, const QString& command) ;
        static QString settingsKey(QAction*) ;
        static QObject* elgibleParent(QObject* parent) ;
        explicit shortcutModel(QObject* parent = 0) ;
        QModelIndex index(int row, int column, const QModelIndex& parent) const ;
        QModelIndex parent(const QModelIndex& child) const ;
        int rowCount(const QModelIndex& parent) const ;
        int columnCount(const QModelIndex& parent) const ;
        QVariant data(const QModelIndex& index, int role) const ;
        QVariant headerData(int section, Qt::Orientation orientation, int role) const ;
        bool setData(const QModelIndex& index, const QVariant& value, int role) ;
        Qt::ItemFlags flags(const QModelIndex& index) const ;
        void applyShortcutsToSettings() ;
};
namespace Ui
{
        class specShortcutDialog;
}
class specShortcutDialog : public QDialog
{
        Q_OBJECT
public:
        explicit specShortcutDialog(QWidget* parent = 0);
        ~specShortcutDialog();
private:
        Ui::specShortcutDialog* ui;
        QMap<QString, QList<QKeySequence> > readSequences(QSettings&, const QString &prefix) const ;
public slots:
        void assignShortcuts() ;
};
#endif
```

src/specshortcutdialog.cpp

```
#include "specshortcutdialog.h"
#include "ui_specshortcutdialog.h"
#include "specappwindow.h"
#include "specplotwidget.h"
#include <QSettings>
#define SETTINGSPREFIX "shortcuts"
shortcutModel::actionItem::actionItem(const QAction & a)
{
        init(a) ;
}
shortcutModel::actionItem::actionItem(const QAction * a)
{
        init(*a) ;
}
void shortcutModel::actionItem::init(const QAction & a)
{
        icon = a.icon() ;
        shortCuts = a.shortcuts() ;
        title = a.iconText() ;
}
bool shortcutModel::actionItem::operator <(const shortcutModel::actionItem& other) const
{
        return title < other.title ;
}
bool shortcutModel::actionItem::operator ==(const shortcutModel::actionItem& other) const
{
        return title == other.title ;
}
QString shortcutModel::actionItem::shortCutString() const
{
        return shortCutList().join(",␣") ;
}
QStringList shortcutModel::actionItem::shortCutList() const
{
        QStringList result ;
        foreach (const QKeySequence shortcut, shortCuts)
```

```
                result << shortcut ;
        return result ;
}
void shortcutModel::actionItem::setShortcuts(const QString & s)
{
        setShortcuts(s.split(QRegExp("\\s*,\\s*")));
}
void shortcutModel::actionItem::setShortcuts(const QStringList & l)
{
        shortCuts.clear();
        foreach(const QString& s, l)
                shortCuts << s ;
}
QString shortcutModel::settingsKey(const QString &category, const QString &command)
{
        return SETTINGSPREFIX + QString("/") + category + "/" + command ;
}
QString shortcutModel::settingsKey(QAction* action)
{
        QObject *parent = elgibleParent(action->parent()) ;
        return settingsKey(parent ? parent->objectName() : QString(), action->iconText()) ;
}
QObject* shortcutModel::elgibleParent(QObject *parent)
{
        while(parent &&
                !qobject_cast<specAppWindow*> (parent) &&
                !qobject_cast<specDockWidget*> (parent))
                parent = parent->parent() ;
        return parent ;
}
#define ITERATEOVERALLACTIONS(DOWITHACTIONS) \
        for (modelDataType::iterator i = modelContent.begin() ; i != modelContent.end() ; ++i) { \
                for(actionItemContainer::iterator a = i.value().begin() ; a != i.value().end() ; ++a►
                    ) { \
        DOWITHACTIONS }}
void shortcutModel::applyShortcutsToSettings()
{
        QSettings settings ;
        ITERATEOVERALLACTIONS(settings.setValue(settingsKey(i.key(), a->title), a->shortCutList());)
}
shortcutModel::shortcutModel(QObject* parent)
        : QAbstractItemModel(parent)
{
        specAppWindow window ;
        specPlotWidget plotWidget(&window) ;
        foreach(QAction * action, window.findChildren<QAction*>())
        {
                if(action->text().isEmpty()) continue ;
                if(action->menu()) continue ;
                parent = elgibleParent(action->parent()) ;
                if(!parent) continue ;
                modelContent[parent->objectName()] << action ;
        }
        QSettings settings ;
        ITERATEOVERALLACTIONS(
                QString key = settingsKey(i.key(), a->title) ;
                if(settings.contains(key))
                        a->setShortcuts(settings.value(key).toStringList());
        )
        foreach(const QString& key, modelContent.keys())
                qSort(modelContent[key]) ;
}
Qt::ItemFlags shortcutModel::flags(const QModelIndex& index) const
{
        return QAbstractItemModel::flags(index) |
               (index.column() == 1 && index.parent().isValid() ?
                Qt::ItemIsEditable :
                Qt::NoItemFlags) ;
}
int shortcutModel::columnCount(const QModelIndex& parent) const
{
        Q_UNUSED(parent)
        return 2 ;
}
QModelIndex shortcutModel::index(int row, int column, const QModelIndex& parent) const
```

```
{
        return createIndex(row, column, parent.isValid() ? parent.row() : -1) ;
}
QModelIndex shortcutModel::parent(const QModelIndex& child) const
{
        return child.internalId() == -1 ?
               QModelIndex() :
               createIndex(child.internalId(), 0, -1) ;
}
int shortcutModel::rowCount(const QModelIndex& parent) const
{
        if(!parent.isValid()) return modelContent.size() ;
        if(parent.parent().isValid()) return 0 ;
        return modelContent[parent.data().toString()].size() ;
}
const shortcutModel::actionItem* shortcutModel::fromIndex(const QModelIndex & index) const
{
        if (!index.parent().isValid()) return 0 ;
        return &(modelContent[index.parent().data().toString()][index.row()]) ;
}
shortcutModel::actionItem* shortcutModel::fromIndex(const QModelIndex & index)
{
        if (!index.parent().isValid()) return 0 ;
        return &(modelContent[index.parent().data().toString()][index.row()]) ;
}
QVariant shortcutModel::data(const QModelIndex& index, int role) const
{
        const actionItem *ai = fromIndex(index) ;
        if(!index.column())
        {
                if(Qt::DecorationRole == role)
                {
                        if(ai) return ai->icon ;
                        return QIcon::fromTheme("folder") ;
                }
                else if(Qt::DisplayRole == role)
                {
                        if(ai) return ai->title ;
                        return modelContent.keys()[index.row()];
                }
        }
        if(!index.parent().isValid()) return QVariant() ;
        if(ai && (Qt::DisplayRole == role || Qt::EditRole == role))
                return ai->shortCutString() ;
        return QVariant() ;
}
QVariant shortcutModel::headerData(int section, Qt::Orientation orientation, int role) const
{
        if(Qt::DisplayRole == role && orientation == Qt::Horizontal)
                return section ? tr("Shortcut") : tr("Action") ;
        return QVariant() ;
}
bool shortcutModel::setData(const QModelIndex& index, const QVariant& value, int role)
{
        actionItem* ai = fromIndex(index) ;
        if(Qt::EditRole != role) return false ;
        if(!index.parent().isValid()) return false ;
        if(!index.column()) return false ;
        if(!ai) return false ;
        QList<QKeySequence> sequences ;
        foreach(const QString & seq, value.toString().split(QRegExp("\\s*,\\s*")))
        {
                if (sequences.contains(seq)) continue ;
                sequences << seq ;
        }
        QMultiMap<QKeySequence, actionItem*> assignment ;
        ITERATEOVERALLACTIONS(foreach(QKeySequence seq, a->shortCuts) assignment.insertMulti(seq, a.►
            operator ->()) ;) ;
        QList<QKeySequence>::iterator i = sequences.begin() ;
        QString group = index.parent().data().toString() ;
        while (i != sequences.end())
        {
                if (!assignment.contains(*i))
                {
                        ++i ;
```

```
                        continue ;
                }
                actionItem* other = 0 ;
                foreach(actionItem* item, assignment.values(*i))
                {
                        if (*item == *ai) continue ;
                        if (modelContent[group].contains(*item)) other = item ;
                }
                if (!other || other == ai)
                {
                        ++i ;
                        continue ;
                }
                if (QMessageBox::Yes ==
                                QMessageBox::question(0,
                                                      tr("Shortcut␣already␣assigned"),
                                                      tr("The␣shortcut␣")
                                                      + i->toString()
                                                      + tr("␣is␣already␣assigned␣to␣\"")
                                                      + other->title
                                                      + tr("\".␣Reassign?"),
                                                      QMessageBox::Yes | QMessageBox::No,
                                                      QMessageBox::No))
                {
                        other->shortCuts.removeAll(*i) ;
                        ++i ;
                }
                else
                        i = sequences.erase(i) ;
        }
        ai->shortCuts = sequences ;
        return true ;
}
specShortcutDialog::specShortcutDialog(QWidget* parent) :
        QDialog(parent),
        ui(new Ui::specShortcutDialog)
{
        ui->setupUi(this);
        setModal(true) ;
        ui->treeView->setModel(new shortcutModel(this)) ;
        ui->treeView->expandAll() ;
        ui->treeView->resizeColumnToContents(0) ;
        ui->treeView->resizeColumnToContents(1) ;
        connect(this, SIGNAL(accepted()), this, SLOT(assignShortcuts())) ;
}
QMap<QString, QList<QKeySequence> > specShortcutDialog::readSequences(QSettings &settings, const ►
    QString& prefix) const
{
        QMap<QString, QList<QKeySequence> > result ;
        foreach(const QString& key, settings.childKeys())
        {
                QList<QKeySequence> seqs ;
                foreach(const QString& seq, settings.value(key).toStringList())
                        seqs << seq ;
                result[prefix + key] = seqs ;
        }
        foreach(const QString& group, settings.childGroups())
        {
                settings.beginGroup(group) ;
                result = result.unite(readSequences(settings, prefix + group + "/")) ;
                settings.endGroup();
        }
        return result ;
}
void specShortcutDialog::assignShortcuts()
{
        shortcutModel* model = findChild<shortcutModel*>() ;
        if(model)
                model->applyShortcutsToSettings();
        QSettings settings ;
        settings.beginGroup(SETTINGSPREFIX) ;
        QMap<QString, QList<QKeySequence> > setShortCuts = readSequences(settings, SETTINGSPREFIX + ►
            QString("/"));
        qDebug() << setShortCuts ;
        foreach(QWidget* tlw, qApp->topLevelWidgets())
```

```
        {
259             foreach(QAction* action, tlw->findChildren<QAction*>())
                {
                        qDebug() << "Searching␣shortcut:" << shortcutModel::settingsKey(action) ;
                        if (setShortCuts.contains(shortcutModel::settingsKey(action)))
                                action->setShortcuts(setShortCuts[shortcutModel::settingsKey(action)►
                                    ]) ;
264             }
        }
}
specShortcutDialog::~specShortcutDialog()
{
269         delete ui;
}
```

src/specshortcutdialog.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>specShortcutDialog</class>
 <widget class="QDialog" name="specShortcutDialog">
5  <property name="geometry">
    <rect>
     <x>0</x>
     <y>0</y>
     <width>400</width>
10   <height>300</height>
    </rect>
   </property>
   <property name="windowTitle">
    <string>Shortcuts</string>
15 </property>
   <layout class="QVBoxLayout" name="verticalLayout">
    <item>
     <widget class="QTreeView" name="treeView"/>
    </item>
20  <item>
     <widget class="QDialogButtonBox" name="buttonBox">
      <property name="orientation">
       <enum>Qt::Horizontal</enum>
      </property>
25    <property name="standardButtons">
       <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
      </property>
     </widget>
    </item>
30 </layout>
 </widget>
 <resources/>
 <connections>
  <connection>
35  <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>specShortcutDialog</receiver>
   <slot>accept()</slot>
   <hints>
40  <hint type="sourcelabel">
     <x>248</x>
     <y>254</y>
    </hint>
    <hint type="destinationlabel">
45   <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
50 <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>specShortcutDialog</receiver>
   <slot>reject()</slot>
55 <hints>
    <hint type="sourcelabel">
     <x>316</x>
```

```
     <y>260</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
 </connections>
</ui>
```

src/specsplitter.h

```
#ifndef SPECSPLITTER_H
#define SPECSPLITTER_H
#include <QSplitter>
class specSplitterHandle : public QSplitterHandle
{
        Q_OBJECT
private:
        void mousePressEvent(QMouseEvent*) ;
public:
        specSplitterHandle(Qt::Orientation orientation, QSplitter* parent) ;
};
class specSplitter : public QSplitter
{
        Q_OBJECT
public:
        explicit specSplitter(Qt::Orientation o, QWidget* parent = 0);
        explicit specSplitter(QWidget* parent = 0) ;
private:
        QSplitterHandle* createHandle() ;
};
#endif
```

src/specsplitter.cpp

```
#include "specsplitter.h"
#include <QContextMenuEvent>
specSplitter::specSplitter(Qt::Orientation o, QWidget* parent)
        : QSplitter(o, parent)
{
}
specSplitter::specSplitter(QWidget* parent)
        : QSplitter(parent)
{
}
specSplitterHandle::specSplitterHandle(Qt::Orientation orientation, QSplitter* parent)
        : QSplitterHandle(orientation, parent)
{}
QSplitterHandle* specSplitter::createHandle()
{
        return new specSplitterHandle(orientation(), this) ;
}
void specSplitterHandle::mousePressEvent(QMouseEvent* e)
{
        QSplitter* p = qobject_cast<QSplitter*> (parent()) ;
        if(p && e->button() == Qt::RightButton && e->modifiers() == Qt::NoModifier)
                p->setOrientation(p->orientation() == Qt::Horizontal ? Qt::Vertical : Qt::Horizontal►
                    ) ;
        QSplitterHandle::mousePressEvent(e) ;
}
```

src/specstreamable.h

```
#ifndef SPECSTREAMABLE_H
#define SPECSTREAMABLE_H
#include <QDataStream>
class specStreamable
{
```

```
public:
        friend QDataStream& operator<< (QDataStream& out, const specStreamable&) ;
        friend QDataStream& operator>> (QDataStream& in, specStreamable&) ;
        virtual ~specStreamable() {}
        void readDirectly(QDataStream& in) { readFromStream(in) ; }
        typedef quint16 type;
        enum streamableType
        {
            none = 0,
            folder = 1,
            logEntry = 2,
            sysEntry = 3,
            kineticItem = 4,
            svgItem = 5,
            legacyDataItem = 6,
            mainWidget = 7,
            mainPlot = 8,
            mainView = 9,
            viewState = 10,
            logWidget = 11,
            kineticWidget = 12,
            actionLibrary = 13,
            undoCommand = 14,
            deleteCommandId = 15,
            newFolderCommandId = 16,
            moveItemsCommandId = 17,
            movePlotCommandId = 19,
            exchangeDataCommandId = 23,
            resizeSVGCommandId = 24,
            newConnectionsCommandId = 31,
            deleteConnectionsCommandId = 32,
            editDescriptorCommandId = 33,
            penColorCommandId = 34,
            symbolStyleCommandId = 35,
            symbolSizeCommandId = 36,
            symbolPenColorCommandId = 37,
            symbolBrushColorCommandId = 38,
            manageConnectionsCommandId = 39,
            manageItemsCommandId = 40,
            dataView = 41,
            multiCommandId = 42,
            genealogyId = 43,
            model = 44,
            descriptor = 45,
            plotStyle = 46,
            metaItem = 47,
            metaModel = 48,
            metaView = 49,
            metaWidget = 50,
            logModel = 51,
            logView = 52,
            range = 53,
            metaRange = 54,
            metaRangeCommand = 55,
            lineWidthCommandId = 58,
            plotTitleCommandId = 59,
            plotYLabelCommandId = 60,
            plotXLabelCommandId = 61,
            fitCurve = 62,
            exchangeFitCommand = 63,
            toggleFitStyleCommand = 64,
            penStyleCommandId = 65,
            descriptorFlagsCommand = 66,
            dataItem = 67,
            deleteDescriptorCommandId = 68,
            renameDescriptorCommandId = 69,
            exchangeFilterCommandId = 70
        };
        virtual type typeId() const = 0;
protected:
        virtual void writeToStream(QDataStream& out) const = 0;
        virtual void readFromStream(QDataStream& in) = 0;
        virtual void writeContents(QDataStream& out) const {Q_UNUSED(out) }
        virtual void readContents(QDataStream& in) {Q_UNUSED(in) }
        virtual specStreamable* factory(const type& t) const {Q_UNUSED(t) ; return 0; }
```

```
        specStreamable* produceItem(QDataStream& in) const;
private:
        type effectiveId() const ;
};
QDataStream& operator<< (QDataStream& out, const specStreamable&) ;
QDataStream& operator>> (QDataStream& in, specStreamable&) ;
#endif
```

src/specstreamable.cpp

```
#include "specstreamable.h"
#include <QDebug>
specStreamable::type specStreamable::effectiveId() const
{
        return type(typeId()) ;
}
QDataStream& operator<< (QDataStream& out, const specStreamable& item)
{
        QByteArray ba ;
        QDataStream stream(&ba, QIODevice::WriteOnly) ;
        stream << (quint32) item.effectiveId() ;
        item.writeToStream(stream) ;
        out << ba ;
        return out ;
}
QDataStream& operator>> (QDataStream& in, specStreamable& item)
{
        QByteArray ba ;
        in >> ba ;
        QDataStream stream(ba) ;
        quint32 ta ;
        stream >> ta ;
        specStreamable::type t(ta) ;
        if(t != item.effectiveId())
        {
                return in ;
        }
        item.readFromStream(stream) ;
        return in ;
}
specStreamable* specStreamable::produceItem(QDataStream& in) const
{
        QByteArray ba ;
        in >> ba ;
        QDataStream stream(ba) ;
        quint32 t ;
        stream >> t ;
        qDebug() << "reading␣type:" << t ;
        specStreamable* item = factory(t) ;
        if(item) item->readFromStream(stream);
        return item ;
}
```

src/spectral/dataitemproperties.h

```
#ifndef DATAITEMPROPERTIES_H
#define DATAITEMPROPERTIES_H
#include <QDialog>
#include "specdataitem.h"
namespace Ui
{
        class dataItemProperties;
}
class dataItemProperties : public QDialog
{
        Q_OBJECT
public:
        dataItemProperties(specDataItem*, QWidget* parent = 0);
        ~dataItemProperties();
        specUndoCommand* changeCommands(QObject*) ;
private:
        Ui::dataItemProperties* ui;
```

```
        specDataItem item ;
        specDataItem* originalItem ;
        QwtPlotCurve highlightSelected, highlightCurrent ;
        bool setupComplete ;
        bool merge, sort ;
        void connectCurve(QwtPlotCurve& c) ;
private slots:
        void refreshPlot() ;
};
#endif
```

src/spectral/dataitemproperties.cpp

```
#include "dataitemproperties.h"
#include "ui_dataitemproperties.h"
#include "specmulticommand.h"
#include "specexchangedatacommand.h"
#include "specexchangefiltercommand.h"
dataItemProperties::dataItemProperties(specDataItem* i, QWidget* parent) :
        QDialog(parent),
        ui(new Ui::dataItemProperties),
        item(*i),
        originalItem(i),
        setupComplete(false)
{
        if(!i) return ;
        item.sortPlotData = false ;
        item.mergePlotData = false ;
        ui->setupUi(this);
        ui->slopeValue->setRange(-INFINITY, INFINITY);
        ui->xShiftValue->setRange(-INFINITY, INFINITY);
        ui->offsetValue->setRange(-INFINITY, INFINITY);
        ui->scalingValue->setRange(-INFINITY, INFINITY);
        item.setPen(QPen()) ;
        item.setSymbol(0) ;
        highlightSelected.setSymbol(new QwtSymbol(QwtSymbol::Ellipse,
                                    QBrush(Qt::blue),
                                    QPen(Qt::NoPen),
                                    QSize(5, 5)));
        highlightSelected.setStyle(QwtPlotCurve::NoCurve);
        highlightCurrent.setSymbol(new QwtSymbol(QwtSymbol::Ellipse,
                                   QBrush(Qt::red),
                                   QPen(Qt::NoPen),
                                   QSize(5, 5)));
        highlightCurrent.setStyle(QwtPlotCurve::NoCurve);
        highlightCurrent.setZ(INFINITY);
        ui->dataPreview->setAutoDelete(false) ;
        item.attach(ui->dataPreview);
        specDataPointFilter filter = item.filter ;
        ui->slopeValue->setValue(filter.getSlope()) ;
        ui->offsetValue->setValue(filter.getOffset()) ;
        ui->xShiftValue->setValue(filter.getXShift()) ;
        ui->scalingValue->setValue(filter.getFactor()) ;
        ui->dataWidget->setRowCount(item.data.size()) ;
        ui->dataWidget->setColumnCount(3) ;
        for(int i = 0 ; i < item.data.size() ; ++i)
        {
                ui->dataWidget->
                setItem(i, 0, new QTableWidgetItem(QString::number(item.data[i].nu)));
                ui->dataWidget->
                setItem(i, 1, new QTableWidgetItem(QString::number(item.data[i].sig)));
                ui->dataWidget->
                setItem(i, 2, new QTableWidgetItem(QString::number(item.data[i].mint)));
        }
        setupComplete = true ;
        refreshPlot();
}
dataItemProperties::~dataItemProperties()
{
        delete ui;
}
void dataItemProperties::refreshPlot()
{
        if(!setupComplete) return ;
```

```
        item.setDataFilter(specDataPointFilter(
                                ui->offsetValue->value(),
                                ui->slopeValue->value(),
                                ui->scalingValue->value(),
                                ui->xShiftValue->value()));
        for(int i = 0 ; i < item.data.size() ; ++i)
        {
                item.data[i].nu = ui->dataWidget->item(i, 0)->text().toDouble() ;
                item.data[i].sig = ui->dataWidget->item(i, 1)->text().toDouble() ;
                item.data[i].mint = ui->dataWidget->item(i, 2)->text().toDouble() ;
        }
        item.revalidate();
        QVector<QPointF> selectedData ;
        foreach(QTableWidgetItem * selectedItem, ui->dataWidget->selectedItems())
        selectedData << item.sample(selectedItem->row()) ;
        highlightSelected.setSamples(selectedData) ;
        highlightCurrent.setSamples(
            ui->dataWidget->currentItem() ?
            QVector<QPointF>() << item.sample(ui->dataWidget->currentRow()) :
            QVector<QPointF>()) ;
        connectCurve(highlightSelected) ;
        connectCurve(highlightCurrent) ;
        ui->dataPreview->replot();
}
void dataItemProperties::connectCurve(QwtPlotCurve& c)
{
        if(c.dataSize()) c.attach(ui->dataPreview) ;
        else c.detach();
}
specUndoCommand* dataItemProperties::changeCommands(QObject* parent)
{
        bool dataUnchanged = true, correctionUnchanged = true ;
        if(item.data.size() == originalItem->data.size())
                for(size_t i = 0 ; i < item.dataSize() ; ++i)
                        dataUnchanged = dataUnchanged &&
                                        item.data[i].exactlyEqual(originalItem->data[i]) ;
        else dataUnchanged = false ;
        correctionUnchanged =
            (item.dataFilter() == originalItem->dataFilter()) ;
        if(correctionUnchanged && dataUnchanged) return 0 ;
        specUndoCommand* parentMulti = 0 ;
        if(!dataUnchanged && !correctionUnchanged)
        {
                parentMulti = new specMultiCommand ;
                parentMulti->setParentObject(parent) ;
        }
        if(!dataUnchanged)
        {
                specExchangeDataCommand* dataCommand = new specExchangeDataCommand(parentMulti) ;
                dataCommand->setParentObject(parent);
                dataCommand->setItem(originalItem, item.data) ;
                if(correctionUnchanged)
                {
                        dataCommand->setText(tr("Modify␣item␣data"));
                        return dataCommand ;
                }
        }
        if(!correctionUnchanged)
        {
                specExchangeFilterCommand* correctionCommand = new specExchangeFilterCommand(►
                    parentMulti) ;
                correctionCommand->setParentObject(parent) ;
                correctionCommand->setItem(originalItem) ;
                correctionCommand->setAbsoluteFilter(item.filter) ;
                if(dataUnchanged)
                        return correctionCommand ;
        }
        parentMulti->setText(tr("Modify␣item␣data␣and␣correction")) ;
        return parentMulti ;
}
```

src/spectral/dataitemproperties.ui

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

```
<ui version="4.0">
 <class>dataItemProperties</class>
 <widget class="QDialog" name="dataItemProperties">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>559</width>
    <height>479</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>Data Set Properties</string>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout">
   <item>
    <widget class="specSplitter" name="splitter">
     <property name="orientation">
      <enum>Qt::Vertical</enum>
     </property>
     <widget class="specPlot" name="dataPreview">
      <property name="frameShape">
       <enum>QFrame::StyledPanel</enum>
      </property>
      <property name="frameShadow">
       <enum>QFrame::Raised</enum>
      </property>
     </widget>
     <widget class="QWidget" name="layoutWidget">
      <layout class="QHBoxLayout" name="horizontalLayout">
       <item>
        <widget class="QTableWidget" name="dataWidget">
         <property name="columnCount">
          <number>3</number>
         </property>
         <column>
          <property name="text">
           <string>x-value</string>
          </property>
         </column>
         <column>
          <property name="text">
           <string>y-value</string>
          </property>
         </column>
         <column>
          <property name="text">
           <string>max. intensity</string>
          </property>
         </column>
        </widget>
       </item>
       <item>
        <widget class="QWidget" name="correctionWidget" native="true">
         <layout class="QGridLayout" name="gridLayout">
          <item row="2" column="0">
           <widget class="QLabel" name="scalingLabel">
            <property name="text">
             <string>Scaling &lt;i&gt; a&lt;/i&gt;</string>
            </property>
            <property name="buddy">
             <cstring>scalingValue</cstring>
            </property>
           </widget>
          </item>
          <item row="3" column="0">
           <widget class="QLabel" name="offsetLabel">
            <property name="text">
             <string>Offset &lt;i&gt;b&lt;/i&gt;</string>
            </property>
            <property name="buddy">
             <cstring>offsetValue</cstring>
            </property>
           </widget>
          </item>
```

```
           <item row="3" column="1">
            <widget class="QDoubleSpinBox" name="offsetValue">
             <property name="decimals">
              <number>10</number>
             </property>
            </widget>
           </item>
           <item row="4" column="0">
            <widget class="QLabel" name="slopeLabel">
             <property name="text">
              <string>Slope &lt;i&gt;c&lt;/i&gt;</string>
             </property>
             <property name="buddy">
              <cstring>slopeValue</cstring>
             </property>
            </widget>
           </item>
           <item row="4" column="1">
            <widget class="QDoubleSpinBox" name="slopeValue">
             <property name="decimals">
              <number>10</number>
             </property>
            </widget>
           </item>
           <item row="5" column="0">
            <widget class="QLabel" name="xShiftLabel">
             <property name="text">
              <string>&lt;i&gt;x&lt;/i&gt;-Shift &lt;i&gt;x&lt;sub&gt;0&lt;/sub&gt;&lt;/i&gt;</string>
             </property>
             <property name="buddy">
              <cstring>xShiftValue</cstring>
             </property>
            </widget>
           </item>
           <item row="5" column="1">
            <widget class="QDoubleSpinBox" name="xShiftValue">
             <property name="decimals">
              <number>10</number>
             </property>
            </widget>
           </item>
           <item row="1" column="0" colspan="2">
            <widget class="QLabel" name="formulaLabel">
             <property name="text">
              <string>&lt;b&gt;Correction:&lt;/b&gt; &lt;i&gt;y&lt;/i&gt; '␣=␣&lt;i&gt;a␣y&lt;/i&gt;␣►
                  +␣&lt;i&gt;b&lt;/i&gt;␣+␣&lt;i&gt;c&lt;/i&gt;␣(&lt;i&gt;x&lt;/i&gt;␣+␣&lt;i&gt;x&lt►
                  ;sub&gt;0&lt;/sub&gt;&lt;/i&gt;)</string>
␣␣␣␣␣␣␣␣␣␣␣␣</property>
␣␣␣␣␣␣␣␣␣␣␣</widget>
␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣␣<item␣row="2"␣column="1">
␣␣␣␣␣␣␣␣␣␣␣<widget␣class="QDoubleSpinBox"␣name="scalingValue">
␣␣␣␣␣␣␣␣␣␣␣␣<property␣name="decimals">
␣␣␣␣␣␣␣␣␣␣␣␣␣<number>10</number>
␣␣␣␣␣␣␣␣␣␣␣␣</property>
␣␣␣␣␣␣␣␣␣␣␣</widget>
␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣␣<item␣row="6"␣column="0">
␣␣␣␣␣␣␣␣␣␣␣<spacer␣name="verticalSpacer">
␣␣␣␣␣␣␣␣␣␣␣␣<property␣name="orientation">
␣␣␣␣␣␣␣␣␣␣␣␣␣<enum>Qt::Vertical</enum>
␣␣␣␣␣␣␣␣␣␣␣␣</property>
␣␣␣␣␣␣␣␣␣␣␣␣<property␣name="sizeHint"␣stdset="0">
␣␣␣␣␣␣␣␣␣␣␣␣␣<size>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣<width>20</width>
␣␣␣␣␣␣␣␣␣␣␣␣␣␣<height>40</height>
␣␣␣␣␣␣␣␣␣␣␣␣␣</size>
␣␣␣␣␣␣␣␣␣␣␣␣</property>
␣␣␣␣␣␣␣␣␣␣␣</spacer>
␣␣␣␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣␣␣␣</layout>
␣␣␣␣␣␣␣␣</widget>
␣␣␣␣␣␣␣</item>
␣␣␣␣␣␣</layout>
␣␣␣␣␣</widget>
```

```
    </widget>
   </item>
   <item>
    <widget class="QDialogButtonBox" name="buttonBox">
     <property name="orientation">
      <enum>Qt::Horizontal</enum>
     </property>
     <property name="standardButtons">
      <set>QDialogButtonBox::Cancel|QDialogButtonBox::Ok</set>
     </property>
    </widget>
   </item>
  </layout>
 </widget>
 <customwidgets>
  <customwidget>
   <class>specPlot</class>
   <extends>QFrame</extends>
   <header>specplot.h</header>
   <container>1</container>
  </customwidget>
  <customwidget>
   <class>specSplitter</class>
   <extends>QSplitter</extends>
   <header>specsplitter.h</header>
   <container>1</container>
  </customwidget>
 </customwidgets>
 <tabstops>
  <tabstop>dataWidget</tabstop>
  <tabstop>buttonBox</tabstop>
  <tabstop>scalingValue</tabstop>
  <tabstop>offsetValue</tabstop>
  <tabstop>slopeValue</tabstop>
  <tabstop>xShiftValue</tabstop>
 </tabstops>
 <resources/>
 <connections>
  <connection>
   <sender>buttonBox</sender>
   <signal>accepted()</signal>
   <receiver>dataItemProperties</receiver>
   <slot>accept()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>222</x>
     <y>464</y>
    </hint>
    <hint type="destinationlabel">
     <x>157</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>buttonBox</sender>
   <signal>rejected()</signal>
   <receiver>dataItemProperties</receiver>
   <slot>reject()</slot>
   <hints>
    <hint type="sourcelabel">
     <x>290</x>
     <y>470</y>
    </hint>
    <hint type="destinationlabel">
     <x>286</x>
     <y>274</y>
    </hint>
   </hints>
  </connection>
  <connection>
   <sender>scalingValue</sender>
   <signal>valueChanged(double)</signal>
   <receiver>dataItemProperties</receiver>
   <slot>refreshPlot()</slot>
```

```
    <hints>
     <hint type="sourcelabel">
      <x>467</x>
      <y>286</y>
     </hint>
     <hint type="destinationlabel">
      <x>473</x>
      <y>478</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>offsetValue</sender>
    <signal>valueChanged(double)</signal>
    <receiver>dataItemProperties</receiver>
    <slot>refreshPlot()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>503</x>
      <y>312</y>
     </hint>
     <hint type="destinationlabel">
      <x>439</x>
      <y>479</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>slopeValue</sender>
    <signal>valueChanged(double)</signal>
    <receiver>dataItemProperties</receiver>
    <slot>refreshPlot()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>490</x>
      <y>339</y>
     </hint>
     <hint type="destinationlabel">
      <x>333</x>
      <y>480</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>xShiftValue</sender>
    <signal>valueChanged(double)</signal>
    <receiver>dataItemProperties</receiver>
    <slot>refreshPlot()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>529</x>
      <y>366</y>
     </hint>
     <hint type="destinationlabel">
      <x>566</x>
      <y>367</y>
     </hint>
    </hints>
   </connection>
   <connection>
    <sender>dataWidget</sender>
    <signal>currentCellChanged(int,int,int,int)</signal>
    <receiver>dataItemProperties</receiver>
    <slot>refreshPlot()</slot>
    <hints>
     <hint type="sourcelabel">
      <x>89</x>
      <y>379</y>
     </hint>
     <hint type="destinationlabel">
      <x>79</x>
      <y>476</y>
     </hint>
    </hints>
   </connection>
```

```
    <connection>
      <sender>dataWidget</sender>
      <signal>itemSelectionChanged()</signal>
      <receiver>dataItemProperties</receiver>
      <slot>refreshPlot()</slot>
      <hints>
       <hint type="sourcelabel">
        <x>41</x>
        <y>300</y>
       </hint>
       <hint type="destinationlabel">
        <x>0</x>
        <y>249</y>
       </hint>
      </hints>
    </connection>
    <connection>
      <sender>dataWidget</sender>
      <signal>itemChanged(QTableWidgetItem*)</signal>
      <receiver>dataItemProperties</receiver>
      <slot>refreshPlot()</slot>
      <hints>
       <hint type="sourcelabel">
        <x>336</x>
        <y>296</y>
       </hint>
       <hint type="destinationlabel">
        <x>557</x>
        <y>239</y>
       </hint>
      </hints>
    </connection>
   </connections>
   <slots>
    <slot>refreshPlot()</slot>
   </slots>
</ui>
```

src/spectral/specdataitem.h

```
#ifndef SPECDATAITEM_H
#define SPECDATAITEM_H
#include "specmodelitem.h"
#include <QHash>
#include <QVector>
#include "specdatapoint.h"
#include "specdescriptor.h"
#include "specstreamable.h"
#include "speclogentryitem.h"
#include "specdatapointfilter.h"
class specDataItem : public specLogEntryItem
{
        friend class dataItemProperties ;
        friend class specLegacyDataItem ;
public:
        typedef QVector<specDataPoint> dataContainer ;
private:
        specDataPointFilter filter ;
        dataContainer data ;
        dataContainer correctedData() const ;
        void readFromStream(QDataStream&) ;
        void writeToStream(QDataStream&) const ;
        type typeId() const { return specStreamable::dataItem ; }
public:
        specDataItem(const dataContainer& data,
                     const QHash<QString, specDescriptor>& description,
                     specFolderItem* par = 0, QString tag = "");
        specDataItem(const specDataItem&) ;
        specDataItem() ;
        const dataContainer& allData() const { return data ; }
        void setData(const dataContainer&) ;
        void swapData(dataContainer&) ;
        dataContainer getDataExcept(const QList<specRange*>& ranges) ;
        void applyCorrection(dataContainer&) const ;
```

```
        void reverseCorrection(dataContainer&) const ;
        QIcon decoration() const ;
        specDataItem& operator+= (const specDataItem& toAdd) ;
        void flatten() ;
        QVector<double> intensityData() const ;
        void refreshPlotData() ;
        specDataPointFilter dataFilter() const;
        void setDataFilter(const specDataPointFilter&) ;
        void addDataFilter(const specDataPointFilter&) ;
        void attach(QwtPlot* plot) ;
        void detach() ;
        void exportData(const QList<QPair<bool, QString> >&, const QList<QPair<spec::value, QString>►
             >&, QTextStream&) ;
        specUndoCommand* itemPropertiesAction(QObject* parentObject) ;
};
class specLegacyDataItem : public specDataItem
{
private:
        type myType ;
        type typeId() const { return myType; }
        void readFromStream(QDataStream& in) ;
public:
        specLegacyDataItem() ;
};
#endif
```

src/spectral/specdataitem.cpp

```
#include "specdataitem.h"
#include <QVariant>
#include <utility-functions.h>
#include <algorithm>
#include "dataitemproperties.h"
#include "specplot.h"
QVector<double> specDataItem::intensityData() const
{
        QVector<double> retval ;
        foreach(const specDataPoint & dataPoint, data)
        retval << dataPoint.mint ;
        return retval ;
}
specDataItem::specDataItem(const QVector<specDataPoint>& dat, const QHash<QString, specDescriptor>& ►
     desc, specFolderItem* par, QString description)
        : specLogEntryItem(desc, par, description),
          data(dat)
{}
specDataItem::specDataItem()
        : specLogEntryItem()
{}
specDataItem::dataContainer specDataItem::correctedData() const
{
        dataContainer result(data) ;
        for(int i = 0 ; i < result.size() ; ++i)
                filter.applyCorrection(result[i]) ;
        return result ;
}
void specDataItem::refreshPlotData()
{
        QVector<QPointF> newData ;
        foreach(const specDataPoint & dataPoint, correctedData())
        newData << dataPoint ;
        setSamples(newData) ;
}
QIcon specDataItem::decoration() const { return QIcon(":/data.png") ; }
void specDataItem::readFromStream(QDataStream& in)
{
        specLogEntryItem::readFromStream(in) ;
        in >> data
           >> filter ;
}
void specDataItem::writeToStream(QDataStream& out) const
{
        specLogEntryItem::writeToStream(out) ;
        out << data
```

```
            << filter ;
}
specDataItem& specDataItem::operator+= (const specDataItem& toAdd)
{
        foreach(QString key, toAdd.description.keys())
        {
                if(description.keys().contains(key))
                {
                        if(description[key].isNumeric())
                        {
                                double total = data.size() + toAdd.data.size() ;
                                description[key] = description[key].numericValue() * data.size() / ►
                                    total + toAdd.description[key].numericValue() * toAdd.data.size►
                                    () / total ;
                        }
                        else if(!descriptor(key, true).contains(toAdd.descriptor(key, true)))
                                description[key] = descriptor(key).append(",␣").append(toAdd.►
                                    descriptor(key)) ;
                }
                else
                        description[key] = toAdd.description[key] ;
        }
        if(!descriptor("").contains(toAdd.descriptor("")))
                changeDescriptor("", toAdd.descriptor("").prepend(descriptor("").isEmpty() ? "" : ►
                    descriptor("").append(",␣"))) ;
        foreach(specDataPoint point, toAdd.correctedData())
        {
                filter.reverseCorrection(point) ;
                data << point ;
        }
        invalidate() ;
        return (*this) ;
}
bool compareDataPoints(const specDataPoint& a, const specDataPoint& b)
{
        return a == b ;
}
void specDataItem::flatten()
{
        qSort(data) ;
        QVector<specDataPoint> newData ;
        averageToNew(data.begin(), data.end(), compareDataPoints, std::back_inserter(newData)) ;
        data.swap(newData) ;
        invalidate() ;
}
void specDataItem::exportData(const QList<QPair<bool, QString> >& headerFormat, const QList<QPair<►
    spec::value, QString> >& dataFormat, QTextStream& out)
{
        revalidate();
        for(int i = 0 ; i < headerFormat.size() ; i++)
                out << (headerFormat[i].first ? headerFormat[i].second : this->descriptor(►
                    headerFormat[i].second)) ;
        out << endl ;
        typedef QPair<spec::value, QString> formatPair ;
        foreach(const specDataPoint & point, correctedData())
        {
                foreach(const formatPair & format, dataFormat)
                {
                        switch(format.first)
                        {
                                case spec::wavenumber: out << point.nu ; break ;
                                case spec::signal: out << point.sig ; break ;
                                case spec::maxInt: out << point.mint ; break ;
                        }
                        out << format.second ;
                }
        }
        out << endl ;
}
QVector<specDataPoint> specDataItem::getDataExcept(const QList<specRange*>& ranges)
{
        QVector<specDataPoint> newData ;
        for(int i = 0 ; i < data.size() ; ++i)
        {
                specDataPoint point = data[i] ;
```

```
                filter.applyCorrection(point) ;
                bool include = true ;
117             foreach(specRange * range, ranges)
                {
                        if(range->contains(point.nu))
                        {
                                include = false ;
122                             break ;
                        }
                }
                if(include)
                        newData << data[i] ;
127     }
        return newData ;
}
void specDataItem::applyCorrection(QVector<specDataPoint>& newData) const
{
132     for(int i = 0 ; i < newData.size() ; ++i)
                filter.applyCorrection(newData[i]) ;
}
void specDataItem::reverseCorrection(QVector<specDataPoint>& newData) const
{
137     for(int i = 0 ; i < newData.size() ; ++i)
                filter.reverseCorrection(newData[i]) ;
}
void specDataItem::swapData(QVector<specDataPoint>& newData)
{
142     data.swap(newData) ;
        qSort(data) ;
        invalidate() ;
}
specDataItem::specDataItem(const specDataItem& other)
147     : specLogEntryItem(other),
          filter(other.filter),
          data(other.data)
{
}
152 specUndoCommand* specDataItem::itemPropertiesAction(QObject* parentObject)
{
        dataItemProperties propertiesDialog(this) ;
        propertiesDialog.exec() ;
        if(propertiesDialog.result() != QDialog::Accepted) return 0 ;
157     return propertiesDialog.changeCommands(parentObject) ;
}
void specDataItem::attach(QwtPlot* pl)
{
        specModelItem::attach(pl) ;
162     specPlot* p = qobject_cast<specPlot*> (plot()) ;
        if(p)
                p->attachToPicker(this) ;
}
void specDataItem::detach()
167 {
        specPlot* p = qobject_cast<specPlot*> (plot()) ;
        if(p)
                p->detachFromPicker(this) ;
        specModelItem::detach() ;
172 }
specLegacyDataItem::specLegacyDataItem()
        : specDataItem(),
          myType(legacyDataItem)
{}
177 void specLegacyDataItem::readFromStream(QDataStream& in)
{
        myType = dataItem ;
        specLogEntryItem::readFromStream(in) ;
        QVector<legacyDatapoint> legacyData ;
182     in >> legacyData
           >> filter ;
        foreach(legacyDatapoint point, legacyData)
        data << point ;
}
187 void specDataItem::setDataFilter(const specDataPointFilter& f)
{
        filter = f ;
```

```
        invalidate();
}
specDataPointFilter specDataItem::dataFilter() const
{
        return filter ;
}
void specDataItem::addDataFilter(const specDataPointFilter& other)
{
        filter += other ;
        invalidate();
}
```

src/spectral/specdatapoint.h

```
#ifndef SPECDATAPOINT_H
#define SPECDATAPOINT_H
#include <QDataStream>
#include <math.h>
#include <QVector>
#include "names.h"
#include <QPointF>
struct specDataPoint;
struct legacyDatapoint ;
QDataStream& operator<< (QDataStream&, const specDataPoint&);
QDataStream& operator>> (QDataStream&, specDataPoint&);
QDataStream& operator>> (QDataStream&, legacyDatapoint&);
struct specDataPoint
{
        double nu, sig, mint ;
        specDataPoint() ;
        specDataPoint(const double& n, const double& s, const double& m) ;
        bool operator< (const specDataPoint&) const ;
        bool operator== (const specDataPoint&) const ;
        bool exactlyEqual(const specDataPoint&) const ;
        bool operator!= (const specDataPoint& toCompare) const ;
        specDataPoint& operator+= (const specDataPoint&);
        specDataPoint& operator/= (const double&) ;
        specDataPoint& operator/ (const double&) ;
        operator QPointF() const ;
        friend QDataStream& operator<< (QDataStream&, const specDataPoint&);
        friend QDataStream& operator>> (QDataStream&, specDataPoint&);
};
struct legacyDatapoint : specDataPoint
{
        friend QDataStream& operator>> (QDataStream&, legacyDatapoint&) ;
};
#endif
```

src/spectral/specdatapoint.cpp

```
#include "specdatapoint.h"
#include <QTextStream>
#include "utility-functions.h"
QDataStream& operator<< (QDataStream& out, const specDataPoint& point)
{ out << point.nu << point.sig << point.mint ; return out ;}
QDataStream& operator>> (QDataStream& in , specDataPoint& point)
{ in >> point.nu >> point.sig >> point.mint ; return in ; }
specDataPoint::specDataPoint() { nu = 0; sig = 0 ; mint = 0 ; }
specDataPoint::specDataPoint(const double& n, const double& s, const double& m)
{ nu = n; sig = s ; mint = m ;}
bool specDataPoint::operator< (const specDataPoint& toCompare) const
{ return nu < toCompare.nu ;}
bool specDataPoint::operator== (const specDataPoint& toCompare) const
{
        return doubleComparison(toCompare.nu, nu) ;
}
bool specDataPoint::operator!= (const specDataPoint& toCompare) const
{ return !(toCompare == *this) ; }
specDataPoint& specDataPoint::operator+= (const specDataPoint& toAdd)
{
        nu += toAdd.nu ; sig += toAdd.sig ; mint = std::max(toAdd.mint, mint) ;
        return (*this) ;
```

```
}
specDataPoint& specDataPoint::operator/= (const double& toDiv)
{
        nu /= toDiv ; sig /= toDiv ;
        return (*this) ;
}
specDataPoint& specDataPoint::operator/ (const double& toDiv)
{
        *this /= toDiv ;
        return *this ;
}
bool specDataPoint::exactlyEqual(const specDataPoint& o) const
{
        return nu == o.nu && sig == o.sig && mint == o.mint ;
}
QDataStream& operator>> (QDataStream& in, legacyDatapoint& d)
{
        double t ;
        return in >> t >> d.nu >> d.sig >> d.mint ;
}
specDataPoint::operator QPointF() const
{
        return QPointF(nu, sig) ;
}
```

src/spectral/specdataview.h

```
#ifndef SPECDATAVIEW_H
#define SPECDATAVIEW_H
#include "specview.h"
#include <QPersistentModelIndex>
class specDataView : public specView
{
        Q_OBJECT
private:
        type typeId() const { return specStreamable::dataView ;}
public:
        specDataView(QWidget* parent = 0);
        ~specDataView();
};
#endif
```

src/spectral/specdataview.cpp

```
#include "specdataview.h"
#include <QTextStream>
specDataView::specDataView(QWidget* parent)
        : specView(parent)
{
        setWhatsThis(tr("This␣list␣contains␣data␣entries.␣␣Use␣the␣import␣button␣to␣import␣data␣from►
            ␣files.␣␣Manage␣items␣by␣dragging␣and␣dropping."));
        setObjectName("mainView");
}
specDataView::~specDataView()
{
}
```

src/spectral/specpefile.h

```
#ifndef SPECPEFILE_H
#define SPECPEFILE_H
#include <QMap>
#include <QHash>
#include <QString>
class QByteArray ;
class specDataItem ;
class QVariant ;
class QString ;
class specPEFile
{
```

```
private:
        typedef QMap<int, QString> labelMapType ;
        typedef qint32 blockSize ;
        typedef qint16 innerCode ;
        typedef qint16 innerLength ;
        template<typename t> static t readInt(const QByteArray& ba, int pos = 0) ;
        static double readDouble(const QByteArray& ba, int pos = 0) ;
        labelMapType* labelMap ;
        QMap<QString, QVariant> data ;
        QString Description ;
        static QVariant readEntry(const QByteArray&, innerCode code) ;
        QByteArray readItem(const QByteArray&, QString label = QString()) ;
        void readFromArray(const QByteArray&) ;
public:
        specPEFile(const QByteArray&);
        operator specDataItem() const;
};
#endif
```

src/spectral/specpefile.cpp

```
#include "specpefile.h"
#include <QVariant>
#include <QString>
#include <QByteArray>
#include <QStringList>
#include <cmath>
#include "specdescriptor.h"
#include "specdatapoint.h"
#include "specdataitem.h"
#include <QtEndian>
enum specPEFile::blockID
{
    DSet2DC1DIBlock = 120,
    HistoryRecordBlock = 121,
    InstrHdrHistoryRecordBlock = 122,
    InstrumentHeaderBlock = 123,
    IRInstrumentHeaderBlock = 124,
    UVInstrumentHeaderBlock = 125,
    FLInstrumentHeaderBlock = 126
} ;
enum specPEFile::typeCodeID
{
    ShortType = 29999,
    UShortType = 29998,
    IntType = 29997,
    UIntType = 29996,
    LongType = 29995,
    BoolType = 29988,
    CharType = 29987,
    CvCoOrdPointType = 29986,
    StdFontType = 29985,
    CvCoOrdDimensionType = 29984,
    CvCoOrdRectangleType = 29983,
    RGBColorType = 29982,
    CvCoOrdRangeType = 29981,
    DoubleType = 29980,
    CvCoOrdType = 29979,
    ULongType = 29978,
    PeakType = 29977,
    CoOrdType = 29976,
    RangeType = 29975,
    CvCoOrdArrayType = 29974,
    EnumType = 29973,
    LogFontType = 29972
} ;
enum specPEFile::dataMemberID
{
    DataSetDataTypeMember = -29839,
    DataSetAbscissaRangeMember = -29838,
    DataSetOrdinateRangeMember = -29837,
    DataSetIntervalMember = -29836,
    DataSetNumPointsMember = -29835,
    DataSetSamplingMethodMember = -29834,
```

```
    DataSetXAxisLabelMember = -29833,
    DataSetYAxisLabelMember = -29832,
    DataSetXAxisUnitTypeMember = -29831,
    DataSetYAxisUnitTypeMember = -29830,
    DataSetFileTypeMember = -29829,
    DataSetDataMember = -29828,
    DataSetNameMember = -29827,
    DataSetChecksumMember = -29826,
    DataSetHistoryRecordMember = -29825,
    DataSetInvalidRegionMember = -29824,
    DataSetAliasMember = -29823,
    DataSetVXIRAccyHdrMember = -29822,
    DataSetVXIRQualHdrMember = -29821,
    DataSetEventMarkersMember = -29820
} ;
enum specPEFile::historyBlockID
{
    HistoryNumberMember = -29839,
    HistoryUserMember = -29838,
    HistoryActionTypeMember = -29837,
    HistoryTimestampMember = -29836,
    HistoryParameterMember = -29835,
    HistoryDescriptionMember = -29834,
    HistoryPreviousMember = -29833
};
specPEFile::specPEFile(const QByteArray& ba)
        : labelMap(0)
{
        readFromArray(ba);
}
template<typename t>
t specPEFile::readInt(const QByteArray& ba, int pos)
{
        if((size_t) ba.size() < pos + sizeof(t)) return 0 ;
        return qFromLittleEndian(* ((t*)(ba.data() + pos))) ;
}
double specPEFile::readDouble(const QByteArray& ba, int pos)
{
        if((size_t) ba.size() < pos + sizeof(double)) return NAN ;
        return * ((double*)(ba.data() + pos)) ;
}
QVariant specPEFile::readEntry(const QByteArray& in, innerCode code)
{
        if(CharType == code)
        {
                innerLength length = readInt<innerLength> (in) ;
                return QString(in.mid(sizeof(length), qMin<innerLength> (length, in.size() - sizeof(►
                    length)))) ;
        }
        if(CvCoOrdRangeType == code)
                return QList<QVariant>() << readDouble(in) << readDouble(in, sizeof(double)) ;
        if(CvCoOrdType == code || DoubleType == code)
                return readDouble(in) ;
        if(LongType == code)
                return readInt<qint32> (in) ;
        if(EnumType == code)
                return readInt<qint32> (in) ;
        if(UIntType == code)
                return readInt<quint16> (in) ;
        if(ShortType == code)
                return readInt<qint16> (in) ;
        if(ULongType == code)
                return readInt<quint32> (in) ;
        if(CvCoOrdArrayType == code)
        {
                blockSize size = qMin(readInt<blockSize> (in), in.size()) ;
                QList<QVariant> data ;
                for(int i = sizeof(blockSize) ; i < size ; i += sizeof(double))
                        data << * ((double*)(in.data() + i)) ;
                return data;
        }
        return QVariant() ;
}
QByteArray specPEFile::readItem(const QByteArray& in, QString label)
{
```

```
        typedef qint16 blockType ;
        blockType type = readInt<blockType> (in) ;
        if(29970 < type)
        {
                if(!label.isEmpty())
                {
                        QVariant newEntry = readEntry(in.mid(sizeof(type)), type) ;
                        if(data.contains(label))
                        {
                                QVariant v = data[label] ;
                                if(v.canConvert(QVariant::StringList))
                                        v = (v.toStringList() << newEntry.toString()) ;
                                else
                                        v = (QStringList() << v.toString() << newEntry.toString()) ;
                                data[label] = v ;
                        }
                        else
                                data[label] = newEntry ;
                }
                return QByteArray() ;
        }
        blockSize size = readInt<blockSize> (in, sizeof(type)) ;
        QByteArray core(in.mid(sizeof(type) + sizeof(size), qMin((size_t) size, in.size() - sizeof(►
            type) - sizeof(size)))),
                   remainder = in.mid(core.size() + sizeof(type) + sizeof(size)) ;
        labelMapType* oldLabels = labelMap ;
        switch(type)
        {
                case DSet2DC1DIBlock:
                case DataSetHistoryRecordMember:
                case HistoryRecordBlock:
                case InstrHdrHistoryRecordBlock:
                case IRInstrumentHeaderBlock:
                        labelMap = new labelMapType ;
                default: ;
        }
        switch(type)
        {
                case DSet2DC1DIBlock:
                        labelMap->insert(-29839, "DataSetDataTypeMember") ;
                        labelMap->insert(-29838, "DataSetAbscissaRangeMember") ;
                        labelMap->insert(-29837, "DataSetOrdinateRangeMember") ;
                        labelMap->insert(-29836, "DataSetIntervalMember") ;
                        labelMap->insert(-29835, "DataSetNumPointsMember") ;
                        labelMap->insert(-29834, "DataSetSamplingMethodMember") ;
                        labelMap->insert(-29833, "DataSetXAxisLabelMember") ;
                        labelMap->insert(-29832, "DataSetYAxisLabelMember") ;
                        labelMap->insert(-29831, "DataSetXAxisUnitTypeMember") ;
                        labelMap->insert(-29830, "DataSetYAxisUnitTypeMember") ;
                        labelMap->insert(-29829, "DataSetFileTypeMember") ;
                        labelMap->insert(-29828, "DataSetDataMember") ;
                        labelMap->insert(-29827, "DataSetNameMember") ;
                        labelMap->insert(-29826, "DataSetChecksumMember") ;
                        labelMap->insert(-29825, "DataSetHistoryRecordMember") ;
                        labelMap->insert(-29824, "DataSetInvalidRegionMember") ;
                        labelMap->insert(-29823, "DataSetAliasMember") ;
                        labelMap->insert(-29822, "DataSetVXIRAccyHdrMember") ;
                        labelMap->insert(-29821, "DataSetVXIRQualHdrMember") ;
                        labelMap->insert(-29820, "DataSetEventMarkersMember") ;
                        break ;
                case HistoryRecordBlock:
                        labelMap->insert(-29839, "HistoryNumberMember") ;
                        labelMap->insert(-29838, "HistoryUserMember") ;
                        labelMap->insert(-29837, "HistoryActionTypeMember") ;
                        labelMap->insert(-29836, "HistoryTimestampMember") ;
                        labelMap->insert(-29835, "HistoryParameterMember") ;
                        labelMap->insert(-29834, "HistoryDescriptionMember") ;
                        labelMap->insert(-29833, "HistoryPreviousMember") ;
                        break ;
                case IRInstrumentHeaderBlock:
                        for(int i = -29700 ; i < -29600 ; ++i)
                                labelMap->insert(i, "Instrument") ;
                        break ;
                default: ;
        }
```

```
        while(!core.isEmpty())
                core = readItem(core, labelMap ? labelMap->value(type) : QString()) ;
        if(labelMap != oldLabels)
        {
                delete labelMap ;
                labelMap = oldLabels ;
        }
        return remainder ;
}
void specPEFile::readFromArray(const QByteArray& in)
{
        if(in.left(4) != "PEPE") return ;
        Description = in.mid(4, 40) ;
        readItem(in.mid(44)) ;
}
specPEFile::operator specDataItem() const
{
        QHash<QString, specDescriptor> description ;
        foreach(const QString & key, data.keys())
        {
                if(key == "DataSetAbscissaRangeMember" ||
                        key == "DataSetOrdinateRangeMember" ||
                        key == "DataSetIntervalMember" ||
                        key == "DataSetDataMember")
                        continue ;
                QVariant v = data[key] ;
                if(QVariant::Bool == v.type()
                        || QVariant::Double == v.type()
                        || QVariant::Int == v.type()
                        || QVariant::UInt == v.type()
                        || QVariant::ULongLong == v.type()
                        || QVariant::LongLong == v.type())
                        description[key] = v.toDouble() ;
                else
                        description[key] = v.toStringList().join("\n") ;
        }
        QVector<specDataPoint> dataPoints ;
        QList<QVariant> limits = data["DataSetAbscissaRangeMember"].toList() ;
        double begin = 0 ;
        if(!limits.isEmpty()) begin = limits.first().toDouble() ;
        double step = data["DataSetIntervalMember"].toDouble() ;
        QList<QVariant> dataList = data["DataSetDataMember"].toList() ;
        foreach(const QVariant & point, dataList)
        {
                dataPoints << specDataPoint(begin, point.toDouble() , 0.) ;
                begin += step ;
        }
        return specDataItem(dataPoints, description, 0, Description) ;
}
```

src/textEditor/specsimpletextedit.h

```
#ifndef SPECSIMPLETEXTEDIT_H
#define SPECSIMPLETEXTEDIT_H
#include <QWidget>
#include <QTextEdit>
#include <QToolBar>
#include <QAction>
#include <QFontComboBox>
#include <QDoubleSpinBox>
class specSimpleTextEdit : public QWidget
{
        Q_OBJECT
private:
        QTextEdit* content ;
        QToolBar* toolBar ;
        QAction* bold ;
        QAction* italic ;
        QFontComboBox* font ;
        QDoubleSpinBox* size ;
private slots:
        void toggleBold() ;
        void toggleItalic() ;
        void formatChange(const QTextCharFormat&) ;
```

```
        void changeFontSize(const double&) ;
        void newContent() ;
public:
        explicit specSimpleTextEdit(QWidget* parent = 0);
        void setText(const QString&) ;
        QString getText() const ;
signals:
        void contentChanged(QString newContent) ;
};
#endif
```

src/textEditor/specsimpletextedit.cpp

```
#include "specsimpletextedit.h"
#include <QVBoxLayout>
specSimpleTextEdit::specSimpleTextEdit(QWidget* parent) :
        QWidget(parent),
        content(new QTextEdit),
        toolBar(new QToolBar),
        bold(new QAction(QIcon::fromTheme("format-text-bold"), tr("Fett"), this)),
        italic(new QAction(QIcon::fromTheme("format-text-italic"), tr("Kursiv"), this)),
        font(new QFontComboBox(this)),
        size(new QDoubleSpinBox(this))
{
        setLayout(new QVBoxLayout) ;
        layout()->addWidget(toolBar) ;
        layout()->addWidget(content) ;
        bold->setCheckable(true) ;
        italic->setCheckable(true) ;
        toolBar->addAction(bold) ;
        toolBar->addAction(italic) ;
        toolBar->addWidget(font) ;
        toolBar->addWidget(size) ;
        QAction* undoAction = new QAction(QIcon::fromTheme("edit-undo"), tr("undo"), this) ;
        QAction* redoAction = new QAction(QIcon::fromTheme("edit-redo"), tr("redo"), this) ;
        undoAction->setDisabled(true) ;
        redoAction->setDisabled(true) ;
        connect(undoAction, SIGNAL(triggered()), content, SLOT(undo())) ;
        connect(redoAction, SIGNAL(triggered()), content, SLOT(redo())) ;
        connect(content, SIGNAL(undoAvailable(bool)), undoAction, SLOT(setEnabled(bool))) ;
        connect(content, SIGNAL(redoAvailable(bool)), redoAction, SLOT(setEnabled(bool))) ;
        toolBar->addAction(undoAction) ;
        toolBar->addAction(redoAction) ;
        formatChange(content->currentCharFormat()) ;
        connect(bold, SIGNAL(triggered()), this, SLOT(toggleBold())) ;
        connect(italic, SIGNAL(triggered()), this, SLOT(toggleItalic())) ;
        connect(font, SIGNAL(currentFontChanged(QFont)), content, SLOT(setCurrentFont(QFont))) ;
        connect(content, SIGNAL(currentCharFormatChanged(QTextCharFormat)), this, SLOT(formatChange(►
            QTextCharFormat))) ;
        connect(size, SIGNAL(valueChanged(double)), this, SLOT(changeFontSize(double))) ;
        connect(content, SIGNAL(textChanged()), this, SLOT(newContent())) ;
}
void specSimpleTextEdit::newContent()
{
        emit contentChanged(getText());
}
void specSimpleTextEdit::toggleBold()
{
        QFont font = content->currentFont() ;
        font.setBold(! font.bold());
        content->setCurrentFont(font) ;
}
void specSimpleTextEdit::toggleItalic()
{
        content->setFontItalic(! content->currentFont().italic());
}
void specSimpleTextEdit::formatChange(const QTextCharFormat& format)
{
        QFont currentFont = format.font() ;
        bold->setChecked(currentFont.bold()) ;
        italic->setChecked(currentFont.italic()) ;
        font->setCurrentFont(currentFont) ;
        size->setValue(currentFont.pointSizeF());
}
```

```
QString specSimpleTextEdit::getText() const
{
        return content->toHtml() ;
}
void specSimpleTextEdit::setText(const QString& text)
{
        content->clear() ;
        content->setHtml(text) ;
}
void specSimpleTextEdit::changeFontSize(const double& newSize)
{
        QFont font = content->currentFont() ;
        font.setPointSizeF(newSize);
        content->setCurrentFont(font) ;
}
```

src/utilities/bzipiodevice.h

```
#ifndef BZIPIODEVICE_H
#define BZIPIODEVICE_H
#include <QIODevice>
struct bzipIODevicePrivate ;
class bzipIODevice : public QIODevice
{
        Q_OBJECT
private:
        QIODevice* internalDevice ;
        bzipIODevicePrivate* zipStream ;
        QByteArray buffer ;
        qint64 lastPos ;
        qint64 readData(char* data, qint64 maxlen) ;
        qint64 writeData(const char* data, qint64 len) ;
        bool bufferToDevice() ;
        void prepareInBuffer(int minsize) ;
        void fillOutBuffer(int minsize) ;
        bool writeBuffer() ;
public:
        explicit bzipIODevice(QIODevice* inputDevice = 0, QObject* parent = 0);
        ~bzipIODevice() ;
        QIODevice* releaseDevice() ;
        bool atEnd() const ;
        void close() ;
        bool isSequential() const ;
        bool open(OpenMode mode) ;
};
#endif
```

src/utilities/bzipiodevice.cpp

```
#include "bzipiodevice.h"
#include <bzlib.h>
struct bzipIODevicePrivate : bz_stream
{
        bzipIODevicePrivate()
        {
                bzalloc = 0 ;
                bzfree = 0 ;
                opaque = 0 ;
        }
        qint64 outPos() const
        {
                return ((qint64) total_out_hi32 << 32) + total_out_lo32 ;
        }
        qint64 inPos() const
        {
                return ((qint64) total_in_hi32 << 32) + total_in_lo32 ;
        }
};
bzipIODevice::bzipIODevice(QIODevice* inputDevice, QObject* parent) :
        QIODevice(parent),
        internalDevice(inputDevice),
        zipStream(new bzipIODevicePrivate),
```

```
        buffer(1, 0),
        lastPos(0)
{
}
bzipIODevice::~bzipIODevice()
{
        close() ;
        delete internalDevice ;
        delete zipStream ;
}
bool bzipIODevice::open(OpenMode mode)
{
        if(isOpen()) return false ;
        if(!internalDevice) return false ;
        if(!QIODevice::open(mode)) return false ;
        if(internalDevice->openMode() != mode && !internalDevice->open(mode)) return false ;
        lastPos = 0 ;
        if(mode == ReadOnly)
        {
                if(BZ2_bzDecompressInit(zipStream, 0, 0) != BZ_OK) return false ;
                zipStream->avail_in = 0 ;
                zipStream->next_in = 0 ;
                return true ;
        }
        if(mode == WriteOnly) return BZ2_bzCompressInit(zipStream, 9, 0, 0) == BZ_OK ;
        return false ;
}
bool bzipIODevice::bufferToDevice()
{
        if(buffer.size() == internalDevice->write(buffer))
        {
                buffer.clear();
                return true ;
        }
        return false ;
}
void bzipIODevice::close()
{
        if(!isOpen()) return ;
        if(openMode() == ReadOnly) BZ2_bzDecompressEnd(zipStream) ;
        if(openMode() == WriteOnly)
        {
                prepareInBuffer(0) ;
                while(BZ2_bzCompress(zipStream, BZ_FINISH) != BZ_STREAM_END)
                {
                        writeBuffer() ;
                        prepareInBuffer(0);
                }
                writeBuffer() ;
                BZ2_bzCompressEnd(zipStream) ;
        }
        QIODevice::close() ;
        internalDevice->close() ;
}
void bzipIODevice::prepareInBuffer(int minsize)
{
        buffer.resize(qMax(buffer.size(), minsize)) ;
        zipStream->next_out = buffer.data() ;
        zipStream->avail_out = buffer.size() ;
        lastPos = zipStream->outPos() ;
}
void bzipIODevice::fillOutBuffer(int minsize)
{
        buffer = internalDevice->read(minsize) ;
        zipStream->next_in = buffer.data() ;
        zipStream->avail_in = buffer.size() ;
}
bool bzipIODevice::writeBuffer()
{
        qint64 bytes = zipStream->outPos() - lastPos ;
        if(internalDevice->write(buffer.data(), bytes) != bytes) return false ;
        lastPos += bytes ;
        return true ;
}
bool bzipIODevice::atEnd() const
```

```
{
        if(internalDevice && zipStream && zipStream->state == BZ_RUN && errorString().isEmpty()) ►
            return false;
        return true ;
}
bool bzipIODevice::isSequential() const
{
        return true ;
}
qint64 bzipIODevice::writeData(const char* data, qint64 len)
{
        if(!len) return 0 ;
        if(!internalDevice) return -1 ;
        if(!isOpen()) return -1 ;
        if(openMode() != WriteOnly) return -1 ;
        zipStream->next_in = const_cast<char*>(data) ;
        zipStream->avail_in = len ;
        while(zipStream->avail_in)
        {
                prepareInBuffer(len) ;
                int rv = BZ2_bzCompress(zipStream, BZ_RUN) ;
                if(rv != BZ_OK && rv != BZ_RUN_OK) break ;
                if(!writeBuffer()) return -1 ;
        }
        return len ;
}
qint64 bzipIODevice::readData(char* data, qint64 maxlen)
{
        if(!maxlen) return 0 ;
        if(!internalDevice) return -1 ;
        if(!isOpen()) return -1 ;
        if(openMode() != ReadOnly) return -1 ;
        zipStream->next_out = data ;
        zipStream->avail_out = maxlen ;
        while(zipStream->avail_out)
        {
                if(!zipStream->avail_in)
                {
                        fillOutBuffer(maxlen) ;
                        if(!zipStream->avail_in) break ;
                }
                int rv = BZ2_bzDecompress(zipStream) ;
                if(rv == BZ_STREAM_END) break ;
                if(rv != BZ_OK) return -1 ;
        }
        qint64 written = zipStream->outPos() - lastPos ;
        lastPos += written ;
        return written ;
}
QIODevice* bzipIODevice::releaseDevice()
{
        close() ;
        QIODevice* dev = internalDevice ;
        internalDevice = 0 ;
        return dev ;
}
```

src/utility-functions.h

```
#ifndef SPECUTILITIES_H
#define SPECUTILITIES_H
#ifdef DOUBLEDEVIATIONCORRECTION
#include <cfloat>
#endif
#include <QList>
#include <QVector>
#include <QHash>
#include <QString>
#include <QStringList>
#include "specdataitem.h"
#include <QTextStream>
#include <QFile>
#include <names.h>
QList<specModelItem*> readLogFile(QFile& file) ;
```

```
QList<specModelItem*> readHVFile(QFile&) ;
QList<specModelItem*> readPEFile(QFile&) ;
QList<specModelItem*> readPEBinary(QFile&) ;
QList<specModelItem*> readJCAMPFile(QFile&) ;
QList<specModelItem*> readSKHIFile(QFile&) ;
QList<specModelItem*> readXYFILE(QFile&) ;
QPair<QString, specDescriptor> readJCAMPldr(QString& first, QTextStream& in) ;
specModelItem* readJCAMPBlock(QTextStream& in) ;
void readJCAMPdata(QTextStream& in, QVector<specDataPoint>& data, double step, double xfactor, ►
    double yfactor) ;
QVector<double> waveNumbers(QTextStream&) ;
QHash<QString, specDescriptor> fileHeader(QTextStream&) ;
specFileImportFunction fileFilter(const QString& fileName);
QVector<double> gaussjinv(QVector<QVector<double> >&, QVector<double>&) ;
bool comparePoints(const QPointF&, const QPointF&) ;
template<typename T, class forwardIterator, typename binaryPredicate>
inline T average(forwardIterator& begin, const forwardIterator& end, binaryPredicate equal, const T ►
    first)
{
        begin++ ;
        T retVal = first ;
        int count = 1 ;
        while(begin != end && equal(*begin, first))
        {
                retVal += *begin++ ;
                ++ count ;
        }
        return retVal /= (double) count ;
}
QwtSymbol* cloneSymbol(const QwtSymbol* original) ;
template<class forwardIterator, class outputIterator, typename binaryPredicate>
inline void averageToNew(forwardIterator begin, forwardIterator end,
                         binaryPredicate equal, outputIterator target)
{
        while(begin != end)
                *target++ = average(begin, end, equal, *begin) ;
}
inline bool doubleComparison(const double& a, const double& b)
{
#ifdef DOUBLEDEVIATIONCORRECTION
        return fabs(a - b) <= 2 * DBL_EPSILON * qMax(qMax(1.0, fabs(a)), fabs(b)) ;
#else
        return a == b ;
#endif
}
#endif
```

src/utility-functions.cpp

```
#include <utility-functions.h>
#include <QTextCodec>
#include <QFileInfo>
#include <QPair>
#include <math.h>
#include <names.h>
#include "specdescriptor.h"
#include "speclogentryitem.h"
#include "speclogmessage.h"
#include <QQueue>
#include <QtDebug>
#include <QPointF>
#include "specdatapoint.h"
#include <QMessageBox>
#include <QObject>
#include <QDoubleValidator>
#include "specprofiler.h"
#include "specpefile.h"
using std::max ;
using std::min ;
bool comparePoints(const QPointF& a, const QPointF& b)
{
        return a.x() < b.x() ;
}
QList<specModelItem*> readJCAMPFile(QFile& file)
```

```
{
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO␣8859-1")) ;
        QString title = in.readLine() ;
        QPair<QString, specDescriptor> titleLine = readJCAMPldr(title, in) ;
        if(titleLine.first != "TITLE") return QList<specModelItem*>() ;
        specModelItem* item = readJCAMPBlock(in) ;
        item->changeDescriptor("", titleLine.second.content()) ;
        return (QList<specModelItem*>() << item) ;
}
QPair<QString, specDescriptor> readJCAMPldr(QString& first, QTextStream& in)
{
        QQueue<QString> toInterpret ;
        do
        {
                toInterpret.enqueue(first) ;
                first = in.readLine() ;
        }
        while(first.left(2) != "##" && !in.atEnd()) ;
        int pos = toInterpret[0].indexOf("=") ;
        QString label = toInterpret[0].mid(2, pos - 2) ;
        toInterpret[0].remove(0, pos + 1) ;
        QString data, nl ;
        do
        {
                data += nl ;
#ifdef STRICT_JCAMP
                data += (toInterpret.dequeue()).left(80) ;
#else
                data += (toInterpret.dequeue());
#endif
                int commentBegins = data.indexOf("$$") ;
                if(commentBegins >= 0)
                        data.truncate(commentBegins) ;
                nl = '\n' ;
        }
        while(!toInterpret.isEmpty()) ;
        return QPair<QString, specDescriptor> (label, specDescriptor(data)) ;
}
specModelItem* readJCAMPBlock(QTextStream& in)
{
        QString first = in.readLine() ;
        QHash<QString, specDescriptor> descriptors ;
        QList<specModelItem*> children ;
        QPair<QString, specDescriptor> ldr = readJCAMPldr(first, in) ;
        descriptors[ldr.first] = ldr.second ;
        QVector<specDataPoint> data;
        while(!in.atEnd())
        {
                QPair<QString, specDescriptor> ldr = readJCAMPldr(first, in) ;
                QString parsedLabel = ldr.first.toUpper().remove(QRegExp("[\\s\\-\\/\\_]")) ;
                if(parsedLabel == "END") break ;
                else if(parsedLabel == "TITLE")
                {
                        children << readJCAMPBlock(in) ;
                        children.last()->changeDescriptor("", ldr.second.content()) ;
                }
                else if(parsedLabel == "XYDATA")
                {
                        QString content = ldr.second.content(true) ;
                        QTextStream dataStream(&content) ;
                        QString typeOfData = dataStream.readLine() ;
                        if(typeOfData == "(X++(Y..Y))")
                                readJCAMPdata(dataStream, data,
                                              (descriptors["LASTX"].content().toDouble() - ►
                                                  descriptors["FIRSTX"].content().toDouble()) / (►
                                                  descriptors["NPOINTS"].content().toDouble() - 1),
                                              descriptors["XFACTOR"].content().toDouble(),
                                              descriptors["YFACTOR"].content().toDouble()
                                             ) ;
                }
                else
                        descriptors[ldr.first] = ldr.second ;
        }
        specModelItem* item = 0 ;
```

```
        if(descriptors["DATA TYPE"].content() == "LINK")
        {
101             item = new specFolderItem() ;
                for(QHash<QString, specDescriptor>::iterator i = descriptors.begin() ; i != ►
                    descriptors.end() ; i++)
                        foreach(specModelItem * child, children)
                        child->changeDescriptor(i.key(), i.value().content()) ;
                item->addChildren(children, 0) ;
106     }
        else
        {
                item = new specDataItem(data, descriptors) ;
                foreach(specModelItem * child, children)
111             delete child ;
        }
        item->invalidate();
        return item ;
}
116 void readJCAMPdata(QTextStream& in, QVector<specDataPoint>& data, double step, double xfactor, ►
    double yfactor)
{
        QRegExp specialCharacters("[?@%A-DF-Za-df-s+\\-\\s]|[Ee](?![+\\-\\s])"),
                posSQZdigits("[A-DF-I@]|E(?![+\\-\\s])"),
                negSQZdigits("[a-df-i]|e(?![+\\-\\s])"),
121             posDIFdigits("[J-R%]"),
                negDIFdigits("[j-r]"),
                posDUPdigits("[S-Zs]") ;
        bool wasDiff = false ;
        QVector<double> yvals, xvals ;
126     while(!in.atEnd())
        {
                QString line = in.readLine() ;
                QString x = line.left(line.indexOf(specialCharacters)) ;
                line.remove(0, line.indexOf(specialCharacters)) ;
131             xvals << (xvals.isEmpty() ? x.toDouble() *xfactor : xvals.last() + step) ;
                if(wasDiff)
                {
                        xvals.remove(xvals.size() - 1) ;
                        int itemLength = line.indexOf(specialCharacters, 1) ;
136                     if(itemLength == -1) itemLength = line.size() ;
                        QString yInput = line.left(itemLength) ;
                        line.remove(0, itemLength) ;
                        yInput.remove(QRegExp("[\\s]")) ;
                        if(yInput.isEmpty()) break ;
141                     QString firstChar = yInput.left(1) ;
                        yInput.remove(0, 1) ;
                        if(firstChar == "?") break ;
                        if(posSQZdigits.exactMatch(firstChar))
                                yInput.prepend(firstChar == "@" ? "+0" :
146                                             QString("%1").arg(firstChar.data()->toAscii() - 'A' +►
                                                    1)) ;
                        else if(negSQZdigits.exactMatch(firstChar))
                                yInput.prepend(QString("-%1").arg(firstChar.data()->toAscii() - 'a' ►
                                    + 1)) ;
                        else
                                yInput.prepend(firstChar) ;
151                     if(yInput.toDouble() *yfactor != yvals.last())
                                break ;
                }
                while(!line.isEmpty())
                {
156                     int itemLength = line.indexOf(specialCharacters, 1) ;
                        if(itemLength == -1) itemLength = line.size() ;
                        QString yInput = line.left(itemLength) ;
                        line.remove(0, itemLength) ;
                        yInput.remove(QRegExp("[\\s]")) ;
161                     if(yInput.isEmpty()) break ;
                        QString firstChar = yInput.left(1) ;
                        yInput.remove(0, 1) ;
                        if(firstChar == "?")
                        {
166                             xvals << NAN ;
                                line.prepend(yInput) ;
                                wasDiff = false ;
                                break ;
```

```
                        }
                        if(posSQZdigits.exactMatch(firstChar))
                        {
                                yInput.prepend(firstChar == "@" ? "+0" :
                                                QString("%1").arg(firstChar.data()->toAscii() - 'A' +►
                                                        1)) ;
                                yvals << yInput.toDouble() *yfactor ;
                                wasDiff = false ;
                        }
                        else if(negSQZdigits.exactMatch(firstChar))
                        {
                                yInput.prepend(QString("-%1").arg(firstChar.data()->toAscii() - 'a' ►
                                    + 1)) ;
                                yvals << yInput.toDouble() *yfactor ;
                                wasDiff = false ;
                        }
                        else if(posDIFdigits.exactMatch(firstChar))
                        {
                                yInput.prepend(firstChar == "%" ? "+0" :
                                                QString("%1").arg(firstChar.data()->toAscii() - 'I')) ;
                                yvals << yInput.toDouble() *yfactor + (yvals.isEmpty() ? NAN : yvals►
                                    .last()) ;
                                wasDiff = true ;
                        }
                        else if(negDIFdigits.exactMatch(firstChar))
                        {
                                yInput.prepend(QString("-%1").arg(firstChar.data()->toAscii() - 'i')►
                                    ) ;
                                yvals << yInput.toDouble() *yfactor + (yvals.isEmpty() ? NAN : yvals►
                                    .last()) ;
                                wasDiff = true ;
                        }
                        else if(posDUPdigits.exactMatch(firstChar))
                        {
                                yInput.prepend(firstChar == QString('s') ? "9" : QString("%1").arg(►
                                    firstChar.data()->toAscii() - 'R')) ;
                                int count = yInput.toInt() - 1 ;
                                if(wasDiff)
                                {
                                        if(yvals.size() >= 2)
                                                for(int i = 0 ; i < count ; i++)
                                                        yvals << 2.*yvals.last() - yvals[yvals.size►
                                                            () - 2] ;
                                        else
                                                for(int i = 0 ; i < count ; i++)
                                                        yvals << NAN ;
                                }
                                else
                                {
                                        if(yvals.size() >= 1)
                                                for(int i = 0 ; i < count ; i++)
                                                        yvals << yvals.last() ;
                                        else
                                                for(int i = 0 ; i < count ; i++)
                                                        yvals << NAN ;
                                }
                        }
                        else
                        {
                                yvals << QString("%1%2").arg(firstChar, yInput).toDouble() *yfactor ;
                        }
                        while(xvals.size() < yvals.size())
                                xvals << xvals.last() + step ;
                }
        }
        if(yvals.size() != xvals.size()) return ;
        for(int i = 0 ; i < xvals.size() ; i++)
                data << specDataPoint(xvals[i], yvals[i], 0) ;
}
QHash<QString, specDescriptor> fileHeader(QTextStream& in)
{
        QStringList headerContent = in.readLine().split(QRegExp(",␣(?=(\\w+\\s)*\\w+:)")) ;
        QHash<QString, specDescriptor> headerItems ;
        for(QStringList::size_type i = 0 ; i < headerContent.size() ; i++)
        {
```

```
                QString content = QStringList(headerContent[i].split(": ").mid(1)).join(": ").remove►
                    (QRegExp("^\"")).remove(QRegExp("\"$")) ;
                headerItems[headerContent[i].split(": ") [0]] = (specDescriptor(content,
                        (content.contains(QRegExp("\\D"))) ? spec::editable : spec::numeric)) ;
241     }
        if(headerItems["Kommentar"].content().contains("@"))
                headerItems["Pump"] = specDescriptor(headerItems["Kommentar"].content().split("@ ") ►
                    [1].split(" nm") [0], spec::numeric) ;
        return headerItems ;
}
246 void fileHeader(QString header, QHash<QString, specDescriptor>& description)
{
        QStringList headerContent = header.split(QRegExp(", (?=(\\w+\\s)*\\w+:)")) ;
        QHash<QString, specDescriptor> headerItems ;
        QDoubleValidator validator ;
251     int a = 0 ;
        foreach(QString headerEntry, headerContent)
        {
                a = 0 ;
                QString name = headerEntry.section(": ", 0, 0),
256                     content = headerEntry.section(": ", 1) ;
                content.remove(QRegExp("^\"")).remove(QRegExp("\"$")) ;
                description[name] = specDescriptor(content,
                                                   validator.validate(content, a) == QValidator::►
                                                       Acceptable ?
                                                   spec::numeric : spec::editable) ;
261     }
        if(description["Kommentar"].content().contains("@"))
                description["Pump"] = specDescriptor(description["Kommentar"].content().section("@ "►
                    , 1, -1).section(" nm", 0, 0), spec::numeric) ;
}
QVector<double> waveNumbers(const QStringList& wns)
266 {
        QVector<double> wavenumbers ;
        foreach(QString wn, wns) wavenumbers += wn.toDouble() ;
        return wavenumbers ;
}
271 QList<specModelItem*> readHVMeasurement(const QString& measurement, QString filename)
{
        QStringList lines = measurement.split("\n") ;
        QList<specModelItem*> newItems ;
        QHash<QString, specDescriptor> headerItems ;
276     headerItems["Datei"] = filename ;
        fileHeader(lines.takeFirst(), headerItems) ;
        QStringList wns = lines.takeFirst().split(" ") ;
        bool polarisatorMessung = wns.takeFirst().toInt() ;
        QVector<QVector<double> > wavenumbers ;
281     QStringList::iterator wnsIt = wns.begin() ;
        for(int i = 0 ; i + 32 <= wns.size() ; i += 32)
        {
                wavenumbers << QVector<double>() ;
                for(int j = 0 ; j < 32 ; ++j)
286                     wavenumbers.last() << (wnsIt++)->toDouble() ;
        }
        int counter = 0 ;
        for(QStringList::iterator it = lines.begin() ; it != lines.end() ; ++it)
        {
291             QStringList dataEntries = it->split(QRegExp("[ ()]"), QString::SkipEmptyParts) ;
                if(dataEntries.size() < 1) continue ;
                headerItems["Zeit"] = dataEntries.takeFirst().toDouble() ;
                QStringList::iterator entry = dataEntries.begin() ;
                for(int i = 0 ; (i + 1) * 32 * 2 <= dataEntries.size() ; ++i)
296                 {
                        QVector<QVector<double> >::iterator currentWns = wavenumbers.begin()
                                + i / (polarisatorMessung ? 2 : 1) ;
                        QVector<specDataPoint> dataPoints ;
                        for(int j = 0 ; j < 32 ; ++j)
301                         {
                                double value = (entry++)->toDouble() ;
                                double mintv = (entry++)->toDouble() ;
                                dataPoints << specDataPoint(currentWns->at(j), value, mintv) ;
                        }
306                     headerItems["nu"] = (dataPoints.size() > 1) ?
                                            (dataPoints.first().nu + dataPoints.last().nu) / 2. :
                                            ((dataPoints.size() == 1) ? dataPoints.first().nu : NAN) ;
```

```
                        if(polarisatorMessung) headerItems["Polarisation"] = i % 2 ;
                        headerItems["Index"] = counter ++ ;
                        newItems << new specDataItem(dataPoints, headerItems) ;
                }
        }
        return newItems ;
}
QList<specModelItem*> readHVFile(QFile& file)
{
        specProfiler profiler("Datei einlesen") ;
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO 8859-1")) ;
        QStringList measurements = in.readAll().split(QRegExp("\n(?=Solvens)"), QString::►
            SkipEmptyParts) ;
        QString filename = QFileInfo(file.fileName()).fileName() ;
        specFolderItem* top = new specFolderItem(0, filename) ;
        if(measurements.size() == 1) top->addChildren(readHVMeasurement(measurements.first(), ►
            filename)) ;
        else
        {
                int counter = 0 ;
                foreach(const QString & measurement, measurements)
                {
                        specFolderItem* folder = new specFolderItem(0, QString::number(counter++)) ;
                        folder->addChildren(readHVMeasurement(measurement, filename)) ;
                        top->addChild(folder, top->children()) ;
                }
        }
        return QList<specModelItem*>() << top ;
}
QPair<QString, QString> interpretString(QString& string)
{
        if(string.left(1) == "\"")
        {
                string.remove(0, 1) ;
                int final = string.indexOf("\"") ;
                QString fileName = string.mid(0, final) ;
                string.remove(0, 2 + final) ;
                return QPair<QString, QString> ("Datei", fileName) ;
        }
        if(string.left(7) == "Messung")
        {
                bool success = (string.mid(8, 1) == "e") ;
                string.remove(0, string.indexOf(".") + 2) ;
                return QPair<QString, QString> ("Erfolg", success ? "Ja" : "Nein") ;
        }
        QString descriptor = string.mid(0, string.indexOf(": ")), content ;
        string.remove(0, qMax(string.indexOf(QRegExp("[^:\\s]"), descriptor.size()), descriptor.size►
            ())) ;
        if(string.left(1) == "\"")
        {
                content = string.mid(0, string.remove(0, 1).indexOf("\"")) ;
                string.remove(0, max(content.size() + 1, string.indexOf(QRegExp("[^,.\\s\\\"]"), ►
                    content.size()))) ;
        }
        else
        {
                content = string.mid(0, string.indexOf(QRegExp("[,.\\D]{4,}"))) ;
                string.remove(0, max(content.size(), string.indexOf(QRegExp("[^,.\\s\\\"]"), content►
                    .size()))) ;
        }
        return QPair<QString, QString> (descriptor, content) ;
}
QString takeDateOrTime(QString& string)
{
        QString returnString(string.left(string.indexOf(" "))) ;
        string.remove(0, 1 + string.indexOf(" ")) ;
        string.remove(QRegExp("^: ")) ;
        return returnString ;
}
QList<specModelItem*> readLogFile(QFile& file)
{
        QTextStream in(&file) ;
        QList<specModelItem*> list ;
        in.setCodec(QTextCodec::codecForName("ISO 8859-1")) ;
```

```
        QList<specModelItem*> logData ;
        QString firstLine, secondLine ;
381     while(!in.atEnd())
        {
                QHash<QString, specDescriptor> descriptors ;
                if(firstLine.isEmpty()) firstLine = in.readLine() ;
                for(QString::iterator i = firstLine.begin() ; i != firstLine.end() ; ++i)
386             {
                        if(!(QChar(*i).isPrint()))
                        {
                                QMessageBox::critical(0, QObject::tr("Binary␣File"),
                                                      QObject::tr("File␣") +
391                                                   file.fileName() +
                                                      QObject::tr("␣seems␣to␣be␣binary.␣␣Aborting␣►
                                                           import␣(found␣non-printable␣character␣at␣►
                                                           position␣") +
                                                      QString::number(in.pos()) +
                                                      QObject::tr(").␣␣Context:\n") + firstLine) ;
                                for(QList<specModelItem*>::iterator j = logData.begin() ; j != ►
                                     logData.end() ; ++j)
396                                      delete *j ;
                                return QList<specModelItem*>() ;
                        }
                }
                QRegExp dateTimeString("^\\d\\d\\.\\d\\d\\.\\d\\d␣\\d\\d:\\d\\d:\\d\\d␣:␣") ;
401             if(!firstLine.isEmpty() && !dateTimeString.exactMatch(firstLine.left(20)))
                {
                        QMessageBox::critical(0, QObject::tr("Not␣a␣log␣file"),
                                              QObject::tr("File␣") +
                                              file.fileName() +
406                                           QObject::tr("␣does␣not␣conform␣with␣the␣log␣file␣►
                                                format␣(no␣date␣and␣time␣at␣beginning␣of␣a␣line␣at►
                                                ␣position␣") +
                                              QString::number(in.pos()) +
                                              QObject::tr(").␣Offending␣line:\n")
                                              + firstLine) ;
                        for(QList<specModelItem*>::iterator j = logData.begin() ; j != logData.end()►
                              ; ++j)
411                             delete *j ;
                        return QList<specModelItem*>() ;
                }
                if(!in.atEnd()) secondLine = in.readLine() ;
                descriptors["Tag"] = specDescriptor(takeDateOrTime(secondLine)) ;
416             descriptors["Uhrzeit"] = specDescriptor(takeDateOrTime(secondLine)) ;
                if(firstLine == "")
                {
                        while(secondLine.count("\"") % 2)
                                secondLine += "\n" + in.readLine() ;
421                     QPair<QString, QString> newDescriptor = interpretString(secondLine) ;
                        descriptors["Wert"] = newDescriptor.second ;
                        QString mainDescriptor = newDescriptor.first ;
                        while(!secondLine.isEmpty())
                        {
426                             newDescriptor = interpretString(secondLine) ;
                                descriptors[newDescriptor.first] = newDescriptor.second ;
                        }
                        logData += new specLogMessage(descriptors, 0, mainDescriptor) ;
                }
431             else
                {
                        descriptors["A-Tag"] = specDescriptor(takeDateOrTime(firstLine)) ;
                        descriptors["A-Zeit"] = specDescriptor(takeDateOrTime(firstLine)) ;
                        while(firstLine.size())
436                     {
                                QPair<QString, QString> newDescriptor = interpretString(firstLine) ;
                                descriptors[newDescriptor.first] = newDescriptor.second ;
                        }
                        if(secondLine.left(7) == "Messung")
441                     {
                                while(secondLine.size())
                                {
                                        QPair<QString, QString> newDescriptor = interpretString(►
                                             secondLine) ;
                                        descriptors[newDescriptor.first] = newDescriptor.second ;
446                             }
```

```
                        }
                        else
                        {
                                firstLine = descriptors["Tag"].content() + "␣" + descriptors["►
                                    Uhrzeit"].content() + "␣:␣" + secondLine ;
451                             descriptors["Tag"] = descriptors["A-Tag"] ;
                                descriptors["Uhrzeit"] = descriptors["A-Zeit"] ;
                        }
                        logData += new specLogEntryItem(descriptors) ;
                }
456     }
        return logData ;
}
specFileImportFunction fileFilter(const QString& fileName)
{
461     QFile file(fileName) ;
        if(!file.open(QFile::ReadOnly | QFile::Text)) return 0 ;
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO␣8859-1")) ;
        QString sample = in.readLine(5) ;
466     QList<specModelItem*> (*pointer)(QFile&) = 0 ;
        if(sample == "PE␣IR") pointer = readPEFile ;
        else if(sample.left(4) == "PEPE") pointer = readPEBinary ;
        else if(sample == "Solve") pointer = readHVFile ;
        else if(sample == "Time␣") pointer = readSKHIFile ;
471     else if(sample.left(2) == "##") pointer = readJCAMPFile ;
        else
        {
                for(int i = 0 ; i < 10 ; ++i)
                        sample += in.readLine() ;
476             if(sample.contains(QRegExp("^\\d\\d\\.\\d\\d\\.\\d\\d␣\\d\\d:\\d\\d:\\d\\d␣:␣"))) ►
                    pointer = readLogFile ;
                else pointer = readXYFILE ;
        }
        file.close() ;
        return pointer ;
481 }
QList<specModelItem*> readPEBinary(QFile& file)
{
        file.close();
        if(!file.open(QFile::ReadOnly)) return QList<specModelItem*>() ;
486     specPEFile peData(file.readAll()) ;
        specDataItem dataItem(peData) ;
        dataItem.changeDescriptor("Datei", QFileInfo(file.fileName()).fileName()) ;
        dataItem.setDescriptorProperties("Datei", spec::def);
        return QList<specModelItem*>() << new specDataItem(dataItem) ;
491 }
QList<specModelItem*> readPEFile(QFile& file)
{
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO␣8859-1")) ;
496     QHash<QString, specDescriptor> headerItems ;
        headerItems["Datei"] = specDescriptor(QFileInfo(file.fileName()).fileName(), spec::def) ;
        while(in.readLine() != "#DATA") ;
        QList<specModelItem*> specData ;
        QStringList buffer ;
501     QVector<specDataPoint> dataPoints ;
        while(!in.atEnd() && (buffer = in.readLine().split(QRegExp("\\s+"))).size() > 1)
                dataPoints += specDataPoint(buffer[0].toDouble(), buffer[1].toDouble(), 0) ;
        specData += new specDataItem(dataPoints, headerItems) ;
        specData.last()->invalidate();
506     return specData ;
}
QVector<double> gaussjinv(QVector<QVector<double> >& A, QVector<double>& b)
{
        QVector<bool> ipiv(A.size(), false) ;
511     QVector<QVector<QVector<double> >::size_type> indxr ;
        QVector<QVector<double>::size_type> indxc ;
        for(QVector<QVector<double> >::size_type i = 0 ; i < A.size() ; i++)
        {
                QVector<QVector<double> >::size_type irow ;
516             QVector<double>::size_type icol ;
                double max = 0 ;
                for(QVector<QVector<double> >::size_type j = 0 ; j < A.size() ; j++)
                {
```

```
                        if(!ipiv[j])
521                     {
                                for(QVector<double>::size_type k = 0 ; k < A[j].size() ; k++)
                                {
                                        if(!ipiv[k] && (max = std::max(fabs(A[j][k]), max)) == fabs(►
                                             A[j][k]))
                                        {
526                                             irow = j ; icol = k ;
                                        }
                                }
                        }
                }
531             ipiv[icol] = true ;
                if(irow != icol)
                        A[irow].swap(A[icol]) ;
                indxr << irow ;
                indxc << icol ;
536             double pivinv = A[icol][icol] ;
                A[icol][icol] = 1 ;
                for(QVector<double>::size_type j = 0 ; j < A[icol].size() ; j ++)
                        A[icol][j] /= pivinv ;
                for(QVector<QVector<double> >::size_type j = 0 ; j < A.size() ; j++)
541             {
                        if(j != icol)
                        {
                                double dummy = A[j][icol] ;
                                A[j][icol] = 0 ;
546                             for(QVector<double>::size_type k = 0 ; k < A[j].size() ; k++)
                                        A[j][k] -= A[icol][k] * dummy ;
                        }
                }
        }
551     for(QVector<QVector<double>::size_type>::size_type i = indxc.size() - 1 ; i >= 0 ; i--)
                if(indxr[i] != indxc[i])
                        for(QVector<QVector<double> >::size_type j = 0 ; j < A.size() ; j++)
                                qSwap(A[j][indxr[i]], A[j][indxc[i]]) ;
        QVector<double> retval ;
556     for(QVector<QVector<double> >::size_type i = 0 ; i < A.size() ; i++)
        {
                double dummy = 0 ;
                for(QVector<double>::size_type j = 0 ; j < A[i].size() ; j++)
                        dummy += A[i][j] * b[j] ;
561             retval << dummy ;
        }
        return retval ;
}
QList<specModelItem*> readSKHIFile(QFile& file)
566 {
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO␣8859-1")) ;
        QVector < QPair < QPair<double, double>,
                QPair<QVector<double>, QVector<double> > > > integrale ;
571     QStringList integralNamen = in.readLine().split("\t") ;
        QVector<double> zeiten ;
        integralNamen.takeFirst() ;
        integralNamen.takeFirst() ;
        foreach(QString integralName, integralNamen)
576     integrale << qMakePair(qMakePair(integralName.section("␣-␣", 0, 0).toDouble(),
                                         integralName.section("␣-␣", 1, 1).toDouble()),
                               qMakePair(QVector<double>(), QVector<double>()));
        QHash<QString, specDescriptor> headerItems ;
        foreach(QString descriptor, in.readLine().split(",␣"))
581     headerItems[descriptor.section(":␣", 0, 0)] = specDescriptor(descriptor.section(":␣", 1, 1),►
              spec::numeric | spec::editable) ;
        headerItems["raw"] = 1 ;
        headerItems["file"] = QFileInfo(file.fileName()).fileName() ;
        while(!in.atEnd())
        {
586             QStringList firstLine = in.readLine().split("\t") ;
                QStringList secondLine = in.readLine().split("\t") ;
                if(firstLine.size() < 2 || secondLine.size() < 2) break ;
                zeiten << firstLine.takeFirst().toDouble() ;
                secondLine.takeFirst() ;
591             firstLine.takeFirst() ;
                secondLine.takeFirst() ;
```

```
                for(int i = 0 ; i < integrale.size() ; ++i)
                {
                        double a = firstLine.isEmpty() ? NAN : firstLine.takeFirst().toDouble() ;
                        double b = secondLine.isEmpty() ? NAN : secondLine.takeFirst().toDouble() ;
                        integrale[i].second.first << a ;
                        integrale[i].second.second << b ;
                }
        }
        QList<specModelItem*> newItems ;
        for(int i = 0 ; i < integrale.size() ; ++i)
        {
                headerItems["begin"] = integrale[i].first.first ;
                headerItems["end"] = integrale[i].first.second ;
                QVector<specDataPoint> rawOne, rawTwo, diff ;
                for(int j = 0 ; j < integrale[i].second.first.size() ; ++j)
                {
                        double a = integrale[i].second.first[j],
                               b = integrale[i].second.second[j],
                               t = zeiten[j] ;
                        rawOne << specDataPoint(t, a, NAN) ;
                        rawTwo << specDataPoint(t, b, NAN) ;
                        diff << specDataPoint(t, a - b, NAN) ;
                }
                headerItems["raw"] = 1 ;
                newItems << new specDataItem(rawOne, headerItems)
                         << new specDataItem(rawTwo, headerItems) ;
                headerItems["raw"] = 0 ;
                newItems << new specDataItem(diff, headerItems) ;
        }
        return newItems ;
}
QList<specModelItem*> readXYFILE(QFile& file)
{
        QTextStream in(&file) ;
        in.setCodec(QTextCodec::codecForName("ISO␣8859-1")) ;
        QList<specModelItem*> newItems ;
        QChar separator ;
        QString content = in.readAll() ;
        if(content.contains('\t')) separator = '\t' ;
        else if(content.contains('␣')) separator = '␣' ;
        else if(content.contains(',')) separator = ',' ;
        else
        {
                QMessageBox::critical(0, QObject::tr("XY␣import␣failed"), QObject::tr("Could␣not␣►
                    find␣valid␣data␣field␣separators␣(tab,␣space,␣comma)")) ;
                return newItems ;
        }
        QList<QStringList> entries ;
        foreach(QString line, content.split("\n"))
        entries << line.split(separator, QString::SkipEmptyParts) ;
        QDoubleValidator validator ;
        int pos = 0 ;
        QStringList headers ;
        if(!entries.isEmpty() && !entries.first().isEmpty() && validator.validate(entries.first().►
            first(), pos) != QValidator::Acceptable)
                headers = entries.takeFirst() ;
        if(entries.isEmpty() || entries.first().isEmpty()) return newItems ;
        int numberOfColums = entries.first().size() ;
        int i = 0 ;
        foreach(QStringList line, entries)
        {
                ++i ;
                if(line.size() != numberOfColums && line.size() > 1)
                {
                        QMessageBox::critical(0, QObject::tr("XY␣import␣failed"),
                                              QObject::tr("The␣number␣of␣columns␣differs␣in␣at␣least►
                                                  ␣one␣line"
                                                      "␣differs␣from␣that␣determined␣for␣the␣►
                                                          first␣line.␣"
                                                      "(Error␣occured␣in␣line␣%1.␣Pre-determined►
                                                          ␣number␣of"
                                                      "␣columns:␣%2.␣The␣line␣in␣question␣has␣%3►
                                                          ␣columns.)").
                                              arg(i).arg(numberOfColums).arg(line.size())) ;
                        return newItems ;
```

```
                }
        }
        QVector<double> xValues ;
        QVector<QVector<double> > yValues(numberOfColums - 1) ;
        foreach(QStringList line, entries)
        {
                if(!(line.size() > 1)) continue ;
                xValues << line.first().toDouble() ;
                for(int i = 0 ; i < numberOfColums - 1 ; ++i)
                        yValues[i] << line[i + 1].toDouble() ;
        }
        QHash<QString, specDescriptor> description ;
        description["File"] = QFileInfo(file.fileName()).fileName() ;
        for(int i = 0 ; i < numberOfColums - 1 ; ++i)
        {
                if(headers.size() > i)
                        description["Column"] = headers[i + (headers.size() == numberOfColums ? 1 : ►
                            0)] ;
                else if(description.contains("Column"))
                        description.remove("Column") ;
                QVector<specDataPoint> points ;
                for(int j = 0 ; j < xValues.size() ; ++j)
                        points << specDataPoint(xValues[j], yValues[i][j], NAN) ;
                newItems << new specDataItem(points, description) ;
        }
        return newItems ;
}
QwtSymbol* cloneSymbol(const QwtSymbol* original)
{
        if(!original) return 0 ;
        return new QwtSymbol(original->style(),
                             original->brush(),
                             original->pen(),
                             original->size()) ;
}
```

D.2 Computation of anharmonicity constants

The following program was used compute anharmonic constants in multiple runs by alternately calling a control file (`sample.py`) and the Psi4[149] quantum chemistry program. An optimized geometry with filename `geometry.xyz` would have to be provided in the directory specified by `__builtin__.prefix`. In the first pass, the program sets up a number of Psi4 jobs equal to the number of Cartesian coordinates in order to compute the Hessian matrix. After having run Psi4 on all of these jobs, the second pass generates mode and frequency information in a sub-directory `results` of `__builtin__.prefix` and more Psi4 jobs to compute the required third and fourth derivatives to the energy. The last pass, after having run the aforementioned Psi4 jobs, generates information on anharmonic constants in the `results` sub-directory.

`sample.py`

```
import __builtin__

__builtin__.atomAssignment = {"Azulen": [0,1,2,3,4,5,6,7,8,9, 19,20,21,22,23,24,25],
                  "Methylen": [10,26,27],
                  "Amid": [11,17,18,35],
                  "Kette␣vorne": [12,13,28,29,30,31],
                  "Kette␣hinten": [16] + [14,15,32,33,34,36],
```

```
                  "Azid": [37,38,39] }

__builtin__.prefix = "mp2/112-straight/"
__builtin__.modes = [ 89, 92, 94, 95 ]
__builtin__.method = "mp2"
__builtin__.basis = "sto-3g"

import universalpy
```

universal.py

```
# -*- coding: utf-8 -*-
"""
Created on Thu Jul 11 17:09:44 2013

@author: hendrik
"""

import geometry4zdetail
import sys
#from compiler.ast import flatten as flatten

m = geometry4zdetail.molecule() #(prefix + "geometry.xyz")
try: m.method = method
except NameError: m.method = "mp2"

try: m.basis = basis
except NameError: m.basis = "cc-pVDZ"

try: m.delta = delta
except NameError: m.delta = 5e-3

try: m.resonanceLimit = resonanceLimit
except NameError: m.resonanceLimit = 10.

try: m.nmdelta = nmdelta
except NameError: m.nmdelta = 0.1 # Achtung:  sollte 0.01 => Schneider, Thiel CPL 1989, 157, p. 367

try: threshold
except NameError: threshold = .5

zeroFilename = "zero"
hessianFilename = "hess"
ffFilename = "fofi"

print "Analyzing␣␣", prefix
print "␣␣Method:␣␣", m.method
print "␣␣Basis:␣␣␣", m.basis
print "␣␣Delta:␣␣␣", m.delta
print "␣␣NmDelta:␣", m.nmdelta
print "␣␣Res.Lim.:", m.resonanceLimit
print "␣␣Modes:␣␣␣", modes

m.readGeometry(prefix + "geometry.xyz")
m.guessFile = prefix + "guess.dat"

m.sowZero(prefix + zeroFilename)
m.reapGradient(prefix + zeroFilename)
m.reapEnergy(prefix + zeroFilename)
m.reapMasses(prefix + zeroFilename)

m.sowHessian(prefix + hessianFilename)
m.reapHessian(prefix + hessianFilename)
m.findNormalCoords()
m.findCoriolisTerms()
for mm in modes:
        m.sowNormalDisplacements(prefix + ffFilename, m.generateListOfDisplacements(mm))
m.reapNormalDisplacements(prefix + ffFilename)


################## Prefix change!
prefix += "results/"
print "Results␣will␣be␣in␣", prefix
```

```
frequencyFile = open(prefix+"frequencies", 'w')
#print "-------- Normal frequencies --------"
for i in range(len(m.normalFrequencies)):
        print >> frequencyFile, i, m.normalFrequencies[i]/geometry4zdetail.conv
frequencyFile.close()

modeFile = open(prefix+"modes", 'w')
for i in range(len(m.normalFrequencies)):
        displ = m.displayContributions(i)
        print >> modeFile, ""
        print >> modeFile, "#------ Mode ",i," ",m.normalFrequencies[i]/geometry4zdetail.conv
        print >> modeFile, "\n".join([ " ".join(["%.5f" % f for f in d]) for d in displ])
modeFile.close()


for mm in modes:
        m.computePotentialConstants(mm)
for mm in modes:
        m.computeAnharmonicConstants(mm)

def printoutConstants(title, collection, outputFileName):
        outputFile = open(prefix + outputFileName, 'w')
        #print >>outputFile, "#============ " + title
        keys = collection.keys()
        keys.sort()
        for k in keys:
                print >> outputFile, " ".join(["%d" % m for m in k]) + " ", collection[k]
        outputFile.close()

printoutConstants("Potential Constants", m.potentialConstants, "potentialConstants")
printoutConstants("Anharmonic Constants", m.anharmonicConstants, "anharmonicConstants")

for mode in modes:
        paired = [(m.normalFrequencies[other]/geometry4zdetail.conv, anharmonicConstant) for other, ►
             anharmonicConstant in
           [(other, m.anharmonicConstants[(mm,other)]) for mm, other in m.anharmonicConstants.keys()►
                if mm == mode]]
        paired.sort()
        nu = m.normalFrequencies[mode]/geometry4zdetail.conv
        diagonal = [p for p in paired if p[0] == nu]
        paired = [p for p in paired if p[0] != nu] + diagonal
        modeAnharmonicities = open(prefix + "anharmonicConstants-" + str(mode), 'w')
        print >> modeAnharmonicities, "\n".join([" ".join([str(i) for i in p]) for p in paired])
        modeAnharmonicities.close()

####### print pgfplots mit Zuordnung
assignAtom = {v:k for k in atomAssignment for v in atomAssignment[k]}

modeAssignment = {'Delocalized': []}
for k in atomAssignment:
        modeAssignment[k] = []
for l in range(len(m.normalDisplacements)):
        cd = m.normalDisplacements[l]
        contribution = {}
        for i in range(len(cd)):
                label = assignAtom[int(i/3)]
                if label in contribution:
                        contribution[label] += cd[i]**2
                else:
                        contribution[label] = cd[i]**2
        contribution = [(contribution[k], k) for k in contribution]
        contribution.sort()
        if (contribution[-1][0] > threshold) :
                #print l, contribution[-1]
                modeAssignment[contribution[-1][1]] += [l]
        else:
                modeAssignment['Delocalized'] += [l]

quantAssign = [(len(modeAssignment[name]), name) for name in modeAssignment]
quantAssign.sort()
quantAssign.reverse()
for mm in modes:
        anharmonicGraph = open(prefix + "anharmonics"+str(mm)+".tex", 'w')
        print >> anharmonicGraph, "\\pgfkeys{/anharmonicMode=",mm+1,"}"
```

```
        for _, k in quantAssign:
                modeAssignment[k].sort()
                print >> anharmonicGraph, "\\addplot table[x index = 0, y index = 1] {"
                for mo in modeAssignment[k]:
                        print >> anharmonicGraph, m.normalFrequencies[mo]/geometry4zdetail.conv, m.►
                            anharmonicConstants[(mm,mo)]
                print >> anharmonicGraph, "};\n\\addlegendentry{" + k + "}\n"
        anharmonicGraph.close()

# Participation ratios:
participation = open(prefix + "participationRatios.tex", 'w')
for _, k in quantAssign:
        modeAssignment[k].sort()
        print >> participation, "\\addplot table[x index = 0, y index = 1] {"
        for mo in modeAssignment[k]:
                print >> participation, m.normalFrequencies[mo]/geometry4zdetail.conv, 1./len(m.►
                    atomicSymbols)/sum([i**4 for i in m.normalDisplacements[mo]])
        print >> participation, "};\n\\addlegendentry{" + k + "}\n"
participation.close()

lmatrix = open(prefix + "Lmatrix", 'w')
for nm in m.normalDisplacements:
        print >> lmatrix, " ".join(["%.5f" % s for s in nm])
lmatrix.close()
```

geometry4zdetail.py

```
# -*- coding: utf-8 -*-
import re
import numpy
import scipy.constants
import glob
import operator
import tarfile
import fnmatch

from numpy import sqrt as sqrt
from numpy import exp as exp
from compiler.ast import flatten as flatten
from numpy import pi as pi
from scipy.constants import c as c0
from scipy.constants import m_e as me
from scipy.constants import m_u as amu
from scipy.constants import h as h
Eh = scipy.constants.value("Hartree energy")
a0 = scipy.constants.value("Bohr radius")
Ehcm = Eh / h / c0 / 100
hckB = h*c0*100/scipy.constants.k


numpy.set_printoptions(precision=2,linewidth=100)
conv = sqrt(a0**2 * amu * 4*pi**2 *c0*100)

class molecule:
    delta = .003
    numericCutoff = 1e-2
    nmdelta = 0.1

    def cleanup(self):
        self.atomicSymbols = []
        self.multiplicity=1
        self.charge=0
        self.coordinates = []
        self.masses = []
        self.hessian = []
        self.gradient = []
        self.coordStep = []
        self.energy = 0
        self.normalFrequencies = []
        self.normalDisplacements = []
        self.coordDisplacements = []
        self.displacedGradients = {}
        self.coriolis = {}
        self.potentialConstants = {}
```

```
        self.anharmonicConstants = {}

    def __init__(self,filename=""):
        self.method = "ccsd(t)"
        self.basis = "cc-pVDZ"
        self.guessFile = ""
        self.archive = ""
        self.cleanup()
        self.resonanceLimit = 1.
        if filename:
            self.readGeometry(filename)

    def haveOpenArchive(self):
        return (self.archive and not self.archive.closed)

    def setArchive(self, archiveName = ""):
        if self.haveOpenArchive():
            self.archive.close()
        if archiveName:
            self.archive = tarfile.open(archiveName)

    def filesInArchive(self):
        return [f.name for f in self.archive.getmembers()] if self.haveOpenArchive() else []

    def readFile(self, filename):
        #print "Trying to read file " + filename + " ", (filename in self.filesInArchive())
        return self.archive.extractfile(filename) if filename in self.filesInArchive() else open(filename)

    def readEnergy(self, prefix):
        return float(self.readFile(prefix + "-energy").read())

    def readGradient(self, prefix):
        return [float(n) for n in self.readFile(prefix + "-gradient")]

    def readMasses(self, prefix):
        return [float(n) for n in self.readFile(prefix + "-masses")]

    def reapGradient(self, prefix):
        self.gradient = self.readGradient(prefix)

    def reapEnergy(self, prefix):
        self.energy = self.readEnergy(prefix)

    def reapMasses(self, prefix):
        self.masses = self.readMasses(prefix)

    def size(self):
        return min(len(self.coordinates),len(self.atomicSymbols))

    def sizer(self):
        return range(self.size())

    def writeInstructionFile(self, fileName, displacements = []):
        mfile = open(fileName + ".in", 'w')
        mfile.write(
"""molecule molec {
""" + self.geometryString() + """
    symmetry c1
}
molec.fix_com(True)
molec.fix_orientation(True)
molec.update_geometry()
set basis """ + self.basis +"""
set molden_write false

geom = molec.geometry()
""" +
    "\n".join(["geom.set(" + str(i) + ", " + str(j) +
                  ", geom.get(" + str(i) + ", " + str(j) + ") + " + ("%.20e" % d) + ")" for i, j, d in displacements ]) +
"""
molec.set_geometry(geom)
import os.path
guessFile = '""" + self.guessFile + """'
```

```
    readGuess = os.path.exists(guessFile)
122 if guessFile and readGuess:
        copy_file_to_scratch('""" + self.guessFile + """', 'psi', 'molec', PSIF_SCF_MOS)
        set guess read
    en = gradient(\"""␣+␣self.method␣+␣"""\")
    if␣guessFile␣and␣not␣readGuess:
127 ␣␣␣␣copy_file_from_scratch('""" + self.guessFile + """', 'psi', 'molec', PSIF_SCF_MOS)

    ef = open('""" + fileName + """-energy','w')
    ef.write("%.20e" % en)
    ef.close()
132
    gf = open('""" + fileName + """-gradient','w')
    grad = PsiMod.get_gradient()
    gf.write("\\n".join(["%.20e" % grad.get(i,j) for i in range(molec.natom()) for j in range(3)]))

137 mf = open('""" +fileName + """-masses', 'w')
    mf.write("\\n".join([\"%.20e\" % molec.mass(i) for i in range(molec.natom())]))
    mf.close()
    """)
        mfile.close()
142
    def readGeometry(self, filename):
        self.cleanup()
        geometryFile = self.readFile(filename)
        gs = geometryFile.read()
147     geometryFile.close()
        lines = re.split("\n",gs)
        if len(lines) < 1: raise Exception("File␣too␣short")
        for l in lines:
            tokens = [t for t in re.split("\\s+",l) if t]
152         if len(tokens) == 2:
                self.charge = int(tokens[0])
                self.multiplicity = int(tokens[1])
            if len(tokens) != 4: continue
            self.atomicSymbols += [tokens[0]]
157         self.coordinates += [[float(x) for x in tokens[1:]]]

    def geometryString(self):
        result = str(self.charge)+"␣"+str(self.multiplicity)+"\n"
        for i in self.sizer():
162         result += "␣" + "␣␣␣".join([self.atomicSymbols[i]] + ["%.20f" % x for x in self.►
                coordinates[i]]) + "\n"
        return result

    def sowZero(self, fileName):
        self.writeInstructionFile(fileName)
167
    def sowHessian(self, prefix):
        for i in self.sizer():
            for j in range(3):
                for s in "+", "-":
172                 self.writeInstructionFile(prefix + s + str(3*i+j),
                                              [(i,j, int(s+"1")*self.delta)])


    def reapHessian(self,prefix, includeGradient = False):
177     self.hessian = [ [(p-m)/2./self.delta
                          for p,m in zip(self.readGradient(prefix + "+" + str(3*i+j)),
                                         self.readGradient(prefix + "-" + str(3*i+j)))]
                          for i in self.sizer()
                          for j in range(3)]
182     # symmetrize:
        for i in range(len(self.hessian)):
            for j in range(i+1,len(self.hessian)):
                ave = (self.hessian[i][j]+self.hessian[j][i])/2.
                self.hessian[i][j] = ave
187             self.hessian[j][i] = ave

    def applyCoordStep(self):
        for i in range(min(len(3*self.coordinates), len(self.coordStep))):
            self.coordinates[int(i/3)][i%3] -= self.coordStep[i]
192     self.coordStep = []

    def applyGradientStep(self, scale):
```

```
        norm = numpy.sqrt(sum([g**2 for g in self.gradient]))
        for i in range(min(len(3*self.coordinates), len(self.gradient))):
            self.coordinates[int(i/3)][i%3] -= self.gradient[i]/norm*scale
        self.coordStep = []

    def findNormalCoords(self):
        muvector = numpy.array([1./sqrt(i) for i in flatten(zip(self.masses, self.masses, self.masses))])
        mu = numpy.diag(muvector)
        mat = mu * numpy.matrix(self.hessian) * mu / c0/100 * Eh
        eigenValues, eigenVectors = numpy.linalg.linalg.eig(numpy.array(mat))
        eigenVectors = zip(*eigenVectors)
        normalModes = zip(eigenValues, eigenVectors)
        normalModes.sort()
        del normalModes[:6]
        mapEvEw = [list(eigenValues).index(i) for i,_ in normalModes]
        normalModes = [(sqrt(f),m) for f,m in normalModes]
        self.normalFrequencies = [nu for nu,_ in normalModes]
        # Displacements in bohr (a0))
        self.coordDisplacements = [muvector*nm[1]/sqrt(conv*nm[0]/h) for nm in normalModes] # Vgl. spectro, Gl. (18).  Cave:  nm[0] = omega*conv, d.h. hier steht eigentlich omega*conv**2
        # was:
        #self.normalDisplacements = [[eigenVectors[mode][i] for i in range(len(eigenVectors))] for mode in mapEvEw]
        self.normalDisplacements = [list(m) for _, m in normalModes]
        #for m in normalModes:
                #print m

    def findCoriolisTerms(self):
        itm = numpy.matrix(3*[3*[0.0]]) #inertia tensor
        i = 0
        for coord in self.coordinates:
            m = self.masses[i]
            for j in range(3):
                for k in range(3):
                    itm[j,k] += m*((j==k)*sum(map(operator.mul,coord,coord))-coord[j]*coord[k])
            i += 1 #TODO: eigentlich werden nur die Diagonalelemente benoetigt (vorausgesetzt, der Traegheitstensor ist diagonal)
            # Achtung: Molekuel muss an den Hauptachsen ausgerichtet sein!
#         print "Traegheitstensor:",itm # TODO: Einheiten pruefen.  Vermute amu*A**2 -- Uebereinstimmung mit Lit.Int. J IR mm Waves 4, 505 (1983)
        B = [ h/8./pi**2/c0/100/itm[i,i]/1e-20/amu for i in range(3)]
        #print "B:",B, itm
        #BA = [h/8./pi**2/c0/100/1e-20/amu/(1e-20+sum([ self.masses[a] * self.coordinates[a][x]**2 for x in range(3) if x != i for a in range(len(self.coordinates))])) for i in range(3)]
        #print B, BA
        for k in range(len(self.normalDisplacements)):
            for l in range(k, len(self.normalDisplacements)):
                zetas = []
                for i in range(3):
                    c1 = (i-1)%3
                    c2 = (c1-1)%3
                    #print "Coriolisconsts:", i, c1, c2
                    zetas += [sum(map(operator.sub,
                          map(operator.mul, self.normalDisplacements[k][c1::3], self.normalDisplacements[l][c2::3]), \
                          map(operator.mul, self.normalDisplacements[k][c2::3], self.normalDisplacements[l][c1::3])))**2 ]
                #print "Coriolis:", k, l, zetas
                # geprueft nach Califano, p. 97, Gl. 4.8.6
                self.coriolis[(l,k)] = sum(map(operator.mul, zetas, B))*(self.normalFrequencies[k]/self.normalFrequencies[l]+self.normalFrequencies[l]/self.normalFrequencies[k])
                self.coriolis[(k,l)] = self.coriolis[(l,k)]

    def sowNormalDisplacements(self, prefix, dispList):
        for disp in dispList:
            dlstring = "".join(["%d" % i for i in disp])
            self.writeInstructionFile(prefix + dlstring,
                                      [(int(d/3), d%3, self.coordDisplacements[i][d] * self.nmdelta * disp[i])
                                      for i in range(len(disp))
                                      for d in range(3*self.size())
                                      if disp[i]])

    def getFileNames(self, pattern):
```

```
        return fnmatch.filter(self.filesInArchive(), pattern) + glob.glob(pattern)

    def reapNormalDisplacements(self, prefix):
        def getDisplacementFromString(string):
            displ = []
            i = 0
            while i < len(string):
                if string[i] != '-':
                    displ += [int(string[i])]
                else:
                    i += 1
                    displ += [-int(string[i])]
                i += 1
            return tuple(displ)
        gradientDisplacements = [(f,getDisplacementFromString(f[len(prefix):-9])) for f in self.►
             getFileNames(prefix + "*-gradient")]
        for fileName,displacement in gradientDisplacements:
            dispFile = self.readFile(fileName)
            self.displacedGradients[displacement] = numpy.array(self.readGradient(fileName[:-9]))

    def generateListOfDisplacements(self, modeNo):
        n = len(self.normalFrequencies)
        return [ [(i==j)*currentDisp + (modeNo==j)*modeDisp for j in range(n)]
            for currentDisp in -1,1
            for modeDisp in -1,0,1
            for i in range(n)]

    def translate(self, toTranslate): ## [(0,+1), (1,-1)] usw.
            result = [0]*len(self.normalFrequencies)
            for t,d in toTranslate:
                result[t] += d
            return tuple(result)

    def computePotentialConstants(self, mode):
        k = mode
        for l in range(len(self.normalFrequencies)):
            # phi_{kkll}
            derlist = [(+2, -1, 0),   ### Weight, disp1, disp2
                       (-2, +1, 0),
                       (+1, +1, +1),
                       (+1, +1, -1),
                       (-1, -1, +1),
                       (-1, -1, -1)]
            grad = sum([w * self.displacedGradients[self.translate([(k,a), (l,b)])]
                           for w, a, b in derlist])
            g2   = sum([w * self.displacedGradients[self.translate([(l,a), (k,b)])] ## symmetrization
                           for w, a, b in derlist])
            self.potentialConstants[k,k,l,l] = (sum(grad*self.coordDisplacements[k]) +
                    sum(g2*self.coordDisplacements[l])) / 4 / self.nmdelta**3 * Ehcm
            # phi_{klm}
            g3 = sum([a*b*self.displacedGradients[self.translate([(k,a), (l,b)])]
                for a in +1, -1
                for b in +1, -1]) / 4 / self.nmdelta**2 * Ehcm
            for m in range(len(self.normalFrequencies)):
                self.potentialConstants[k,l,m] = sum(g3*self.coordDisplacements[m])
            # phi_{llm}
            g4 = (sum([self.displacedGradients[self.translate([(l,a)])]
                for a in +1, -1]
                ) - self.gradient - self.gradient)/ self.nmdelta**2 * Ehcm
            for m in range(len(self.normalFrequencies)):
                self.potentialConstants[l,l,m] = sum(g4*self.coordDisplacements[m])

    def computeAnharmonicConstants(self, mode):
        def conditionalAddition(phi, delta, faktor):
            return phi/delta if abs(phi**2/delta**3/faktor) < self.resonanceLimit else 0
        k = mode
        xkk = self.potentialConstants[k,k,k,k]
        ok = self.normalFrequencies[k] /conv
        for l in range(len(self.normalFrequencies)):
            ol = self.normalFrequencies[l] /conv
            phih = self.potentialConstants[k,k,l]**2/2.
            xkk -= phih * (1./(2.*ok+ol)+4./ol) - conditionalAddition(phih, (2*ok-ol), 64) # phi^4/►
                 delta^3/256 (phi^2 ist schon halbiert)
            if l == k : continue
            xkl = self.potentialConstants[k,k,l,l]
```

```
            for m in range(len(self.normalFrequencies)):
                om = self.normalFrequencies[m] /conv
                phi = self.potentialConstants[k,l,m]**2/2.
                imkterm = phi/(ok+ol+om) +\
                        conditionalAddition(phi, (-ok+ol+om), 16) +\
                        conditionalAddition(phi, (ok-ol+om), 16) -\
                        conditionalAddition(phi, (ok+ol-om), 16)
                xkl -= self.potentialConstants[k,k,m]*self.potentialConstants[l,l,m] / om \
                       + imkterm
            xkl /= 4.
            xkl += self.coriolis[k,l]
            self.anharmonicConstants[k,l] = \
            self.anharmonicConstants[l,k] = xkl
        self.anharmonicConstants[k,k] = xkk / 16.

    def printContributions(self, mode):
        displ = [(numpy.linalg.norm(self.normalDisplacements[mode][3*i:3*(i+1)]),
                self.atomicSymbols[i], i,
                self.normalDisplacements[mode][3*i:3*(i+1)])
                for i in self.sizer()]
        displ.sort(reverse = True)
        return displ

    def displayContributions(self, mode):
        displ = [self.coordinates[i] +
                self.normalDisplacements[mode][3*i:3*(i+1)]
                for i in self.sizer()]
        return displ
```

D.3 Halide–noble gas cluster energies

The cluster computation program could be called by itself and take input from the standard input or with the input filename as its only parameter. The basic input specifications are:

- A semicolon or a line break ends an assignment or command.
- An equality sign (=) assigns the value to its right to the variable name left of the equality sign explicitly.
- A colon (`:`) copies the *content* of the variable right of the colon to the variable whose name is left of the colon.
- Lists of values have to be delimited by curly braces (`{` and `}`). They may span multiple lines and may be nested.
- White space characters (tab, space) are generally discarded, but may be required to separate values in lists.
- Comments are introduced by a number sign (`#`) and extend to the end of a line.
- Certain variable names are used for program settings (see `variablehash.cpp`).

- Command names cannot be used as variable names. Anything following a command name up to the next line break or semicolon is ignored. Therefore, commands do not take arguments, but rather use reserved variables for their parameters.

When the program requires a variable for computation, it will check if the variable's value corresponds to the name of another variable. If that is the case, it will check that variable instead until a variable is found, whose value is not the name of another variable. This allows for elegant structuring of the input by referring to other variables instead of giving values explicitly. For instance, a potential energy contribution is generally a list of three elements: the name of the type of energy contribution, a list of the symbols of the atoms involved, and a list of parameters specific to that type of contribution (e.g. Lennard-Jones parameters). The following code would specify a harmonic potential (`hb` stands for "harmonic bond") $V_{\mathrm{hb}} = k/2 \times (r - r_e)^2$ between two atoms of type `H` with an equilibrium distance of $r_e = 1.4$ Å and a force constant of $k = 36000$ meV Å^{-2} (the basic units are – with some exceptions for conveniently entering literature potentials – meV for energies and Å for distances):

```
pot = {hb {H H} {1.4 36000}}
```

The program operates on the system stored in the variable name `cluster`, which is supposed to be a list with two elements, of which the first is a list of potential energy contributions and the second is a list of atoms. An atom, in turn, is a list containing – in this order – at least an atomic symbol and three Cartesian coordinate values, and then optionally a mass, a charge, dipole and quadrupole polarizabilities, values of $C_6\, B/\alpha$ in $e\, a_0^8$, and an effective number of electrons N. Units for these values are the ones used in Table C.6. For example, with the previous definition of a harmonic potential, one could set up a diatomic molecule as follows:

```
cluster = {
        { pot }
        {
                { H 0  0   0  1.01 }
                { H 0  0  1.4 1.01 }
                { H 0 1.4  0  1.01 }
        }
}
```

Here, the program would automatically set up three contributions to the potential energy: one for each pair of `H` atoms.

Note that certain potentials do not require any symbols of participating atoms, as they are computed for the entire cluster (namely induction non-additivity, `nind`, and dispersion multipoles, `ind`). Conversely, some potentials do not require parameters, as

they obtain the required values from the atoms (e.g. the Coulomb potential `coul` and the two aforementioned potentials). The names of the various potentials can be found in `variablehash.cpp`.

Finally, command names are

- `energy` for the computation of the energy of the current system,
- `gradient` for a computation of the gradient of the current system,
- `optimize` for a gradient optimization of the geometry,
- `hessian` for computing vibrational frequencies and zero point energies,
- `sphereOptimize` for a parallelized optimization of multiple geometries where the last atom is initially placed on grid points of a spherical surface as described in Figure 4.1 on page 108,
- `boxOptimize` for a similar optimization with a cubic grid,
- `deleteAtom` for deleting the atom of index `atomIndex`, and
- `insertAtom` for inserting the atom specified by `insertingAtom` at position `atomIndex` (`0` denotes the first atom).

For illustrative purposes, `ArIsample.inp` – the input for the first iterations of a calculation of the Ar_nI cluster series – is given below.

`ArIsample.inp`

```
# Ar-Ar Potential: Aziz, Slamann, Mol. Phys. 58, 679-697
ArAr = { hfdb { Ar Ar} {2.26210716e5  10.77874743  1.10785136  0.56072459  0.34602794  64.3  1640.  ►
    51000.  -1.8122004  -0.128422  1.36  12.34208816 3.7565  3.3527} }

# Yourshaw, Zhao, Neumark, J. Chem. Phys., 105 (2), 1996, 351.
ArIAnion = { mmsv { Ar I } { 45.8  4.07  5.70  4.45  1.08   1.62  98400  715000  12800.  162000. } }
ArINeutral ={ pwam {Ar I}
  {942.671 state 1 1 1 -2 1 1
  18.8  3.95  7.15  6.18  1.01   1.62  98400  715000  12800.  162000.
  13.9  4.18  7.25  6.30  1.04   1.62  98400  715000  12800.  162000.
  16.0  4.11  6.90  6.40  1.04   1.64  98400  715000  12800.  162000. }}

# ALL dipole- and quadrupole polarizabilites from Haettig, Hess JPC 100, 6243 (Edelgase, TDMP2)
ArAtom = { Ar  0 0 3.98 39.948 0 11.15 26.39 808.1 5.90}

# Yourshaw, Zhao, Neumark, J. Chem. Phys., 105 (2), 1996, 351.
# Axilrod-Teller (alpha_d, C, N) Anion/Neutral (Yourshaw1996, N aus Lenzer1999xei):
# 52.7/36.1 254 7.79/6.5
# Parameter aus Lenzer1999xei:
IAnion  = { I 0 0 0 126.90447 -1 69.34 495 0 7.253}
IAtom   = { I 0 0 0 126.90447  0 33.1 0   0 6.5 }

# new value from Lenzer2001 JCP 115, 3578
ArExc = {exc {halide Ar Ar} {0.927 6.50}}
```

```
# Global parameters
recenter = 1
state = 2
gradientThreshold = 1e-12
energyThreshold = 1e-12
cluster = { potential geometry }
geometry = { anion }

atomToReplace = 0
replacementAtom = neutral
potential = anionicPotential
verbosity = 10
boxSize = 4
boxStep = 2
maxSteps = 100

anionicPotential = {
        ArAr
        ArIAnion
        {nind {} {} }
        {exc {halide Ar Ar} {0.936 6.50}}
        {ind  {} {} }
        {ax-t {I Ar Ar} {179e3} }
}

neutralPotential = {
        ArAr
        ArINeutral
        {ax-t {I Ar Ar} {129e3} }
}

ArAtom = { Ar  0 0 3.98 39.948 0 11.08 27.11 834.4 5.90}
IAnion  = { I 0 0 0 126.90447 -1 52.7 254 0 7.253}
IAtom   = { I 0 0 0 126.90447  0 36.1 0   0 6.5 }

anion   = IAnion
neutral = IAtom
insertingAtom = ArAtom
halide = I
#azimuthStep = 15
#altitudeStep = 15
atomIndex = 1000
insertAtom
potential = anionicPotential
sphereOptimize
optimize
energy
hessian
anionicCluster : cluster
oldAtom : insertingAtom
insertingAtom = neutral

atomIndex = 0
deleteAtom
insertAtom
potential = neutralPotential
state = 1
optimize
hessian

cluster : anionicCluster
deleteAtom
insertAtom
state = 2
optimize
hessian

cluster : anionicCluster
deleteAtom
insertAtom
state = 3
optimize
hessian

cluster : anionicCluster
```

```
insertingAtom = oldAtom
atomIndex = 1000
insertAtom
potential = anionicPotential
sphereOptimize
optimize
energy
hessian
anionicCluster : cluster
oldAtom : insertingAtom
insertingAtom = neutral

atomIndex = 0
deleteAtom
insertAtom
potential = neutralPotential
state = 1
optimize
hessian
```

src/src.pro

```
SOURCES += main.cpp \
        atom.cpp \
 coord.cpp \
 cluster.cpp \
 energyfunction.cpp \
 mmsv.cpp \
 hfdb.cpp \
 harmonic.cpp \
 pwam.cpp \
 coulomb.cpp \
 lennardjones.cpp \
 nonaddinduction.cpp \
 axilrod.cpp \
 exchange.cpp \
 induction.cpp \
 internalpair.cpp \
 pwam-exp.cpp \
 pw02.cpp \
    gsloptimizer.cpp \
    variablehash.cpp \
    computationmodule.cpp \
    commoninteraction.cpp \
    v2pot.cpp \
    esmsvw.cpp

HEADERS += \
atom.h \
 coord.h \
 cluster.h \
 energyfunction.h \
 mmsv.h \
 hfdb.h \
 harmonic.h \
 pwam.h \
 coulomb.h \
 lennardjones.h \
 nonaddinduction.h \
 axilrod.h \
 exchange.h \
 induction.h \
 internalpair.h \
 pwam-exp.h \
 pw02.h \
    gsloptimizer.h \
    constants.h \
    variablehash.h \
    computationmodule.h \
    commoninteraction.h \
    v2pot.h \
    esmsvw.h

LIBS += -lgsl -lgslcblas
```

```
TEMPLATE = app
CONFIG += warn_on \
          thread \
          qt \
          debug
TARGET = parser

QT += gui

OBJECTS_DIR = ../build/objects
MOC_DIR = ../build/moc
DESTDIR = ../build/target
DLLDESTDIR = ../build/target
UI_DIR = ../build/uic
UI_HEADERS_DIR = ../build/uic-headers
UI_SOURCES_DIR = ../build/uic-src
RCC_DIR = ../build/resources

DEFINES += CHOLESKYINNONADDITIVE

QMAKE_CXXFLAGS += -O0
```

src/atom.h

```
#ifndef ATOM_H
#define ATOM_H
#include <QString>
#include <QVector>
#include "coord.h"
#include <math.h>
class atom : public coord
{
        QString name ;
        double Mass, Charge, Ad, Aq, c6ba, Neff ;
public:
        atom(QString, coord, double mass, QVector<double>) ;
        ~atom();
        atom& operator= (const QVector<double>&) ;
        atom& operator= (const coord&) ;
        atom& operator-= (const coord&) ;
        QString type() const ;
        double mass() const ;
        double charge() const ;
        double dipolePolarizability() const ;
        double quadrupolePolarizability() const ;
        double dispersionHyperPolarizability() const ;
        double effectiveElectronNumber() const ;
        double alpha_d() const ;
        double alpha_q() const ;
        bool operator< (const atom&) const ;
};
inline double deviationBond(const QVector<atom*>& atoms, const double& equilibriumValue)
{ return *atoms[0] % *atoms[1] - equilibriumValue ;}
inline double deviationAngle(const QVector<atom*>& atoms, const double& equilibriumValue)
{
        double b = ((*atoms[0] - *atoms[1]) > (*atoms[2] - *atoms[1])) - equilibriumValue;
        b = fabs(b) <= M_PI ? b : copysign(2.*M_PI - fabs(b), -b) ;
        return b ;
}
inline double deviationDieder(const QVector<atom*>& atoms, const double& equilibriumValue)
{
        coord middle = *atoms[2] - *atoms[1] ;
        double b = (((*atoms[0] - *atoms[1]) / middle) * (*atoms[3] - *atoms[2]) <= 0 ? 1 : -1) * ►
             (((*atoms[0] - *atoms[1]) / middle) > ((*atoms[3] - *atoms[2]) / middle)) - ►
             equilibriumValue ;
        b = fabs(b) <= M_PI ? b : copysign(2.*M_PI - fabs(b), -b) ;
        return b ;
}
inline double deviationInversion(const QVector<atom*>& atoms, const double& equilibriumValue)
{
        coord r1, r2, r3, n, m, d = *atoms[3] ;
        r1 = *atoms[0] - d ;
        r2 = *atoms[1] - d ;
```

```
        r3 = *atoms[2] - d ;
        n = !(r2 / r3) ;
        m = (!r1 - n * (n * !r1)) ;
        double b = (r1 * n < 0. ? -1 : 1) * (r1 > (!r1 - n * (n * !r1)) * (m * (!r2 + !r3) > 0 ? -1 : 1)) - equilibriumValue;
        b = fabs(b) <= M_PI ? b : copysign(2.*M_PI - fabs(b), -b) ;
        return b ;
}
#endif
```

src/atom.cpp

```
#include "atom.h"
#include "constants.h"
atom::atom(QString type, coord pos, double mass, QVector<double> elstat)
        : coord(pos), name(type), Mass(mass)
{
        QVector<double*> electrostatics ;
        electrostatics << &Charge << &Ad << &Aq << &c6ba << &Neff ;
        for(int i = 0 ; i < electrostatics.size() && i < elstat.size() ; i++)
                *electrostatics[i] = (i < elstat.size() ? elstat[i] : 0.) ;
}
atom::~atom()
{
}
atom& atom::operator= (const QVector<double>& list)
{
        coord::operator= (list) ;
        return *this ;
}
atom& atom::operator= (const coord& coords)
{
        coord::operator= (coords) ;
        return *this ;
}
atom& atom::operator-= (const coord& coords)
{
        coord::operator-= (coords) ;
        return *this ;
}
bool atom::operator< (const atom& other) const
{
        return (type() < other.type()) ;
}
QString atom::type() const { return name ;}
double atom::mass() const {return Mass ;}
double atom::charge() const {return Charge ;}
double atom::dipolePolarizability() const { return Ad * parser::a0e3 ;}
double atom::quadrupolePolarizability() const { return Aq * parser::a0e5 ;}
double atom::dispersionHyperPolarizability() const { return c6ba * parser::a0e8 ;}
double atom::effectiveElectronNumber() const { return Neff ; }
double atom::alpha_d() const { return Ad ; }
double atom::alpha_q() const { return Aq ; }
```

src/axilrod.h

```
#ifndef AXILROD_H
#define AXILROD_H
#include "energyfunction.h"
class axilrodContribution : public energyContribution
{
        double C ;
public:
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
        void initialize(const QVector<atom*>& atoms) ;
        axilrodContribution() ;
};
class axilrod : public energyFunction
{
```

```
protected:
        energyContribution* newInteraction() ;
        void doSort(QStringList&) ;
public:
        int numAtoms() ;
        void printDescription() const ;
        axilrod* create() const { return new axilrod() ;}
};
#endif
```

src/axilrod.cpp

```
#include "axilrod.h"
#include <cmath>
#include "constants.h"
QVector<double> axilrodContribution::parameters() const
{ return (QVector<double>() << C) ;}
axilrodContribution::axilrodContribution()
        : C(NAN) {}
void axilrodContribution::setParameters(const QVector<double>& params)
{
        if(params.size() == 1)
                C = params[0] ;
}
void axilrodContribution::initialize(const QVector<atom*>& atoms)
{
        if(!isnan(C)) return ;
        if(atoms.size() != 3) return ;
        double eta[3] ;
        double alphas = 1 ;
        for(int i = 0 ; i < 3 ; ++i)
        {
                eta[i] = sqrt(atoms[i]->effectiveElectronNumber() / atoms[i]->alpha_d()) ;
                alphas *= atoms[i]->alpha_d() ;
        }
        C = 1.5 * alphas * eta[0] * eta[1] * eta[2] * (eta[0] + eta[1] + eta[2])
            / (eta[0] + eta[1]) / (eta[0] + eta[2]) / (eta[1] + eta[2])
            * parser::a0e9 * parser::Eh ;
}
double axilrodContribution::energy(const QVector<atom*>& atoms)
{
        if(atoms.size() != 3) return 0. ;
        coord r12 = *atoms[1] - *atoms[0], r13 = *atoms[2] - *atoms[0], r23 = *atoms[2] - *atoms[1] ;
        return C * (3.* (r12 < r13) * - (r12 < r23) * (r13 < r23) + 1.) / pow(~r12, 3) / pow(~r13, ►
            3) / pow(~r23, 3) ;
}
void axilrodContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Partners:␣␣" ;
        for(int i = 0 ; i < partners().size() ; i++)
                cout << partners() [i] << ",␣" ;
        cout << ".␣␣Parameter:␣" << C << endl ;
}
energyContribution* axilrod::newInteraction()
{ return new axilrodContribution() ; }
void axilrod::doSort(QStringList& list)
{ qSort(list) ; }
int axilrod::numAtoms()
{ return 3 ; }
void axilrod::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Axilrod-Teller-Wechselwirkungspotential.␣␣Funktionale␣Form:␣␣V␣=␣C*(3*cos(theta_i)*►
            cos(theta_j)*cos(theta_k)␣+␣1)␣/␣r12^3␣/␣r13^3␣/␣r23^3."
             << "Beitraege:" << endl ;
        foreach(energyContribution * pointer, interactions)
        pointer->printDescription() ;
}
```

src/cluster.h

```
#ifndef CLUSTER_H
#define CLUSTER_H
#include "atom.h"
#include "energyfunction.h"
#include <QVector>
#include <QPair>
#include <QHash>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
class cluster
{
private:
        QVector<atom*> atoms ;
        QVector<energyFunction*> energies ;
        QVector<QPair<energyContribution*, QVector<atom*> > > contributions ;
        QVector<QPair<energyFunction*, QVector<int> > > copyContributions ;
        QVector<QVector<int> > allCombinations(const int&) ;
        QVector<QVector<atom*> > combinationsOfAtoms(const int& depth) ;
        bool energyValid ;
        bool gradientValid ;
        double lastEnergy ;
        QVector<double> lastGradient ;
        void invalidate() ;
        void clear() ;
public:
        cluster();
        ~cluster();
        cluster(const cluster&) ;
        bool wouldInvertAtoms(const QVector<double>& newCoords) ;
        void addAtom(atom*) ;
        void addEnergyFunction(energyFunction*) ;
        void printAtoms(bool distances = false) ;
        QVector<double> coordinates() ;
        const QVector<atom*>& atomVector() const { return atoms ; }
        gsl_vector* gslCoordinates() ;
        void setCoordinates(const gsl_vector*) ;
        void setCoordinates(const QVector<double>& coords) ;
        double energy() ;
        void centerOnFirstAtom() ;
        QVector<double> force(double step = 1e-5) ;
        QVector<QVector<double> >* hessian(double step = 1e-5) ;
        gsl_matrix* gslHessian(double step = 1e-5) ;
        void printEnergy() ;
        bool initialize() ;
        QVector<double> invMasses() ;
        int size() ;
        cluster& operator= (const cluster&) ;
};
#endif
```

src/cluster.cpp

```
#include "cluster.h"
#include <QTextStream>
#include <math.h>
#include <QTime>
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
cluster::cluster()
        : energyValid(false),
          gradientValid(false)
{
}
cluster::~cluster()
{
        clear() ;
}
void cluster::clear()
{
        foreach(atom * a, atoms)
        delete a ;
        foreach(energyFunction * e, energies)
        delete e ;
        atoms.clear() ;
```

```
        energies.clear() ;
        contributions.clear() ;
}
bool cluster::wouldInvertAtoms(const QVector<double>& newCoords)
{
        QVector<double> oldCoords = coordinates() ;
        QVector<coord> oldVectors ;
        QVector<QVector<int> > dist = allCombinations(2) ;
        for(QVector<QVector<int> >::size_type i = 0 ; i < dist.size() ; i++)
                oldVectors << *atoms[dist[i][0]] - *atoms[dist[i][1]] ;
        setCoordinates(newCoords) ;
        bool wouldInvert = false ;
        for(QVector<QVector<int> >::size_type i = 0 ; i < dist.size() ; i++)
                wouldInvert |= oldVectors[i] * (*atoms[dist[i][0]] - *atoms[dist[i][1]]) < 0 ;
        setCoordinates(oldCoords) ;
        return wouldInvert ;
}
void cluster::addAtom(atom* newAtom)
{
        if(newAtom)
                atoms << newAtom ;
        invalidate();
}
void cluster::printAtoms(bool distances)
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout.setFieldWidth(15) ;
        for(QVector<atom*>::size_type i = 0; i < atoms.size() ; i++)
                cout << atoms[i]->type() <<
                     atoms[i]->operator[](0) <<
                     atoms[i]->operator[](1) <<
                     atoms[i]->operator[](2) <<
                     atoms[i]->mass() << endl ;
        if(!distances) return ;
        cout << "----␣Distances:␣----" << endl ;
        QVector<QVector<int> > dist = allCombinations(2) ;
        for(QVector<QVector<int> >::size_type i = 0 ; i < dist.size() ; i++)
                cout << atoms[dist[i][0]]->type() << "␣␣" << atoms[dist[i][1]]->type() << "\t" << * ►
                     (atoms[dist[i][0]]) % * (atoms[dist[i][1]]) << endl ;
}
QVector<QVector<int> > cluster::allCombinations(const int& depth)
{
        QVector<QVector<int> > retList ;
        if(depth > atoms.size() || depth == 0)
                return retList ;
        if(abs(depth) == 1)
                for(int i = 0 ; i < atoms.size() ; i++)
                        retList << (QVector<int>() << i) ;
        QVector<QVector<int> > templateList = allCombinations(depth + (depth > 0 ? -1 : 1)) ;
        for(int i = 0 ; i < templateList.size() ; i++)
        {
                QVector<int> temporaryList = templateList[i] ;
                for(int j = (depth > 0 ? temporaryList.last() + 1 : 0) ; j < atoms.size() ; j++)
                        if(!temporaryList.contains(j))
                                retList << (QVector<int> (temporaryList) << j) ;
        }
        return retList ;
}
QVector<QVector<atom*> > cluster::combinationsOfAtoms(const int& depth)
{
        QVector<QVector<atom*> > finalList ;
        if(depth == 0)
                return (QVector<QVector<atom*> >() << atoms) ;
        QVector<QVector<int> > initialList = allCombinations(depth) ;
        foreach(QVector<int> list, initialList)
        {
                QVector<atom*> atomList ;
                foreach(int i, list)
                atomList << atoms[i] ;
                finalList << atomList ;
        }
        return finalList ;
}
void cluster::addEnergyFunction(energyFunction* energy)
{
```

```
        if(energy)
                energies << energy ;
        invalidate();
}
bool cluster::initialize()
{
        QStringList atomNames ;
        for(int i = 0 ; i < atoms.size() ; i++)
                atomNames << atoms[i]->type() ;
        for(int i = 0 ; i < energies.size() ; i++)
        {
                if(energies[i]->numAtoms() >= 0)
                {
                        QVector<QVector<atom*> > list = combinationsOfAtoms(energies[i]->numAtoms()) ;
                        for(int j = 0 ; j < list.size() ; j++)
                        {
                                energyContribution* function = energies[i]->explicitFunction(list[j]) ;
                                if(function)
                                {
                                        contributions << QPair<energyContribution*, QVector<atom*> >(function, list[j]) ;
                                        QVector<int> indices ;
                                        foreach(atom * pointer, list[j])
                                        indices << atoms.indexOf(pointer) ;
                                        copyContributions << QPair<energyFunction*, QVector<int> > (energies[i], indices) ;
                                }
                        }
                }
                else
                {
                        QVector<energyContribution*> functions = energies[i]->functions() ;
                        foreach(energyContribution * contribution, functions)
                        {
                                QStringList names = contribution->partners() ;
                                bool possible = true ;
                                foreach(QString name, names)
                                if(!(possible = possible && atomNames.contains(name))) break ;
                                if(possible)
                                {
                                        QVector<atom*> list ;
                                        for(int j = 0 ; j < names.size() ; j++)
                                                list << atoms[atomNames.indexOf(names[j])] ;
                                        energyContribution* function = energies[i]->explicitFunction(list) ;
                                        if(function)
                                        {
                                                contributions << QPair<energyContribution*, QVector<atom*> > (function, list) ;
                                                QVector<int> indices ;
                                                foreach(atom * pointer, list)
                                                indices << atoms.indexOf(pointer) ;
                                                copyContributions << QPair<energyFunction*, QVector<int> > (energies[i], indices) ;
                                        }
                                }
                        }
                }
        }
        invalidate();
        if(contributions.isEmpty() || atoms.isEmpty())
                return false ;
        return true ;
}
double cluster::energy()
{
        if(energyValid) return lastEnergy ;
        double energy = 0 ;
        for(int i = 0 ; i < contributions.size() ; i++)
                energy += contributions[i].first->energy(contributions[i].second);
        lastEnergy = energy ;
        energyValid = true ;
        return energy ;
}
```

```
166 void cluster::printEnergy()
    {
            QTextStream cout(stdout, QIODevice::WriteOnly) ;
            for(int i = 0 ; i < energies.size() ; i++)
            {
171                 QVector<energyContribution*> contribs = energies[i]->functions() ;
                    double energy = 0 ;
                    for(int j = 0 ; j < contributions.size() ; j++)
                    {
                            energyContribution* pointer = contributions[j].first ;
176                         if(contribs.contains(pointer) || (energyContribution*) energies[i] == pointer)
                                    energy += pointer->energy(contributions[j].second) ;
                    }
                    energies[i]->printDescription() ;
                    cout << "Energy␣contribution␣" << i << ":␣␣" << energy << endl ;
181         }
    }
    QVector<double> cluster::coordinates()
    {
            QVector<double> coords ;
186         for(int i = 0 ; i < atoms.size() ; i++)
                    for(int j = 0 ; j < 3 ; j++)
                            coords << atoms[i]->operator[](j) ;
            return coords ;
    }
191 gsl_vector* cluster::gslCoordinates()
    {
            gsl_vector* coords = gsl_vector_alloc(atoms.size() * 3) ;
            double* coordData = coords->data ;
            for(int i = 0 ; i < atoms.size() ; i++)
196                 for(int j = 0 ; j < 3 ; j++)
                            coordData[i * 3 + j] = atoms[i]->operator[](j) ;
            return coords ;
    }
    void cluster::setCoordinates(const QVector<double>& coords)
201 {
            for(int i = 0 ; 3 * i + 2 < coords.size() && i < atoms.size() ; i++)
                    * (atoms[i]) = coords.mid(3 * i, 3) ;
            invalidate();
    }
206 void cluster::invalidate()
    {
            energyValid = false ;
            gradientValid = false ;
    }
211 void cluster::setCoordinates(const gsl_vector* const coords)
    {
            double* coordData = coords->data ;
            for(size_t i = 0 ; 3 * i + 2 < coords->size && i < (size_t) atoms.size() ; i++)
                    for(int j = 0 ; j < 3 ; j++)
216                         atoms[i] ->operator[](j) = coordData[3 * i + j] ;
            invalidate();
    }
    QVector<double> cluster::invMasses()
    {
221         QVector<double> list ;
            for(int i = 0 ; i < atoms.size() ; i++)
            {
                    double val = 1. / sqrt(atoms[i]->mass()) ;
                    list << val << val << val ;
226         }
            return list ;
    }
    cluster::cluster(const cluster& ref)
    { *this = ref ; }
231 cluster& cluster::operator= (const cluster& original)
    {
            foreach(atom * pointer, original.atoms)
            atoms << new atom(*pointer) ;
            foreach(energyFunction * pointer, original.energies)
236         energies << pointer->clone() ;
            initialize() ;
            lastEnergy = original.lastEnergy ;
            energyValid = original.energyValid ;
            lastGradient = original.lastGradient ;
```

```
        gradientValid = original.energyValid ;
        return *this ;
}
QVector<double> cluster::force(double step)
{
        if(gradientValid) return lastGradient ;
        if(!energyValid) energy() ;
        QVector<double> derivative(3 * atoms.size(), 0.) ;
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        for(int j = 0 ; j < contributions.size() ; j++)
        {
                QVector<double>* tempgrad = contributions[j].first->force(contributions[j].second, ►
                    step) ;
                if(!tempgrad) continue ;
                for(int k = 0 ; k < copyContributions[j].second.size() ; k++)
                {
                        int i = copyContributions[j].second[k] * 3, kk = k * 3 ;
                        for(int l = 0 ; l < 3 ; l++)
                                derivative[i + l] += (*tempgrad) [kk + l] ;
                }
                delete tempgrad ;
        }
        lastGradient = derivative ;
        gradientValid = true ;
        return derivative ;
}
QVector<QVector<double> >* cluster::hessian(double step)
{
        QVector<QVector<double> >* hesse = new QVector<QVector<double> > (3 * atoms.size(), QVector<►
            double> (3 * atoms.size(), 0.));
        QVector<double> mu = invMasses() ;
        for(int k = 0 ; k < contributions.size() ; k++)
        {
                QVector<QVector<double> >* tempLaplace = contributions[k].first->laplace(►
                    contributions[k].second, step) ;
                if(!tempLaplace) continue ;
                for(int l = 0 ; l < copyContributions[k].second.size() ; l++)
                {
                        int i = copyContributions[k].second[l] * 3, ll = l * 3 ;
                        for(int ii = 0 ; ii < 3 ; ii++)
                        {
                                for(int m = 0 ; m < copyContributions[k].second.size() ; m++)
                                {
                                        int j = copyContributions[k].second[m] * 3, mm = m * 3 ;
                                        for(int jj = 0 ; jj < 3 ; jj++)
                                                (*hesse) [i + ii][j + jj] += (*tempLaplace) [ll + ii►
                                                    ][mm + jj] ;
                                }
                        }
                }
                delete tempLaplace ;
        }
        for(int i = 0 ; i < atoms.size() ; i++)
        {
                for(int j = 0 ; j < atoms.size() ; j++)
                {
                        double mus = mu[i * 3] * mu[j * 3] ;
                        for(int ii = 3 * i ; ii < 3 * (i + 1) ; ii++)
                                for(int jj = 3 * j ; jj < 3 * (j + 1) ; jj++)
                                        (*hesse) [ii][jj] *= mus ;
                }
        }
        return hesse ;
}
gsl_matrix* cluster::gslHessian(double step)
{
        QVector<QVector<double> >* h = hessian(step) ;
        gsl_matrix* hess = gsl_matrix_alloc(h->size(), h->size()) ;
        for(int i = 0 ; i < h->size() ; ++i)
                for(int j = 0 ; j < h->at(i).size() ; ++j)
                        gsl_matrix_set(hess, i, j, h->at(i).at(j)) ;
        delete h ;
        return hess ;
}
int cluster::size()
```

```
{ return atoms.size() ; }
void cluster::centerOnFirstAtom()
{
        coord newCenter = *atoms[0] ;
        for(int i = 0 ; i < atoms.size() ; i++)
                *atoms[i] -= newCenter ;
}
```

src/commoninteraction.h

```
#ifndef COMMONINTERACTION_H
#define COMMONINTERACTION_H
#include "energyfunction.h"
class commonInteraction : public energyFunction, public energyContribution
{
protected:
        energyContribution* newInteraction() ;
public:
        ~commonInteraction();
        energyContribution* explicitFunction(QVector<atom*>) ;
        int numAtoms() ;
        void doSort(QStringList&) ;
};
#endif
```

src/commoninteraction.cpp

```
#include "commoninteraction.h"
commonInteraction::~commonInteraction()
{
        interactions.clear();
}
energyContribution* commonInteraction::newInteraction()
{
        return this ;
}
energyContribution* commonInteraction::explicitFunction(QVector<atom*> a)
{
        Q_UNUSED(a)
        return this ;
}
int commonInteraction::numAtoms()
{
        return 0 ;
}
void commonInteraction::doSort(QStringList& l)
{
        Q_UNUSED(l)
}
```

src/computationmodule.h

```
#ifndef COMPUTATIONMODULE_H
#define COMPUTATIONMODULE_H
#include "variablehash.h"
#include <QString>
class cluster ;
class computationModule
{
protected:
        virtual QString name() const = 0 ;
        cluster* Cluster ;
        const variableHash::computationSettings* Settings ;
        virtual void run() = 0 ;
        static bool collinear(const QVector<double>& coords) ;
        void printEnergy(double e, bool newl = true) const ;
public:
        computationModule() ;
        void setSettings(const variableHash::computationSettings*) ;
        void setCluster(cluster*) ;
```

```
        void execute() ;
        static QHash<QString, computationModule*> allModules() ;
        virtual ~computationModule() {}
};
#endif
```

src/computationmodule.cpp

```
#include "computationmodule.h"
#include <QTime>
#include <gsl/gsl_sort_vector.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_permutation.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_multimin.h>
#include <gsl/gsl_eigen.h>
#include "constants.h"
#include "gsloptimizer.h"
#include <QThreadPool>
#include "variablehash.h"
QTextStream out(stdout, QIODevice::WriteOnly) ;
bool computationModule::collinear(const QVector<double>& coords)
{
        if(coords.size() == 6)
                return true ;
        QList<coord> cartCoords ;
        for(int i = 0 ; i < coords.size() / 3 ; i++)
                cartCoords << coord(coords.mid(i * 3, 3)) ;
        coord reference = cartCoords[1] - cartCoords[0] ;
        double refNorm = ~reference ;
        bool collinearity = true ;
        for(int i = 2 ; i < cartCoords.size() ; i++)
        {
                coord test = cartCoords[i] - cartCoords[0] ;
                collinearity &= fabs(test * reference) == ~test * refNorm ;
        }
        return collinearity ;
}
computationModule::computationModule()
        : Cluster(0),
          Settings(0)
{}
void computationModule::setSettings(const variableHash::computationSettings* s)
{
        Settings = s ;
}
void computationModule::setCluster(cluster* c)
{
        Cluster = c ;
}
void computationModule::execute()
{
        QTime time ;
        time.start();
        out << "----------␣" << name() << "␣----------" << endl ;
        if(!Cluster)
        {
                out << "Error:␣no␣valid␣cluster␣defined" << endl ;
                return ;
        }
        if(!Settings)
        {
                out << "Error:␣no␣settings␣defined" << endl ;
                return ;
        }
        if(Settings->verbosity > 1)
        {
                out << "␣---␣Cluster␣---␣" << endl ;
                Cluster->printAtoms() ;
                out << "␣---------------␣" << endl ;
        }
        run() ;
        if(Settings->verbosity > 0)
                out << "Timing:␣" << time.elapsed() << "␣ms" << endl ;
```

```
67          out << "----------␣" << name() << "␣done.␣----------" << endl ;
    }
    class energyComputation : public computationModule
    {
            QString name() const { return "Energy" ; }
72          void run()
            {
                    out << "Energy:␣" ;
                    printEnergy(Cluster->energy()) ;
                    if(Settings->verbosity > 1)
77                          Cluster->printEnergy();
            }
    };
    class gradientComputation : public computationModule
    {
82          QString name() const { return "Gradient" ; }
            void run()
            {
                    out << "Gradient:" << endl ;
                    double norm = 0 ;
87                  foreach(const double & d, Cluster->force(Settings->gradientStep))
                    {
                            norm += d*d ;
                            out << d << endl ;
                    }
92                  out << "Gradient␣norm:␣" << sqrt(norm) << endl ;
            }
    };
    class geometryOptimization : public computationModule
    {
97          QString name() const { return "Geometry␣optimization" ; }
            void run()
            {
                    int i = 0 ;
                    gslOptimizer::runOptimization(Cluster,
102                                               Settings,
                                                  i,
                                                  &out) ;
                    out << "Final␣energy:␣" ;
                    printEnergy(Cluster->energy()) ;
107         }
    };
    class zpeComputation : public computationModule
    {
            QString name() const { return "Zero␣point␣energy␣computation" ; }
112         void run()
            {
                    double energy = Cluster->energy() ;
                    gsl_matrix* hessian = Cluster->gslHessian(Settings->gradientStep) ;
                    gsl_vector* eigenWerte = gsl_vector_alloc(hessian->size1) ;
117                 gsl_matrix* eigenVektoren = gsl_matrix_alloc(hessian->size1, hessian->size2) ;
                    gsl_eigen_symmv_workspace* w = gsl_eigen_symmv_alloc(hessian->size1) ;
                    gsl_eigen_symmv(hessian, eigenWerte, eigenVektoren, w) ;
                    gsl_eigen_symmv_free(w) ;
                    gsl_matrix_free(hessian) ;
122                 gsl_eigen_symmv_sort(eigenWerte, eigenVektoren, GSL_EIGEN_SORT_VAL_ASC) ;
                    double zpe = 0. ;
                    bool coll = collinear(Cluster->coordinates()) ;
                    if(Settings->verbosity > 2)
                            out << "Frequencies/meV:" << endl ;
127                 for(size_t i = coll ? 5 : 6 ; i < eigenWerte->size ; ++i)
                    {
                            double f = parser::freq / M_PI * parser::cm / 2. * sqrt(gsl_vector_get(►
                                eigenWerte, i)) ;
                            if(Settings->verbosity > 2)
                                    printEnergy(f) ;
132                         zpe += f / 2. ;
                    }
                    gsl_vector_free(eigenWerte) ;
                    if(Settings->verbosity > 3)
                    {
137                         out << "Eigenvectors:" << endl ;
                            for(size_t i = 0 ; i < eigenVektoren->size1 ; i++)
                            {
                                    for(size_t j = 0 ; j < eigenVektoren->size2 ; ++j)
```

```
                                        out << gsl_matrix_get(eigenVektoren, i, j) << "\t";
                                out << endl ;
                        }
                        out << endl ;
                }
                gsl_matrix_free(eigenVektoren) ;
                out << "Zero point energy: " ;
                printEnergy(zpe) ;
                out << "Total energy:      " ;
                printEnergy(energy + zpe) ;
        }
};
class bulkOptimizer : public computationModule
{
public:
        typedef QPair<QString, cluster*> geometryDefinition ;
        typedef QVector<geometryDefinition> geometryCollection ;
private:
        QVector<gslOptimizer*> threads ;
        virtual geometryCollection geometries(const int&) const = 0 ;
        virtual QString parameterHeader() const = 0;
        void run()
        {
                double bestEnergy = 0 ;
                int bestIndex = -1;
                int calls = 0 ;
                do
                {
                        geometryCollection collection = geometries(calls) ;
                        foreach(geometryDefinition geometry, collection)
                                threads << new gslOptimizer(Settings,
                                                            geometry.second) ;
                        QThreadPool pool ;
                        if(Settings->verbosity > 1)
                                out << "Optimizing " << threads.size() << " geometries, " << pool.►
                                    maxThreadCount() << " at a time." << endl ;
                        foreach(gslOptimizer * thread, threads)
                        pool.start(thread) ;
                        pool.waitForDone();
                        if(Settings->verbosity > 0)
                                out << "Optimizations finished." << endl
                                    << parameterHeader() << "\tenergy\tsteps" << endl ;
                        for(int i = 0 ; i < threads.size() ; ++i)
                        {
                                double e = threads[i]->finalEnergy() ;
                                int s = threads[i]->steps() ;
                                if(Settings->verbosity > 0)
                                        out << collection[i].first << "\t" << e << "\t"
                                            << s << (s < Settings->maxSteps ? " " : "*")
                                            << endl ;
                                if(s < Settings->maxSteps && e < bestEnergy && e > Settings->►
                                    lowerLimit)
                                        bestEnergy = e, bestIndex = i ;
                                ++i ;
                        }
                        if(bestIndex < 0)
                        {
                                out << "No valid best geometry found, trying again." << endl ;
                                clearThreads();
                                ++calls ;
                        }
                }
                while(bestIndex < 0) ;
                out << "Best energy found was: " ;
                printEnergy(bestEnergy, false) ;
                out << " in attempt "
                    << bestIndex
                    << " of "
                    << threads.size()
                    << endl ;
                if(bestIndex < threads.size())
                        Cluster->setCoordinates(threads[bestIndex]->coordinates()) ;
                clearThreads() ;
        }
        void clearThreads()
```

```
        {
                foreach(gslOptimizer * thread, threads)
                delete thread ;
                threads.clear();
        }
};
class sphereOptimize : public bulkOptimizer
{
        QString parameterHeader() const { return "phi\ttheta" ; }
        QString name() const { return "Optimization of last added atom on sphere" ; }
        geometryCollection geometries(const int& calls) const
        {
                geometryCollection collection ;
                double r = 0 ;
                foreach(atom * a, Cluster->atomVector())
                        if((~ *a) > r)
                                r = (~*a) ;
                r *= pow(2 + 2*Settings->includeNextShell, calls) ;
                if(Settings->verbosity > 0)
                        out << "Sphere radius is " << r << " Ang." << endl ;
                generateGeometries(r, collection);
                if (Settings->includeNextShell)
                {
                        if (Settings->verbosity > 0)
                                out << "Adding second sphere with doubled radius." << endl ;
                        generateGeometries(2.*r, collection);
                }
                return collection ;
        }
        void generateGeometries(const double& r, geometryCollection& collection) const
        {
                QVector<double> initialCoords = Cluster->coordinates() ;
                for(double theta = 0 ; theta <= 180 ; theta += Settings->altitudeStep)
                {
                        for(double phi = 0 ; phi < 360 ; phi += Settings->azimuthStep / sin(theta / ►
                            180 * M_PI))
                        {
                                initialCoords[initialCoords.size() - 3] = r * cos(phi / 180 * M_PI) ►
                                    * sin(theta / 180 * M_PI) ;
                                initialCoords[initialCoords.size() - 2] = r * sin(phi / 180 * M_PI) ►
                                    * sin(theta / 180 * M_PI) ;
                                initialCoords[initialCoords.size() - 1] = r * cos(theta / 180 * M_PI►
                                    ) ;
                                cluster* testCluster = new cluster(*Cluster) ;
                                testCluster->setCoordinates(initialCoords) ;
                                collection << qMakePair(QString::number(phi) + "\t" + QString::►
                                    number(theta),
                                                        testCluster) ;
                        }
                }
        }
};
class boxOptimize : public bulkOptimizer
{
        QString parameterHeader() const { return "x\ty\tz" ; }
        QString name() const { return "Optimization of last added atom in a box" ; }
        geometryCollection geometries(const int& calls) const
        {
                geometryCollection collection ;
                double bsize = fabs(Settings->boxSize) * pow(2, calls) ;
                double bstep = fabs(Settings->boxStep) * pow(2, calls) ;
                if(Settings->verbosity > 0)
                        out << "Box size is " << bsize << " Ang, step size is " << bstep << endl ;
                QVector<double> initialCoords = Cluster->coordinates() ;
                for(double x = -bsize ; x <= bsize ; x += bstep)
                {
                        for(double y = -bsize ; y <= bsize ; y += bstep)
                        {
                                for(double z = -bsize ; z <= bsize ; z += bstep)
                                {
                                        initialCoords[initialCoords.size() - 3] = x ;
                                        initialCoords[initialCoords.size() - 2] = y ;
                                        initialCoords[initialCoords.size() - 1] = z ;
                                        cluster* testCluster = new cluster(*Cluster) ;
                                        testCluster->setCoordinates(initialCoords) ;
```

```
                                        collection << qMakePair(QString::number(x) + "\t"
                                                                + QString::number(y) + "\t"
                                                                + QString::number(z),
287                                                             testCluster) ;
                                }
                        }
                }
                return collection ;
292     }
};
void computationModule::printEnergy(double e, bool newl) const
{
        out << e << "␣meV␣(" << e / parser::cm << "␣cm-1)" ;
297     if(newl) out << endl ;
}
QHash<QString, computationModule*> computationModule::allModules()
{
        QHash<QString, computationModule*> all ;
302     all["energy"] = new energyComputation ;
        all["optimize"] = new geometryOptimization ;
        all["hessian"] = new zpeComputation ;
        all["sphereOptimize"] = new sphereOptimize ;
        all["gradient"] = new gradientComputation ;
307     all["boxOptimize"] = new boxOptimize ;
        return all ;
}
```

src/constants.h

```
1   #ifndef CONSTANTS_H
    #define CONSTANTS_H
    #include <cmath>
    namespace parser
    {
6           const double Eh = 27211.38505,
                         a0 = 0.52917721092,
                         freq = 103.61197566988695,
                         cm = 0.123984193 ;
            const double a0e3 = pow(a0, 3),
11                       a0e5 = pow(a0, 5);
            const double a0e8 = a0e3* a0e5,
                         a0e9 = a0e3* a0e3* a0e3 ;
    }
    #endif
```

src/coord.h

```
    #ifndef COORD_H
    #define COORD_H
    #include <QVector>
    #include <QTextStream>
5   class coord
    {
            double x[3] ;
    public:
            coord(const double& xi = 0, const double& yi = 0, const double& zi = 0);
10          coord(const QVector<double>&) ;
            ~coord();
            void setCoord(const int&, const double&) ;
            double coordValue(const int&) ;
            double& operator[](const int&) ;
15          const double& operator[](const int&) const ;
            double operator* (const coord&) const ;
            coord operator* (const double&) const ;
            coord operator/ (const double&) const ;
            coord operator- (const coord&) const ;
20          coord operator+ (const coord&) const ;
            coord operator-() const ;
            coord operator/ (const coord&) const ;
            coord& operator/= (const double&) ;
            coord& operator*= (const double&) ;
25          double operator~() const ;
```

```
        coord operator!() const ;
        double operator% (const coord&) const ;
        double operator> (const coord&) const ;
        double operator< (const coord&) const ;
        coord& operator= (const QVector<double>&) ;
        coord& operator-= (const coord&) ;
        coord& operator+= (const coord&) ;
};
QTextStream& operator<< (QTextStream& out, const coord& a) ;
#endif
```

src/coord.cpp

```
#include "coord.h"
#include <math.h>
coord::coord(const double& xi, const double& yi, const double& zi)
{
        x[0] = xi ;
        x[1] = yi ;
        x[2] = zi ;
}
coord::coord(const QVector<double>& list)
{
        *this = list ;
}
coord::~coord()
{
}
double& coord::operator[](const int& i) { return x[i] ; }
const double& coord::operator[](const int& i) const { return x[i] ; }
double coord::operator> (const coord& other) const
{
        const coord& here = *this ;
        return acos(qMin(qMax((!here) * (!other), -1.), 1.)) ;
}
double coord::operator< (const coord& other) const
{
        const coord& here = *this ;
        return qMin(qMax((!here) * (!other), -1.), 1.) ;
}
double coord::operator* (const coord& other) const
{
        double dotProd = 0 ;
        for(int i = 0 ; i < 3 ; i++)
                dotProd += x[i] * other[i] ;
        return dotProd ;
}
coord coord::operator- (const coord& other) const
{ return coord(x[0] - other[0], x[1] - other[1], x[2] - other[2]) ; }
coord coord::operator+ (const coord& other) const
{ return coord(x[0] + other[0], x[1] + other[1], x[2] + other[2]) ; }
coord coord::operator* (const double& other) const
{ return coord(x[0] * other, x[1] * other, x[2] * other) ; }
coord& coord::operator *= (const double& d)
{
        x[0] *= d ;
        x[1] *= d ;
        x[2] *= d ;
        return *this ;
}
coord& coord::operator /= (const double& d)
{
        return *this *= 1. / d ;
}
coord coord::operator/ (const double& other) const
{ return *this * (1. / other) ; }
coord coord::operator-() const
{ return coord(-x[0], -x[1], -x[2]) ; }
coord coord::operator/ (const coord& other) const
{ return coord(x[1] * other[2] - x[2] * other[1], x[2] * other[0] - x[0] * other[2], x[0] * other[1]►
      - x[1] * other[0]) ; }
double coord::operator~() const
{
        double norm = 0 ;
```

```
        for(int i = 0 ; i < 3 ; i++)
                norm += pow(x[i], 2) ;
        return sqrt(norm) ;
}
coord coord::operator!() const
{
        const double norm = ~*this ;
        return coord(x[0] / norm, x[1] / norm, x[2] / norm) ;
}
double coord::operator% (const coord& other) const
{
        return ~(*this - other) ;
}
coord& coord::operator= (const QVector<double>& list)
{
        for(int i = 0 ; i < list.size() && i < 3 ; i++)
                x[i] = list[i] ;
        return *this ;
}
coord& coord::operator-= (const coord& other)
{
        return (*this += - other) ;
}
coord& coord::operator+= (const coord& other)
{
        for(int i = 0 ; i < 3 ; i ++)
                x[i] += other.x[i] ;
        return *this ;
}
void coord::setCoord(const int& index, const double& value)
{ x[index] = value ; }
double coord::coordValue(const int& index)
{ return x[index] ;}
QTextStream& operator<< (QTextStream& out, const coord& a)
{
        out << "(" << a[0] << "," << a[1] << "," << a[2] << ")" ;
        return out ;
}
```

src/coulomb.h

```
#ifndef COULOMB_H
#define COULOMB_H
#include "commoninteraction.h"
class coulomb : public commonInteraction
{
public:
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
        int numAtoms() ;
        energyContribution* explicitFunction(QVector<atom*>) ;
        coulomb* create() const { return new coulomb() ;}
};
#endif
```

src/coulomb.cpp

```
#include "coulomb.h"
#include "constants.h"
void coulomb::setParameters(const QVector<double>&) {}
QVector<double> coulomb::parameters() const { return QVector<double>() ;}
double coulomb::energy(const QVector<atom*>& atoms)
{
        return atoms[0]->charge() * atoms[1]->charge() / (*atoms[0] % *atoms[1]) * parser::Eh * ►
            parser::a0 ;
}
void coulomb::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Coulomb-Wechselwirkung" << endl ;
```

```
}
int coulomb::numAtoms() { return 2 ;}
energyContribution* coulomb::explicitFunction(QVector<atom*> atoms)
{
        if(atoms.size() != 2) return 0 ;
        if(atoms[0]->charge() == 0. || atoms[1]->charge() == 0.) return 0 ;
        return this ;
}
```

src/energyfunction.h

```
#ifndef ENERGYFUNCTION_H
#define ENERGYFUNCTION_H
#include <QList>
#include <QStringList>
#include "atom.h"
class energyContribution
{
protected:
        static QVector<double>* bMatrixBond(const QVector<atom*>&) ;
        static QVector<double>* bMatrixAngle(const QVector<atom*>&) ;
        static QVector<double>* bMatrixDieder(const QVector<atom*>&) ;
        static QVector<double>* bMatrixInversion(const QVector<atom*>&) ;
        QStringList partner ;
public:
        void setPartners(QStringList&) ;
        virtual void setParameters(const QVector<double>&) = 0 ;
        virtual QVector<double> parameters() const = 0 ;
        virtual double energy(const QVector<atom*>&) = 0 ;
        virtual QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        virtual QVector<QVector<double> >* laplace(const QVector<atom*>&, double step, bool refresh ►
            = false) ;
        virtual void printDescription() const ;
        virtual void initialize(const QVector<atom*>& atoms) ;
        const QStringList& partners() const ;
        virtual ~energyContribution() {} ;
};
class energyFunction
{
protected:
        QVector<energyContribution*> interactions ;
        virtual energyContribution* newInteraction() = 0;
        virtual void doSort(QStringList&) ;
public:
        bool addInteraction(QStringList types, QVector<double> parameters) ;
        energyFunction();
        energyFunction* clone() const;
        virtual energyFunction* create() const = 0 ;
        virtual int numAtoms() = 0 ;
        virtual void printDescription() const ;
        virtual energyContribution* explicitFunction(QVector<atom*>) ;
        QVector<energyContribution*> functions() { return interactions; }
        virtual ~energyFunction();
};
#endif
```

src/energyfunction.cpp

```
#include "energyfunction.h"
#include <QStringList>
#include <QTextStream>
#include <math.h>
energyFunction::energyFunction()
{}
energyFunction::~energyFunction()
{
        foreach(energyContribution * c, interactions)
        delete c ;
        interactions.clear() ;
}
QVector<double>* energyContribution::bMatrixBond(const QVector<atom*>& atoms)
{
```

```
        QVector<double>* bMatrixValues = new QVector<double>;
        coord r = !(*atoms[1] - *atoms[0]) ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << -r[j] ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << r[j] ;
        return bMatrixValues ;
}
QVector<double>* energyContribution::bMatrixAngle(const QVector<atom*>& atoms)
{
        QVector<double>* bMatrixValues = new QVector<double> ;
        coord r1 = (*atoms[0] - *atoms[1]),
                r2 = (*atoms[2] - *atoms[1]) ;
        double cphi = cos(r1 > r2),
                sphi = sin(r1 > r2) ;
        if(cphi == -1 || cphi == 1)
        {
                r1 = r1 + (r1 / coord(1., 0., 0.) + r1 / coord(0., 1., 0.) + r1 / coord(0., 0., 1.))►
                     * 1e-10 ;
                cphi = cos(r1 > r2) ;
                sphi = sin(r1 > r2) ;
        }
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << (cphi * (!r1) [j] - (!r2) [j]) / ~r1 / sphi ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << ((~r1 - cphi* ~r2) * (!r1) [j] + (~r2 - ~r1 * cphi) * (!r2) [j]) /►
                     ~r1 / ~r2 / sphi ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << (cphi * (!r2) [j] - (!r1) [j]) / (~r2) / sphi ;
        return bMatrixValues ;
}
QVector<double>* energyContribution::bMatrixDieder(const QVector<atom*>& atoms)
{
        QVector<double>* bMatrixValues = new QVector<double> ;
        coord r1 = (*atoms[1] - *atoms[0]),
                r2 = (*atoms[2] - *atoms[1]),
                r3 = (*atoms[3] - *atoms[2]);
        double sphi1 = sin(-r1 > r2),
                cphi1 = cos(-r1 > r2),
                sphi2 = sin(-r2 > r3),
                cphi2 = cos(-r2 > r3);
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << - (!r1 / !r2) [j] / ~r1 / sphi1 / sphi1 ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << (~r2 - ~r1 * cphi1) / ~r2 / ~r1 / sphi1 / sphi1 * (!r1 / !r2) [j] +
                                  cphi2 / ~r2 / sphi2 / sphi2 * (!r3 / !r2) [j] ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << (~r2 - ~r3 * cphi2) / ~r2 / ~r3 / sphi2 / sphi2 * (!r3 / !r2) [j] +
                                  cphi1 / ~r2 / sphi1 / sphi1 * (!r1 / !r2) [j] ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << - ((!r3) / (!r2)) [j] / (~r3) / sphi2 / sphi2 ;
        return bMatrixValues ;
}
QVector<double>* energyContribution::bMatrixInversion(const QVector<atom*>& atoms)
{
        QVector<double>* bMatrixValues = new QVector<double>;
        coord r1 = *atoms[0] - *atoms[3],
                r2 = *atoms[1] - *atoms[3],
                r3 = *atoms[2] - *atoms[3];
        coord n = !(r2 / r3),
                m = (!r1 - n * (n * !r1));
        double theta = (r1 * n < 0. ? -1 : 1) * (r1 > (!r1 - n * (n * !r1)) * (m * (!r2 + !r3) > 0 ?►
              -1 : 1)) ;
        double sphi = sin(r2 > r3),
                cphi = cos(r2 > r3),
                ttheta = tan(theta),
                ctheta = cos(theta);
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << ((!r2 / !r3) / ctheta / sphi - !r1 * ttheta) [j] / ~r1 ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << ((!r3 / !r1) / ctheta / sphi - (!r2 - !r3 * cphi) *ttheta / sphi /►
                     sphi) [j] / ~r2 ;
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << ((!r1 / !r2) / ctheta / sphi - (!r3 - !r2 * cphi) *ttheta / sphi /►
                     sphi) [j] / ~r3 ;
```

```
        for(int j = 0 ; j < 3 ; j++)
                *bMatrixValues << - ((*bMatrixValues) [j] + (*bMatrixValues) [3 + j] + (*►
                    bMatrixValues) [6 + j]) ;
87      return bMatrixValues ;
}
QVector<double>* energyContribution::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        Q_UNUSED(refresh)
92      QVector<double>* derivatives = new QVector<double>;
        double reference = energy(atoms) ;
        for(int j = 0 ; j < atoms.size() ; j++)
        {
                for(int i = 0 ; i < 3 ; i++)
97              {
                        (*atoms[j]) [i] += step ;
                        *derivatives << (reference - energy(atoms)) / step ;
                        (*atoms[j]) [i] -= step ;
                }
102     }
        return derivatives ;
}
QVector<QVector<double> >* energyContribution::laplace(const QVector<atom*>& atoms, double step, ►
    bool refresh)
{
107     QVector<QVector<double> >* laplacian = new QVector<QVector<double> > (3 * atoms.size(), ►
            QVector<double> (3 * atoms.size(), 0.)) ;
        for(int i = 0 ; i < 3 * atoms.size() ; i++)
        {
                (*atoms[i / 3]) [i % 3] += step ;
                QVector<double>* uderivedForce = force(atoms, step, refresh) ;
112             (*atoms[i / 3]) [i % 3] -= 2.*step ;
                QVector<double>* lderivedForce = force(atoms, step, refresh) ;
                (*atoms[i / 3]) [i % 3] += step ;
                if(!uderivedForce || !lderivedForce)
                {
117                     delete uderivedForce ;
                        delete lderivedForce ;
                        delete laplacian ;
                        return 0 ;
                }
122             for(int j = 0 ; j < lderivedForce->size() ; j ++)
                {
                        double value = - ((*uderivedForce) [j] - (*lderivedForce) [j]) / 2. / step ;
                        (*laplacian) [i][j] = value ;
                        (*laplacian) [j][i] = value ;
127             }
                delete uderivedForce ;
                delete lderivedForce ;
        }
        return laplacian ;
132 }
bool energyFunction::addInteraction(QStringList types, QVector<double> params)
{
        doSort(types) ;
        foreach(energyContribution * function, interactions)
137     {
                if(function->partners() == types)
                {
                        function->setParameters(params) ;
                        return false ;
142             }
        }
        interactions << newInteraction() ;
        interactions.last()->setPartners(types) ;
        interactions.last()->setParameters(params) ;
147     return true ;
}
energyContribution* energyFunction::explicitFunction(QVector<atom*> atoms)
{
        QStringList types ;
152     for(int i = 0 ; i < atoms.size() ; i++)
                types << atoms[i]->type() ;
        doSort(types) ;
        foreach(energyContribution * function, interactions)
        {
```

```
157                 if(function->partners() == types)
                    {
                            function->initialize(atoms) ;
                            return function ;
                    }
162         }
            return 0 ;
    }
    void energyContribution::setPartners(QStringList& list)
    { partner = list ; }
167 void energyFunction::doSort(QStringList& list)
    { qSort(list) ; }
    const QStringList& energyContribution::partners() const
    {
            return partner ;
172 }
    void energyFunction::printDescription() const
    {
            QTextStream cout(stdout, QIODevice::WriteOnly) ;
            cout << "Keine Beschreibung vorhanden." << endl ;
177         cout << "Beitraege:" << endl ;
            foreach(energyContribution * pointer, interactions)
            pointer->printDescription() ;
    }
    void energyContribution::printDescription() const
182 {
            QTextStream cout(stdout, QIODevice::WriteOnly) ;
            cout << "Keine Beschreibung vorhanden." << endl ;
    }
    void energyContribution::initialize(const QVector<atom*>& atoms)
187 { Q_UNUSED(atoms) }
    energyFunction* energyFunction::clone() const
    {
            energyFunction* other = create() ;
            foreach(energyContribution * contrib, interactions)
192         other->addInteraction(contrib->partners(), contrib->parameters()) ;
            return other ;
    }
```

src/esmsvw.h

```
 1  #ifndef ESMSVW_H
    #define ESMSVW_H
    #include "energyfunction.h"
    class esmsvwContribution : public energyContribution
    {
 6  private:
            double epsilon, rm, bm, C0, A, alpha, x1, x2, x3, x4 ;
            double a1, a2, a3, a4 ;
            double b1, b2, b3, b4 ;
    public:
11          void setParameters(const QVector<double> &) ;
            QVector<double> parameters() const ;
            double energy(const QVector<atom *> &) ;
            void printDescription() const ;
    };
16  class esmsvw : public energyFunction
    {
    protected:
            energyContribution* newInteraction() ;
    public:
21          void printDescription() const;
            int numAtoms() ;
            esmsvw* create() const { return new esmsvw() ; }
    };
    #endif
```

src/esmsvw.cpp

```
#include "esmsvw.h"
int esmsvw::numAtoms()
{
```

```
        return 2 ;
}
void esmsvw::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "ESMSvW potential " ;
        foreach(energyContribution * pointer, interactions)
        {
                cout << "(";
                QStringList ps = pointer->partners() ;
                foreach(QString p, ps)
                cout << p << "," ;
                cout << ") " ;
        }
        cout << endl ;
}
energyContribution* esmsvw::newInteraction()
{
        return new esmsvwContribution ;
}
#define ASSIGNMENTMACRO double *paras[] = { &epsilon, &rm, &bm, &C0, &A, &alpha, &x1, &x2, &x3, &x4 } ;
#define ASSIGNMENTMACROC const double *paras[] = { &epsilon, &rm, &bm, &C0, &A, &alpha, &x1, &x2, &x3, &x4 } ;
void esmsvwContribution::setParameters(const QVector<double> &params)
{
        ASSIGNMENTMACRO
        for (int i = 0 ; i < 10 ; ++i)
                *(paras[i]) = (i < params.size() ? params[i] : 0) ;
        a1 = log(A) - alpha*(x1-1) ;
        a2 = (log(exp(-2.*bm*(x2-1))-2.*exp(-bm*(x2-1)))-a1)/(x2-x1) ;
        a3 = (a2+alpha)/(x2-x1) ;
        a4 = ((-bm*(1+exp(-2*bm*(x2-1)-a1-a2*(x2-x1)))-a2)/(x2-x1)-a3)/(x2-x1) ;
        b1 = exp(-bm*(x3-1)) - 2*exp(-bm*(x3-1)) ;
        b2 = (-C0/epsilon/pow(rm*x4,6) - b1 ) / (x4-x3) ;
        b3 = (2*bm*(exp(-bm*(x3-1))-exp(-2*bm*(x3-1)))-b2)/(x3-x4) ;
        b4 = ((6*C0/epsilon/pow(rm,6)/pow(x4,7)-b2)/(x4-x3)-b3)/(x4-x3) ;
}
void esmsvwContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "ESMSvW contribution for: " ;
        for (int i = 0 ; i < partner.size() ; ++i)
                cout << partner[i] << " " ;
        ASSIGNMENTMACROC ;
        for (int i = 0 ; i < 10 ; ++i)
                cout << *(paras[i]) << " " ;
        cout << endl ;
}
QVector<double> esmsvwContribution::parameters() const
{
        ASSIGNMENTMACROC ;
        QVector<double> result ;
        for (int i = 0 ; i < 10 ; ++i)
                result << *(paras[i]) ;
        return result ;
}
double esmsvwContribution::energy(const QVector<atom *> & atoms)
{
        if (atoms.size() != 2) return 0 ;
        double r = *(atoms[0]) % *(atoms[1]) ;
        double x = r / rm ;
        return epsilon * ( x <= x1 ? A*exp(-alpha*(x-1)) :
                                     x < x2 ? exp(a1+(x-x1)*(a2+(x-x2)*(a3+(x-x1)*a4))) :
                                              x <= x3 ? exp(-2*bm*(x-1)) - 2*exp(-bm*(x-1)) :
                                                        x < x4 ? b1 + (x-x3)*(b2+(x-x4)*(b3+(x-x3)*b4)) :
                                                                 - C0/epsilon/pow(r,6)) ;
}
```

src/exchange.h

```
#ifndef EXCHANGE_H
#define EXCHANGE_H
```

```
#include "energyfunction.h"
class exchangeContribution : public energyContribution
{
private:
        double beta, threshold, lowerThresh ;
public:
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
};
class exchange : public energyFunction
{
protected:
        energyContribution* newInteraction() ;
        void doSort(QStringList&) ;
public:
        int numAtoms() ;
        void printDescription() const ;
        exchange* create() const { return new exchange() ;}
};
#endif
```

src/exchange.cpp

```
#include "exchange.h"
#include <math.h>
#include "constants.h"
void exchangeContribution::setParameters(const QVector<double>& params)
{
        if(params.size() >= 2)
        {
                beta = params[0] ;
                threshold = params[1] ;
                if (params.size() >=3) lowerThresh = params[2] ;
                else lowerThresh = threshold/10. ;
        }
}
QVector<double> exchangeContribution::parameters() const
{ return (QVector<double>() << beta << threshold) ;}
double exchangeContribution::energy(const QVector<atom*>& atoms)
{
        if(atoms.size() != 3) return 0. ;
        atom* charge = 0, *ex1 = 0, *ex2 = 0 ;
        for(int i = 0 ; i < atoms.size() ; i++)
        {
                if(atoms[i]->charge() && !charge)
                        charge = atoms[i] ;
                else if(ex1)
                        ex2 = atoms[i] ;
                else
                        ex1 = atoms[i] ;
        }
        if(!charge) return 0 ;
        double rij = *ex1 % *ex2, ri0 = *ex1 % *charge, rj0 = *ex2 % *charge, rC0 = *charge % ((*ex1►
             + *ex2) / 2.);
        if(rij > threshold || rC0 < lowerThresh)
                return 0. ;
        return (erf(beta * ri0) / ri0
                + erf(beta * rj0) / rj0
                - 2.*erf(beta * rC0) / rC0)
                        / (exp(.5 * pow(beta, 2) * pow(rij, 2)) - 1.) * parser::Eh * parser::a0 ;
}
void exchangeContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Partners:␣␣" ;
        for(int i = 0 ; i < partners().size() ; i++)
                cout << partners() [i] << ",␣" ;
        cout << "␣␣Parameters:␣␣beta=" << beta << "␣␣threshold␣=␣R_{ij}>" << threshold << "␣␣lower␣►
             threshold␣=␣R_{C0}␣<␣" << lowerThresh << endl ;
}
energyContribution* exchange::newInteraction()
{ return new exchangeContribution() ; }
```

```
void exchange::doSort(QStringList& list)
{ qSort(list) ; }
int exchange::numAtoms()
{ return 3 ; }
void exchange::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Austausch-Wechselwirkungspotential.␣␣Funktionale␣Form:␣"
             << "V=e^2*S_{ij}^2/(1-S_{ij}^2)*(erf(beta*R_{i0})/R_{i0}+erf(beta*R_{j0})/R_{j0}-2*erf(►
                 beta*R_{C0})/R_{C0}).␣␣Beitraege:" << endl ;
        foreach(energyContribution * pointer, interactions)
        pointer->printDescription() ;
}
```

src/gsloptimizer.h

```
#ifndef GSLOPTIMIZER_H
#define GSLOPTIMIZER_H
#include <QRunnable>
#include "cluster.h"
#include "variablehash.h"
class gslOptimizer : public QRunnable
{
private:
        cluster* Cluster ;
        int i ;
        const variableHash::computationSettings* settings ;
        double energy ;
public:
        gslOptimizer(const variableHash::computationSettings* s,
                     cluster* C);
        ~gslOptimizer() ;
        void run() ;
        inline double finalEnergy() { return energy ; }
        QVector<double> coordinates() ;
        inline bool operator< (gslOptimizer& other) { return energy < other.finalEnergy() ; }
        inline int steps() { return i ; }
        static double gslEnergy(const gsl_vector* coords, void* clusterPointer) ;
        static void gslGradient(const gsl_vector* coords, void* clusterPointer, gsl_vector* ►
            derivatives) ;
        static void gslBoth(const gsl_vector* coords, void* clusterPointer, double* envalue, ►
            gsl_vector* derivatives) ;
        static void runOptimization(cluster* Cluster,
                                    const variableHash::computationSettings *s,
                                    int& steps,
                                    QTextStream* out) ;
};
#endif
```

src/gsloptimizer.cpp

```
#include "gsloptimizer.h"
#include <gsl/gsl_multimin.h>
#include <gsl/gsl_blas.h>
gslOptimizer::gslOptimizer(const variableHash::computationSettings *s,
                           cluster* C)
        : Cluster(C),
          settings(s)
{
        setAutoDelete(false) ;
}
void gslOptimizer::runOptimization(cluster* Cluster,
                                   const variableHash::computationSettings* s,
                                   int& steps,
                                   QTextStream* out)
{
        double energy = INFINITY ;
        gsl_multimin_fdfminimizer* minimizer = 0 ;
#define OPTIMIZERPICKER(NUM,MINIMIZER,VERBOSETITLE) \
        case NUM: \
                minimizer = gsl_multimin_fdfminimizer_alloc( \
                        MINIMIZER, \
```

```
                        Cluster->size() * 3) ; \
                if (out && s->verbosity > 2) \
                        *out << "Using " << VERBOSETITLE << " optimizer." << endl ; \
                break ;
        switch(s->optimizerType)
        {
                OPTIMIZERPICKER(1, gsl_multimin_fdfminimizer_conjugate_fr, "Fletcher-Reeves") ;
                OPTIMIZERPICKER(2, gsl_multimin_fdfminimizer_conjugate_pr, "Polak-Ribiere") ;
                OPTIMIZERPICKER(3, gsl_multimin_fdfminimizer_steepest_descent, "steepest descent") ;
                OPTIMIZERPICKER(4, gsl_multimin_fdfminimizer_vector_bfgs, "Broyden-Fletcher-Goldfarb-Shanno, version 1") ;
        default:
                        if (out)
                                *out << "ERROR: Did not recognize optimizer choice: " << s->optimizerType << endl ;
                OPTIMIZERPICKER(0, gsl_multimin_fdfminimizer_vector_bfgs2, "Broyden-Fletcher-Goldfarb-Shanno, version 2") ;
        } ;
        gsl_multimin_function_fdf optFunction ;
        optFunction.n = 3 * Cluster->size() ;
        optFunction.f = &gslEnergy ;
        optFunction.df = &gslGradient ;
        optFunction.fdf = &gslBoth ;
        optFunction.params = Cluster ;
        gsl_vector* gslCoords = Cluster->gslCoordinates() ;
        gsl_multimin_fdfminimizer_set(minimizer,
                                      &optFunction,
                                      gslCoords,
                                      1e-3,
                                      0.1) ;
        int fieldWidth = 0 ;
        QTextStream::FieldAlignment alignment = QTextStream::AlignLeft ;
        if(out)
        {
                fieldWidth = out->fieldWidth() ;
                alignment = out->fieldAlignment() ;
                out->setFieldWidth(15) ;
                out->setFieldAlignment(QTextStream::AlignRight) ;
                *out << "Step" << "Gradient" << "Energy" << "Change" << "Max step" << endl
                     << "---" << "---"
                     << (energy = Cluster->energy())
                     << "---" << endl ;
        }
        for(steps = 0 ; steps < s->maxSteps ; ++steps)
        {
                gsl_vector *oldCoords = Cluster->gslCoordinates() ;
                int status = gsl_multimin_fdfminimizer_iterate(minimizer) ;
                double change = energy - minimizer->f ;
                double norm = gsl_blas_dnrm2(minimizer->gradient) ;
                double maxStep = qMax(gsl_vector_max(minimizer->dx),
                                      -gsl_vector_min(minimizer->dx)) ;
                if(out)
                        *out << steps << norm << minimizer->f << change << maxStep << endl ;
                if(status)
                {
                        if(out)
                                *out << "GSL optimizer quit with status: "
                                     << status
                                     << endl ;
                        if (Cluster->energy() < s->lowerLimit)
                                Cluster->setCoordinates(oldCoords);
                        delete oldCoords ;
                        break ;
                }
                if(maxStep > s->maxStep)
                {
                        if(out)
                                *out << "Step too large, scaling back." << endl ;
                        gsl_vector_memcpy(gslCoords, oldCoords) ;
                        gsl_vector_scale(minimizer->dx, s->maxStep / maxStep) ;
                        gsl_vector_add(gslCoords, minimizer->dx) ;
                        gsl_multimin_fdfminimizer_set(minimizer,
                                                      &optFunction,
                                                      gslCoords,
                                                      1e-3,
```

```
                                                        0.1) ;
                }
                delete oldCoords ;
                if(fabs(change) < s->energyThreshold
                                && GSL_SUCCESS == gsl_multimin_test_gradient(minimizer->gradient, s►
                                    ->gradientThreshold))
                {
                        if(out) *out << "Energy␣and␣gradient␣converged." << endl ;
                        break ;
                }
                energy = minimizer->f ;
        }
        gsl_multimin_fdfminimizer_free(minimizer) ;
        gsl_vector_free(gslCoords) ;
        if(out)
        {
                out->setFieldWidth(fieldWidth) ;
                out->setFieldAlignment(alignment);
        }
}
void gslOptimizer::run()
{
        runOptimization(Cluster,
                        settings,
                        i,
                        0);
        energy = Cluster->energy() ;
}
QVector<double> gslOptimizer::coordinates()
{
        return Cluster->coordinates() ;
}
gslOptimizer::~gslOptimizer()
{
        delete Cluster ;
}
double gslOptimizer::gslEnergy(const gsl_vector* coords, void* clusterPointer)
{
        ((cluster*) clusterPointer)->setCoordinates(coords) ;
        return ((cluster*) clusterPointer)->energy() ;
}
void gslOptimizer::gslGradient(const gsl_vector* coords, void* clusterPointer, gsl_vector* ►
    derivatives)
{
        ((cluster*) clusterPointer)->setCoordinates(coords) ;
        QVector<double> gradient = ((cluster*) clusterPointer)->force() ;
        for(int i = 0 ; i < gradient.size() ; i++)
                (derivatives->data) [i] = -gradient[i] ;
}
void gslOptimizer::gslBoth(const gsl_vector* coords, void* clusterPointer, double* envalue, ►
    gsl_vector* derivatives)
{
        *envalue = gslEnergy(coords, clusterPointer) ;
        gslGradient(coords, clusterPointer, derivatives) ;
}
```

src/harmonic.h

```
#ifndef HARMONIC_H
#define HARMONIC_H
#include "energyfunction.h"
namespace Harmonic
{
        enum Type { bond = 2 , angle = 3 , dieder = 4 , inversion = 5 } ;
}
class harmContribution : public energyContribution
{
private:
        double r, k;
        double(*value)(const QVector<atom*>&, const double&) ;
        QVector<double>* (*bMatrix)(const QVector<atom*>&) ;
public:
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        QVector<QVector<double> >* laplace(const QVector<atom*>&, double step, bool refresh = false) ;
```

```
        void printDescription() const ;
        harmContribution(Harmonic::Type type = Harmonic::bond) ;
        double energy(const QVector<atom*>&) ;
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
};
class harmonic : public energyFunction
{
private :
        Harmonic::Type id ;
protected :
        energyContribution* newInteraction() ;
        void doSort(QStringList&) ;
public:
        void printDescription() const ;
        harmonic(Harmonic::Type type = Harmonic::bond) ;
        harmonic* create() const { return new harmonic(id) ;}
        int numAtoms() ;
};
#endif
```

src/harmonic.cpp

```
#include "harmonic.h"
#include <math.h>
#include <QStringList>
#include <QTextStream>
#include "atom.h"
harmContribution::harmContribution(Harmonic::Type type)
{
        switch(type)
        {
                case Harmonic::bond :
                        value = deviationBond ;
                        bMatrix = bMatrixBond ;
                        break ;
                case Harmonic::angle :
                        value = deviationAngle ;
                        bMatrix = bMatrixAngle ;
                        break ;
                case Harmonic::dieder :
                        value = deviationDieder ;
                        bMatrix = bMatrixDieder ;
                        break ;
                case Harmonic::inversion:
                        value = deviationInversion ;
                        bMatrix = bMatrixInversion ;
                        break ;
                default :
                        value = deviationBond ;
                        bMatrix = bMatrixBond ;
        }
}
void harmContribution::setParameters(const QVector<double>& list)
{
        if(list.size() < 2) return ;
        r = list[0];
        k = list[1] ;
}
QVector<double> harmContribution::parameters() const
{ return (QVector<double>() << r << k) ;}
QVector<double>* harmContribution::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        QVector<double>* derivatives = bMatrix(atoms);
        double pre = value(atoms, r) * k ;
        for(int i = 0 ; i < 3 * atoms.size() ; i++)
                (*derivatives) [i] *= -pre ;
        return derivatives ;
}
double harmContribution::energy(const QVector<atom*>& atoms)
{ return .5 * k * pow(value(atoms, r), 2) ; }
energyContribution* harmonic::newInteraction()
```

```
{ return new harmContribution(id) ; }
int harmonic::numAtoms()
{ return -qMin(4, (int) id) ; }
harmonic::harmonic(Harmonic::Type type)
        : id(type) { ; }
void harmContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Parameter:␣␣" ;
        if(value == deviationBond)
                cout << "␣Bond:␣␣r0=" << r << ",␣k=" << k << ".␣␣" << endl ;
        else if(value == deviationAngle)
                cout << "␣Angle:␣␣phi0=" << r / M_PI * 180. << ",␣k=" << k << ".␣␣" << endl ;
        else if(value == deviationDieder)
                cout << "␣Dieder:␣␣phi0=" << r / M_PI * 180. << ",␣k=" << k << ".␣␣" << endl ;
        else if(value == deviationInversion)
                cout << "␣Inversion:␣␣phi0=" << r / M_PI * 180. << ",␣k=" << k << ".␣␣" << endl ;
}
void harmonic::doSort(QStringList& list)
{ Q_UNUSED(list) }
void harmonic::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Harmonisches␣Potential␣für␣" ;
        if(id == Harmonic::bond)
                cout << "Bindungen.␣␣2␣Atome.␣␣Funktion:␣␣k/2*(r-r0)^2." ;
        else if(id == Harmonic::angle)
                cout << "Winkel.␣␣3␣Atome.␣␣Funktion:␣␣k/2*(phi-phi0)^2.";
        else if(id == Harmonic::dieder)
                cout << "Dieder.␣␣4␣Atome.␣␣Funktion:␣␣k/2*(phi-phi0)^2.";
        else if(id == Harmonic::inversion)
                cout << "improper␣dihedrals.␣␣4␣Atome.␣␣Funktion:␣␣k/2*(theta-theta0)^2.";
        cout << "␣␣Beitraege:" << endl ;
        foreach(energyContribution * pointer, interactions)
        pointer->printDescription() ;
}
QVector<QVector<double> >* harmContribution::laplace(const QVector<atom*>& atoms, double step, bool ►
    refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        QVector<double>* derivatives = bMatrix(atoms);
        QVector<QVector<double> >* laplacian = new QVector<QVector<double> > (atoms.size() * 3, ►
            QVector<double>()) ;
        for(int i = 0 ; i < atoms.size() * 3 ; i++)
        {
                for(int j = i ; j < atoms.size() * 3 ; j++)
                {
                        double value = (*derivatives) [i] * (*derivatives) [j] * k ;
                        (*laplacian) [i] << value ;
                        if(i != j)(*laplacian) [j] << value ;
                }
        }
        return laplacian ;
}
```

src/hfdb.h

```
#ifndef HFDB_H
#define HFDB_H
#include "energyfunction.h"
class hfdbContribution : public energyContribution
{
private:
        double f(double) ;
        double csum(double) ;
        double df(double) ;
        double dcsum(double) ;
        double A, alpha, c6, c8, c10, C6, C8, C10, betas, beta, D, epsilon, rm, sigma ;
public:
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        double energy(const QVector<atom*>&) ;
        void setParameters(const QVector<double>&) ;
        void printDescription() const ;
```

```
        QVector<double> parameters() const ;
};
class hfdb : public energyFunction
{
protected:
        energyContribution* newInteraction() ;
public:
        hfdb();
        int numAtoms() ;
        hfdb* create() const { return new hfdb() ;}
        ~hfdb();
};
#endif
```

src/hfdb.cpp

```
#include "hfdb.h"
#include <math.h>
#include <QStringList>
hfdb::hfdb()
        : energyFunction()
{
}
hfdb::~hfdb()
{
}
void hfdbContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "HFDB␣contribution␣for:␣␣";
        for(int i = 0 ; i < partner.size() ; i++)
                cout << partner[i] << "␣␣" ;
        cout << "Parameters:␣␣" ;
        const double* paras[14] ;
        paras[0] = & A ;
        paras[1] = & alpha ;
        paras[2] = & c6 ;
        paras[3] = & c8 ;
        paras[4] = & c10 ;
        paras[5] = & C6 ;
        paras[6] = & C8 ;
        paras[7] = & C10 ;
        paras[8] = & betas ;
        paras[9] = & beta ;
        paras[10] = & D ;
        paras[11] = & epsilon ;
        paras[12] = & rm ;
        paras[13] = & sigma ;
        for(int i = 0 ; i < 10 ; i++)
                cout << * (paras[i]) << "␣␣";
        cout << endl ;
}
QVector<double> hfdbContribution::parameters() const
{
        const double* paras[14] ;
        paras[0] = & A ;
        paras[1] = & alpha ;
        paras[2] = & c6 ;
        paras[3] = & c8 ;
        paras[4] = & c10 ;
        paras[5] = & C6 ;
        paras[6] = & C8 ;
        paras[7] = & C10 ;
        paras[8] = & betas ;
        paras[9] = & beta ;
        paras[10] = & D ;
        paras[11] = & epsilon ;
        paras[12] = & rm ;
        paras[13] = & sigma ;
        QVector<double> pars ;
        for(int i = 0 ; i < 14 ; i++)
                pars << * (paras[i]) ;
        return pars ;
}
```

```
void hfdbContribution::setParameters(const QVector<double>& list)
{
        double* paras[14] ;
        paras[0] = & A ;
        paras[1] = & alpha ;
        paras[2] = & c6 ;
        paras[3] = & c8 ;
        paras[4] = & c10 ;
        paras[5] = & C6 ;
        paras[6] = & C8 ;
        paras[7] = & C10 ;
        paras[8] = & betas ;
        paras[9] = & beta ;
        paras[10] = & D ;
        paras[11] = & epsilon ;
        paras[12] = & rm ;
        paras[13] = & sigma ;
        for(int i = 0 ; i < 14 ; i++)
                * (paras[i]) = (i < list.size() ? list[i] : 0.) ;
}
double hfdbContribution::f(double x)
{ return x < D ? exp(-pow(D / x - 1., 2)) : 1. ; }
double hfdbContribution::df(double x)
{ return x < D ? 2.* (pow(D, 2) / pow(x, 3) - D / pow(x, 2)) * exp(-pow(D / x - 1., 2)) : 0. ; }
double hfdbContribution::csum(double x)
{ return c6 / pow(x, 6) + c8 / pow(x, 8) + c10 / pow(x, 10) ; }
double hfdbContribution::dcsum(double x)
{ return -6.*c6 / pow(x, 7) - 8.*c8 / pow(x, 9) - 10.*c10 / pow(x, 11) ; }
QVector<double>* hfdbContribution::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        QVector<double>* derivatives = bMatrixBond(atoms);
        double r = * (atoms[0]) % * (atoms[1]) ;
        double x = r / rm ;
        double pre = epsilon / rm * (A * (2.*betas * x - alpha) * exp(-alpha * x + betas * pow(x, 2)►
            ) - df(x) * csum(x) - f(x) * dcsum(x));
        for(int i = 0 ; i < 3 * atoms.size() ; i++)
                (*derivatives) [i] *= -pre ;
        return derivatives ;
}
double hfdbContribution::energy(const QVector<atom*>& atoms)
{
        if(atoms.size() != 2)
                return 0. ;
        double r = * (atoms[0]) % * (atoms[1]) ;
        double x = r / rm ;
        return epsilon * (A * exp(-alpha * x + betas * pow(x, 2)) - f(x) * csum(x)) ;
}
energyContribution* hfdb::newInteraction()
{
        energyContribution* newContrib = new hfdbContribution ;
        return newContrib ;
}
int hfdb::numAtoms()
{
        return 2 ;
}
```

src/induction.h

```
#ifndef INDUCTION_H
#define INDUCTION_H
#include "commoninteraction.h"
class induction : public commonInteraction
{
private:
        QVector<QVector<QVector<double> > > Ta ;
        QVector<QVector<QVector<QVector<double> > > > Tab ;
public:
        double energy(const QVector< atom* >&);
        void printDescription() const;
        void setParameters(const QVector< double >&);
        QVector<double> parameters() const ;
```

```
        energyContribution* explicitFunction(QVector< atom* > arg1);
        induction* create() const { return new induction() ; }
};
#endif
```

src/induction.cpp

```
#include "induction.h"
#include <math.h>
#include "constants.h"
double induction::energy(const QVector< atom* >& atoms)
{
        int charge = 0;
        for(charge = 0 ; charge < atoms.size() ; ++charge)
                if(atoms[charge]->charge())
                        break ;
        if(charge == atoms.size()) return 0 ;
        double en = 0 ;
        for(int i = 0 ; i < atoms.size() ; ++i)
        {
                if(i == charge) continue ;
                coord toCharge = *atoms[charge] - *atoms[i] ;
                double tC = ~toCharge ;
                coord dipoles ;
                double quadrupoles = 0 ;
                for(int k = 0 ; k < atoms.size() ; ++k)
                {
                        if(k == charge || k == i) continue ;
                        coord r = *atoms[i] - *atoms[k] ;
                        double rr = ~r ;
                        dipoles += r * pow(rr, -8) ;
                        quadrupoles += pow(rr, -6) * (3 * pow(toCharge * r / rr / tC, 2) - 1) ;
                }
                en += atoms[i]->dispersionHyperPolarizability() * pow(tC, -3)
                      * (1.5 * (dipoles * toCharge) - 0.125 * quadrupoles) ;
        }
        return en * atoms[charge]->charge() * parser::Eh * parser::a0 ;
}
void induction::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Induktions-Wechselwirkung" << endl ;
}
void induction::setParameters(const QVector< double >&) {}
QVector<double> induction::parameters() const {return QVector<double>() ;}
energyContribution* induction::explicitFunction(QVector< atom* > atoms)
{
        for(int i = 0 ; i < atoms.size() ; i++)
                if(atoms[i]->dispersionHyperPolarizability())
                        return this ;
        return 0 ;
}
```

src/internalpair.h

```
#ifndef INTERNALPAIR_H
#define INTERNALPAIR_H
#include "energyfunction.h"
#include "harmonic.h"
#include <QPair>
class ipContribution : public energyContribution
{
private:
        double k ;
        double rA, rB ;
        double(*valueA)(const QVector<atom*>&, const double&) ;
        double(*valueB)(const QVector<atom*>&, const double&) ;
        QVector<double>* (*bMatrixA)(const QVector<atom*>&) ;
        QVector<double>* (*bMatrixB)(const QVector<atom*>&) ;
        QVector<atom*> setA, setB ;
        int separation ;
        void split(const QVector<atom*>&) ;
```

```
        void assignPointers(const Harmonic::Type&, double(*&)(const QVector<atom*>&, const double&),►
             QVector<double>* (*&)(const QVector<atom*>&)) ;
public:
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        QVector<QVector<double> >* laplace(const QVector<atom*>&, double step, bool refresh = false) ;
        void printDescription() const ;
        ipContribution(Harmonic::Type, Harmonic::Type) ;
        double energy(const QVector<atom*>&) ;
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
};
class internalPair : public energyFunction
{
private :
        Harmonic::Type idA, idB ;
protected :
        energyContribution* newInteraction() ;
        void doSort(QStringList&) ;
public:
        void printDescription() const ;
        internalPair(Harmonic::Type, Harmonic::Type) ;
        int numAtoms() ;
        internalPair* create() const { return new internalPair(idA, idB) ;}
};
#endif
```

src/internalpair.cpp

```
#include "internalpair.h"
#include <math.h>
#include <QStringList>
#include <QTextStream>
#include "atom.h"
void ipContribution::assignPointers(const Harmonic::Type& type, double(*& values)(const QVector<atom►
    *>&, const double&), QVector<double>* (*& bMatrices)(const QVector<atom*>&))
{
        switch(type)
        {
                case Harmonic::bond :
                        values = deviationBond ;
                        bMatrices = bMatrixBond ;
                        break ;
                case Harmonic::angle :
                        values = deviationAngle ;
                        bMatrices = bMatrixAngle ;
                        break ;
                case Harmonic::dieder :
                        values = deviationDieder ;
                        bMatrices = bMatrixDieder ;
                        break ;
                case Harmonic::inversion:
                        values = deviationInversion ;
                        bMatrices = bMatrixInversion ;
                        break ;
                default :
                        values = deviationBond ;
                        bMatrices = bMatrixBond ;
        }
}
ipContribution::ipContribution(Harmonic::Type A, Harmonic::Type B)
{
        assignPointers(A, valueA, bMatrixA) ;
        assignPointers(B, valueB, bMatrixB) ;
        separation = qMin(4, (int) A) ;
}
void ipContribution::split(const QVector<atom*>& atoms)
{
        setA.clear() ; setB.clear() ;
        for(int i = 0 ; i < separation ; i++) setA << atoms[i] ;
        for(int i = separation ; i < atoms.size() ; i++) setB << atoms[i] ;
}
void ipContribution::setParameters(const QVector<double>& list)
{
        if(list.size() < 3) return ;
```

```
        rA = list[0] ;
        rB = list[1] ;
        k = list[2] ;
}
QVector<double> ipContribution::parameters() const
{ return (QVector<double>() << rA << rB << k) ;}
QVector<double>* ipContribution::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        split(atoms) ;
        QVector<double>* forces = bMatrixA(setA);
        double pre = -fabs(valueB(setB, rB)) * k ;
        for(int i = 0 ; i < 3 * separation ; i++)
                (*forces) [i] *= pre ;
        QVector<double>* derivatives = bMatrixB(setB);
        pre = -fabs(valueA(setA, rA)) * k ;
        for(int i = 0 ; i < 3 * atoms.size() - 3 * separation ; i++)
                (*forces) << (*derivatives) [i] * pre ;
        delete derivatives ;
        return forces ;
}
double ipContribution::energy(const QVector<atom*>& atoms)
{
        split(atoms) ;
        return k * valueA(setA, rA) * valueB(setB, rB) ;
}
energyContribution* internalPair::newInteraction()
{ return new ipContribution(idA, idB) ; }
int internalPair::numAtoms()
{ return -qMin(4, (int) idA) - qMin(4, (int) idB) ; }
internalPair::internalPair(Harmonic::Type typeA, Harmonic::Type typeB)
        : idA(typeA), idB(typeB) { ; }
void ipContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Kraftkonstante:␣␣" << k << "␣␣Partner:␣␣";
        cout << ".␣Typ:␣␣" ;
        if(valueA == deviationBond) cout << "␣Bindung␣" ;
        if(valueA == deviationAngle) cout << "␣Winkel␣" ;
        if(valueA == deviationDieder) cout << "␣Dieder␣" ;
        if(valueA == deviationInversion) cout << "␣Inversion␣" ;
        cout << "␣und␣" ;
        if(valueB == deviationBond) cout << "␣Bindung␣" ;
        if(valueB == deviationAngle) cout << "␣Winkel␣" ;
        if(valueB == deviationDieder) cout << "␣Dieder␣" ;
        if(valueB == deviationInversion) cout << "␣Inversion␣" ;
        cout << endl ;
}
void internalPair::doSort(QStringList& list)
{ Q_UNUSED(list) }
void internalPair::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Gemischtes␣Potential␣für␣" ;
        foreach(energyContribution * pointer, interactions)
        pointer->printDescription() ;
}
QVector<QVector<double> >* ipContribution::laplace(const QVector<atom*>& atoms, double step, bool ►
    refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        split(atoms) ;
        QVector<double>* deviationsA = bMatrixA(setA) ;
        QVector<double>* deviationsB = bMatrixB(setB) ;
        QVector<QVector<double> >* laplacian = new QVector<QVector<double> > (3 * atoms.size(), ►
            QVector<double> (3 * atoms.size(), 0.)) ;
        for(int i = 0 ; i < separation * 3 ; i++)
        {
                for(int j = separation * 3 ; j < atoms.size() * 3 ; j++)
                {
                        double value = k * (*deviationsA) [i] * (*deviationsB) [j - separation * 3] ;
                        (*laplacian) [i][j] = value ;
```

```
                        (*laplacian) [j][i] = value ;
                }
        }
        cout << "Listen:␣␣" << setA.size() << "␣␣" << setB.size() << "␣␣" << atoms.size() << "␣␣" <<►
             separation << endl ;
        for(int i = 0 ; i < deviationsA->size() ; i++)
                cout << "DevA:␣␣" << (*deviationsA) [i] << endl ;
        for(int i = 0 ; i < deviationsB->size() ; i++)
                cout << "DevB:␣␣" << (*deviationsB) [i] << endl ;
        for(int i = 0 ; i < atoms.size() * 3 ; i++)
        {
                cout << "Laplacian:␣␣" ;
                for(int j = 0 ; j < atoms.size() * 3 ; j++)
                        cout << (*laplacian) [i][j] << "␣␣" ;
                cout << endl ;
        }
        return laplacian ;
}
```

src/lennardjones.h

```
#ifndef LENNARDJONES_H
#define LENNARDJONES_H
#include <energyfunction.h>
#include <QHash>
#include <QString>
#include <QPair>
#include <QStringList>
class ljContribution : public energyContribution
{
        double epsilon, sigma ;
public:
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
} ;
class lennardjones : public energyFunction
{
protected:
        energyContribution* newInteraction() ;
        void doSort(QStringList&) ;
public:
        int numAtoms() ;
        void printDescription() const ;
        lennardjones* create() const { return new lennardjones() ;}
};
#endif
```

src/lennardjones.cpp

```
#include "lennardjones.h"
#include <math.h>
void ljContribution::setParameters(const QVector<double>& params)
{
        if(params.size() != 2) return ;
        epsilon = 4.*params[0] ;
        sigma = params[1] ;
}
QVector<double> ljContribution::parameters() const
{ return (QVector<double>() << epsilon / 4. << sigma) ;}
double ljContribution::energy(const QVector<atom*>& atoms)
{
        double x = sigma / (*atoms[0] % *atoms[1]) ;
        return epsilon * (pow(x, 12) - pow(x, 6)) ;
}
void ljContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << partner[0] << "␣␣" << partner[1] << "␣␣4*epsilon␣=␣" << epsilon << "␣␣sigma␣=␣" << ►
             sigma << endl ;
}
```

```
energyContribution* lennardjones::newInteraction()
{ return new ljContribution ; }
void lennardjones::doSort(QStringList& list)
{ qSort(list) ; }
int lennardjones::numAtoms()
{ return 2 ; }
void lennardjones::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Lennard-Jones-Potential.␣␣Beitraege:" << endl ;
        foreach(energyContribution * pointer, interactions)
        pointer->printDescription() ;
}
```

src/main.cpp

```
#include <QTextStream>
#include <QFile>
#include <QString>
#include <QStringList>
#include <QHash>
#include <QVariant>
#include "variablehash.h"
int main(int argc, char* argv[])
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        QFile inputFile ;
        if(argc > 1)
        {
                inputFile.setFileName(argv[1]) ;
                inputFile.open(QFile::ReadOnly) ;
        }
        if(!inputFile.isOpen())
                inputFile.open(stdin, QFile::ReadOnly) ;
        QTextStream inputStream(&inputFile) ;
        QString input = inputStream.readAll() ;
        inputStream.setDevice(NULL) ;
        inputFile.close() ;
        cout << "----------␣Input:␣␣␣␣␣␣␣----------" << endl ;
        cout << input ;
        cout << "----------␣end␣of␣input␣----------" << endl ;
        QStringList inputLines = input.split("\n", QString::SkipEmptyParts)
                                 .filter(QRegExp("^[^#]"))
                                 .join("\n")
                                 .split(QRegExp("[;\\n]"), QString::SkipEmptyParts) ;
        cout << "======================Initiating␣computation==================" << endl ;
        variableHash variables(inputLines) ;
        cout << "======================Computation␣done====================" << endl << "Variable␣►
            tree:" << endl ;
        cout << variables.currentContent() ;
        return 0;
}
```

src/mmsv.h

```
#ifndef MMSV_H
#define MMSV_H
#include "energyfunction.h"
class mmsvContribution : public energyContribution
{
private:
        double rm, epsilon, beta1, beta2, x1, x2, C6, C8, B4, B6 ;
        double sw(double) ;
        static double morse(double x, double beta) ;
        double w(double) ;
        double dsw(double) ;
        static double dmorse(double x, double beta , double rm) ;
        double dw(double) ;
        bool isAnion ;
public :
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        void setParameters(const QVector<double>&) ;
```

```
        QVector<double> parameters() const ;
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
        mmsvContribution(bool anion = true) ;
};
class mmsv : public energyFunction
{
private:
        bool isAnion ;
protected:
        energyContribution* newInteraction() ;
public:
        void printDescription() const ;
        mmsv(bool anion = true);
        int numAtoms() ;
        mmsv* create() const { return new mmsv() ;}
        ~mmsv();
};
#endif
```

src/mmsv.cpp

```
#include "mmsv.h"
#include <math.h>
#include <QStringList>
#include <QTextStream>
mmsvContribution::mmsvContribution(bool anion)
        : energyContribution(),
          isAnion(anion)
{}
void mmsvContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "MMSV␣contribution␣for:␣␣";
        for(int i = 0 ; i < partner.size() ; i++)
                cout << partner[i] << "␣␣" ;
        cout << "Parameters:␣␣" ;
        const double* paras[10] ;
        paras[0] = & epsilon ;
        paras[1] = & rm ;
        paras[2] = & beta1 ;
        paras[3] = & beta2 ;
        paras[4] = & x1 ;
        paras[5] = & x2 ;
        paras[6] = & C6 ;
        paras[7] = & C8 ;
        paras[8] = & B4 ;
        paras[9] = & B6 ;
        for(int i = 0 ; i < 10 ; i++)
                cout << * (paras[i]) << "␣␣";
        cout << endl ;
}
QVector<double> mmsvContribution::parameters() const
{
        const double* paras[10] ;
        paras[0] = & epsilon ;
        paras[1] = & rm ;
        paras[2] = & beta1 ;
        paras[3] = & beta2 ;
        paras[4] = & x1 ;
        paras[5] = & x2 ;
        paras[6] = & C6 ;
        paras[7] = & C8 ;
        paras[8] = & B4 ;
        paras[9] = & B6 ;
        QVector<double> pars ;
        for(int i = 0 ; i < 10 ; i++)
                pars << * (paras[i]) ;
        return pars ;
}
void mmsvContribution::setParameters(const QVector<double>& list)
{
        double* paras[10] ;
        paras[0] = & epsilon ;
```

```
        paras[1] = & rm ;
        paras[2] = & beta1 ;
        paras[3] = & beta2 ;
        paras[4] = & x1 ;
        paras[5] = & x2 ;
        paras[6] = & C6 ;
        paras[7] = & C8 ;
        paras[8] = & B4 ;
        paras[9] = & B6 ;
        for(int i = 0 ; i < 10 ; i++)
                * (paras[i]) = (i < list.size() ? list[i] : 0) ;
}
QVector<double>* mmsvContribution::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        Q_UNUSED(step)
        Q_UNUSED(refresh)
        QVector<double>* derivatives = bMatrixBond(atoms);
        double r = * (atoms[0]) % * (atoms[1]) ;
        double x = r / rm ;
        double pre = epsilon * (x <= 1. ? dmorse(x, beta1, rm) :
                                (x <= x1 ? dmorse(x, beta2, rm) :
                                 (x < x2 ? dsw(x) * (morse(x, beta2) - w(x)) + sw(x) * (dmorse(x, ►
                                     beta2, rm) - dw(x)) + dw(x) : dw(x))));
        for(int i = 0 ; i < 3 * atoms.size() ; i++)
                (*derivatives) [i] *= -pre ;
        return derivatives ;
}
double mmsvContribution::dmorse(double x, double beta, double rm)
{ return -2.*beta / rm * exp(2.*beta * (1. - x)) + 2.*beta / rm * exp(beta * (1. - x)) ; }
double mmsvContribution::dsw(double x)
{ return -.5 * M_PI / (x2 - x1) * sin(M_PI * (x - x1) / (x2 - x1)) / rm ;}
double mmsvContribution::dw(double x)
{return isAnion ? 4.*B4 / epsilon / pow(rm * x, 5) + 6.*B6 / epsilon / pow(rm * x, 7) : 6.*C6 / ►
     epsilon / pow(rm * x, 7) + 8.*C8 / epsilon / pow(rm * x, 9) ; }
double mmsvContribution::sw(double x)
{ return .5 * (cos(M_PI * (x - x1) / (x2 - x1)) + 1.) ; }
double mmsvContribution::morse(double x, double beta)
{ return exp(2 * beta * (1 - x)) - 2.*exp(beta * (1 - x)) ; }
double mmsvContribution::w(double x)
{ return isAnion ? -B4 / epsilon / pow(rm * x, 4) - B6 / epsilon / pow(rm * x, 6) : -C6 / epsilon / ►
     pow(rm * x, 6) - C8 / epsilon / pow(rm * x, 8) ; }
double mmsvContribution::energy(const QVector<atom*>& atoms)
{
        if(atoms.size() != 2)
                return 0. ;
        double r = * (atoms[0]) % * (atoms[1]) ;
        double x = r / rm ;
        double retval = epsilon * (x <= 1. ? morse(x, beta1) :
                                   (x <= x1 ? morse(x, beta2) :
                                    (x < x2 ? sw(x) * morse(x, beta2) + (1. - sw(x)) * w(x) : w(x))));
        return retval;
}
energyContribution* mmsv::newInteraction()
{
        return new mmsvContribution(isAnion) ;
}
mmsv::mmsv(bool anion)
        : energyFunction(),
          isAnion(anion)
{
}
mmsv::~mmsv()
{
}
int mmsv::numAtoms()
{
        return 2 ;
}
void mmsv::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "MMSV␣potential␣" ;
        foreach(energyContribution * pointer, interactions)
        {
                cout << "(";
```

```
                QStringList ps = pointer->partners() ;
                foreach(QString p, ps)
                cout << p << "," ;
                cout << ")␣" ;
        }
        cout << endl ;
}
```

src/nonaddinduction.h

```
#ifndef NONADDINDUCTION_H
#define NONADDINDUCTION_H
#include "commoninteraction.h"
#include <gsl/gsl_vector.h>
#include <gsl/gsl_matrix.h>
struct tValues
{
        double x, y, z, r ;
        double x2, x3, y2, y3, z2, z3, x4, y4, z4 ;
        double r3, r5, r7, r9, r11 ;
        double Tx, Ty, Tz ;
        double Txx, Tyy, Tzz, Txy, Txz, Tyz ;
        double Txxx, Tyyy, Tzzz, Txxy, Txxz, Txyy, Txzz, Tyyz, Tyzz, Txyz;
        double Txxxy, Txxxz, Txyyy, Txzzz, Tyyyyz, Tyzzzz, Txxxyy, Txxxzz, Txxyyy, Txxxyz, Txyyyz, Txyzz ;
        double Txxxxy, Txxxxz, Txyyyy, Txzzzz, Tyyyyz, Tyzzzz, Txxxyy, Txxxzz, Txxyyy, Txxzzz, ►
             Tyyyzz, Tyyzzz,
               Txxxyz, Txyyyz, Txyzzz, Txxyyz, Txxyzz, Txyyzz;
        double Txxxx, Tyyyy, Tzzzz, Txxxxx, Tyyyyy, Tzzzzz ;
        inline tValues() {}
        inline tValues(const coord& coords)
        {
                x = coords[0]; y = coords[1]; z = coords[2]; r = ~coords ;
                x2 = pow(x, 2); x3 = pow(x, 3); y2 = pow(y, 2); y3 = pow(y, 3); z2 = pow(z, 2); z3 =►
                     pow(z, 3) ; x4 = pow(x2, 2); y4 = pow(y2, 2); z4 = pow(z2, 2) ;
                r3 = pow(r, -3); r5 = -3.*pow(r, -5); r7 = 15.*pow(r, -7); r9 = -105.*pow(r, -9); ►
                     r11 = 945.*pow(r, -11) ;
                Tx = x * r3 ; Ty = y * r3 ; Tz = z * r3 ;
                Txx = x2 * r5 + r3 ; Tyy = y2 * r5 + r3 ; Tzz = z2 * r5 + r3 ;
                Txy = x * y * r5; Txz = x * z * r5; Tyz = y * z * r5 ;
                Txxx = x3 * r7 + 3.*x * r5; Tyyy = y3 * r7 + 3.*y * r5; Tzzz = z3 * r7 + 3.*z * r5;
                Txxy = x2 * y * r7 + y * r5; Txxz = x2 * z * r7 + z * r5; Txyy = x * y2 * r7 + x * ►
                     r5; Txzz = x * z2 * r7 + x * r5;
                Tyyz = y2 * z * r7 + z * r5; Tyzz = y * z2 * r7 + y * r5; Txyz = x * y * z * r7 ;
                Txxxx = (x4 * r9 + 6.*x2 * r7 + 3.*r5) ; Tyyyy = (y4 * r9 + 6.*y2 * r7 + 3.*r5) ; ►
                     Tzzzz = (z4 * r9 + 6.*z2 * r7 + 3.*r5) ;
                Txxxy = (x3 * y * r9 + 3.*x * y * r7); Txxxz = (x3 * z * r9 + 3.*x * z * r7); Txyyy ►
                     = (x * y3 * r9 + 3.*x * y * r7);
                Txzzz = (x * z3 * r9 + 3.*x * z * r7); Tyyyz = (y3 * z * r9 + 3.*y * z * r7); Tyzzz ►
                     = (y * z3 * r9 + 3.*y * z * r7);
                Txxyy = (x2 * y2 * r9 + (y2 + x2) * r7 + r5); Txxzz = (x2 * z2 * r9 + (x2 + z2) * r7►
                     + r5); Tyyzz = (y2 * z2 * r9 + (y2 + z2) * r7 + r5);
                Txxyz = (x2 * y * z * r9 + y * z * r7); Txyyz = (x * y2 * z * r9 + x * z * r7); ►
                     Txyzz = (x * y * z2 * r9 + x * y * r7) ;
                Txxxxy = (x4 * y * r11 + 6.*x2 * y * r9 + 3.*y * r7) ; Txxxxz = (x4 * z * r11 + 6.*►
                     x2 * z * r9 + 3.*z * r7) ; Txyyyy = (y4 * x * r11 + 6.*x * y2 * r9 + 3.*x * r7) ►
                     ;
                Txzzzz = (x * z4 * r11 + 6.*x * z2 * r9 + 3.*x * r7) ; Tyyyyz = (y4 * z * r11 + 6.*►
                     y2 * z * r9 + 3.*z * r7) ; Tyzzzz = (y * z4 * r11 + 6.*y * z2 * r9 + 3.*y * r7) ►
                     ;
                Txxxyy = (x3 * y2 * r11 + (3.*x * y2 + x3) * r9 + 3.*x * r7) ; Txxxzz = (x3 * z2 * ►
                     r11 + (3.*x * z2 + x3) * r9 + 3.*x * r7) ; Txxyyy = (y3 * x2 * r11 + (3.*y * x2 ►
                     + y3) * r9 + 3.*y * r7) ;
                Txxzzz = (x2 * z3 * r11 + (3.*z * x2 + z3) * r9 + 3.*z * r7) ; Tyyyzz = (y3 * z2 * ►
                     r11 + (3.*y * z2 + y3) * r9 + 3.*y * r7) ; Tyyzzz = (y2 * z3 * r11 + (3.*z * y2 ►
                     + z3) * r9 + 3.*z * r7) ;
                Txxxyz = (x3 * y * z * r11 + 3.*x * y * z * r9) ; Txyyyz = (x * y3 * z * r11 + 3.*x ►
                     * z * y * r9) ; Txyzzz = (x * y * z3 * r11 + 3.*x * y * z * r9) ;
                Txxyyz = (x2 * y2 * z * r11 + (y2 + x2) * z * r9 + z * r7) ; Txxyzz = (x2 * y * z2 *►
                      r11 + (x2 + z2) * y * r9 + y * r7) ; Txyyzz = (x * y2 * z2 * r11 + (y2 + z2) * ►
                     x * r9 + x * r7) ;
                Txxxxx = (x4 * x * r11 + 10.*x3 * r9 + 15.*x * r7) ; Tyyyyy = (y4 * y * r11 + 10.*y3►
                      * r9 + 15.*y * r7) ; Tzzzzz = (z4 * z * r11 + 10.*z3 * r9 + 15.*z * r7) ;
        }
};
```

```
class nonAddInduction : public commonInteraction
{
private:
        QVector<QVector<tValues*> > Ts ;
        QVector<gsl_vector*> pairMoments ;
        gsl_vector* totalMoments, *moments ;
        int dimension, halideIndex ;
        void fillQVector(const tValues*, double*) ;
        void fillQDerivatives(const tValues*, double*) ;
        void fillMatrix(const tValues*, double*) ;
        void fillDerivativeMatrix(const tValues*, double*, int coord) ;
        void copyPairInteraction(double* s, double* d) ;
        void copyPairVector(double* s, double* d) ;
        void outputVector(gsl_vector*) ;
        void outputMatrix(gsl_matrix*) ;
        void clear() ;
public:
        nonAddInduction();
        ~nonAddInduction();
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh = false) ;
        double energy(const QVector< atom* >& arg1);
        void printDescription() const;
        void setParameters(const QVector< double >& arg1) { Q_UNUSED(arg1) }
        QVector<double> parameters() const {return QVector<double>() ;}
        energyContribution* explicitFunction(QVector< atom* > arg1);
        nonAddInduction* create() const { return new nonAddInduction() ;}
};
#endif
```

src/nonaddinduction.cpp

```
#include "nonaddinduction.h"
#include "constants.h"
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_permutation.h>
#include <gsl/gsl_linalg.h>
#include <gsl/gsl_blas.h>
void nonAddInduction::fillQVector(const tValues* t, double* vec)
{
        vec[0] = t->Tx ; vec[1] = t->Ty ; vec[2] = t->Tz ;
        vec[3] = t->Txx; vec[4] = t->Tyy; vec[5] = t->Tzz;
        vec[6] = t->Txy; vec[7] = t->Txz; vec[8] = t->Tyz;
}
void nonAddInduction::fillQDerivatives(const tValues* t, double* row)
{
        row[0] = t->Txx ; row[1] = t->Txy ; row[2] = t->Txz ;
        row[3] = t->Txxx ; row[4] = t->Txyy ; row[5] = t->Txzz ;
        row[6] = t->Txxy ; row[7] = t->Txxz ; row[8] = t->Txyz ;
        row += dimension ;
        row[0] = t->Txy ; row[1] = t->Tyy ; row[2] = t->Tyz ;
        row[3] = t->Txxy ; row[4] = t->Tyyy ; row[5] = t->Tyzz ;
        row[6] = t->Txyy ; row[7] = t->Txyz ; row[8] = t->Tyyz ;
        row += dimension ;
        row[0] = t->Txz ; row[1] = t->Tyz ; row[2] = t->Tzz ;
        row[3] = t->Txxz ; row[4] = t->Tyyz ; row[5] = t->Tzzz ;
        row[6] = t->Txyz ; row[7] = t->Txzz ; row[8] = t->Tyzz ;
}
void nonAddInduction::fillMatrix(const tValues* t, double* matrix)
{
        matrix[0] = t->Txx ; matrix[1] = t->Txy ; matrix[2] = t->Txz ;
        matrix[3] = -t->Txxx ; matrix[4] = -t->Txyy ; matrix[5] = -t->Txzz ;
        matrix[6] = -t->Txxy ; matrix[7] = -t->Txxz; matrix[8] = -t->Txyz ;
        matrix += dimension;
        matrix[0] = t->Txy ; matrix[1] = t->Tyy; matrix[2] = t->Tyz ;
        matrix[3] = -t->Txxy ; matrix[4] = -t->Tyyy ; matrix[5] = -t->Tyzz ;
        matrix[6] = -t->Txyy ; matrix[7] = -t->Txyz; matrix[8] = -t->Tyyz;
        matrix += dimension ;
        matrix[0] = t->Txz ; matrix[1] = t->Tyz ; matrix[2] = t->Tzz ;
        matrix[3] = -t->Txxz ; matrix[4] = -t->Tyyz ; matrix[5] = -t->Tzzz ;
        matrix[6] = -t->Txyz ; matrix[7] = -t->Txzz; matrix[8] = -t->Tyzz;
        matrix += dimension ;
        matrix[0] = t->Txxx ; matrix[1] = t->Txxy ; matrix[2] = t->Txxz ;
        matrix[3] = -t->Txxxx ; matrix[4] = -t->Txxyy ; matrix[5] = -t->Txxzz ;
        matrix[6] = -t->Txxxy ; matrix[7] = -t->Txxxz ; matrix[8] = -t->Txxyz ;
```

```
        matrix += dimension ;
        matrix[0] = t->Txyy ; matrix[1] = t->Tyyy ; matrix[2] = t->Tyyz ;
        matrix[3] = -t->Txxyy ; matrix[4] = -t->Tyyyy ; matrix[5] = -t->Tyyzz ;
        matrix[6] = -t->Txyyy ; matrix[7] = -t->Txyyz ; matrix[8] = -t->Tyyyz ;
        matrix += dimension ;
        matrix[0] = t->Txzz ; matrix[1] = t->Tyzz ; matrix[2] = t->Tzzz ;
        matrix[3] = -t->Txxzz ; matrix[4] = -t->Tyyzz ; matrix[5] = -t->Tzzzz ;
        matrix[6] = -t->Txyzz ; matrix[7] = -t->Txzzz ; matrix[8] = -t->Tyzzz ;
        matrix += dimension ;
        matrix[0] = t->Txxy ; matrix[1] = t->Txyy ; matrix[2] = t->Txyz ;
        matrix[3] = -t->Txxxy ; matrix[4] = -t->Tyyyy ; matrix[5] = -t->Txyzz ;
        matrix[6] = -t->Txxyy ; matrix[7] = -t->Txxyz ; matrix[8] = -t->Txyyz ;
        matrix += dimension ;
        matrix[0] = t->Txxz ; matrix[1] = t->Txyz ; matrix[2] = t->Txzz ;
        matrix[3] = -t->Txxxz ; matrix[4] = -t->Tyyyz ; matrix[5] = -t->Tzzzz ;
        matrix[6] = -t->Txxyz ; matrix[7] = -t->Txxzz ; matrix[8] = -t->Txyzz ;
        matrix += dimension ;
        matrix[0] = t->Txyz ; matrix[1] = t->Tyyz ; matrix[2] = t->Tyzz ;
        matrix[3] = -t->Txxyz ; matrix[4] = -t->Tyyyz ; matrix[5] = -t->Tyzzz ;
        matrix[6] = -t->Txyyz ; matrix[7] = -t->Txyzz ; matrix[8] = -t->Tyyzz ;
}
void nonAddInduction::fillDerivativeMatrix(const tValues* t, double* matrix, int coor)
{
        if(coor == 0)
        {
                matrix[0] = t->Txxx ; matrix[1] = t->Txxy ; matrix[2] = t->Txxz ;
                matrix[3] = -t->Txxxx ; matrix[4] = -t->Txxyy ; matrix[5] = -t->Txxzz ;
                matrix[6] = -t->Txxxy ; matrix[7] = -t->Txxxz ; matrix[8] = -t->Txxyz ;
                matrix += dimension;
                matrix[0] = t->Txxy ; matrix[1] = t->Txyy ; matrix[2] = t->Txyz ;
                matrix[3] = -t->Txxxy ; matrix[4] = -t->Txyyy ; matrix[5] = -t->Txyzz ;
                matrix[6] = -t->Txxyy; matrix[7] = -t->Txxyz ; matrix[8] = -t->Txyyz ;
                matrix += dimension ;
                matrix[0] = t->Txxz ; matrix[1] = t->Txyz ; matrix[2] = t->Txzz ;
                matrix[3] = -t->Txxxz ; matrix[4] = -t->Txyyz ; matrix[5] = -t->Txzzz ;
                matrix[6] = -t->Txxyz ; matrix[7] = -t->Txxzz; matrix[8] = -t->Txyzz;
                matrix += dimension ;
                matrix[0] = t->Txxxx ; matrix[1] = t->Txxxy ; matrix[2] = t->Txxxz ;
                matrix[3] = -t->Txxxxx ; matrix[4] = -t->Txxxyy ; matrix[5] = -t->Txxxzz ;
                matrix[6] = -t->Txxxxy ; matrix[7] = -t->Txxxxz ; matrix[8] = -t->Txxxyz ;
                matrix += dimension ;
                matrix[0] = t->Txxyy ; matrix[1] = t->Txyyy ; matrix[2] = t->Txyyz ;
                matrix[3] = -t->Txxxyy; matrix[4] = -t->Txyyyy ; matrix[5] = -t->Txyyzz ;
                matrix[6] = -t->Txxyyy ; matrix[7] = -t->Txxyyz ; matrix[8] = -t->Txyyyz ;
                matrix += dimension ;
                matrix[0] = t->Txxzz ; matrix[1] = t->Txyzz ; matrix[2] = t->Txzzz ;
                matrix[3] = -t->Txxxzz ; matrix[4] = -t->Txyyzz ; matrix[5] = -t->Txzzzz ;
                matrix[6] = -t->Txxyzz ; matrix[7] = -t->Txxzzz ; matrix[8] = -t->Txyzzz ;
                matrix += dimension ;
                matrix[0] = t->Txxxy ; matrix[1] = t->Txxyy ; matrix[2] = t->Txxyz ;
                matrix[3] = -t->Txxxxy ; matrix[4] = -t->Txxyyy ; matrix[5] = -t->Txxyzz ;
                matrix[6] = -t->Txxxyy ; matrix[7] = -t->Txxxyz ; matrix[8] = -t->Txxyyz ;
                matrix += dimension ;
                matrix[0] = t->Txxxz ; matrix[1] = t->Txxyz ; matrix[2] = t->Txxzz ;
                matrix[3] = -t->Txxxxz ; matrix[4] = -t->Txxyyz ; matrix[5] = -t->Txxzzz ;
                matrix[6] = -t->Txxxyz ; matrix[7] = -t->Txxxzz ; matrix[8] = -t->Txxyzz ;
                matrix += dimension ;
                matrix[0] = t->Txxyz ; matrix[1] = t->Txyyz ; matrix[2] = t->Txyzz ;
                matrix[3] = -t->Txxxyz ; matrix[4] = -t->Txyyyz ; matrix[5] = -t->Txyzzz ;
                matrix[6] = -t->Txxyyz ; matrix[7] = -t->Txxyzz ; matrix[8] = -t->Txyyzz ;
        }
        else if(coor == 1)
        {
                matrix[0] = t->Txxy ; matrix[1] = t->Txyy ; matrix[2] = t->Txyz ;
                matrix[3] = -t->Txxxy ; matrix[4] = -t->Txyyy ; matrix[5] = -t->Txyzz ;
                matrix[6] = -t->Txxyy ; matrix[7] = -t->Txxyz ; matrix[8] = -t->Txyyz ;
                matrix += dimension;
                matrix[0] = t->Txyy ; matrix[1] = t->Tyyy ; matrix[2] = t->Tyyz ;
                matrix[3] = -t->Txxyy ; matrix[4] = -t->Tyyyy ; matrix[5] = -t->Tyyzz ;
                matrix[6] = -t->Txyyy; matrix[7] = -t->Txyyz ; matrix[8] = -t->Tyyyz ;
                matrix += dimension ;
                matrix[0] = t->Txyz ; matrix[1] = t->Tyyz ; matrix[2] = t->Tyzz ;
                matrix[3] = -t->Txxyz ; matrix[4] = -t->Tyyyz ; matrix[5] = -t->Tyzzz ;
                matrix[6] = -t->Txyyz ; matrix[7] = -t->Txyzz; matrix[8] = -t->Tyyzz;
                matrix += dimension ;
```

```
                matrix[0] = t->Txxxy ; matrix[1] = t->Txxyy ; matrix[2] = t->Txxyz ;
                matrix[3] = -t->Txxxxy ; matrix[4] = -t->Txxyyy ; matrix[5] = -t->Txxyzz ;
                matrix[6] = -t->Txxxyy ; matrix[7] = -t->Txxxyz ; matrix[8] = -t->Txxyyz ;
                matrix += dimension ;
                matrix[0] = t->Txyyy ; matrix[1] = t->Tyyyy ; matrix[2] = t->Tyyyz ;
                matrix[3] = -t->Txyyyy ; matrix[4] = -t->Tyyyyy ; matrix[5] = -t->Tyyyzz ;
                matrix[6] = -t->Txyyyy ; matrix[7] = -t->Txyyyz ; matrix[8] = -t->Tyyyyz ;
                matrix += dimension ;
                matrix[0] = t->Txyzz ; matrix[1] = t->Tyyzz ; matrix[2] = t->Tyzzz ;
                matrix[3] = -t->Txxyzz ; matrix[4] = -t->Tyyyzz ; matrix[5] = -t->Tyzzzz ;
                matrix[6] = -t->Txyyzz ; matrix[7] = -t->Txyzzz ; matrix[8] = -t->Tyyzzz ;
                matrix += dimension ;
                matrix[0] = t->Txxyy ; matrix[1] = t->Txyyy ; matrix[2] = t->Txyyz ;
                matrix[3] = -t->Txxxyy ; matrix[4] = -t->Txyyyy ; matrix[5] = -t->Txyyzz ;
                matrix[6] = -t->Txxyyy ; matrix[7] = -t->Txxyyz ; matrix[8] = -t->Txyyyz ;
                matrix += dimension ;
                matrix[0] = t->Txxyz ; matrix[1] = t->Txyyz ; matrix[2] = t->Txyzz ;
                matrix[3] = -t->Txxxyz ; matrix[4] = -t->Txyyyz ; matrix[5] = -t->Txyzzz ;
                matrix[6] = -t->Txxyyz ; matrix[7] = -t->Txxyzz ; matrix[8] = -t->Txyyzz ;
                matrix += dimension ;
                matrix[0] = t->Txyyz ; matrix[1] = t->Tyyyz ; matrix[2] = t->Tyyzz ;
                matrix[3] = -t->Txxyyz ; matrix[4] = -t->Tyyyyz ; matrix[5] = -t->Tyyzzz ;
                matrix[6] = -t->Txyyyz ; matrix[7] = -t->Txyyzz ; matrix[8] = -t->Tyyyzz ;
        }
        else if(coor == 2)
        {
                matrix[0] = t->Txxz ; matrix[1] = t->Txyz ; matrix[2] = t->Txzz ;
                matrix[3] = -t->Txxxz ; matrix[4] = -t->Txyyz ; matrix[5] = -t->Txzzz ;
                matrix[6] = -t->Txxyz ; matrix[7] = -t->Txxzz ; matrix[8] = -t->Txyzz ;
                matrix += dimension;
                matrix[0] = t->Txyz ; matrix[1] = t->Tyyz ; matrix[2] = t->Tyzz ;
                matrix[3] = -t->Txxyz ; matrix[4] = -t->Tyyyz ; matrix[5] = -t->Tyzzz ;
                matrix[6] = -t->Txyyz; matrix[7] = -t->Txyzz ; matrix[8] = -t->Tyyzz ;
                matrix += dimension ;
                matrix[0] = t->Txzz ; matrix[1] = t->Tyzz ; matrix[2] = t->Tzzz ;
                matrix[3] = -t->Txxzz ; matrix[4] = -t->Tyyzz ; matrix[5] = -t->Tzzzz ;
                matrix[6] = -t->Txyzz ; matrix[7] = -t->Txzzz; matrix[8] = -t->Tyzzz;
                matrix += dimension ;
                matrix[0] = t->Txxxz ; matrix[1] = t->Txxyz ; matrix[2] = t->Txxzz ;
                matrix[3] = -t->Txxxxz ; matrix[4] = -t->Txxyyz ; matrix[5] = -t->Txxzzz ;
                matrix[6] = -t->Txxxyz ; matrix[7] = -t->Txxxzz ; matrix[8] = -t->Txxyzz ;
                matrix += dimension ;
                matrix[0] = t->Txyyz ; matrix[1] = t->Tyyyz ; matrix[2] = t->Tyyzz ;
                matrix[3] = -t->Txxyyz; matrix[4] = -t->Tyyyyz ; matrix[5] = -t->Tyyzzz ;
                matrix[6] = -t->Txyyyz ; matrix[7] = -t->Txyyzz ; matrix[8] = -t->Tyyyzz ;
                matrix += dimension ;
                matrix[0] = t->Txzzz ; matrix[1] = t->Tyzzz ; matrix[2] = t->Tzzzz ;
                matrix[3] = -t->Txxzzz ; matrix[4] = -t->Tyyzzz ; matrix[5] = -t->Tzzzzz ;
                matrix[6] = -t->Txyzzz ; matrix[7] = -t->Txzzzz ; matrix[8] = -t->Tyzzzz ;
                matrix += dimension ;
                matrix[0] = t->Txxyz ; matrix[1] = t->Txyyz ; matrix[2] = t->Txyzz ;
                matrix[3] = -t->Txxxyz ; matrix[4] = -t->Txyyyz ; matrix[5] = -t->Txyzzz ;
                matrix[6] = -t->Txxyyz ; matrix[7] = -t->Txxyzz ; matrix[8] = -t->Txyyzz ;
                matrix += dimension ;
                matrix[0] = t->Txxzz ; matrix[1] = t->Txyzz ; matrix[2] = t->Txzzz ;
                matrix[3] = -t->Txxxzz ; matrix[4] = -t->Txyyzz ; matrix[5] = -t->Txzzzz ;
                matrix[6] = -t->Txxyzz ; matrix[7] = -t->Txxzzz ; matrix[8] = -t->Txyzzz ;
                matrix += dimension ;
                matrix[0] = t->Txyzz ; matrix[1] = t->Tyyzz ; matrix[2] = t->Tyzzz ;
                matrix[3] = -t->Txxyzz ; matrix[4] = -t->Tyyyzz ; matrix[5] = -t->Tyzzzz ;
                matrix[6] = -t->Txyyzz ; matrix[7] = -t->Txyzzz ; matrix[8] = -t->Tyyzzz ;
        }
}
void nonAddInduction::copyPairInteraction(double* source, double* destination)
{
        for(int i = 0 ; i < 9 ; i++)
        {
                for(int j = 0 ; j < 9 ; j++)
                        destination[j] = source[j] ;
                source += dimension ;
                destination += 18 ;
        }
}
inline void nonAddInduction::copyPairVector(double* source, double* destination)
{
```

```
194         for(int i = 0 ; i < 9 ; i++)
                destination[i] = source[i] ;
    }
    void nonAddInduction::outputVector(gsl_vector* vec)
    {
199         QTextStream cout(stdout, QIODevice::WriteOnly) ;
            cout << "Vector:\t" ;
            for(size_t i = 0 ; i < vec->size ; i++)
                cout << vec->data[i] << "\t" ;
            cout << endl ;
204 }
    void nonAddInduction::outputMatrix(gsl_matrix* matrix)
    {
            QTextStream cout(stdout, QIODevice::WriteOnly) ;
            for(size_t i = 0 ; i < matrix->size1 ; i++)
209         {
                cout << i << ":\t" ;
                for(size_t j = 0 ; j < matrix->size2 ; j++)
                    cout << matrix->data[i * matrix->size2 + j] << "\t" ;
                cout << endl ;
214         }
            cout << endl ;
    }
    nonAddInduction::nonAddInduction()
            : commonInteraction(),
219           totalMoments(NULL),
              moments(NULL),
              dimension(0),
              halideIndex(-1)
    {}
224 nonAddInduction::~nonAddInduction()
    {
            clear() ;
            interactions.clear() ;
    }
229 void nonAddInduction::clear()
    {
            if(totalMoments) gsl_vector_free(totalMoments) ;
            if(moments) gsl_vector_free(moments) ;
            totalMoments = 0 ;
234         moments = 0 ;
            foreach(gsl_vector * pairMoment, pairMoments)
            gsl_vector_free(pairMoment) ;
            pairMoments.clear() ;
            dimension = 0 ;
239         foreach(QVector<tValues*> vector, Ts)
            {
                foreach(tValues * pointer, vector)
                delete pointer ;
                vector.clear() ;
244         }
            Ts.clear() ;
            dimension = 0 ;
    }
    void nonAddInduction::printDescription() const
249 {
            QTextStream cout(stdout, QIODevice::WriteOnly) ;
            cout << "Non-additive␣induction␣function." << endl ;
    }
    energyContribution* nonAddInduction::explicitFunction(QVector< atom* > atoms)
254 {
            if(dimension) clear() ;
            dimension = 9 * atoms.size() ;
            totalMoments = gsl_vector_alloc(dimension) ;
            moments = gsl_vector_alloc(dimension) ;
259         pairMoments.resize(atoms.size() - 1) ;
            for(int i = 0 ; i < pairMoments.size() ; i++)
                pairMoments[i] = gsl_vector_alloc(18) ;
            Ts.resize(atoms.size() - 1) ;
            for(int i = 0 ; i < Ts.size() ; i++)
264         {
                Ts[i].resize(Ts.size() - i) ;
                for(int j = 0 ; j < Ts[i].size() ; j++)
                    Ts[i][j] = new tValues ;
            }
```

```
        halideIndex = 0 ;
        for(halideIndex = 0 ; halideIndex < atoms.size() && !atoms[halideIndex]->charge() ; ►
            halideIndex++) ;
        return this ;
}
double nonAddInduction::energy(const QVector<atom*>& atoms)
{
        if(atoms.size() < 3) return 0 ;
        if(atoms.size() != dimension / 9) explicitFunction(atoms) ;
        if(halideIndex == atoms.size()) return 0. ;
        for(int i = 0 ; i < atoms.size() - 1 ; i++)
                for(int j = 0 ; i + j + 1 < atoms.size() ; j++)
                        * (Ts[i][j]) = *atoms[i + j + 1] - *atoms[i] ;
        gsl_matrix* interactionMatrix = gsl_matrix_alloc(dimension, dimension) ;
        for(int i = 0 ; i < atoms.size() - 1 ; i++)
                for(int j = i + 1 ; j < atoms.size() ; j++)
                        fillMatrix(Ts[i][j - i - 1], &interactionMatrix->data[(i + j * dimension) * ►
                            9]) ;
        double* part ;
        for(int i = 0 ; i < atoms.size() ; i++)
        {
                double alpha = 1. / atoms[i]->dipolePolarizability(),
                       C = 3. / atoms[i]->quadrupolePolarizability() ;
                part = &interactionMatrix->data[i * 9 * (dimension + 1)] ;
                for(int j = 0 ; j < 9 ; j++)
                        for(int k = 0 ; k < 9 ; k++)
                                part[j * dimension + k] = 0. ;
                for(int j = 0 ; j < 3 ; j++)
                        part[j * (dimension + 1)] = alpha ;
                for(int j = 3 ; j < 6 ; j++)
                        part[j * (dimension + 1)] = C ;
                C /= 2. ;
                for(int j = 6 ; j < 9 ; j++)
                        part[j * (dimension + 1)] = C ;
        }
        part = interactionMatrix->data ;
        for(int i = 0 ; i < dimension ; i++)
                for(int j = 0 ; j < i ; j++)
                        part[i + j * dimension] = part[j + i * dimension] ;
        gsl_vector* chargeVector = gsl_vector_alloc(dimension) ;
        part = chargeVector->data ;
        for(int i = 0 ; i < atoms.size() ; i++)
        {
                if(i == halideIndex)
                {
                        for(int j = 0 ; j < 9 ; j++)
                                part[halideIndex * 9 + j] = 0 ;
                        continue ;
                }
                fillQVector(Ts[qMin(halideIndex, i)][qMax(halideIndex, i) - qMin(halideIndex, i) - ►
                    1], &part[i * 9]) ;
        }
        gsl_matrix* pairMatrix = gsl_matrix_alloc(18, 18) ;
        gsl_permutation* pairPermutation = gsl_permutation_alloc(18) ;
        gsl_vector* allPairMoments = gsl_vector_alloc(dimension) ;
        int s ;
        for(int i = 0 ; i < atoms.size() ; i++)
        {
                if(i == halideIndex)
                {
                        for(int j = 0 ; j < 9 ; j++)
                                allPairMoments->data[j + halideIndex * 9] = 0 ;
                        continue ;
                }
                int j = i - (i > halideIndex) ;
                copyPairInteraction(&interactionMatrix->data[halideIndex * 9 * (dimension + 1)], ►
                    pairMatrix->data) ;
                copyPairInteraction(&interactionMatrix->data[9 * (halideIndex * dimension + i)], &►
                    pairMatrix->data[9]);
                copyPairInteraction(&interactionMatrix->data[9 * (halideIndex + i * dimension)], &►
                    pairMatrix->data[162]) ;
                copyPairInteraction(&interactionMatrix->data[i * 9 * (dimension + 1)], &pairMatrix->►
                    data[171]) ;
                copyPairVector(&chargeVector->data[halideIndex * 9], pairMoments[j]->data) ;
                copyPairVector(&chargeVector->data[i * 9], &pairMoments[j]->data[9]) ;
```

```
                gsl_linalg_LU_decomp(pairMatrix, pairPermutation, &s) ;
                gsl_linalg_LU_svx(pairMatrix, pairPermutation, pairMoments[j]) ;
339             part = pairMoments[j]->data ;
                for(int k = 0 ; k < 9 ; k++)
                        allPairMoments->data[k + i * 9] = part[9 + k];
        }
        gsl_permutation_free(pairPermutation) ;
344     gsl_set_error_handler_off() ;
#ifdef CHOLESKYINNONADDITIVE
        if(gsl_linalg_cholesky_decomp(interactionMatrix) != GSL_EDOM)
                gsl_linalg_cholesky_solve(interactionMatrix, chargeVector, moments) ;
        else
349     {
                gsl_matrix_free(interactionMatrix) ;
                gsl_vector_free(chargeVector) ;
                gsl_vector_free(allPairMoments) ;
                gsl_matrix_free(pairMatrix) ;
354             clear() ;
                return INFINITY ;
        }
#else
        gsl_permutation* permutation = gsl_permutation_alloc(interactionMatrix->size1) ;
359     gsl_linalg_LU_decomp(interactionMatrix, permutation, &s) ;
        gsl_linalg_LU_solve(interactionMatrix, permutation, chargeVector, moments) ;
        gsl_permutation_free(permutation) ;
#endif
        gsl_matrix_free(interactionMatrix) ;
364     gsl_vector_memcpy(totalMoments, moments) ;
        gsl_vector_sub(totalMoments, allPairMoments) ;
        double retVal ;
        gsl_blas_ddot(totalMoments, chargeVector, &retVal) ;
        gsl_vector_free(chargeVector) ;
369     gsl_vector_free(allPairMoments) ;
        gsl_matrix_free(pairMatrix) ;
        return retVal / (-2.) * parser::Eh * parser::a0 ;
}
QVector<double>* nonAddInduction::force(const QVector<atom*>& atoms, double step, bool refresh)
374 {
        Q_UNUSED(step)
        if(atoms.size() < 3) return 0 ;
        if(atoms.size() != dimension / 9 || refresh)
        {
379             if(INFINITY == energy(atoms))
                        return 0 ;
        }
        if(halideIndex == atoms.size()) return new QVector<double> (atoms.size() * 3, 0.) ;
        gsl_matrix* qDerivatives = gsl_matrix_calloc(3 * atoms.size(), dimension) ;
384     for(int i = 0 ; i < atoms.size() ; i++)
        {
                if(i == halideIndex) continue ;
                fillQDerivatives(Ts[qMin(halideIndex, i)][qMax(halideIndex, i) - qMin(halideIndex, i►
                    ) - 1], &qDerivatives->data[i * (9 + 3 * dimension)]) ;
        }
389     gsl_vector* derivatives = gsl_vector_alloc(3 * atoms.size()) ;
        gsl_blas_dgemv(CblasNoTrans, 1., qDerivatives, totalMoments, 0., derivatives) ;
        gsl_vector* muDerivative = gsl_vector_alloc(dimension),
                  *muPairDerivative = gsl_vector_alloc(18) ;
        for(int i = 0 ; i < atoms.size() ; i++)
394     {
                if(i == halideIndex) continue ;
                for(int k = 0 ; k < 3 ; k++)
                {
                        gsl_matrix* globalInteractions = gsl_matrix_calloc(dimension, dimension) ;
399                     for(int j = 0 ; j < atoms.size() ; j++)
                        {
                                if(j == i) continue ;
                                fillDerivativeMatrix(Ts[qMin(j, i)][qMax(j, i) - qMin(j, i) - 1], &►
                                    globalInteractions->data[(qMin(i, j) + qMax(i, j) *dimension) * ►
                                    9], k) ;
                                if(j > i)
404                             {
                                        double* toChange = &globalInteractions->data[(qMin(i, j) + ►
                                            qMax(i, j) * dimension) * 9] ;
                                        for(int k = 0 ; k < 9 ; k++)
                                        {
```

```
                                        for(int l = 0 ; l < 9 ; l++)
                                                toChange[l] *= -1. ;
                                        toChange += dimension ;
                                }
                        }
                        double* source = &globalInteractions->data[(qMin(i, j) + qMax(i, j) ►
                             * dimension) * 9],
                                *destination = &globalInteractions->data[(qMax(i, j) + qMin(►
                                     i, j) * dimension) * 9] ;
                        for(int k = 0 ; k < 3 ; k++)
                        {
                                for(int l = 0 ; l < 3 ; l++)
                                        destination[l] = source[l] ;
                                for(int l = 3 ; l < 9 ; l++)
                                        destination[l] = -source[l] ;
                                source += dimension ;
                                destination += dimension ;
                        }
                        for(int k = 0 ; k < 6 ; k++)
                        {
                                for(int l = 0 ; l < 3 ; l++)
                                        destination[l] = -source[l] ;
                                for(int l = 3 ; l < 9 ; l++)
                                        destination[l] = source[l] ;
                                source += dimension ;
                                destination += dimension ;
                        }
                }
                gsl_matrix* pairInteraction = gsl_matrix_calloc(18, 18) ;
                copyPairInteraction(&globalInteractions->data[i * 9 + 9 * dimension * ►
                     halideIndex], &pairInteraction->data[9]) ;
                copyPairInteraction(&globalInteractions->data[halideIndex * 9 + 9 * ►
                     dimension * i], &pairInteraction->data[162]) ;
                gsl_blas_dgemv(CblasNoTrans, 1., pairInteraction, pairMoments[i - (i > ►
                     halideIndex)], 0., muPairDerivative) ;
                gsl_blas_dgemv(CblasNoTrans, 1., globalInteractions, moments, 0., ►
                     muDerivative) ;
                double global, pair ;
                gsl_blas_ddot(muPairDerivative, pairMoments[i - (i > halideIndex)], &pair) ;
                gsl_blas_ddot(muDerivative, moments, &global) ;
                derivatives->data[3 * i + k] += (pair - global) / 2. ;
                gsl_matrix_free(pairInteraction) ;
                gsl_matrix_free(globalInteractions) ;
        }
    }
    double halideForce[] = {0., 0., 0.} ;
    for(int j = 0 ; j < 3 ; j++)
    {
        double* forceData = derivatives->data + j ;
        double& halideForceComponent = halideForce[j] ;
        for(int i = 0 ; i < atoms.size() ; i++)
        {
            if(i != halideIndex)
                halideForceComponent -= *forceData ;
            forceData += 3 ;
        }
    }
    for(int i = 0 ; i < 3 ; i++)
        derivatives->data[3 * halideIndex + i] = halideForce[i] ;
    gsl_vector_scale(derivatives, parser::Eh * parser::a0) ;
    QVector<double>* retVal = new QVector<double> (atoms.size() * 3) ;
    for(int i = 0 ; i < 3 * atoms.size() ; i++)
        (*retVal) [i] = derivatives->data[i] ;
    gsl_matrix_free(qDerivatives) ;
    gsl_vector_free(derivatives) ;
    gsl_vector_free(muDerivative) ;
    gsl_vector_free(muPairDerivative) ;
    return retVal ;
}
```

src/pw02.h

```
#ifndef PW02_H
#define PW02_H
```

```
#include <QPair>
#include "commoninteraction.h"
class pw02 : public commonInteraction
{
private :
        double e, rm, bm, c0, A, a1, x1, x2, x3, x4, A2, al2, be2, c2 ;
        double a1, a2, a3, a4, b1, b2, b3, b4 ;
        double delta ;
        int ew ;
public :
        double energy(const QVector<atom*>&) ;
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        void printDescription() const ;
        void addChild(energyFunction*, QList<double>) ;
        pw02* create() const { return new pw02() ;}
};
#endif
```

src/pw02.cpp

```
#include "pw02.h"
#include <math.h>
#include <mmsv.h>
void pw02::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Pairwise␣additive␣matrix␣potential␣based␣on␣V0/V2." << endl ;
}
double pw02::energy(const QVector<atom*>& atoms)
{
        double vzero = 0 , a = 0, b = 0, c = 0, g = 0, h = 0 ;
        for(int i = 0 ; i < atoms.size() - 1 ; i++)
        {
                QVector<atom*> combination ;
                coord relativePosition = *atoms[i + 1] - *atoms[0] ;
                double r = pow(~relativePosition, 2), x = relativePosition[0], y = relativePosition►
                     [1], z = relativePosition[2] ;
                double re = sqrt(r) ;
                double xv = re / rm ;
                double vtwo = -A2 * exp(-al2 * re - be2 * r) + c2 / pow(re, 6) ;
                vzero += (xv <= x1 ? A * exp(-al * (xv - 1.)) :
                          (xv < x2 ? exp(a1 + (xv - x1) * (a2 + (xv - x2) * (a3 + (xv - x1) * a4))) :
                           (xv <= x3 ? exp(-2.*bm * (xv - 1.)) - 2.*exp(-bm * (xv - 1.)) :
                            (xv < x4 ? b1 + (xv - x3) * (b2 + (xv - x4) * (b3 + (xv - x3) * b4)) :
                             - c0 / e / pow(rm * xv, 6))))) ;
                a += (3.*pow(z, 2) / r - 1.) * vtwo ;
                b += z * x / r * vtwo ;
                c += (pow(y, 2) - pow(x, 2)) / r * vtwo ;
                g += z * y / r * vtwo ;
                h += x * y / r * vtwo ;
        }
        vzero *= e ;
        b *= -.6 / M_SQRT2 ;
        c *= .3 ;
        g *= .6 / M_SQRT2 ;
        h *= .6 ;
        double k = (27e2 * a * (pow(b, 2) + pow(g, 2) - pow(h, 2) - pow(c, 2)) + 27.*pow(a, 3) + 1e3►
              * pow(delta, 3) - 54e3 * b * g * h + 27e3 * c * (pow(g, 2) - pow(b, 2))) / pow(30., 3),
               A = sqrt(pow(delta, 2) / 9. + (pow(c, 2) + 2.*pow(g, 2) + pow(h, 2) + 2.*pow(b, 2)) /►
                    3. + pow(.1 * a, 2)) ;
        double phi = acos(k / pow(A, 3)) / 3 ;
        switch(ew)
        {
                case 1: return vzero - A * (cos(phi) - sqrt(3.) * sin(phi)) + delta / 3. ;
                case 2: return vzero - A * (cos(phi) + sqrt(3.) * sin(phi)) + delta / 3. ;
                default: return vzero + 2.*A * cos(phi) - 2. / 3.*delta ;
        }
}
void pw02::setParameters(const QVector<double>& params)
{
        if(params.size() < 2) return ;
        delta = params[0] ;
        ew = params[1] ;
```

```
        if(params.size() < 14) return ;
        e = params[2] ;
        rm = params[3] ;
        bm = params[4] ;
        c0 = params[5] ;
        A = params[6] ;
        al = params[7] ;
        x1 = params[8] ;
        x2 = params[9] ;
        x3 = params[10] ;
        x4 = params[11] ;
        A2 = params[12] ;
        al2 = params[13] ;
        be2 = params[14] ;
        c2 = params[15] ;
        a1 = log(A) - al * (x1 - 1.) ;
        a2 = log(exp(-2.*bm * (x2 - 1.) - a1) - 2.*exp(-bm * (x2 - 1.) - a1)) / (x2 - x1) ;
        a3 = (a2 + al) / (x2 - x1) ;
        a4 = ((-2.*bm * (1. + 1. / (exp(-bm * (x2 - 1.)) - 2.)) - a2) / (x2 - x1) - a3) / (x2 - x1) ;
        b1 = exp(-2.*bm * (x3 - 1.)) - 2.*exp(-bm * (x3 - 1.)) ;
        b2 = (c0 / e / pow(rm, 6) / pow(x4, 6) + b1) / (x3 - x4) ;
        b3 = (2.*bm * (exp(-2.*bm * (x3 - 1)) - exp(-bm * (x3 - 1.))) + b2) / (x4 - x3) ;
        b4 = ((6.*c0 / e / pow(rm, 6) / pow(x4, 7) - b2) / (x4 - x3) - b3) / (x4 - x3) ;
}
QVector<double> pw02::parameters() const
{
        QVector<double> params ;
        params << delta << ew ;
        params << e << rm << bm << c0 << A << al << x1 << x2 << x3 << x4 << A2 << al2 << be2 << c2 ;
        return params ;
}
```

src/pwam.h

```
#ifndef PWAM_H
#define PWAM_H
#include <QPair>
#include "commoninteraction.h"
#include "mmsv.h"
class pwam : public commonInteraction
{
private :
        QVector<double> permparams ;
        void reset(const QVector<double>& params) ;
protected:
        QVector<QVector<QPair<energyContribution*, QPair<double, double> > > > contributions ;
        QList<energyFunction*> functions ;
        QList<QPair<double, double> > weights ;
        double delta ;
        int ew ;
public :
        QVector<double>* force(const QVector<atom*>&, double step, bool refresh) ;
        double energy(const QVector<atom*>&) ;
        void setParameters(const QVector<double>&) ;
        QVector<double> parameters() const ;
        void printDescription() const ;
        energyContribution* explicitFunction(QVector<atom*>) ;
        void addChild(energyFunction*, QList<double>) ;
        pwam* create() const { return new pwam() ;}
        ~pwam() ;
};
#endif
```

src/pwam.cpp

```
#include "pwam.h"
#include <math.h>
#include <mmsv.h>
#include <QDebug>
#include "esmsvw.h"
#include "v2pot.h"
void pwam::printDescription() const
```

```
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Pairwise␣additive␣matrix␣potential." << endl ;
}
pwam::~pwam()
{
        reset(QVector<double>());
}
QVector<double>* pwam::force(const QVector<atom*>& atoms, double step, bool refresh)
{
        return commonInteraction::force(atoms, step, refresh) ;
}
double pwam::energy(const QVector<atom*>& atoms)
{
        if(functions.isEmpty()) return 0 ;
        if(atoms.size() == 2 && contributions.size() == 1 && contributions[0].size() == 3)
                return contributions[0][qBound(0,abs(ew) - 1,2)].first->energy(atoms) + (ew < 0 ? ►
                    delta / 3. * (abs(ew) < 3 ? -1 : 2) : 0.) ;
        double vzero = 0 , a = 0, b = 0, c = 0, g = 0, h = 0 ;
        double xx = 0, yy = 0, zz = 0, xy = 0, xz = 0, yz = 0, v2 = 0 ;
        for(int i = 0 ; i < atoms.size() - 1 ; i++)
        {
                QVector<atom*> combination ;
                combination << atoms[0] << atoms[i + 1] ;
                double vtwo = 0 ;
                for(int j = 0; j < contributions[i].size(); j++)
                {
                        double contribution = contributions[i][j].first->energy(combination) ;
                        vzero += contribution * contributions[i][j].second.first ;
                        vtwo += contribution * contributions[i][j].second.second ;
                }
                coord relativePosition = *atoms[i + 1] - *atoms[0] ;
                double r = pow(~relativePosition, 2), x = relativePosition[0], y = relativePosition►
                    [1], z = relativePosition[2] ;
                a += (3.*pow(z, 2) / r - 1.) * vtwo ;
                b += z * x / r * vtwo ;
                c += (pow(y, 2) - pow(x, 2)) / r * vtwo ;
                g += z * y / r * vtwo ;
                h += x * y / r * vtwo ;
                v2 += vtwo ;
                xx += x*x/r*vtwo ;
                yy += y*y/r*vtwo ;
                zz += z*z/r*vtwo ;
                xy += x*y/r*vtwo ;
                xz += x*z/r*vtwo ;
                yz += y*z/r*vtwo ;
        }
        a *= .1 ;
        b *= .6 ;
        c *= .3 ;
        g *= .6 ;
        h *= .6 ;
        double k = a * (.5 * (pow(b, 2) + pow(g, 2)) - pow(h, 2) - pow(c, 2)) + pow(a, 3) + pow(►
            delta / 3., 3) + b * g * h + .5 * c * (pow(g, 2) - pow(b, 2)),
               A = sqrt(pow(delta / 3., 2) + (pow(c, 2) + pow(g, 2) + pow(h, 2) + pow(b, 2)) / 3. + ►
                   pow(a, 2)) ;
        double phi = acos(k / pow(A, 3)) / 3 ;
        return vzero + 2.*A * cos(phi - 4.*M_PI / 3.*(abs(ew))) + (ew > 0 ? delta / 3. * (ew < 3 ? 1►
            : -2) : 0.) ;
}
void pwam::setParameters(const QVector<double>& params)
{
        if(params.size() < 2) return ;
        delta = params[0] ;
        ew = params[1] ;
        if (params.size() == 16)
        {
                reset(params) ;
                weights << QPair<double, double>(1,0)
                                << QPair<double, double>(0,1) ;
                functions << new esmsvw ;
                functions.last()->addInteraction(partner, params.mid(2,10)) ;
                functions << new v2pot ;
                functions.last()->addInteraction(partner, params.mid(12)) ;
        }
```

```
        if (params.size() == 38)
        {
                reset(params) ;
                for(int i = 2 ; i < 8 ; i += 2)
                        weights << QPair<double, double> (params[i] / 3., params[i + 1] * 5. / 3.) ;
                for(int i = 8 ; i < 38 ; i += 10)
                {
                        functions << new mmsv(false) ;
                        functions.last()->addInteraction(partner, params.mid(i, 10)) ;
                }
        }
}
QVector<double> pwam::parameters() const
{
        return permparams ;
}
void pwam::reset(const QVector<double> &params)
{
        weights.clear();
        permparams = params ;
        while (!functions.isEmpty())
                delete functions.takeLast() ;
        contributions.clear();
}
void pwam::addChild(energyFunction* function, QList<double> weight)
{
        if(weight.size() != 2) return ;
        functions << function ;
        weights << QPair<double, double> (weight[0] / 3., weight[1] * 5. / 3.) ;
}
energyContribution* pwam::explicitFunction(QVector<atom*> atoms)
{
        if(functions.isEmpty()) return 0 ;
        for(int i = 1 ; i < atoms.size() ; i++)
        {
                contributions << QVector<QPair<energyContribution*, QPair<double, double> > >() ;
                QVector<atom*> combination ;
                combination << atoms[i] << atoms[0] ;
                for(int j = 0 ; j < functions.size() ; j++)
                {
                        energyContribution* contribution = functions[j]->explicitFunction(►
                            combination) ;
                        if(contribution)
                                contributions.last() << QPair<energyContribution*, QPair<double, ►
                                    double> > (contribution, weights[j]) ;
                }
        }
        return this ;
}
```

src/pwam-exp.h

```
#ifndef PWAMEXP_H
#define PWAMEXP_H
#include <QPair>
#include "pwam.h"
#include "mmsv.h"
class pwamexp : public pwam
{
private :
        QVector<QVector<QPair<energyContribution*, QPair<double, double> > > > contributions ;
        QList<energyFunction*> functions ;
        QList<QPair<double, double> > weights ;
        QVector<double> evs ;
        double delta ;
        int ew ;
public :
        double energy(const QVector<atom*>&) ;
        void printDescription() const ;
        pwamexp* create() const { return new pwamexp() ;}
        pwamexp() { evs.resize(3) ;}
};
#endif
```

src/pwam-exp.cpp

```
#include "pwam-exp.h"
#include <math.h>
#include <mmsv.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_complex_math.h>
#include <gsl/gsl_eigen.h>
#include <gsl/gsl_sort_vector.h>
#include <QDebug>
void pwamexp::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "Pairwise␣additive␣matrix␣potential␣(explicit␣matrix␣evaluation)." << endl ;
}
double pwamexp::energy(const QVector<atom*>& atoms)
{
        if(functions.isEmpty()) return 0 ;
        double vzero = 0 , a = 0, b = 0, c = 0, g = 0, h = 0 ;
        for(int i = 0 ; i < atoms.size() - 1 ; i++)
        {
                QVector<atom*> combination ;
                combination << atoms[i + 1] << atoms[0] ;
                double vtwo = 0 ;
                for(int j = 0; j < contributions[i].size(); j++)
                {
                        double contribution = contributions[i][j].first->energy(combination) ;
                        vzero += contribution * contributions[i][j].second.first ;
                        vtwo += contribution * contributions[i][j].second.second ;
                }
                coord relativePosition = *atoms[i + 1] - *atoms[0] ;
                double r = pow(~relativePosition, 2), x = relativePosition[0], y = relativePosition►
                    [1], z = relativePosition[2] ;
                a += (3.*pow(z, 2) / r - 1.) * vtwo ;
                b += z * x / r * vtwo ;
                c += (pow(y, 2) - pow(x, 2)) / r * vtwo ;
                g += z * y / r * vtwo ;
                h += x * y / r * vtwo ;
        }
        b *= -.6 / M_SQRT2 ;
        c *= .3 ;
        g *= .6 / M_SQRT2 ;
        h *= .6 ;
        gsl_matrix_complex* matrix = gsl_matrix_complex_calloc(6, 6);
        gsl_matrix_complex_set(matrix, 0, 0, gsl_complex_rect(vzero + 2. / 3.*delta, 0)) ;
        gsl_matrix_complex_set(matrix, 0, 2, gsl_complex_rect(-M_SQRT2 * .1 * a, 0)) ;
        gsl_matrix_complex_set(matrix, 0, 3, gsl_complex_rect(b, g)) ;
        gsl_matrix_complex_set(matrix, 0, 4, gsl_complex_rect(-1. / sqrt(3.) *b, 1. / sqrt(3.) *g)) ;
        gsl_matrix_complex_set(matrix, 0, 5, gsl_complex_rect(sqrt(2. / 3.) *c, sqrt(2. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 1, 1, gsl_complex_rect(vzero + 2. / 3.*delta, 0)) ;
        gsl_matrix_complex_set(matrix, 1, 2, gsl_complex_rect(b, -g)) ;
        gsl_matrix_complex_set(matrix, 1, 3, gsl_complex_rect(M_SQRT2 * .1 * a, 0)) ;
        gsl_matrix_complex_set(matrix, 1, 4, gsl_complex_rect(-sqrt(2. / 3.) *c, +sqrt(2. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 1, 5, gsl_complex_rect(-sqrt(1. / 3.) *b, -sqrt(1. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 2, 0, gsl_complex_rect(-M_SQRT2 * .1 * a, 0)) ;
        gsl_matrix_complex_set(matrix, 2, 1, gsl_complex_rect(b, g)) ;
        gsl_matrix_complex_set(matrix, 2, 2, gsl_complex_rect(vzero + .1 * a - delta / 3., 0)) ;
        gsl_matrix_complex_set(matrix, 2, 4, gsl_complex_rect(sqrt(2. / 3.) *b, -sqrt(2. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 2, 5, gsl_complex_rect(sqrt(1. / 3.) *c, sqrt(1. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 3, 0, gsl_complex_rect(b, -g)) ;
        gsl_matrix_complex_set(matrix, 3, 1, gsl_complex_rect(M_SQRT2 * .1 * a, 0)) ;
        gsl_matrix_complex_set(matrix, 3, 3, gsl_complex_rect(vzero + .1 * a - delta / 3., 0)) ;
        gsl_matrix_complex_set(matrix, 3, 4, gsl_complex_rect(sqrt(1. / 3.) *c, -sqrt(1. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 3, 5, gsl_complex_rect(-sqrt(2. / 3.) *b, -sqrt(2. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 4, 0, gsl_complex_rect(-1. / sqrt(3.) *b, -1. / sqrt(3.) *g)) ;
        gsl_matrix_complex_set(matrix, 4, 1, gsl_complex_rect(-sqrt(2. / 3.) *c, -sqrt(2. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 4, 2, gsl_complex_rect(sqrt(2. / 3.) *b, sqrt(2. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 4, 3, gsl_complex_rect(sqrt(1. / 3.) *c, sqrt(1. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 4, 4, gsl_complex_rect(vzero - .1 * a - delta / 3., 0)) ;
        gsl_matrix_complex_set(matrix, 5, 0, gsl_complex_rect(sqrt(2. / 3.) *c, -sqrt(2. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 5, 1, gsl_complex_rect(-sqrt(1. / 3.) *b, sqrt(1. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 5, 2, gsl_complex_rect(sqrt(1. / 3.) *c, -sqrt(1. / 3.) *h)) ;
        gsl_matrix_complex_set(matrix, 5, 3, gsl_complex_rect(-sqrt(2. / 3.) *b, sqrt(2. / 3.) *g)) ;
        gsl_matrix_complex_set(matrix, 5, 5, gsl_complex_rect(vzero - .1 * a - delta / 3., 0)) ;
        gsl_vector* eigenvalues = gsl_vector_alloc(6) ;
```

```
        gsl_eigen_herm_workspace* ws = gsl_eigen_herm_alloc(6) ;
        gsl_eigen_herm(matrix, eigenvalues, ws) ;
        gsl_eigen_herm_free(ws) ;
        gsl_matrix_complex_free(matrix) ;
        double k = (27e2 * a * (pow(b, 2) + pow(g, 2) - pow(h, 2) - pow(c, 2)) + 27.*pow(a, 3) + 1e3►
              * pow(delta, 3) - 54e3 * b * g * h + 27e3 * c * (pow(g, 2) - pow(b, 2))) / pow(30., 3),
               A = sqrt(pow(delta, 2) / 9. + (pow(c, 2) + 2.*pow(g, 2) + pow(h, 2) + 2.*pow(b, 2)) /►
                    3. + pow(.1 * a, 2)) ;
        double phi = acos(k / pow(A, 3)) / 3 ;
        evs[0] = vzero - A * (cos(phi) - sqrt(3.) * sin(phi)) ;
        evs[1] = vzero - A * (cos(phi) + sqrt(3.) * sin(phi)) ;
        evs[2] = vzero + 2.*A * cos(phi) ;
        qDebug() << "Eigene Formel:" << evs[0] << evs[1] << evs[2] ;
        gsl_sort_vector(eigenvalues) ;
        qDebug() << "Matrix direkt:"
                 << gsl_vector_get(eigenvalues,0)
                 << gsl_vector_get(eigenvalues,2)
                 << gsl_vector_get(eigenvalues,4) ;
        gsl_vector_free(eigenvalues) ;
        return gsl_vector_get(eigenvalues, 2*ew) ;
}
```

src/v2pot.h

```
#ifndef V2POT_H
#define V2POT_H
#include "energyfunction.h"
class v2potContribution : public energyContribution
{
private:
        double A2, alpha2, beta2, C2;
public:
        QVector<double> parameters() const ;
        void setParameters(const QVector<double> &) ;
        double energy(const QVector<atom *> &) ;
        void printDescription() const ;
};
class v2pot : public energyFunction
{
protected:
        energyContribution* newInteraction() { return new v2potContribution ; }
public:
        void printDescription() const ;
        int numAtoms() { return 2 ; }
        v2pot* create() const {return new v2pot ; }
};
#endif
```

src/v2pot.cpp

```
#include "v2pot.h"
QVector<double> v2potContribution::parameters() const
{
        return QVector<double>() << A2 << alpha2 << beta2 << C2 ;
}
void v2potContribution::setParameters(const QVector<double> &ps)
{
        A2 = ps.size() > 0 ? ps[0] : 0 ;
        alpha2 = ps.size() > 1 ? ps[1] : 0 ;
        beta2 = ps.size() > 2 ? ps[2] : 0 ;
        C2 = ps.size() > 3 ? ps[3] : 0 ;
}
double v2potContribution::energy(const QVector<atom *> &atoms)
{
        if (atoms.size() != 2) return 0 ;
        double r = *(atoms[0]) % *(atoms[1]) ;
        return -A2 * exp(-alpha2 * r - beta2*r*r) + C2/pow(r,6) ;
}
void v2potContribution::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "V2 contribution for:  ";
```

```
        for(int i = 0 ; i < partner.size() ; i++)
                cout << partner[i] << "␣␣" ;
        cout << "␣Parameters:␣␣" << A2 << alpha2 << beta2 << C2;
}
void v2pot::printDescription() const
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        cout << "V2␣potential␣" ;
        foreach(energyContribution * pointer, interactions)
        {
                cout << "(";
                QStringList ps = pointer->partners() ;
                foreach(QString p, ps)
                cout << p << "," ;
                cout << ")␣" ;
        }
        cout << endl ;
}
```

src/variablehash.h

```
#ifndef VARIABLEHASHB_H
#define VARIABLEHASHB_H
#include <QHash>
#include <QString>
#include <QVariant>
#include <QStringList>
#include <cluster.h>
class computationModule ;
class variableHash : public QHash<QString, QVariant>
{
public:
        struct computationSettings
        {
                double energyThreshold ;
                double gradientThreshold ;
                double gradientStep ;
                double boxSize;
                double boxStep;
                double maxStep;
                double perturbativeStep;
                double imgLimit;
                double lowerLimit ;
                double azimuthStep ;
                double altitudeStep ;
                int maxSteps ;
                bool printEigenvectors ;
                bool recenter ;
                bool includeNextShell ;
                int verbosity ;
                int optimizerType ;
                void read(const variableHash&, QTextStream& out) ;
        };
private:
        computationSettings settings ;
        QString variableContent(const QVariant& variable, int indentation = 0) ;
        QVariant buildVariant(QStringList&) ;
        QHash<QString, computationModule*> commands ;
        QHash<QString, energyFunction*> functions ;
        cluster* generateCluster(QVariant&) ;
        cluster* setupCluster(const QString&) ;
        energyFunction* generateFunction(const QVariant&) ;
        atom* generateAtom(const QVariant&) ;
        void readSettings() ;
        void insertAtom(QTextStream& out) ;
        void deleteAtom(QTextStream& out) ;
        void setGeometry(const QVector<double>& coords, QTextStream& out) ;
        void setVariable(QString& key, QStringList& followingLines) ;
public:
        variableHash(QStringList) ;
        ~variableHash() ;
        static QString popNextElement(QString&) ;
        void assignVariable(QString&, QStringList&) ;
        QString currentContent() ;
```

```
        QVariant resolveVariable(const QVariant&) const;
        QVariant resolveVariable(const QString&) const ;
        QVariant resolveVariable(const char variant[]) const ;
};
#endif
```

src/variablehash.cpp

```
#include "variablehash.h"
#include <QList>
#include "computationmodule.h"
#include "lennardjones.h"
#include "mmsv.h"
#include "hfdb.h"
#include "harmonic.h"
#include "pwam.h"
#include "coulomb.h"
#include "axilrod.h"
#include "nonaddinduction.h"
#include "induction.h"
#include "exchange.h"
#include "pwam-exp.h"
#include "internalpair.h"
#include "pw02.h"
variableHash::variableHash(QStringList inputLines)
{
        functions["mmsv"] = new mmsv ;
        functions["hfdb"] = new hfdb ;
        functions["hb"] = new harmonic(Harmonic::bond) ;
        functions["ha"] = new harmonic(Harmonic::angle) ;
        functions["hd"] = new harmonic(Harmonic::dieder) ;
        functions["hi"] = new harmonic(Harmonic::inversion) ;
        functions["pwam"] = new pwam() ;
        functions["coul"] = new coulomb() ;
        functions["lj"] = new lennardjones() ;
        functions["ax-t"] = new axilrod ;
        functions["nind"] = new nonAddInduction() ;
        functions["ind"] = new induction() ;
        functions["exc"] = new exchange() ;
        functions["bobo"] = new internalPair(Harmonic::bond, Harmonic::bond) ;
        functions["boan"] = new internalPair(Harmonic::bond, Harmonic::angle) ;
        functions["bodi"] = new internalPair(Harmonic::bond, Harmonic::dieder) ;
        functions["anan"] = new internalPair(Harmonic::angle, Harmonic::angle) ;
        functions["andi"] = new internalPair(Harmonic::angle, Harmonic::dieder) ;
        functions["didi"] = new internalPair(Harmonic::dieder, Harmonic::dieder) ;
        functions["pwamexp"] = new pwamexp() ;
        functions["pw02"] = new pw02() ;
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        commands = computationModule::allModules() ;
        while(!inputLines.isEmpty())
        {
                settings.read(*this, cout) ;
                QString input = inputLines.takeFirst().trimmed() ;
                if(settings.verbosity > 3)
                        cout << "processing:␣␣" << input << endl ;
                if(input.isEmpty()) continue ;
                QString object = input.section(QRegExp("(\\s+|=)"), 0, 0) ;
                input.remove(0, object.size()).remove(QRegExp("^\\s+")) ;
                if(object == "deleteAtom")
                {
                        deleteAtom(cout) ;
                        continue ;
                }
                if(object == "insertAtom")
                {
                        insertAtom(cout) ;
                        continue ;
                }
                if(commands.contains(object))
                {
                        if(settings.verbosity > 2)
                                cout << "Executing␣command␣" << object << endl ;
                        cluster* Cluster = setupCluster("cluster") ;
                        if(!Cluster) continue ;
```

```
                        computationModule* command = commands[object] ;
                        command->setCluster(Cluster) ;
                        command->setSettings(&settings) ;
                        command->execute();
                        if(settings.recenter)
                                Cluster->centerOnFirstAtom();
                        setGeometry(Cluster->coordinates(), cout) ;
                        delete Cluster ;
                        continue ;
                }
                QString temp = popNextElement(input) ;
                if(temp == "=")
                {
                        inputLines.prepend(input) ;
                        assignVariable(object, inputLines) ;
                }
                else if(temp == ":")
                {
                        inputLines.prepend(input) ;
                        setVariable(object, inputLines) ;
                }
                else
                        cout << "Error:␣␣not␣an␣assignment␣or␣command:␣␣" << temp << endl ;
        }
}
void variableHash::setGeometry(const QVector<double>& coords, QTextStream& out)
{
        if(coords.size() / 3 == resolveVariable(resolveVariable("cluster").toList() [1]).toList().►
            size())
        {
                QList<QVariant> oldCluster = resolveVariable("cluster").toList() ;
                if(oldCluster.empty())
                {
                        out << "Cluster␣not␣defined;␣cannot␣set␣geometry." << endl ;
                        return ;
                }
                QList<QVariant> atoms = resolveVariable(oldCluster[1]).toList() ;
                for(int i = 0 ; i < atoms.size() ; i++)
                {
                        QList<QVariant> currentAtom = resolveVariable(atoms[i]).toList() ;
                        for(int j = 0 ; j < 3 ; j++)
                                currentAtom[j + 1] = coords[3 * i + j] ;
                        atoms[i] = currentAtom ;
                }
                oldCluster[1] = atoms ;
                operator[]("cluster") = oldCluster ;
                if(settings.verbosity > 2)
                {
                        out << "New␣cluster␣geometry:" << endl ;
                        out << variableContent(operator[]("cluster"), 1) ;
                }
        }
        else
                out << "Setting␣geometry␣failed!" << endl ;
}
void variableHash::deleteAtom(QTextStream& out)
{
        if(resolveVariable("cluster") == QVariant())
        {
                out << "Cannot␣delete␣atom.␣␣\"cluster\"␣undefined." << endl ;
                return ;
        }
        QList<QVariant> oldCluster = resolveVariable("cluster").toList() ;
        QList<QVariant> atoms = resolveVariable(oldCluster[1]).toList() ;
        if(atoms.empty())
        {
                out << "No␣atoms;␣cannot␣delete." << endl ;
                return ;
        }
        int deletePos = qBound(0, resolveVariable("atomIndex").toInt(), atoms.size() - 1) ;
        out << "Deleting␣atom␣"
            << "␣at␣position␣"
            << deletePos << endl ;
        atoms.removeAt(deletePos) ;
        oldCluster[1] = atoms ;
```

```
        operator[]("cluster") = oldCluster ;
        if(settings.verbosity > 1)
                out << variableContent(operator[]("cluster"), 1) ;
}
void variableHash::insertAtom(QTextStream& out)
{
        if(resolveVariable("cluster") == QVariant())
        {
                out << "Cannot␣add␣atom.␣␣\"cluster\"␣undefined." << endl ;
                return ;
        }
        QList<QVariant> oldCluster = resolveVariable("cluster").toList() ;
        QList<QVariant> atoms = resolveVariable(oldCluster[1]).toList() ;
        QVariant insertingAtom = operator[]("insertingAtom") ;
        int insertPos = qBound(0, resolveVariable("atomIndex").toInt(), atoms.size()) ;
        out << "Inserting␣atom␣"
            << insertingAtom.toString()
            << "␣at␣position␣"
            << insertPos << endl ;
        atoms.insert(insertPos, insertingAtom) ;
        oldCluster[1] = atoms ;
        operator[]("cluster") = oldCluster ;
        if(settings.verbosity > 1)
                out << variableContent(operator[]("cluster"), 1) ;
}
QString variableHash::popNextElement(QString& input)
{
        QRegExp controlCharacters("[\\{\\};=]") ;
        input.remove(QRegExp("^\\s+")) ;
        if(input.isEmpty()) return QString() ;
        if(controlCharacters.exactMatch(input.left(1)))
        {
                QString retVal = input.left(1) ;
                input.remove(0, 1) ;
                return retVal ;
        }
        if(input[0] == '\"')
        {
                input.remove(0, 1) ;
                QString retVal = input.left(input.indexOf("\"")) ;
                input.remove(0, input.indexOf('\"') + 1) ;
                return retVal ;
        }
        QString retVal = input.section(QRegExp("(\\s+|[\\{\\};=])"), 0, 0) ;
        input.remove(0, retVal.size()) ;
        return retVal ;
}
QVariant variableHash::buildVariant(QStringList& followingLines)
{
        QTextStream cout(stdout, QIODevice::WriteOnly) ;
        QString value ;
        while(!followingLines.isEmpty() && (value = popNextElement(followingLines[0])).isEmpty())
                followingLines.takeFirst() ;
        if(value.isEmpty() || value == ";" || value == "}") return true ;
        if(value == "{")
        {
                QList<QVariant> retVal ;
                do
                {
                        if(followingLines.isEmpty())
                        {
                                cout << "ERROR:␣no␣closing␣bracket␣at␣end.␣␣Adding␣closing␣bracket."►
                                     << endl ;
                                followingLines << "}" ;
                        }
                        value = popNextElement(followingLines[0]) ;
                        if(value.isEmpty())
                                followingLines.takeFirst() ;
                        else if(value != "}")
                        {
                                value = "\"" + value + "\"" ;
                                followingLines.prepend(value) ;
                                retVal << buildVariant(followingLines) ;
                        }
                }
```

```
                while(value != "}") ;
                return retVal ;
        }
        return value;
}
void variableHash::assignVariable(QString& key, QStringList& followingLines)
{
        operator[](key) = buildVariant(followingLines) ;
}
void variableHash::setVariable(QString& key, QStringList& followingLines)
{
        QVariant temp = buildVariant(followingLines) ;
        if(temp.type() == QVariant::String && contains(temp.toString()))
                temp = resolveVariable(temp) ;
        operator[](key) = temp ;
}
QString variableHash::currentContent()
{
        QString retVal ;
        for(int i = 0 ; i < size() ; i++)
                retVal += keys() [i] + "␣=␣" + variableContent(operator[](keys() [i]), 0) + "\n";
        return retVal ;
}
QString variableHash::variableContent(const QVariant& variable, int indentation)
{
        QString retVal(indentation, '␣') ;
        if(variable.type() == QVariant::Double)
                retVal += QString("%1").arg(variable.toDouble()) ;
        else if(variable.type() == QVariant::String)
                retVal += variable.toString() ;
        else if(variable.type() == QVariant::List)
        {
                retVal += "{" ;
                for(int i = 0 ; i < variable.toList().size() ; i++)
                        retVal += variableContent(variable.toList() [i], indentation + 1) ;
                retVal += "}\n" ;
        }
        return retVal ;
}
QVariant variableHash::resolveVariable(const QVariant& variant) const
{
        if(variant.type() == QVariant::String)
        {
                if(contains(variant.toString()))
                        return resolveVariable(operator[](variant.toString())) ;
                else
                        return variant ;
        }
        return variant ;
}
QVariant variableHash::resolveVariable(const QString& variant) const
{ return resolveVariable(operator[](variant)) ; }
QVariant variableHash::resolveVariable(const char variant[]) const
{ return resolveVariable(operator[](variant)) ; }
cluster* variableHash::generateCluster(QVariant& input)
{
        QList<QVariant> list = resolveVariable(input).toList() ;
        if(list.size() < 2) return NULL ;
        QList<QVariant> energyFunctions = resolveVariable(list[0]).toList(),
                        atoms = resolveVariable(list[1]).toList() ;
        if(energyFunctions.isEmpty() || atoms.isEmpty()) return NULL ;
        cluster* retVal = new cluster ;
        for(int i = 0 ; i < energyFunctions.size() ; i++)
                retVal->addEnergyFunction(generateFunction(energyFunctions[i])) ;
        for(int i = 0 ; i < atoms.size() ; i++)
                retVal->addAtom(generateAtom(atoms[i])) ;
        if(! retVal->initialize()) return NULL ;
        return retVal ;
}
energyFunction* variableHash::generateFunction(const QVariant& input)
{
        QList<QVariant> list = resolveVariable(input).toList() ;
        if(list.size() < 1) return NULL ;
        if(! functions.contains(resolveVariable(list[0]).toString())) return NULL ;
        energyFunction* retVal = functions[resolveVariable(list[0]).toString()]->create() ;
```

```
        for(int i = 1 ; i + 1 < list.size() ; i += 2)
        {
                QVector<double> parameters ;
                QStringList partners ;
                foreach(const QVariant & v, resolveVariable(list[i]).toList())
                partners << resolveVariable(v).toString() ;
                QList<QVariant> params = resolveVariable(list[i + 1]).toList() ;
                for(int j = 0 ; j < params.size() ; j++)
                        parameters << resolveVariable(params[j]).toDouble() ;
                retVal->addInteraction(partners, parameters) ;
        }
        return retVal ;
}
atom* variableHash::generateAtom(const QVariant& input)
{
        QList<QVariant> list = resolveVariable(input).toList() ;
        int i = 0 ;
        if(list.size() < 4) return NULL ;
        QString label = resolveVariable(list[i++]).toString() ;
        QVector<double> coords, elstat ;
        double mass = 0.;
        while(i < 4)
                coords << resolveVariable(list[i++]).toDouble() ;
        if(i < list.size())
                mass = resolveVariable(list[i++]).toDouble() ;
        if(i < list.size())
                while(i < list.size())
                        elstat << resolveVariable(list[i++]).toDouble() ;
        return new atom(label, coords, mass, elstat) ;
}
cluster* variableHash::setupCluster(const QString& clusterName)
{
        QTextStream out(stdout, QIODevice::WriteOnly) ;
        if(!contains(clusterName))
        {
                out << "ERROR:␣␣no␣cluster␣defined.␣␣Define␣variable␣\"" << clusterName << "\"␣►
                    properly." << endl ;
                return NULL ;
        }
        cluster* Cluster = generateCluster(operator[](clusterName)) ;
        if(!Cluster)
        {
                out << "ERROR:␣␣variable␣\"" << clusterName << "\"␣not␣properly␣defined.␣␣Did␣you␣►
                    include␣geometry␣list␣and␣forces?" << endl ;
                return NULL ;
        }
        return Cluster ;
}
enum vartype { boolT, doubleT, intT } ;
struct assignment
{
        QString variableName ;
        void* variable ;
        QString unit ;
        QString vname ;
        vartype type ;
        double defaultValue ;
        assignment(QString name, void* var, QString u, QString vn, double def = 0, vartype t = ►
            doubleT)
                : variableName(name),
                  variable(var),
                  unit(u),
                  vname(vn),
                  type(t),
                  defaultValue(def)
        {}
};
void variableHash::computationSettings::read(const variableHash& source, QTextStream& out)
{
        verbosity = 10 ;
        QList<assignment> variables ;
        variables << assignment("␣␣Verbosity␣level", &verbosity, "", "verbosity", 0, intT)
                  << assignment("␣␣Energy␣threshold", &energyThreshold, "meV", "energyThreshold", 1e►
                      -12)
```

```
                << assignment("  Gradient threshold", &gradientThreshold, "meV/Ang", "►
                    gradientThreshold", 1e-12)
                << assignment("  Maximum number of steps", &maxSteps, "", "maxSteps", 50, intT)
                << assignment("  Maximum size of any single step", &maxStep, "Ang", "maxStep", 1)
                << assignment("  Step size", &gradientStep, "Ang", "gradientStep", 1e-3)
                << assignment("  Output of eigenvectors is", &printEigenvectors, "", "►
                    printEigenvectors", 0, boolT)
                << assignment("  Recentering of cluster is", &recenter, "", "recenter", 0, boolT)
                << assignment("  Box side for geometry searches reaches from/to -/+", &boxSize, "►
                    Ang.", "boxSize", 10)
                << assignment("  Length of box step", &boxStep, "Ang.", "boxStep", 2)
                << assignment("  Length of perturbative step", &perturbativeStep, "", "►
                    perturbativeStep", .1)
                << assignment("  Limit for imaginary modes", &imgLimit, "cm-1", "imaginaryLimit", ►
                    -.1)
                << assignment("  Lower limit for acceptable energies", &lowerLimit, "meV", "►
                    lowerLimit", -1e7)
                << assignment("  Step size for azimuth (sphere optimize)", &azimuthStep, "deg", "►
                    azimuthStep", 30)
                << assignment("  Step size for altitude (sphere optimize)", &altitudeStep, "deg", ►
                    "altitudeStep", 30)
                << assignment("  Include next shell (sphere optimize)", &includeNextShell, "", "►
                    includeNextShell", 0, boolT)
                << assignment("  Optimizer type", &optimizerType, "", "optimizerType", 0, intT);
        out << "  --- Reading settings ---" << endl ;
        foreach(const assignment & a, variables)
        {
#define PRINTVARCONDITION if (verbosity > 1 || a.variable == &verbosity)
                PRINTVARCONDITION out << a.variableName << ": " ;
                QVariant value = (source.contains(a.vname) ? source.resolveVariable(a.vname) : a.►
                    defaultValue) ;
                if(a.type == doubleT)
                {
                        double v = value.toDouble();
                        PRINTVARCONDITION out << v ;
                        * ((double*) a.variable) = v ;
                }
                if(a.type == intT)
                {
                        int v = value.toInt() ;
                        PRINTVARCONDITION out << v ;
                        * ((int*) a.variable) = v ;
                }
                if(a.type == boolT)
                {
                        bool v = value.toBool() ;
                        PRINTVARCONDITION out << (v ? "on " : "off") ;
                        * ((bool*) a.variable) = v ;
                }
                PRINTVARCONDITION out << " " << a.unit << " (variable \"" << a.vname << "\")" << ►
                    endl ;
        }
        out << "  ---------------------" << endl ;
}
variableHash::~variableHash()
{
        foreach(computationModule * cm, commands.values())
        delete cm ;
        foreach(energyFunction * ef, functions.values())
        delete ef ;
}
```

Acknowledgment

This work would not have been possible without much help and support on many levels and in many respects. I am indebted to Prof. Jürgen Troe for the opportunity to pursue this work in his group. Further, I would like to thank Prof. Dirk Schwarzer for the continued guidance, academic support, and inspiration. Prof. Jörg Schroeder's advice was indispensable for this work, as well. My gratitude shall also be extended to Prof. Alec Wodtke, who supported the continuation of this work, provided interesting insights into a very different field of research, and enabled quantum chemical computations.

My predecessors Dr. Christian Reichardt and Dr. Tim Schäfer built the optical apparatus, and – more importantly – offered their help and support whenever necessary. Dr. Aliaksandr Kandratsenka was available at all times for any discussion pertaining to theory or computers. Dr. Matthäus Kopczynski sparked my interest in the halide–noble gas clusters.

Martin Fechner, Ansgar Esztermann, Heiko Niemeier, and Petra Küster helped countless times in all computational matters. Help from the workshops of precision mechanics (headed by Rainer Schürkötter) and electronics (headed by Frank Meyer), as well as the department's own crew, Reinhardt Bürsing and Florian Lange, is gratefully acknowledged.

Finally, in daily operations of the experiments and beyond, it was a privilege to work with some of the finest individuals and lab mates one might hope for, namely Prof. Alexander Horn, Thomas Auth, and Jörn Werdecker with the optical setup, as well as our lab students Thomas Nick, Claudia Orben, and Robert Medel. To a yet higher degree, I owe thanks to Daniel Fischer for his invaluable help in conducting the experimental work and Sven Kaufmann for proofreading this work and offering his help whenever possible. Both Daniel and Sven were also instrumental in advancing the data processing software to the fairly operational state it has by now reached. Last, but most certainly not least, Inge Dreger shall be thanked for proofreading, administrative and motivational support, and being a nice neighbor.

Finally, through all this time, my wife Wensi Vennekate excelled at being supportive and at the same time lenient with all the frustration she had to endure.

Thank you!

Curriculum Vitae

Personal Details

Born September 29, 1983 in Haan (Germany)
Married since 10/2008
one son (born May 2011)
Citizen of Germany

Education

since 01/2009	Doctoral studies in the group of Prof. Jürgen Troe/Prof. Alec Wodtke, Max-Planck-Institute for Biophysical Chemistry, Göttingen
12/2008	Diploma thesis *Investigation of the Photoinduced Decomposition of Acetylbenzoyl and Phthaloyl Peroxid via Femtosecond Infrared Spectroscopy* in the group of Prof. Jürgen Troe, Max-Planck-Institute for Biophysical Chemistry, Göttingen (grade: A)
04/2006	Vordiplom (grade: A)
since 04/2004	Georg-August University, Göttingen, Germany; subject: chemistry
07/2003–03/2004	Military draft service
2001 – 2003	Gymnasium Petrinum Dorsten (Abitur)
2000 – 2001	Colonel White High School, Dayton, Ohio (Diploma of Graduation)
1994 – 2000	Gymnasium Petrinum Dorsten

Publications

2012	Hendrik Vennekate, Dirk Schwarzer, Joel Torres-Alacan, Oliver Krahe, Alexander C. Filippou, Frank Neese and Peter Vöhringer: *Ultrafast primary processes of an iron-(III) azido complex in solution induced with 266 nm light* in *Physical Chemistry Chemical Physics*, volume 14, issue 18, pages 6165–6172.
2011	Hendrik Vennekate, Arne Walter, Daniel Fischer, Jörg Schroeder and Dirk Schwarzer: *Photodecarbonylation of Diphenylcyclopropenone – a Direct Pathway to Electronically Excited Diphenylacetylene?* in *Zeitschrift für Physikalische Chemie*, volume 225, pages 1089–1104.
2010	Peter Sebald, Hendrik Vennekate, Rainer Oswald, Peter Botschwina and Hermann Stoll: *A theoretical study of ZnH_2: a case of very strong DarlingDennison resonance* in *Molecular Physics*, volume 108, issue 3-4, pages 487–499.

Presentations

22.03.2012	Presentation on *Diphenylcyclopropenone and Diphenylacetylene: Reaction and Relaxation Dynamics via UV-Pump-IR-Probe-Spectroscopy* at the *VIIIth Prague Workshop on Photoinduced Molecular Processes*.

05.10.2010	Presentation on *Recent UV-IR pump-probe experiments: Photolysis of aromatic peroxo esters and diphenylcyclopropenone* as visitor to the group of Prof. Xueming Yang at the Dalian Institute of Chemical Physics, PR China.
27. – 30.09.2009	Poster on *Diphenylcyclopropenone and Diphenylacetylene: Reaction and Relaxation Dynamics* and *Acetyl Benzoyl Peroxide and Phthaloyl Peroxide: Photolysis* at the *International Bunsen Discussion Meeting on Dynamics in complex molecular environments*.

Teaching

2010–2012	Introductory class *Programming in C++* (three times)
2006–2010	Teaching assistant for the introductory lecture *Theoretical Chemistry* (four times)
Summer 2008	Teaching assistant for the advanced lecture *Electronic Spectroscopy and Reaction Dynamics* (once)
Summer 2006	Teaching assistant for the introductory lecture *Experimental Chemistry II: Organic Chemistry* (once)

Activities and Memberships

08/2010–04/2012	Spokesperson of the doctoral students of the Faculty of Chemistry in the GAUSS (Georg-August University School of Science) Ph.D. program
since 05/2006	Gesellschaft Deutscher Chemiker e.V. (member of the Society of German Chemists)
since 03/2005	Deutschsprachige Anwendervereinigung TEX e.V. (member of the German TEX Users Group)
since 01/2005	Deutsche Bunsengesellschaft für Physikalische Chemie (member of the German Bunsen Society for Physical Chemistry)
11/2004–12/2008	Studienstiftung des deutschen Volkes (student scholar of the German National Academic Foundation)
since 07/2003	Deutsche Physikalische Gesellschaft (member of the German Physical Society)

Promovierenden-Erklärung der Georg-August-Universität Göttingen

Ich, Hendrik Vennekate, geboren am 29. September 1983 in Haan, beabsichtige, eine Dissertation zum Thema *S_2 State Photodissociation of Diphenylcyclopropenone, Vibrational Energy Transfer along Aliphatic Chains, and Energy Calculations of Noble Gas–Halide Clusters* an der Georg-August-Universität Göttingen anzufertigen. Dabei werde ich von Herrn Prof. Dr. Dirk Schwarzer betreut. Ich gebe folgende Erklärung ab:

1. Die Gelegenheit zum vorliegenden Promotionsvorhaben ist mir nicht kommerziell vermittelt worden. Insbesondere habe ich keine Organisation eingeschaltet, die gegen Entgelt Betreuerinnen und Betreuer für die Anfertigung von Dissertationen sucht oder die mir obliegenden Pflichten hinsichtlich der Prüfungsleistungen für mich ganz oder teilweise erledigt.

2. Hilfe Dritter wurde bis jetzt und wird auch künftig nur in wissenschaftlich vertretbarem und prüfungsrechtlich zulässigem Ausmaß in Anspruch genommen. Insbesondere werden alle Teile der Dissertation selbst angefertigt; unzulässige fremde Hilfe habe ich dazu weder unentgeltlich noch entgeltlich entgegengenommen und werde dies auch zukünftig so halten.

3. Die Richtlinien zur Sicherung der guten wissenschaftlichen Praxis an der Universität Göttingen werden von mir beachtet.

4. Eine entsprechende Promotion wurde an keiner anderen Hochschule im In- oder Ausland beantragt; die eingereichte Dissertation oder Teile von ihr wurden nicht für ein anderes Promotionsvorhaben verwendet.

Mir ist bekannt, dass unrichtige Angaben die Zulassung zur Promotion ausschließen bzw. später zum Verfahrensabbruch oder zur Rücknahme des erlangten Grades führen.

Göttingen, den