DOASIS
A Framework Design for DOAS

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Diplom-Informatiker
Stefan G. Kraus
aus Heidelberg

Mannheim, Juli 2006
Dekan: Prof. Dr. Matthias Krause, Universität Mannheim
Referent: Prof. Dr. Bernd Jähne, Universität Heidelberg
Korreferent: Prof. Dr. Reinhard Männer, Universität Mannheim

The book
Stefan Kraus

DOASIS
A Framework Design for DOAS

is published by
Shaker Verlag
(http://www.shaker.eu)


Details about how to obtain this book can be found here:
The Differential Optical Absorption Spectroscopy (DOAS) is a simple, sensitive and robust method to measure trace gas concentrations by analyzing the spectrum of light that has passed through the atmosphere. Due to these advantages, DOAS gains more and more popularity for environmental monitoring in both cities and on a global scale. To use DOAS, an efficient software basis is necessary. The existing DOAS tools either concentrate only on a certain part of the whole DOAS analysis process or lack support of new devices or evaluation methods.

This work will show a way to unify the already well known measurement and evaluation methods. The DOASIS Framework (DOAS Intelligent System) covers the whole range from hardware support, data management to the evaluation process. This framework offers an easy to extend basis to cover the DOAS evaluation process as a whole including automation support through a standardized scripting language.

One of the major aspects in a DOAS analysis is the evaluation of the measured data. Especially in scientific environments it is necessary to have an easy, fast and reliable way to test and integrate new evaluation methods. A framework will be presented that supports these needs for the non-linear evaluation process of a DOAS analysis while still providing the performance and stability of already existing software tools.

Besides the improvements in the evaluation process from a technical point of view, a new method for evaluating spectral data is proposed. Using the Laplace pyramid an optimal bandwidth filter is determined for \( NO_2 \) evaluation. Instead of using a polynomial to remove certain broadband structures in a spectrum and preprocessing the spectra with high- and low-pass filters, which both require user estimated parameters, the bandpass filter eliminates these.
Zusammenfassung


Diese Arbeit zeigt einen Weg, die vorhandenen Mess- und Auswertungswerkzeuge zu vereinheitlichen. Das DOASIS Framework (DOAS Intelligent System) umfaßt die gesamte Palette von der hardware-seitigen Ansteuerung von Spektrographen, über die Aufzeichnung von Spektren und deren Verwaltung, bis hin zur Datenauswertung. Das hier vorgestellte Framework bietet durch seine einfache Erweiterbarkeit eine gute Basis für die Integration neuer Geräte und Auswertungsmethoden, die über eine standardisierte Skriptsprache vollkommen automatisiert werden können.

Einer der wichtigsten Punkte einer DOAS Analyse ist die Auswertung der gemessenen Spektren. Gerade im Forschungsbereich ist es wichtig, einfach, schnell und zuverlässig neue Auswertungsverfahren anzuwenden. Eine neue Methode zur Erstellung von Auswertungsfunktionen wurde entwickelt, die eine einfache Änderung der Auswertungsfunktionen als auch der Auswertungsmethoden beinhaltet. Der Anwender kann z.B. durch einfaches Hinzufügen oder Rekombinieren von Basisfunktionen neue Auswertungsmethoden entwickeln, ohne dabei die gesamte Auswertungsroutine anpassen zu müssen.

Weiterhin wird ein Ansatz zur direkten Verbesserung der DOAS Analyse vorgestellt. Mit Hilfe der Laplace Pyramide wird ein optimaler Bandpaßfilter für eine \( NO_2 \) Auswertung ermittelt. Die Vorverarbeitung mit einem Laplace-Bandpaßfilter macht die Verwendung eines Polynoms zur Eliminierung von breitbandigen Strukturen und die Vorverarbeitung mit Hoch- und Tiefpaßfiltern, die beide vom Anwender geschätzte Parameter benötigen, überflüssig.
Acknowledgements

During this PhD thesis I learned a lot of new and interesting things and I enjoyed working on this interdisciplinary topic that combines physics and computer science. Therefore I would like to thank my supervisors Prof. Dr. Bernd Jähne and Prof. Dr. Ulrich Platt for giving me the chance to work on this interesting area and for all the ideas and suggestions I got from them. Additionally I want to mention Prof. Dr. Rainer Männer who kindly did the second survey of this thesis.

Many thanks to the people at the IUP for testing all the releases and the great enhancement ideas they suggested. Especially the Satellite-group of Thomas Wagner who were the first audience that had the luck to test and validate the initial releases.

Special thanks go to Alexander Hoffmann, Ralf Kraus, Harald Metzger from Hoffmann Messtechnik GmbH who always were very supportive in providing test environments and hardware components and to Bernd Kromer who I always enjoyed discussing with. Also I would like to thank my colleagues from the IWR, especially Christoph Garbe, Tobias Dierig and Hagen Spies for the fruitful discussions we had about algorithms and implementation details. And Günther Balschbach who worked with me to fight server crashes and network problems.

A big thanks goes out to my parents Erika and Herbert for all of their support and making it possible to study. Last but not least I want to thank my wife Heike for keeping her faith in me all over the years although a lot of nights passed by with endless debugging sessions.

Wiesloch, July 2006
# Contents

1 Introduction

## I Basics

2 DOAS Differential Optical Absorption Spectroscopy
   2.1 Lambert-Beer’s Law ........................................ 6
   2.2 Differential Optical Density ............................. 7

3 DOAS Measurements and Evaluation
   3.1 Artificial Light Source Measurements .................. 11
   3.2 Direct Light and Stray Light Measurements .......... 12
      3.2.1 Satellite Based Measurements ..................... 12
      3.2.2 Light Path Length ................................. 14
      3.2.3 Vertical Column Density and Air Mass Factor ... 15
      3.2.4 Clouds ............................................. 16
      3.2.5 Fraunhofer Spectrum ............................... 17
      3.2.6 Ring Spectrum ..................................... 17
      3.2.7 $I_0$-Effect ...................................... 18
      3.2.8 Undersampling ..................................... 18
   3.3 DOAS Evaluation ........................................... 20
      3.3.1 Dark Current ...................................... 20
      3.3.2 Detector Offset ................................... 20
      3.3.3 Light Source Correction ............................ 21
      3.3.4 Noise Reduction and Broad Band Structures ... 22
      3.3.5 Preparation of the Absorption Cross-Sections ... 23
      3.3.6 Fitting .............................................. 26
II DOASIS

4 DOASIS Framework

4.1 Requirements

4.2 Application Environment
  4.2.1 Java
  4.2.2 .NET Framework

4.3 Performance Consideration

4.4 User Interface
  4.4.1 Concepts
  4.4.2 Comparison

5 Module Structure

5.1 Generic Modules
  5.1.1 Reflection

5.2 Serialization

5.3 Event Model

5.4 Multithreaded Device Drivers

6 Automation

6.1 JScript .NET
  6.1.1 Debugging JScripts

6.2 Running Scripts

6.3 Extending JScript
  6.3.1 DOASIS JScript Extensions

III Analysis

7 DOASIS Evaluation
7.1 Levenberg-Marquardt Nonlinear Fitting 75
   7.1.1 Further Optimizations to Levenberg-Marquardt 80
7.2 Fit Framework 82
   7.2.1 Class Structure 85
   7.2.2 Modular Extensions 87
   7.2.3 Performance Considerations 87
   7.2.4 Numerical Considerations 88
   7.2.5 Evaluation Setup 90
   7.2.6 Performance Analysis and Comparison 91

8 Multiscale Analysis 101
   8.1 Laplace Pyramid 101
     8.1.1 Evaluation Analysis 104

9 Summary And Outlook 111

IV Appendix

A Application Description 115
   A.1 Console Application 115
   A.2 Graphical User Interface 115
     A.2.1 The Main Window 116
     A.2.2 Properties and Statistics 116
     A.2.3 Specbar 117
     A.2.4 Spectrographs and Devices 118
     A.2.5 Console Output 119
     A.2.6 JScripts, MiniScripts and QuickScripts 119
     A.2.7 Fitting 121

B Extend DOASIS 123
   B.1 Menu Structure 123
# List of Figures

2.1 The spectra range of GOME an several absorption cross-section. ............................................. 5  
2.2 Definition of the Optical Density. ......................... 6  
3.1 Schematical DOAS measurement setup with an artifical light source. ................................. 11  
3.2 Principal of satellite based measurements. ......... 13  
3.3 Structure of the orbits recorded by GOME. ........ 14  
3.4 The effects of clouds on the light path length. .... 16  
3.5 Example of the Undersampling Effect. ................. 19  
3.6 Process flow to prepare a literature reference spectrum for a DOAS evaluation. .................... 25  
3.7 Example of a DOAS evaluation. ........................ 28  
4.1 The main window of the DOASIS graphical user interface. .............................................. 42  
4.2 Auto-Hiding Algorithm for menus. ..................... 45  
4.3 Generic event dispatching. .............................. 46  
5.1 DoasCore Class Structure. ............................... 52  
5.2 The namespace dependency structure of DOASIS. .. 55  
5.3 Reflection mechanism. ....................................... 58  
5.4 Event handling mechanism in DOASIS. ................. 62  
5.5 DOASIS dynamic driver proxy creation for non-reentrant drivers. .................................... 64  
7.1 The Levenberg-Marquardt algorithm. ................. 79  
7.2 The DOASIS Fit Framework Structure. .................. 86  
7.3 General usage of the DOASIS Fit Framework. ........ 90
7.4 Comparison of the DOASIS DOAS Fit against the DOASIS DOAS Fit with additional offset polynomial.

7.5 Comparison of the DOASIS DOAS Fit against the DOASIS Intensity Fit.

7.6 Comparison of the DOASIS DOAS Fit against the DOASIS Intensity Fit with additional offset polynomial.

7.7 Comparison of the DOASIS DOAS Fit against the formerly used DOAS Fit of the Satellite Group.

7.8 Comparison of the DOASIS DOAS Fit against the WinDOAS DOAS Fit.

7.9 Comparison of the DOASIS DOAS Fit against the WinDOAS DOAS Fit with additional offset polynomial.

7.10 Comparing the evaluation speed of DOASIS against other DOAS evaluation implementations.

8.1 Example of a Laplace pyramid of a GOME spectrum.

8.2 The Laplace Pyramid algorithm.

8.3 Comparison of the DOASIS DOAS Fit against level three of the Laplace Analysis.

8.4 Comparison of the DOASIS DOAS Fit against level two of the Laplace Analysis.

8.5 Comparison of the DOASIS DOAS Fit against level one of the Laplace Analysis.

8.6 Comparison of the DOASIS DOAS Fit against level zero of the Laplace Analysis.

8.7 Comparison of the DOASIS DOAS Fit against level zero of the Laplace Analysis with an additional binomial low pass filter.

A.1 Command window of the DoasConsole component.

A.2 DOASIS main application window.

A.3 DOASIS menu item and toolbar definition.

A.4 Spectrum properties display.

A.5 The DOASIS Specbar.
A.6  Spectrograph and device control.                  120
A.7  Console output component in the DOASIS GUI.      120
A.8  DOASIS JScript .NET controlling.                  121
A.9  Fit setup dialog.                                 122
F.1   DOASIS running on Linux.                         155
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>Available DOAS Tools</td>
<td>34</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of the user interfaces of the available DOAS tools</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Available file formats of DOASIS.</td>
<td>53</td>
</tr>
<tr>
<td>B.1</td>
<td>Configuration settings to define a parent menu item.</td>
<td>124</td>
</tr>
<tr>
<td>B.2</td>
<td>Configuration settings to define a menu item.</td>
<td>125</td>
</tr>
<tr>
<td>B.3</td>
<td>Configuration settings to define a file type.</td>
<td>126</td>
</tr>
<tr>
<td>B.4</td>
<td>Configuration values to define a new spectrograph device.</td>
<td>128</td>
</tr>
<tr>
<td>B.5</td>
<td>Configuration values to define a new device.</td>
<td>129</td>
</tr>
<tr>
<td>B.6</td>
<td>Possible interfaces to define functional blocks of a device object.</td>
<td>129</td>
</tr>
<tr>
<td>D.1</td>
<td>Available intrinsic data types in JScript.</td>
<td>140</td>
</tr>
<tr>
<td>D.2</td>
<td>Available operators in JScript.</td>
<td>141</td>
</tr>
</tbody>
</table>
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveX</td>
<td>Extended version of COM with support for user interfaces</td>
</tr>
<tr>
<td>AOT</td>
<td>Ahead-Of-Time</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>CIL</td>
<td>Common Intermediate Language</td>
</tr>
<tr>
<td>CLI</td>
<td>Common Language Infrastructure</td>
</tr>
<tr>
<td>CLR</td>
<td>Common Language Runtime</td>
</tr>
<tr>
<td>CLS</td>
<td>Common Language Specification</td>
</tr>
<tr>
<td>COM</td>
<td>Component Object Model</td>
</tr>
<tr>
<td>CTS</td>
<td>Common Type System</td>
</tr>
<tr>
<td>DOAS</td>
<td>Differential Optical Absorption Spectroscopy</td>
</tr>
<tr>
<td>DOD</td>
<td>Differential Optical Density</td>
</tr>
<tr>
<td>FIR</td>
<td>Finite Impulse Response</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>HMI</td>
<td>Human Machine Interface</td>
</tr>
<tr>
<td>I/O</td>
<td>Input/Output</td>
</tr>
<tr>
<td>JIT</td>
<td>Just-In-Time</td>
</tr>
<tr>
<td>MDI</td>
<td>Multiple Document Interface</td>
</tr>
<tr>
<td>OD</td>
<td>Optical Density</td>
</tr>
<tr>
<td>PMD</td>
<td>Polarisation Monitoring Device</td>
</tr>
<tr>
<td>SCD</td>
<td>Slant Column Density</td>
</tr>
<tr>
<td>SDI</td>
<td>Single Document Interface</td>
</tr>
<tr>
<td>SDK</td>
<td>Software Development Kit</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>UML</td>
<td>Unified Modeling Language</td>
</tr>
<tr>
<td>VCD</td>
<td>Vertical Column Density</td>
</tr>
<tr>
<td>VES</td>
<td>Virtual Execution System</td>
</tr>
</tbody>
</table>
1 Introduction

Environmental monitoring gets more and more important in todays societies. Especially the effects caused by air pollution like global warming and other climate changes are of major interest and require an observation of the so called Greenhouse gases like $CO_2$ and $CH_4$.

To get an overview about the global distribution, satellite based measurements are required. But for local identification of pollution sources, ground based measurements are used. Preferable the method used for both kinds of measurement should be the same to get comparable results.

One of the methods to archive this is the Differential Optical Absorption Spectroscopy (DOAS) which will be explained in detail in chapter 2. DOAS allows the measurement of species concentrations in a non-destructive way by measuring the absorption of light at different wavelengths when it passes through the measurement media. The evaluation of the measured Optical Densities requires several preprocessing and post processing steps and a non-linear fitting algorithm based on the Lambert-Beer’s Law which describes how light is absorbed by the different species.

The major part of this work will focus on the evaluation algorithm and the requirements to acquire, prepare and evaluate spectral data and how the evaluation process overall could be improved in algorithmic and software engineering manners. Up to now the parts of a DOAS measurement were done in separate steps using several little programs that do a small part of the acquire and evaluation process. This causes a very inefficient work flow until the desired results get extracted. To overcome these limitations in chapter 4 a framework called DOASIS will be introduced which combines all required measurement and evaluation steps into one process and gives a basis which can easily be extended to new needs of algorithmic, data management and hardware demands.

Many users of the DOAS method have quite different needs to their measurement setup and therefore have different processing and evaluation procedures. In chapter 6 the automation support of the framework will be explained and how it integrates into the framework itself and other components. Scripting is done based on the JScript syntax which is easy to understand and even unexperienced users can setup their measurement functions quickly.
Another important task was flexibility of the framework. It should be possible to add functionality provided by the user without the need to modify the framework at all. Even yet still unknown devices and evaluation methods can be added using reflection techniques as shown in chapter 5. New device drivers for spectrometers and other hardware components can easily be added through a common interface that ensures to create the necessary UI components automatically. Multiple instances of the same driver can be created to enable concurrent access to several devices of the same type at the same time although the device driver is only capable of controlling a single device.

But also the algorithms were improved in several ways. Especially the evaluation process got enhanced in several ways like explained in chapter 7. A flexible way is introduced to change the model used to extract the required information from the measured spectra. This evaluation procedures still fits the need of fast data processing but also provides a structured, object orientated environment.

Finally in chapter 8 a new evaluation algorithm will be introduced to overcome several heuristically determined parameters of the standard analysis using a multiscale preprocessing of the spectral data.

In appendix A detailed informations about the user interfaces will be given. Appendix B and C describe how to extend the functionality of the DOASIS framework and how to exchange data with other applications. A quick guide through the JScript scripting capabilities will be given in D and a brief overview about the available documentation can be found in E. Finally in appendix F informations about support of other operating systems can be found.
Part I

Basics
2 DOAS Differential Optical Absorption Spectroscopy

One of today’s methods to monitor air pollution and observe local and global atmospheric changes is the Differential Optical Absorption Spectroscopy (DOAS) [Platt, 1994]. DOAS measurements can be done quite easily. A single spectrometer that measures either sunlight or the light of an artificial light source that has passed through a certain gas volume at a given spectral wavelength range, is sufficient for a simple measurement. Of course in reality it is not that simple. There is still the need to solve several optical and electrical problems. But basically a spectrum is recorded.

Unfortunately the spectral data obtained from a spectrograph does not directly reflect the desired information. To get the concentration of several species within a given gas volume it is necessary to know the effects of matter to electro-magnetical rays and therefore how the species change the light that passes by. This behavior is called absorption and is described in the Lambert-Beer’s Law.

![Figure 2.1: The spectral range of GOME and several examples of species absorptions present in this range.](image)
2.1 Lambert-Beer’s Law

The Lambert-Beer’s Law describes how the light intensity at a certain wavelength is weakened when it passes through a layer of substances. The source intensity $I_0(\lambda)$ is absorbed by means of the integrated substance’s concentration $c(s)$ along the light path $s$. The wavelength dependent absorption strength of the substance is given by the so called cross-section $\sigma(\lambda, T)$, which defines the light absorption at a specific wavelength $\lambda$ and temperature $T$ in an infinite thin layer. The absorption structures of several species are shown in figure 2.1.

$$I(\lambda) = I_0(\lambda)e^{-\sigma(\lambda,T) \int c(s)ds} \quad (2.1)$$

The cross-section of a specific substance can be derived from literature data or from lab measurements. For later evaluation it is essential to know the cross-sections as accurate as possible.

**Figure 2.2:** Definition of the Optical Density. The Optical Density is defined by the logarithm of the fraction between the light source’s intensity $I_0(\lambda)$ and the measured light intensity $I(\lambda)$. The Differential Optical Density is defined as the fraction between the interpolated source intensity $I^*(\lambda)$ and the measured intensity $I(\lambda)$. 
Often the concentration \( c(s) \) and the light path length \( l = \int ds \) are not known by default. Therefore the concentration \( c(s) \) and the light path length \( l \) are combined to the Slant Column Density \( S \).

\[
S = \int c(s) ds
\] (2.2)

The \( S \) describes the substance’s concentration within the whole volume that is described by the light path. So if we know the light path length exactly, we can determine the substance’s concentration directly from the \( S \). Unfortunately the exact light path length is only known in static measurement setups. For other measurement setups like satellite observation more sophisticated methods are required to estimate the light path length.

The absorption given by equation (2.1) only describes the absorption of a single substance. In real world measurement we have a lot of absorbing substances that influence the light simultaneously. Fortunately the absorption of several substances \( i \) can be expressed by the sum of their absorption cross-sections \( \sigma_i(\lambda, T) \) in regard to their specific concentration \( c_i(s) \) respectively their \( S_i \).

\[
I(\lambda) = I_0(\lambda)e^{-\sum \sigma_i(\lambda, T) S_i}
\] (2.3)

This equation shows that to determine the \( S \) of a substance the light source’s intensity \( I_0(\lambda) \) needs to be known very well. However, the light source’s spectrum is often not known, thus, another approach is needed to evaluate the measured spectra without any knowledge.

### 2.2 Differential Optical Density

First of all a measure for the absorption strength needs to be defined. Therefore we define the Optical Density (OD) which is the logarithm of the ratio between the light source’s intensity \( I_0(\lambda) \) and the measured intensity \( I(\lambda) \).
\[ D(\lambda) = \ln \frac{I_0(\lambda)}{I(\lambda)} \] (2.4)

To overcome the problem of the unknown light source intensity the Lambert-Beer’s Law needs to be modified in a way that the absorptions and respectively the concentrations of substances are described in a way without an exact knowledge of \( I_0(\lambda) \). A closer look at the spectral data shows that the relevant information about the absorption strength is only contained in the high frequency structures of the spectrum. The low frequency parts are defined by the light source itself and influences from scattering. Therefore we split the spectral data into a low frequency part \( \sigma^B_i(\lambda) \) and a high frequency part \( \sigma'_i(\lambda) \).

\[
\sigma(\lambda) = \sigma^B_i(\lambda) + \sigma'_i(\lambda)
\] (2.5)

Another problem of equation (2.3) is the fact that none of the \( S_i \) is known and all of them need to be determined by the measured data. Therefore equation (2.3) is rewritten as a set of linear equation which looks like:

\[
\ln I(\lambda) = \ln I_0(\lambda) - \sum \sigma_i(\lambda, T) S_i
\] (2.6)

Combining equation (2.5) and equation (2.6) leads to:

\[
\ln I(\lambda) = \ln I_0(\lambda) - \sum \sigma^B_i(\lambda, T) S_i - \sum \sigma'_i(\lambda) S_i
\] (2.7)

The low frequency part, which consists of the light source’s intensity \( I_0(\lambda) \) and the low frequency part of the absorption cross-section \( \sigma^B_i(\lambda) \), can be modeled by an appropriate set of basis functions \( \beta_j(\lambda) \).

\[
\sum \alpha_j \beta_j(\lambda) = \ln I_0(\lambda) - \sum \sigma^B_i(\lambda) S_i
\] (2.8)
where the set of basis functions if commonly represented by a polynomial \( P(\lambda) \) of degree \( m \).

\[
P(\lambda) = \sum_{j=0}^{m} a_j \lambda^j = \sum_j \alpha_j \beta_j(\lambda)
\] (2.9)

The linear equation system from equation (2.6) does no longer need the light source’s intensity \( I_0(\lambda) \) anymore and can be written as:

\[
\ln I(\lambda) = -\sum_i \sigma'_i(\lambda) S_i + \sum_{j=0}^{m} a_j \lambda^j
\] (2.10)

which can easily be solved for the \( S_i \) and polynomial coefficients \( a_j \) using a linear least square method.

Since the absorption no longer depends on the absolute knowledge of the source intensity \( I_0(\lambda) \) we define the Differential Optical Density now:

\[
D'(\lambda) = \ln \frac{I_0^B(\lambda)}{I(\lambda)}
\] (2.11)

Using a single measured spectrum several species’ concentrations can be determined at once. So DOAS is a very powerful method for concentration measurements in a high spatial and temporal resolution. Figure 2.2 shows the dependencies between \( I_0(\lambda) \) and \( I(\lambda) \) and how the Optical Density and the Differential Optical Density are defined.
DOAS Differential Optical Absorption Spectroscopy
3  DOAS Measurements and Evaluation

Now that we have the tools to evaluate the measured spectra data, we should have a look at some measurement setups and discuss their advantages and problems. The different measurement methods require different preparation of the recorded data and lead to different evaluation problems. The following section will describe the different measurement methods and point out some extensions required to successfully evaluate and improve accuracy of the measured spectra.

3.1 Artificial Light Source Measurements

Basically DOAS was developed to measure the concentration of air pollutants in urban areas. For this kind of measurement a fixed device setup can be used as shown in figure 3.1.

An artificial light source is used in this setup. This has the advantage that several scattering effects do not need to be taken into account during evaluation. Also the structure of the light source is normally well known and can therefore directly be removed from the measured spectrum. Other advantages of an artificial light source are the well known light path and a higher independence from weather conditions especially during night time measurements. Also very localized measurements can be done using this kind of setup.

But there are some disadvantages, too. To outshine the surrounding stray light very bright light sources are required. This high light intensity and relatively short light path increase the detection limit due to a higher photon
noise whereas only very small absorptions are present. In any case at least two measurement sites are required. On one side the detector and on the other side either the light source itself or a reflector.

3.2 Direct Light and Stray Light Measurements

In cases where it is not possible to use an artificial light source stray light measurements are done. Stray light measurements use as light source the sun. So measurements through the whole atmosphere can be done using just a detector.

The difference between direct light and stray light measurements is the viewing angle of the detector. While in direct light measurements the detector 'looks' directly into the light source in stray light measurements only the light scattered due to Mie- and Rayleigh-Scattering is recorded.

The direct light measurement setup looks more or less the same like a setup with an artificial light source except that as light source normally the sun is used. Stray light measurements have the advantage of a much longer light path and therefore of stronger absorptions. This reduces the detection limit and increases the measurement accuracy and even very weak absorbers can be detected.

3.2.1 Satellite Based Measurements

One of the most established applications of a stray light measurements are satellite based setups. The satellite scans over the earth’s surface in orbits. Each orbit is build up of pixels of a certain height and width and it contains the spectra information of the light reflected by the appropriate region on the surface. Several orbits are recorded until the whole surface has been scanned. Due to this set of orbits global distribution maps for different kinds of species like NO$_2$, CO$_2$ or BrO can be generated [Wagner and Platt, 1998] and spatial-temporal analysis algorithms can be applied [Wenig, 2001].

The most famous DOAS application from satellite so far is the Global Ozone Monitoring Experiment (GOME) [Hahne et al., 1995] aboard the ERS-2 satellite [ESA, 1995a], which was launched in April 1995. It operates at a height of 780km on a sun-synchronous orbit. The orbit’s pixels are build up by three individual spectral scans which cover a ground area of 320km
Figure 3.2: Principal of satellite based measurements. The light gets emitted by the sun and is reflected by the Earth’s surface. The satellite looks on the surface and records the reflected light which has passed through the atmosphere.

(east-west) by 40km (north-south). Figure 3.3 shows how GOME scans the Earth’s atmosphere ([ESA, 1995d]. The device looks nearly perpendicularly to the surface. This mode is called nadir. Of course spectral data can only be recorded on the day-light side of the earth. Therefore about three days are needed to obtain a global coverage.

GOME scans the earth’s surface in orbits. Each orbit consists of a set of horizontal scans. During each scan three pixels are recorded by the four spectrometers which operate in the ranges of 240nm-316nm, 311nm-405nm, 405nm-611nm and 595nm-793nm. While the spectrometer is moved back to its starting position for the next three pixels, a backscan pixel is recorded as well. Additionally 16 Polarisation Monitoring Device (PMD) pixels, which are used to get a visible representation.

In March 2002 another, more advanced instrument was launched: The SCanning Imaging Absorption SpectroMeter for Atmosperic CHartographY (SCIAMACHY) on board ENVISAT [ESA, 2002]. Compared to GOME it has an extended spectral range which allows the detection or further species like the Greenhouse gases $CH_4$ and $N_2O$. Also the spatial resolution has
been increased and besides the nadir operating mode the limb mode has been added. In limb mode the device operates like it is done in direct light measurements. It 'looks' through the atmosphere tangentially on a light source which can not only be the sun but other stars, too. Combining these two operating modes SCIAMACHY can distinguish between the troposphere and the stratosphere whereas GOME basically covers the stratosphere mainly.

### 3.2.2 Light Path Length

Stray light measurements and especially satellite measurements lead to several other evaluation problems. While in direct light measurements the light path is quite well known, the length of the light path in stray light measurements highly depend of the number of scattering events that occurred to the photon until it reaches the detector.
Since the accuracy of an evaluation highly depends on the knowledge of the light path length several methods are available to, at least, get the statistical distribution of the light path length. Most of these methods are based on Monte-Carlo models. The path of a single photon through the atmosphere is simulated by modeling the geometric structure and the possible scattering events that may occur.

To get valid statistics a lot of photons need to be simulated. Unfortunately this is a very time consuming task.

### 3.2.3 Vertical Column Density and Air Mass Factor

As given in equation (2.3) the DOAS method gives us the Slant Column Density (SCD), which is the integrated concentration of a substance along the light path. The Slant Column Density is not an appropriate measure to be compared to other results, since it depends on the measurement setup and therefore in stray light measurement on the viewing angle of the detector and the so called Solar Zenith Angle (SZA).

The SZA describes the angle between the zenith and the current position of the sun in regard to the current detector position. With higher SZA the Slant Column Density increases, since the light path length increases, too. Figure 3.2 shows the dependencies of SCD, VCD and SZA.

To get the concentration independent of the viewing geometry respectively the SZA, the Vertical Column Density (VCD) is used. The VCD describes the species concentration within the air column directly above the detector when looking perpendicular through the air. There is a linear dependency between the Slant Column Density $S_i$ and the Vertical Column Density $V_i$. This dependency is described by the Air Mass Factor $A_i$ for a given species $i$, wavelength $\lambda$ and Solar Zenith Angle $\theta$.

$$V_i = \frac{S_i(\lambda, \theta)}{A_i(\lambda, \theta)} \quad (3.1)$$

The Air Mass Factor (AMF) depends highly on the current Solar Zenith Angle due to the effects of the SZA to the Slant Column Densities. Besides the geometric influences on the light path the AMF needs to take into
account several other effects like scattering at aerosols. Therefore the AMF depends on the species and wavelength, too.

### 3.2.4 Clouds

Clouds have tremendous effect on the light path length and the Air Mass Factor. Light that passes through a cloud gets scattered several times resulting in an increased absorption strength [Wagner et al., 1997]. The Monte-Carlo Models mentioned in section 3.2.2 have been extended to get the distribution of the light path length at different cloud heights.

![Diagram of cloud effects](image)

**Figure 3.4:** The different effects of clouds on the light path length. *a)* The light gets scattered inside the cloud, but still passes through it. *b)* The light gets scattered inside the cloud and never passes through the atmosphere below the cloud. *c)* The light directly gets reflected by the top of the cloud.

But clouds do not only affect the light path length. Most of the light is just reflected by the cloud’s top shielding any atmosphere below it. Figure 3.4 examplifies the different light path scenarios when clouds are in place.

Remembering the size of a ground pixel of the GOME instrument it is very likely that a pixel is partly covered by clouds. So the evaluation results
are always missing some parts of the troposphere which is shielded by a cloud. At least to make assumptions about concentrations in the troposphere algorithms have been developed to estimate the cloud coverage and its influence on the evaluation. [Wenig, 1999] estimates a cloud-free pixel by averaging data over a long period of time. This cloud-free data will then be used as reference to estimate the concentrations of a substance below a cloud. Other approaches use the knowledge about the $O_3$ distribution in the atmosphere. Knowing the current $O_3$ concentration the top height of the cloud can be determined and therefore correction factors for the missing tropospheric data can be estimated.

### 3.2.5 Fraunhofer Spectrum

Another effect on stray light measurements is caused by the light source itself. Since in most stray light measurements the sun is used as light source, the measured spectrum already contains absorption structures even if no atmosphere is in the light path. The sun’s photosphere absorb parts of the light emitted by itself. These absorptions are called Fraunhofer lines.

To correct these unwanted absorptions during a measurement a Fraunhofer spectrum is recorded by directly looking into the sun with the detector. The correction is done by either dividing the measured spectrum by the Fraunhofer spectrum to eliminate the light source structures or by using the Fraunhofer spectrum as an additional cross-section reference during evaluation.

### 3.2.6 Ring Spectrum

But the measured Fraunhofer lines are not as sharp as expected. Due to Raman-scattering in the atmosphere the Fraunhofer absorption lines get ‘filled up’. This effect is called Ring-effect and needs to be corrected during evaluation as well [Bussemer, 1993]. Normally a Ring spectrum is calculated from the reference Fraunhofer spectrum already recorded. This Ring spectrum will then be used as another additional cross-section reference.
3.2.7 \( I_0 \)-Effect

The Fraunhofer spectrum has many narrow absorption lines which often overlap with absorption structured of other species. So the Fraunhofer spectrum does not fulfill the assumption from equation (2.7) that the light source only consists of low frequency, broad band structures and it is very likely that the Fraunhofer structures overlap with absorption structures of other substances. This problem is called \( I_0 \)-effect since it arises from a problematic light source \( I_0 \).

3.2.8 Undersampling

Detector pixels do an integration over a certain wavelength range. So very narrow absorption lines cannot be recorded correctly if the resolution of the detector is below the Nyquist-frequency of the absorption line. If the sampling theorem is not met, the original signal cannot be reconstructed correctly. So undersampling a spectrum at different wavelength grids will result in different representations that cannot be transformed back to the original signal. For DOAS evaluation this is normally not a problem, since both the measurement spectrum and the reference spectrum are recorded on the same device, thus contain the same error.

If at least one of the reference spectra is not sampled on the same wavelength grid as the undersampled measurement spectrum, a certain error is introduced to the DOAS evaluation. Figure 3.5 tries to demonstrate the error that can appear when a signal gets undersampled at slightly shifted samples. But why does a wavelength recalibration of the reference spectrum not fix this problem? Recalibration means to shift and stretch the data using some sort of interpolation method. Since the signal is only available in an undersampled representation, the interpolation method cannot correctly reconstruct the original signal and the recalibrated reference spectrum will have a certain error compared to spectra directly recorded on the correct wavelength grid.

It is not always possible to record the reference spectra using the same device as for measurement. For example, in the GOME instrument the solar reference spectrum is recorded on a slightly shifted wavelength grid due to technical issues and a Doppler shift introduced by the way the solar reference gets recorded and, unfortunately, GOME undersamples the Fraunhofer
lines. Additionally wavelength shifts occur due to temperature drifts over one orbit, while the solar reference is only recorded once for each orbit.

[Slijkhuis et al., 1999; Chance et al., 2005] propose to add an *Undersampling Correction* (UC) cross-section to the DOAS evaluation. The Undersampling Correction cross-section is generated by convoluting a high resolution spectrum at two different, undersampled wavelength grids using the slit function of the instrument. The second resulting spectrum is shifted back to the wavelength grid of the first spectrum using cubic spline interpolation. The UC cross-section is the difference between the first and the second spectrum.

This method gives an estimated about the error introduced to a DOAS evaluation by the interpolation method used to shift reference spectra that do not exactly match the measurement spectrum wavelength calibration.

*Figure 3.5:* The given original signal is sampled using 8 pixels on two different grids $\alpha$ and $\beta$. The second grid $\beta$ is shifted by half a pixel against grid $\alpha$. Since the original signal was undersampled, the signal reconstructions of both samples using a cubic spline interpolation differ compared to the original signal and compared to each other. To accommodate these differences in a DOAS evaluation, the second sample is shifted and resampled to grid $\alpha$. The difference between the two samples is the Undersampling Correction cross-section.
3.3 DOAS Evaluation

Before we can use the equation system (2.10) to determine the Slant Column Densities of several substances, the measured data needs to be prepared very precisely. Low detection limits can only be archived, if all influences caused by the detector, the light source and the whole measurement setup are eliminated. The following steps will describe several preprocessing steps that may need to be done. Of course, depending on the measurement setup it is not always possible to have the required data and therefore some corrections cannot be done. Normally this should only have an effect on the resulting errors respectively the detection limit achieved.

3.3.1 Dark Current

The dark current reflects the structures caused by the electronics of the detector itself. These structures can be recorded quite easily by recording a spectrum while the detector does not receive any light. The so recorded Dark Current spectrum \( D(\lambda) \) can be subtracted from the measured spectrum whereas the integration time, which is the product of the number of scans done and the exposure time of a single scan, needs to be taken into account.

\[
I_c(\lambda) = I(\lambda) - \frac{i_i}{d_i} D(\lambda)
\]

(3.2)

where

\( i_i \): Integration time of measured spectrum

\( d_i \): Integration time of dark current spectrum

The dark current spectrum is recorded using long integration times to reduce the effects of noise in the signal.

3.3.2 Detector Offset

The detector has a certain offset level when it samples data. The values of each pixel is actually not zero if no light hits the pixel but it has a certain value defined by the detector electronics. This offset is necessary to avoid
problems with the analogue-digital converts when values near zero should be sampled.

The Offset spectrum $O(\lambda)$ is recorded using very short exposure times and can just be subtracted like the Dark Current spectrum. But since the offset is independent from the integration time and only occurs for each scan once, the correction factor is given by the ratio between the number of scans of the measured spectrum and the offset spectrum

$$I_c(\lambda) = I(\lambda) - \frac{i_s}{o_s}O(\lambda) \quad (3.3)$$

where

- $i_s$: Number of scans in the measured spectrum
- $o_s$: Number of scans in the offset spectrum

### 3.3.3 Light Source Correction

Since the DOAS assumption that the structures of the light source do only consist of broad band structures is not always valid, it is essential to remove any kind of absorption structures that come from the light source. Therefore, first of all a spectrum is recorded by directly looking into the light source. During long term measurements this should be done at regular intervals, since the structure of the light source may change during time.

The influences of the light source can be removed in two ways now:

1. Divide the measured spectrum by the light source spectrum.
   In this case special care needs to be taken about noise in the signal. Dividing by noisy structures may create unwanted absorption structures in the resulting spectrum.
2. Use the light source spectrum as an additional absorption cross-section during evaluation.
   Especially when the light source has a lot of high frequency absorption structures, like it does when using a Fraunhofer spectrum, it’s more practicable to use the light source spectrum as an additional cross-section during evaluation. The SCD of the light source spectrum should be near one then.
3.3.4 Noise Reduction and Broad Band Structures

Any kind of measurement always contains a certain amount of noise either generated by the device itself or due to the photon noise statistics. Although the noise contained in a measured spectrum should not have any influence on the evaluation result, especially when detecting very weak absorbers it is convenient to try to remove or reduce the amount of noise in the spectrum.

Noise can be reduced by doing some sort of smoothing which is done using a low pass filter to remove the higher frequency parts of the spectrum. Several methods can be used to do the low pass filtering. Here are just a few possibilities:

- **Polynomial Regression**
  A polynomial of degree $m$ is fitted to the data. The resulting polynomial describes the low pass filtered spectrum. The fit can be done using a least squares method to determine the polynomials coefficients.

  $LF(I(\lambda)) = \sum_{i=0}^{m} \gamma_i \lambda^i$  \hspace{1cm} (3.4)

- **Binomial Mask Filtering**
  A binomial mask of a certain width is applied to the data. The definition of the binomial coefficients is given in [Bronstein and Semendjajew, 1989]. In detail a $B^1$ mask is applied $n$-times which equals a binomial mask $B^n$ but without the need to create a mask of the appropriate width. Smoothing with a binomial mask is similar to a convolution with a Gaussian distribution curve.

  $LF(I(\lambda)) = I(\lambda) \ast B^n$  \hspace{1cm} (3.5)

- **Savitzky-Golay Filter**
  The Savitzky-Golay filter is a special case of the polynomial regression filter. Instead of fitting one polynomial to the whole data set at once, for each sample a polynomial fit against the next $n$ neighbors of the current sample is done. The result of the polynomial at the current data point is then taken as the new filtered data sample. So for $m$ data samples $m$ least square fits would be necessary. But instead a FIR
filter can be constructed that does the polynomial fitting and evaluation automatically, thus the filter reduces to a simple convolution with the filter mask [Press et al., 1995].

Since DOAS requires a separation of the measured data into a low frequency part and a high frequency part, a removal of the unwanted low frequency part is wanted. To achieve this a high pass filter is applied to the measured spectrum. Its purpose is to remove all the low frequency parts that are not required for evaluation. The high pass filter is done by doing a low pass filter on the spectrum and dividing the spectrum with its low pass filtered variant.

\[
HF(I(\lambda)) = \frac{I(\lambda)}{LF(I(\lambda))}
\]

The remaining low frequency structures are covered by the polynomial used during evaluation.

### 3.3.5 Preparation of the Absorption Cross-Sections

Special care needs to be taken about the species cross-sections used for evaluation. The accuracy of the whole evaluation depends on the accuracies of each cross-section. To get the information about the absorption structures and strengths of a specific species, two methods can be used:

- **Record a cross-section.**
  This is the easiest method to get a cross-section. A probe of the desired substance is put into the direct light path of the device and a spectrum is recorded. The probe needs to be of a well known concentration. Also the light path length should be well known. The resulting SCD for the given cross-section needs to be corrected by:

\[
\sigma(\lambda) = \frac{\tau(\lambda)}{c_r I_r} \quad (3.7)
\]

\[
S = S_c c_r I_r \quad (3.8)
\]
where $\tau(\lambda)$ is the cross-section recorded and $c_r$ and $l_r$ are the well known measurement conditions.

The major advantage of this method is that the reference spectrum matches exactly the device’s transfer function. Any kind of sampling effects, blurring or detector noise are already present in the reference spectrum.

Of course, the structures of the light source still need to be removed from the reference spectrum as described in the previous section.

- **Use a literature spectrum.**

The absorption structures of many species are available as high resolution cross-sections. These cross-sections are available in several databases or from literature. In many cases where it is not appropriate to record a reference spectrum like in satellite based measurements, using these high resolution spectra is the only way to do a successful evaluation. However, the high resolution spectra cannot directly be used for a DOAS evaluation. It is necessary to convolute the cross-sections with the transfer function of the device used to record the measurement spectra. Any kind of distortions and sampling effect that occur to the recorded spectra should be represented by the transfer function as well.

Therefore the transfer function is determined by recording a very sharp spectral line like a Hg-spectrum has several of it. The shape of the line describes the transfer function of the device. Assuming an optimal transfer function of the detector, its shape should be similar to a Gaussian function. Using this assumption a Gaussian curve that fits the recorded spectral line best is used as 'ideal' transfer function for convolution.

Unfortunately the transfer function is not constant over the whole spectral range. Normally due to optical effects the transfer function is only symmetric in the middle of the spectral range of the detector. Towards the borders the transfer function gets more and more asymmetric. So a transfer function is recorded at several location within the spectra range of the detector and the convolution is then done with different transfer functions at different spectral intervals. The Gaussian function used for convolution either is kept constant over an interval or gets interpolated linearly.

Also there are methods available that try to describe the asymmetric transfer functions using a Voigt Function which’s parameters are described by a polynomial over the spectral range of the detector. The Voigt function can describe the changes in the symmetry smoothly.
If it is not possible to record a sharp spectral line, the transfer function is determined by doing a deconvolution of a recorded Fraunhofer spectrum. At least the convoluted cross-sections needs to be resampled to the wavelength mapping of the measured spectrum as shown in figure 3.6.

**Figure 3.6:** Process flow to prepare a literature reference spectrum for a DOAS evaluation. The aperture function of the device is determined by recording a Hg spectrum. One of the Hg lines in the spectrum is selected as convolution kernel. The convolution kernel needs to be normalized and offset corrected so it can be used for convolution with the high resolution literature spectrum.

As soon as a cross-section spectrum is available for a specific substance, an evaluation can be done.
3.3.6 Fitting

When all cross-section and the measured spectrum are prepared, the evaluation can be done. As shown in equation (2.10), the DOAS evaluation reduces to a simple least squares fit. However, this requires a constant wavelength mapping of all spectra involved, which is not necessarily given [Stutz, 1996].

Due to temperature drifts and other influences in the optical parts and the detector itself the spectral data may get shifted and squeezed or stretched over time. For example when a satellite moves from the dark side of the earth to the daylight side, the temperature rises by more than 100 degrees. Of course it is tried to stabilize the measurement conditions. But not every distortion of the measurements can be eliminated.

The drifts in the spectral data can’t be ignored during evaluation especially when the absorption structures are only few channels wide. For example, assuming an absorption structure of Gaussian shape that has a Full-With-Half-Height of 1 channel, a shift of the measured spectrum of one channel in regard to the wavelength calibration of the cross-sections would falsify the result by 20%.

Therefore we need to extend the DOAS evaluation to be more flexible about calibration errors:

\[
\ln I(\lambda) = - \sum_i S_i \sigma_i(\Gamma_i(\lambda)) + P(\lambda) \quad (3.9)
\]

where

\[
\Gamma_i(\lambda) = \sum_{k=0}^{m} \gamma_k \lambda^k \quad (3.10)
\]

is a polynomial of degree \( m \) that corrects the calibration errors of each cross-section \( \sigma_i \).

Normally it is sufficient to have a degree of 1 for the correction polynomial \( \Gamma(\lambda) \) which is equal to a spectral shift \( s_i = \gamma_0 \) and a squeezing or stretch-
ing factor \( t_i = \gamma_1 \) for each cross-section \( \sigma_i \). Equation (3.10) can then be rewritten as

\[
\ln I(\lambda) = - \sum_i S_i \sigma_i (s_i + t_i \lambda) + P(\lambda)
\] (3.11)

Unfortunately this extension leads to another problem. It is no longer possible to have a set of linear equations and use a simple least squares method to solve these equations. The calibration correction introduces a set of nonlinear parameters of \( s_i \) and \( t_i \), which need to be determined during evaluation as well.

Currently an iterative method based on the Levenberg-Marquardt algorithm is used to solve this nonlinear problem, which will be explained in detail in chapter 7.1. Due to the nature of nonlinear fitting algorithms the result is not necessarily correct. The algorithm may not find the global optimum like the linear least squares method does, since it tries to get closer to the solution by estimating a parameter gradient in every iteration step depending on the former iteration. A new parameter set is expected to be more accurate when the sum of the squared differences

\[
\chi^2 = \int \left[ \ln I(\lambda) + \sum_i S_i \sigma_i (\Gamma_i(\lambda)) - P(\lambda) \right]^2 d\lambda
\] (3.12)

between the measured data and the cross-sections gets smaller respectively \( \chi^2 \rightarrow 0 \). The algorithm terminates when the \( \chi^2 \) does not improve by a certain amount \( \epsilon \) anymore

\[
\chi_i^2 - \chi_{i-1}^2 < \epsilon
\] (3.13)

or a maximum number of iterations is reached. So the algorithm terminates always and reports a result that is expected to be optimal although it only represents a local minimum of the \( \chi^2 \)-error.
Also it is necessary to have independent parameters. Any dependency in the model parameters may produce ambiguous results and prevents the algorithm to continue evaluation. Figure 3.7 shows an example DOAS evaluation where several cross-sections are fitted against a measured spectrum.

![Figure 3.7: Example of a DOAS evaluation. The measured spectrum is separated into the components represented by the cross-sections included in the evaluation. The results show the fitted reference overlapped with the residual to get a visible feedback about the fit results quality.](image)

To reduce the risk of a false result, several enhancements to the algorithm have been made. Most of them result in reducing the number of nonlinear parameters and therefore in stabilizing the result. A more detailed description of the evaluation algorithm will be given in section 7 where new extensions and enhancements introduced by this work are explained.

### 3.4 Evaluation Problems

The more complex an evaluation scenario gets the more likely the evaluation fails for some reason. Problems can arise at any stage of a DOAS evaluation.
starting at controlling the detector and recording a spectrum and ending at
the DOAS fit.

First of all the handling of the different kinds of detectors used to do a DOAS
measurement is a source for problems. The different kinds of detectors
require different kinds of hardware to support. Most of the detectors are
shipped with controlling software from the manufacturer. But this software
is often not feasible for an automatic measurement and is often hard to
adapt to certain measurement setups as well as it is not convenient to
prepare a measurement separately for every kind of hardware used.

The next source of problems may arise during the preparation of the neces-
sary species’s cross-sections. For the different kinds of preparation methods
already little software packages exist. But this requires to convert the data
depending on the format that is supported by a specific tool as well as the
configuration of these different tools. These steps are a very fault-prone
task. During conversion information may get lost or due to a falsely con-
figured tool errors creep in notelessly.

At least the DOAS fit is one of the biggest sources for evaluation prob-
lems. Due to the more and more complex evaluation scenarios used in an
evaluation, the number of unknown parameters increases and the nonlinear
evaluation algorithm gets more and more unstable. Of course the algorithm
itself will always produce stable and correct results, if the parameters are
independent from each other. Unfortunately the cross-sections often con-
tain similar structures that can cause dependencies in the parameters. Or
a reference spectrum contains periodic structures and it is not clear if a
positive or negative shift is the correct way to correct calibration errors.

The DOAS method is based on a separation of the signal into a high and low
frequency part. However, it is not clearly defined how this separation should
be done. According to the DOAS definition from equation (2.7) The high
frequency part must contain all frequencies that describe the absorption
structure of a substance. Currently the separation is done using a high-
and low-pass filter. The filters need to be tuned manually by the user.

The polynomial used during evaluation represents the remaining broad band
structures that are not required for the evaluation but are still present in
the data even after the high-pass filtering. The order of this polynomial
needs to be defined by the user manually as well. Thus choosing a too
low order for the polynomial increases the residual of the evaluation and
therefore the result’s error and inaccuracy. On the other hand a too high polynomial order may create a polynomial which’s structures overlap with absorption structures of one or more cross-sections.

The choice of these heuristic parameters gets more important the more cross-sections are used during evaluation. Especially if these cross-sections contain both very broad band structures and very narrow high frequency absorptions.

The focus of this work is to improve the DOAS measurements and evaluation overall. Therefore first of all a framework was developed to combine all parts of a DOAS measurement into a single packet. So all steps from recording the data up to the evaluation should be done in a single process without the error-prone handling of several different preparation steps. One focus was to get a user friendly interface so even beginners can quickly do their measurements and evaluations. But on the other side the framework should be easily extensible to support new devices, measurement setups and algorithms without the need to adapt the whole evaluation process to the new needs.

Several additions have been made to the evaluation algorithm itself to increase the stability of the evaluation. The more complex parameter handling allows to define several restrictions to avoid ambiguities and ensure proper results. The evaluation itself can easily be exchanged with other optimization algorithms. Support for more complex evaluation models is included as well as a new evaluation method that does the separation of the signal into high and low frequency parts using a set of band bass filters to avoid the heuristic filters used currently.

The following chapters will describe these enhancements and point out their advantages as well as disadvantages.
Part II

DOASIS
4 DOASIS Framework

The DOAS analysis requires several steps to get from the measured spectrum to the concentrations of the desired substances. Current solutions for this work-flow only provide a very fixed set of procedures to perform the required tasks.

Some tools have support for one or two kinds of detectors. Automated measurements can be defined by a proprietary scripting language. Other tools are required to do the preprocessing of the spectral data. Depending on the evaluation method required, even another program is needed. To adapt the evaluation process to the different needs, some groups even use specialized data processing and evaluation software with hard coded application flow.

All of these tools available have another big disadvantage: Every change required to one of the evaluation steps needs to be integrated and implemented by the developer himself. It’s not possible to extend the functionality of a tool without the need to rewrite it.

For effective DOAS measurements it’s necessary to have a unique framework that provides support for different kinds of detectors and device components. Also it must offer all the mathematical and operational functionality required for data management, evaluation preprocessing and the DOAS evaluation itself. The user should even be able to extend the functionality with new algorithms without any modification of the framework’s core.

Therefore the DOAS Intelligent System (DOASIS) was developed. It tries to cover all the needs of a DOAS researcher. First of all the requirements of the framework should be defined in detail.

4.1 Requirements

A framework is only useful if it deals with all or at least the most common needs of the user. So what does a DOAS user need? The framework should cover at least these items.

• Easy-To-use
  First of all the framework should not be too complicated or too abstract. Most DOAS users will not be very experienced software devel-
<table>
<thead>
<tr>
<th>Tool</th>
<th>Description</th>
</tr>
</thead>
</table>
| MFC    | - DOS-based tool  
|        | - Support for serial HMT controllers  
|        | - Graphical mode to display spectra  
|        | - Proprietary scripting support  
|        | - Basic spectrum mathematics  
|        | - Fitting of the standard DOAS model included |
| XDOAS  | - Linux-based variant of MFC  
|        | - Additional support for Ocean Optics USB devices  
|        | - GUI of MFC ported to X-Windows |
| WinDOAS| - Windows-based tool  
|        | - For evaluation purposes only  
|        | - All mathematics to prepare and evaluate spectra  
|        | - Several fitting models available |
| Falt2Std| - DOS-based tool  
|        | - Convolutes a cross-section with the aperture function |

*Table 4.1: Available DOAS Tools*

opers. Therefore the structure should be easy to understand and easy to be used.

**User Interface**

As already mentioned in the previous item, the users will not be computer experts. So it’s not enough to provide a set of methods and packages. A graphical user interface is required that presents all the functionality contained in the framework in a convenient way so even beginners can quickly start doing their measurements and evaluations. New functionality should show up automatically as well as it should be possible to integrate new user defined methods seamlessly into the user interface.
• **Scalability**
  The framework should be extendable by developers as well as by the user himself. The DOAS community constantly improves their evaluation methods and introduces new DOAS applications. A user should therefore be able to integrate his own evaluation methods and application procedures. The framework should also not be limited to the basic set of functionality that covers data recording, data handling, mathematical operations and evaluation.

• **Automation**
  Each user will do the measurements in slightly different ways depending on the current application, the framework should be able to run these different measurements automatically. The user needs to define his application and the framework will do the required tasks without further user interaction.

• **Hardware Support**
  The framework should be capable to operate with all kinds of devices currently used and should allow an easy integration of new devices.

• **Interoperability**
  Todays operating system already provide a great set of functionality to the user. Therefore it should be possible to use all components that are present on a specific system. Therefore the framework should be able to access several kinds of external resources and offer its functionality to external components as well. Also it’s important to communicate with other available DOAS tools which is done via file exchange normally. Filters for the different kinds of file formats are required.

• **Remoting**
  Most measurements are not done in the lab but on locations widely spread across the measurement area. Besides the requirement to have the measurement running automatically, support for remote access would be convenient. For example, while a machine at the measurement site records the data, a control center would control the device and evaluate the data somewhere in a local lab. The evaluation of huge sets of data could be speed up by doing the evaluation in parallel on different machines.

• **Spectrum Mathematics**
  Besides the support of the basic mathematical operations for spectra
like dividing two spectra, all the methods required for preprocessing the measured data and preparing cross-sections should be included as well.

- **Evaluation**
  Of course the major task of an DOAS framework should be the evaluation of the measured data. Support for the currently established methods should be present. But due to the scientific nature of DOAS measurements the user should be capable to try new evaluation methods and models in a simple way.

### 4.2 Application Environment

Besides the functional requirements a decision has to be made in what kind of environment DOASIS should be available. Modern applications use an object oriented approach to solve the given task. Object oriented programming has several advantages that already cover some of the requirements of DOASIS. The inheritance of objects is a very powerful way to offer a common set of methods while still being able to extend the functionality to further needs [Goos, 1999; Pomberger and Blaschek, 1996; Rumbaugh et al., 1993].

To be as flexible as possible it is also required to operate with methods respectively objects that are not known at development time. Modern application environments allow this by providing methods to dynamically query an object’s type, inheritance, methods and properties without knowing anything about the object itself. This mechanism is called Reflection [Buschmann et al., 1996] and is included in Java and the .NET framework. Other object orientated developments languages like C++ only provide very limited reflection methods [Stroustrup, 1997]. Functional programming languages don’t know reflection at all.

The next sections will give an overview about the application environments suitable for DOASIS.

#### 4.2.1 Java

Java is an object oriented programming language developed by [Sun Microsystems, Inc., 1995]. While other programming languages come with an compiler that directly transforms the program code to an executable valid
for a specific processor type, Java is compiled into a binary meta language. The complete language definition is given by [Gosling et al., 1996].

The meta language has several advantages. The compiler does not need to generate executables for every processor type separately. As soon as an interpreter for the meta code is available for a system, it can run all Java applications without the need to change anything. To speed up execution nowadays so called Just-In-Time (JIT)-compilers are used. These compilers transform the Java meta code into the binary code required by the current machine as soon as the code gets executed. So the first time a method is called, it needs some time until it is compiled into the machines native code. But any further call to it runs at full speed. Another big advantage of using a JIT-compiler gets important during code optimization: When a compiler produces directly binary code for a specific machine it can only take into account a static machine setup for code optimization. Specific parameters like cache sizes, current memory and CPU utilizations cannot be used, since these parameters are not known during the optimization cycle. The JIT-compiler can take these system parameters into account and optimize code especially for the current system.

The Java language itself is based on the C++ language with several improvements. First of all to avoid buffer overflows and invalid pointers, Java uses references to objects only. Any kind of data used is represented by an object except some intrinsic data types like integer numbers or boolean values. Objects only have members and methods, which can be overloaded by derived objects. While C++ is a very powerful language with many possibilities, code often gets complicated and hard to maintain. Java has therefore skipped a lot of features available in C++ like templates, polymorphism and pointer operations. Classes can be grouped logically into packages which can be hierarchically structured.

The developer does no longer need to manage the lifetime of the object by himself. A garbage collector removes all objects that are no longer used by the system. Several strategies for garbage collection can be found in [Jones and Lins, 1996], [Wilson, 1992] and [Cohen, 1981]. Although it is often assumed that garbage collection requires more processing time, modern garbage collection strategies outperform a manual memory management and reduce the risk of memory leaks. For example, freeing allocated space does not need to be done when ever an object is released. The garbage collector will clean up unreferenced objects on demand asynchronously without
interfering with the main application. Java also provides a mechanism that validates access to objects and array elements. Thus buffer overflows are impossible anymore and access violations do no longer mix up an applications behavior but get reported directly. This makes development and bug removal a lot easier.

Additionally Java ships with a huge set of packages for several kinds of developer’s needs like optimized sorting algorithms [Knuth, 1973]. A complete description of the available package can be found in [Chandrasekhar et al., 2004].

4.2.2 .NET Framework

The .NET framework was developed by Microsoft and released in 2000 [Microsoft Corporation, 2000c]. It has several benefits compared to other application environments. The primary goal in developing the .NET framework was to get rid of the evolutionary grown Windows API and provide a well structured object oriented way to the Windows functionality.

.NET operates in a similar way like Java. All .NET code is compiled into a meta language which is described in the Common Language Infrastructure (CLI) standard [ECMA International, 2002b]. The CLI defines the Common Intermediate Language (CIL) which is the actual byte code that can be executed in a CLI environment. A high-level application will be compiled to the intermediate byte-code and will be compiled to native code if it gets executed on a specific machine. Unlike Java the CLI also includes various meta information and a language independent type description in each assembly. The CLI therefore eases to create and use CIL byte-code in various high-level programming languages like C#, C++, J#, VisualBasic .NET and Delphi .NET.

To reduce the time required by the JIT-compiler during program execution, .NET uses Ahead-Of-Time (AOT) compilation as well. AOT-compilation simply means that not only the code that is currently executed is compiled into native code, but also code that is kept in packages currently not in use. Even a cache for system-wide packages is available to reduce application startup time. The packages do not only contain the program code but also any kind of additional resources. Therefore the code packages are called assemblies.
In general, .NET uses the advantages of Java and extends the ideas in several ways. While Java applications can only be written in the Java programming language, .NET only defines the meta language to describe the program. Therefore, .NET applications can be written in several kinds of programming languages, like C++, J# (an adapted version of Java), C# (based on Java but with several language improvements like event handling, etc.) [ECMA International, 2002a], VisualBasic and other variants of well-known languages. All .NET applications use a garbage collector as well and run in a so-called managed environment. The managed environment ensures proper access to arrays and objects like Java does. The basic set of utility packages available in Java are provided in .NET as well [Microsoft Corporation, 2000d].

But .NET has several advantages in regard to Java: On one hand, .NET is far more interoperable than Java. While Java provides support to access C-like functions in native code modules and only proprietary support of the COM interface, .NET makes it possible to use code written in different kinds of programming languages and available through several interfaces.

On the other hand the meta language of .NET describes the objects in a binary compatible way. So it's possible to write an object e.g. in Delphi and than using it in a J# application. As long as the binary .NET code is available the code can be mixed in various kinds. Native C-like methods can be directly called through the P-invocation mechanism. Unlike the Java-Native-Interface (JNI) the .NET framework takes care about marshaling the method’s parameters appropriately.

Finally, since many other developers offer their components using the Component Object Model (COM) interface, .NET also provides an easy way to access external COM objects from a .NET application. By default all .NET objects can be made available as COM objects as well so they can be used by other developers.

Also the idea of using type information at runtime, which is already present in the Reflection package and will be discussed in more detail in section 5.1.1, is available in .NET as well. This mechanism got extended by allowing to define additional attributes to classes, properties and methods. So more information can be made available to applications that try to get information about an object from its type information. A simple attribute, for example, is the Description attribute. Using the Description attribute an explaining text can be connected to a property or method which then
the .NET framework is only available for Windows platforms currently. But there are two Open Source projects under development that make the .NET framework available under Linux (see appendix F). A wide range of commercial and free-ware tools for development ([Heise Zeitschriften Verlag, 2003]) and the fact that most manufacturers ship their devices with Windows drivers only emphases a Windows development environment currently. Actually, in regard to hardware support we’re stuck on Windows machines anyway.

Taking the interoperability support of .NET into account, the needs of the DOASIS framework find in .NET their best match.

4.3 Performance Consideration

Both Java and .NET do not directly compile to the native machine code of a specific processor. Both use a meta language to describe the actual program code which will be compiled to native code during execution time. Early versions of the Java Virtual Machines only interpreted the Java code which slowed down execution dramatically. Fortunately many optimization have been done since then. The already mention JIT technology brought great performance improvements to the Java world and was therefore also used for .NET.

JIT not only improved performance by simply compiling the meta code into native machine code. Unlike C or C++ compilers the target processor type does not need to be specified during compilation time. The JIT compiler can generate optimized native code especially for the type of processor it is currently running on. It is even possible to distinguish different generations of a processor and use special code optimizations for each.

It is a common rumor that such monitored environments like Java or .NET cannot run an algorithm as fast as native C or C++ applications can do. The lack of pointer operations and the array boundaries tests, for example, need more operations than simple C code does that accesses array elements by simple increasing a pointer. However, if certain design and coding rules are mets during development, managed code can be as fast as a native C application or even outperform it [Schanzer, 2003; Noriskin, 2003]. For ex-
ample, the boundaries check during array access, of course, costs additional performance. But the JIT compiler optimizes this additional code in a way that the index check can be done asynchronously by the processor while it waits for data being transferred from main memory into its cache. Thus with increasing array sizes the overhead for array bounds checking converges to nearly zero.

To get the best performance out of a managed application, these things should be avoided:

- Frequent creation and disposal for objects: Creating a new object or disposing an old one, needs the garbage collector to run additional code and update its internal structures. Try to reuse existing objects instead or creating new ones.

- Error reporting through exceptions: Exception handling is a very complex and therefore costly operation. Exception handling is only useful when an operation should be aborted, but not to report a certain state. [Goodenough, 1975] defines when exceptions are appropriate and when not.

- Non const loops: Try to define a loop using const values. If const values are used, the JIT compiler can apply several optimizations. For example, if the JIT can detect that a loop doing an array access has a certain upper limit, it can omit the boundaries check. In non-const loops the boundaries check always has to be used.

- By value parameters: Using by-value parameters requires the compiler to create a copy of the value on the stack. Using reference parameters where possible is the preferred way (and therefore is the default parameter type in Java and .NET).

- Too many indirections: A object oriented language is used to project a certain problem to a object hierarchy. Using too many object indirections through delegate classes and to deep object trees prevent the JIT compiler from from doing in place code substitution and optimizing a method call. Try to keep the object structure and inheritance as simply as possible.
4.4 User Interface

Today’s applications run in a graphical environment and therefore interact with the user through a Graphical User Interface (GUI). Such GUIs increase the efficiency of an application especially when unexperienced users start working with it [Schmitt, 1983; Blaschek, 2002]. Additionally the UI needs to take care of interactions or interventions by the user. Several Runtime Tactics need to be included to increase usability, for example, by providing a cancel functionality to long running tasks [Bass et al., 2003].

From a developer’s point of view modifications on the core modules of the application should only require minor or better no changes in the UI. The Design-Time Tactic here is to separate the UI as much as possible from the core modules [Bass et al., 2003]. As already mentioned in section 4.1 the framework should therefore provide an easy to understand user interface. The interface should also adapt itself to the objects and their properties and methods available in the framework.

Figure 4.1: The main window of the DOASIS graphical user interface. It shows the selected spectrum and additional information in the properties windows on the right panel. The script and fitting control can be found in the panels at the bottom of the window.
4.4.1 Concepts

The concepts behind the design of the DOASIS GUI are quite simple: Show as much information as possible but do not display unnecessary information [Schmitt, 1996]. The most important information are the spectra themselves. The major part of the GUI is used to display the spectra in several modes. The main UI windows of DOASIS can be seen in figure 4.1 and will be explained in more detail in appendix A.

The user can adapt the look and feel of the GUI to his needs very easy. All panels that show additional spectrum information or provide access to devices and automation support can be placed freely in the GUI, so the user can decide where what kind of information shows up.

The most common operations are available through toolbars. If an operation requires the input of additional parameters, they can be defined via the menu structure. Activating the same operation from a toolbar reuses the last set of parameters automatically. This makes the usage of repeating tasks quite comfortable. For example, prepare the correction of the dark current once and reuse the operations several times without the need to specify the dark current spectrum every time again.

Adaptive Menu Structure

The DOASIS GUI offers a lot of functions through menus, toolbars and access panels. Normally a user does not need the whole set of functionality and therefore the GUI would be more efficient, if it only offers the elements the user really needs [Kraus, 2003].

On the other hand it should be possible for any user to extend and modify the functionality of the GUI. Therefore special care needs to be taken about how the menus are integrated.

But why should we develop our own algorithm to get auto adaptive menus? First of all there are only a few solutions available and all of them follow the standard scheme to implement menus: The developer has to write a method that gets called on selection of the associated menu item. The menu event handlers need to be present during development time and the developer has to take care about how and when a menu item gets displayed or is hidden from the user. Since the goal of the DOASIS GUI is to be open to
extensions outside the basic application and take care about the handling of user needs a new method was needed.

In any case the development of a menu structure start with defining the complete layout of the menu items. The structure of the application menus is predefined in the applications configuration. A simple hierarchical tree structure is used to define the layout. A detailed description about the structure definition and the possible item definitions is given in appendix B.1.

Using a dynamic configuration makes it easy to change the menu layout and add new methods, but the application still does not adapt its behavior automatically to the user’s needs like other modern applications do.

A menu item is considered to be useless for a user if he does not use the item for a certain time. But as soon as a hidden item is used again, it should be visible immediately. A quite simple approach to this problem is using a reference counter. Each menu item gets a reference counter associated. Every time the application starts the reference counters of all menu items are decremented. As soon as an item’s reference counter is zero, the item is considered to be hidden by default. If a hidden item is used again, the reference counter is reseted to its starting value. The inactivity timeout is therefore defined by the number of application starts. More complex schemes could be used as well, which may take other ergonomical and psychological information into account.

But it’s not enough to decide which item should be hidden and which not. It is also possible to always keep a valid structure definition even though complete parts of a menu get hidden. The hierarchical structure of menus requires the handling of all parent nodes of an item as well during an selection event. Figure 4.2 shows the complete algorithm used for handling the auto hide feature and to keep the menu structure valid.

So far the adaption to the user needs is given. But this algorithm only works if it gets executed in each event handler for each menu item. Also it is still required that each menu item gets its own event handler. A centralized management is more appropriate here. The GUI therefore has a centralized event dispatcher, which’s state diagramm can be found in figure 4.3. The event dispatcher catches all events from the menu items and their associated toolbar buttons and short cut keys. On each event received by the dispatcher the dispatcher first starts adapting the menu
structure according to the auto hiding algorithm. As soon as the menu layout is modified, the dispatcher needs to know what method is associated with the currently selected menu item. This information is also given in the configuration as well where the method is actually located. The dispatcher then simply needs to query for the given method declaration and then call it.
Again generalization is achieved by using reflection. The .NET framework provides easy methods to include additional program libraries like ActiveX controls or other .NET assemblies. So the dispatcher becomes the central point where functionality from other developers can directly be included in the DOASIS GUI. The developer only has to define an additional menu entry in the configuration and what method from which library should be used on a selection of the new menu item. There is no need to recompile the framework or the GUI at all.

![Diagram](image)

**Figure 4.3:** The central UI event handling is done through a generic event handler. This event handler ensures that all parameters necessary to call the specialized event handler are available. Depending on the kind of event and if default values are present, the user is requested to define the parameters.

Using a central event dispatcher has other side effects, too. Most menu items just provide a user interface to one method present in a program library. The only purpose of a menu item event handler is to ask the user for any additional information required to call the underlying method and then
execute the associated function. DOASIS therefore has some generic dialog elements that make queries for method parameter quite easy. A developer does no longer need to define a special dialog for every menu item to get the required information. DOASIS takes care about this automatically. The centralized dispatcher queries for the required method parameters, generate an appropriate dialog and presents this to the user. All information collected is then send to the underlying function and can be extracted there.

Other common functionality can be included in the dispatcher:

- The dispatcher can update the undo history every time an action is executed.
- The dispatcher can directly decide whether the user gets asked for additional parameters or the last ones are used again. Such a feature is quite handy when a menu item gets called by a toolbar button or from a shortcut. Often the user wants to prepare a function once and then reuse the same function over and over again (e.g., correction of the detector offset).

So the DOASIS GUI provides a very powerful mechanism to automatically adapt to the user’s needs but still making it very simple for developers to include their additional functionality.

**Generic Panels**

Due to the nature of the DOASIS framework information is made available through objects that provide different sets of properties. The standard approach to this problem would be to define an interface and any object that is connected to the information panel must implement this interface.

This method, however, has some limitations. Any additional information given by an object that is not covered by the interface does not show up in the panel. Also any change in the interface definition requires an adaption of the associated panel.

A more elegant way to extract data from an object of unknown type is given through the reflection namespace. Since reflection allows to query the complete type information of an object a generic way can be used to get all public available properties of it.

Fortunately the .NET framework already ships such an GUI element called PropertyGrid. The PropertyGrid queries the public properties of an ob-
ject and displays them in a tabular way. It’s even possible to distinguish between read-only and read/write properties. For most basic types editors are available and an object’s property can be modified directly from the GUI. The .NET framework also provides several ways to adapt the behavior of such a generic control to the needs of the developer.

So this method is a very powerful way to link the user interface to the framework. Any modification made to the framework’s object will immediately show up in the GUI without the needs to adapt anything.

**Console mode**

Since a graphical user interface is not applicable for all systems, the DOASIS framework provides a console interface, too. Although the console application only provides a limited user interface, it’s sufficient to run automatic measurements.

Often the measurement systems run with very low powered system in regard to memory size and CPU power. In such a case a resource eating GUI is not well suited and a simple console application is sufficient.

### 4.4.2 Comparison

So what are the actual benefits of this new GUI? Currently there is only one other DOAS tool available that has a GUI like it is expected from modern applications: WinDOAS. The other tools either have no GUI or console based graphical displays that will be deprecated in a few years. A detailed overview about the GUI features of the different tools is given in table 4.2.

So the DOASIS framework provides an advanced easy-to-use user interface that supports all the needs required by the underlying framework and of course the needs of the user.
<table>
<thead>
<tr>
<th>Component</th>
<th>DOASIS</th>
<th>MFC</th>
<th>XDOAS</th>
<th>WinDOAS</th>
<th>Falt2Std</th>
</tr>
</thead>
<tbody>
<tr>
<td>UI Style</td>
<td>MDI windows used</td>
<td>SDI display used</td>
<td>MDI windows used</td>
<td>MDI windows used</td>
<td>No UI</td>
</tr>
<tr>
<td>UI Handling</td>
<td>Application menus and shortcuts available</td>
<td>Command style input required</td>
<td>Mixture of menus and command style input</td>
<td>Application menus available</td>
<td>Cfg. file and cmd. parameters</td>
</tr>
<tr>
<td>Generic UI</td>
<td>Generic information panels</td>
<td>Fixed screen display</td>
<td>Fixed display</td>
<td>Fixed display, but configurable selection of information possible</td>
<td>-</td>
</tr>
<tr>
<td>Adaptive UI</td>
<td>Auto adaptive menu structure available</td>
<td>No menu available</td>
<td>Fixed menu structure</td>
<td>Fixed menu structure</td>
<td>-</td>
</tr>
<tr>
<td>UI Layout</td>
<td>Docking windows available</td>
<td>Fixed screen layout</td>
<td>Fixed screen layout</td>
<td>Fixed screen layout</td>
<td>-</td>
</tr>
</tbody>
</table>
5 Module Structure

DOASIS basically provides several abstraction layers like the Device and the HMI name-spaces do. These abstractions are necessary to ensure that the automation procedures, algorithms and data management run independent from the currently used user interface or detector used. For example, a message presented to the user should be done in the same way regardless if the message is displayed in a dialog or written on a console.

Although the detectors supported are not completely equal to use, every detectors purpose is to record a spectrum. So the framework provides a way that ensures this basic functionality is available for all detectors used. Only specialized controlling parameters extend the detector’s functionality. This also applies to several other kinds of hardware modules available like controlling motors, heatings or other kinds of measurement supporting hardware. As already described in chapter 4.4 the user interface is capable to query these extended functionality and allows the user to control them as well as the common functions. [Bass et al., 2003] call this technique Information Hiding: Provide only the information that is most likely not to change when the inner parts of a module are exchanged or enhanced.

The abstraction and modularity is given by the different objects and interfaces available in DOASIS. To work efficiently with these kinds of functionality it is very useful to build logical groups that represent a specific functional package or module [Buschmann et al., 1996]. Therefore the DOASIS framework is build up of several namespaces which are contained in the assembly DoasCore (figure 5.2). An UML definition of some of the core namespaces is shown in figure 5.1.

- **Device Namespace**
  The complete support for the different kinds of detectors and other devices is located in this namespace. Currently the following devices are supported:
  - Hoffmann Messtechnik Serial Controller
  - Hoffmann Messtechnik MiniDOAS Controller
  - Hoffmann Messtechnik MultiStepper Controller
  - Hoffmann Messtechnik MiniStepper Controller
  - Slotted Disc Controller
  - Triax Spectrograph
**Figure 5.1:** The UML Class Structure of the DoasCore assembly shows that each module is designed to be as simple as possible. The class structure only has a maximum of six hierarchy levels in a module and the modules only communicate through well defined interfaces.
<table>
<thead>
<tr>
<th>File Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SP2</td>
<td>Binary format of DOASIS which includes all relevant informations of a spectrum using a binary stream format.</td>
</tr>
<tr>
<td>SPE</td>
<td>Binary format of DOASIS prior version 3.0.</td>
</tr>
<tr>
<td>MFC</td>
<td>Binary format of the MFC program.</td>
</tr>
<tr>
<td>STD</td>
<td>Text based spectrum data format that includes additional spectrum properties besides the spectral data itself. DOASIS has support for an extended version of this format to include additional spectrum properties and wavelength calibration information.</td>
</tr>
<tr>
<td>LV1</td>
<td>Support of GOME Level 1 Files which contain the spectral data of one complete orbit.</td>
</tr>
<tr>
<td>IVL</td>
<td>File format for spectral data from the IVL institute.</td>
</tr>
<tr>
<td>XS, DAT, KTZ, REF</td>
<td>Different types of high resolution cross-section data.</td>
</tr>
<tr>
<td>RAW</td>
<td>Simple binary representation of spectral data.</td>
</tr>
<tr>
<td>TXT, ASC, CSV</td>
<td>Text based file formats that only contains the spectral information.</td>
</tr>
<tr>
<td>XML</td>
<td>XML based format of a spectrum object using a SOAP stream formatter.</td>
</tr>
</tbody>
</table>

**Table 5.1: Available file formats of DOASIS.**

- Several Acton Spectrographs
- Jobin Yvon Spectrographs
- Ocean Optics USB2000
- Ocean Optics ADC1000

Also basic support for the following communication protocols is available:
- Serial Communication (RS232)
- Parallel Port Communication
- USB Communication (either using vendor supplied drivers or native USB communication and direct interaction with USB microcontrollers [Eberhardt et al., 1999; Flik and Liebig, 1994; Tan, 1997]).

**I/O Namespace**

Imports and exports spectral data using a wide set of file formats, which are listed in table 5.1 and will be explained in depth in appendix C.
DOASIS supports serialization for all relevant objects. So new file type export and import filters can be added by implementing a stream formatter class.

- **Math Namespace**
  All spectrum mathematics are contained in the Math namespace. Besides the basic mathematical methods like adding or dividing a spectrum, the required procedures to preprocess a spectrum and prepare it for evaluation are provided by this module.
  The complete evaluation framework presented in chapter 7 resides in this namespace, too. Calibration methods, advanced analysis functions and signal generators are exposed by this namespace. The SZA conversion needed for stray light measurements and the obligatory time and geographical coordinate transformations can be used through the appropriate classes.

- **Spectra Namespace**
  A spectrum not only consists of the spectral data, but also of several additional information like the time and location of the measurement, wavelength calibration information and specific measurement information. The management of these additional informations and the spectral data itself is implemented in the spectrum object of this namespace.
  Long term measurements normally record several thousands of spectra, which need to be managed as well. Specialized collections and hash tables simplify this task.

- **Script Namespace**
  Contains everything to run automation scripts. A more detailed description of this package will be given in chapter 6.

- **Utils Namespace**
  Everything else like configuration access and data management is covered by this namespace.

- **HMI Namespace**
  The HMI namespace represents all extensions of DOASIS to interact with the user. Since DOASIS supports a graphical and a console based user interface, a developer should not need to take care about the different possibilities to visualize data or query additional information from the user. The HMI-objects redirect the applications input and output automatically to the current user interface.
But DOASIS is not only build up by the `DoasCore` assembly, which contains the namespaces described above. The `DoasCore` assembly is more like the heart of the framework. The user interface gets access to the spectrum data through the `DoasCore` assembly. Also it will be informed about changes and modification using the event model explained in section 5.3.

In cases where access to one of the supported devices is required, the `DoasCore` assembly is used as a middle ware layer. The functionality of the hardware is presented by an abstracted object within the `Device` namespace. Any access to the device itself is done using an instance of one of the classes available in this namespace. The object itself does not directly control the hardware, but delegates the request to one of the underlying device drivers.

All mathematical and data management components are located in the `DoasCore` assembly as well. The complete set of necessary components to cover the DOAS measurement and evaluation process as a whole are present in this central library.
5.1 Generic Modules

DOASIS should be capable to include extensions from other developers or components of a system without the need to create an adapted version of DOASIS. This interoperability and extendability of DOASIS requires a fairly generic module definition.

Besides the possibilities of the .NET framework to directly include other assemblies especially the I/O namespace, the device namespace and the extensions of the user interface need further extension methods.

First of all all of the extensions to DOASIS are defined in the configuration as described in appendix B. In the configuration a developer or user only has to define the package / assembly that contains the new objects and methods. Second the type of the new objects must be given. Factories for the different kinds of modules handle the creation of objects of the desired type regardless if these types are contained in the DOASIS framework or from an external source.

This information is enough for the DOASIS framework to give a user access to the new functions. For example, to include support for a new file type it is enough to create a stream formatter for the file type and add the type definition to the configuration. The file factory enumerates all known types and therefore the new type as well. So a spectrum can be saved in the new file types format just by using the new file name’s extension.

5.1.1 Reflection

The major part of generalization is realized through the Reflection namespace of the .NET framework. Reflection is also part of other modern programming languages and allows a developer to get the complete information about the available objects during runtime. The functionality of the Reflection namespace includes

- **Type identification** to identify an object.
- **Assembly identification** to dynamically add new packages at runtime.
- **Inheritance information** to ensure the presence of required basic object functionality.
- **Interface implementation** for a function specific usage of the object.
- **Object method information** including method parameter data as well.
- **Object property information** to get or set object parameters.

Of course, *Reflection* also reports the current access level of properties and methods, whether they are private, protected or public available.

But Reflection does not only allow to get the type information of an object. Knowing what methods an object supports is not enough to use an object of unknown type in a meaningful way. This information only makes sense, if it is possible to use the methods and properties somehow through defined interfaces [Buschmann et al., 1996]. A method call made during development time gets compiled into the binary representation of the application or library including the handling of the method’s parameters and return value. Type conversions and syntax checking is done by the compiler automatically. Unfortunately when a method of an object that gets loaded during runtime should be called, thus the compiler does not know the object type at all during development time, requires other methods. So to say the compiler cannot generate the method call statically and everything required to do the method call dynamically needs to be implemented.

Reflection makes this task quite simple as well. Every object is associated with a type object that describes the object. This type object also has a method called **Invoke**. Using **Invoke** makes it easy to do any kind of action on the object. **Invoke** is some sort of a generic method that is supported by every object and delegates a call to the desired object method or object property.

Using **Invoke** a correct handling of a methods parameters and return values is ensured as well.

Figure 5.3 shows how in general an object of a type not known during development time is accessed. This process also can be defined by these simple steps:

1. Ensure the assembly that contains the class definition of the desired object is loaded.
2. Create a type representation of the desired object.
3. Create an object of the given type using the system’s **Activator** by specifying the needed constructor.
Figure 5.3: General work flow to programmatically handle method calls to methods and objects that are unknown by development time.

4. Call a method, read or write to a property of the new object using the type definition’s Invoke method and the appropriate parameters’ type conversion.

Here is an example code of how to use the Activator and the Invoke methods:

```csharp
string szMethod = confMenu.GetString("Method", null);
string szType = confMenu.GetString("Type", null);

// try to get the object type
Type typeMenu = Type.GetType(szType);

// try to get a type from the ProgID since it may references a COM object
if (typeMenu == null)
    typeMenu = Type.GetTypeFromProgID(szType);

// try to get an instance of the defined type
object objMenuType = Activator.CreateInstance(typeMenu);

// find the method to call
MethodInfo miMethod = typeMenu.GetMethod(szMethod);

// invoke the method with the standard event handler arguments
typeMenu.InvokeMember(szMethod, BindingFlags.InvokeMethod, null, objMenuType, null);
```
So Reflection makes it very simple to develop generic applications.

## 5.2 Serialization

Also the serialization mechanism is used in DOASIS. Serialization is a mechanism to map an object into some sort of data stream or to read an object from such an stream. In this mechanism the serialization process is separated from the creation of the stream. An object that is serializeable only needs to provide a standardized way to store the object’s content into an repository or read its state from a given repository.

The serialization process will redirect the object’s repository to a so called stream formatter. The formatter is actually the part that is responsible for an appropriate representation of the objects data. The other way round the formatter must be able to parse a given data stream back into a correct object representation somehow.

But the serialization mechanism does not only handle a single object. Any dependent objects get serialized as well. To ensure the correct order of the objects contained, the serialization mechanism first builds an object tree that describes all objects. Then one object after another gets serialized into the data stream using the formatter.

The complete object tree gets parsed by the formatter in case the objects need to be reconstructed again. Each object is first deserialized and at the end the references are adapted to build the object tree as described in the stream.

The .NET framework already has two kinds of serialization formatters present. The **BinaryFormatter** is capable to convert an object tree into a binary representation. The binary formatter does a so called inner serialization by not only handling the public available properties of an object: All of the object’s members are contained in this stream by default.

A more compatible stream is generated by the **SoapFormatter** which generates an XML representation of the object tree compatible to the SOAP standard. The SOAP formatter does an outer serialization only, thus only
serializing the public available properties of an object. So in this case an object’s state must be fully recoverable by its public members only.

Using serialization makes it quite simple to store or load data represented by several objects.

```csharp
// create the output stream
System.IO.FileStream fsFile = null;
try {
    // create the file stream
    fsFile = new System.IO.FileStream(szFileName, System.IO.FileMode.Create);
    // create a stream formatter
    IFormatter formatterSpec = new ExtendedStandardFile();
    // serialize the spectrum into the file stream
    // using the output formatter
    formatterSpec.Serialize(fsFile, specData);
} catch (System.Exception eFile) {
    Console.WriteLine(eFile.Message);
} finally {
    // close the file
    if (fsFile != null)
        fsFile.Close();
}
```

By default every object is serializable. If an object should be serialized the Reflection mechanism is used to read all data from an object or to write the data back into it. Whether an object’s member gets involved in serialization or not can be controlled using the NonSerialized attribute. The serialization process will skip all members that have the NonSerialized attribute.

However, sometimes it is not enough to just hide some members of an object from serialization. An object can get complete control over its serialization by implementing the ISerializeable interface. The object then, of course, needs to provide code that does the serialization and reconstruction completely.

But why use serialization at all? A simple function that reads the necessary data from an object and store it in some file format would do the same. In serialization the object representation is independent from the data stream used to hold the object’s data. Thus new file formats can be added by just providing a new stream formatter. The reconstruction of the objects itself or the construction of the object tree is done by the serialization mechanism automatically. Further more the formatter does not need to know into what sort of data stream it writes or reads the object’s data. An object can be made persistent in several ways by changing the data stream. Regardless
if a file stream or a database is used, the information gets correctly stored by the serialization mechanism. Even remote access to an object can be handled through serialization. For example, by-value parameters will be serialized during a remote call, get serialized into a stream on the server, sent to the client and deserialized back into client side local objects.

Also changes in the objects that get serialized would require to adapt every file handling function for each file type. Fortunately, the new object’s members get serialized automatically. Reflection reports the new object’s member to the serialization formatter which stores the new data like any other before. Of course in such a case a version conflict between older serialization streams that do not contain the new object’s member and newer stream would happen. Therefore several mechanisms are available to ensure version compatibility and even make it possible to produce objects of different types from the ones used to create a serialization stream.

### 5.3 Event Model

Besides the modules for device access and several other features, DOASIS also does the complete management of the spectral data. The spectral data, however, can be modified from several instances and other parts of the framework need to get informed about any change in the data. This especially applies to the user interface which needs to update the displayed information as soon as a change occurs.

To avoid development errors it is not very useful to have a developer make sure that every time a spectrum or a property has been modified all dependent modules get informed about the change. Since such changes will occur very often in new evaluation algorithms and other spectrum processing methods, it is very likely that this manual updates are forgotten and the application ends up with an inconsistent behavior.

A hierarchical event handling model is already present in DOASIS that ensures to inform all modules about relevant changes. Again the underlying .NET framework makes event handling very easy. Event handler and delegates can be defined and a standard event sink model can be build up quickly. So basically every component provides several event sources that get called when ever the associated event occurs. For example, a spectrum object provides events for changes in the spectral data or the spectrum
properties and many other kinds of events. A module, like the user interface, that needs to get informed about a change in a spectrum just adds its event handler, which is represented by a delegate object, to the desired event of the spectrum object. The complete management is done by the .NET framework automatically.

Figure 5.4 illustrates the event dependencies with the DOASIS Framework.

Figure 5.4: DOASIS relies on events that each spectrum creates when either the spectrum data itself or one of its properties are changed. A hierarchical structure of event handlers allows the application to react on changes that happen. While the console application does not require to react on changes at all, the GUI needs to update its display and refresh the informations presented when ever a spectrum is modified.

5.4 Multithreaded Device Drivers

DOAS measurements do not only require to access one spectrometer. Tomographic measurements and tracking of plumes can only be done when several devices record spectra at the same time. Otherwise a time based correlation of the data is hard to find out. Especially in plumes that have a high velocity.
The framework therefore needs to allow access to several devices in parallel. Normally this is not a problem, since different devices are accessed through different instances of a device driver which should not interfere each other [Wettstein, 1993]. But this functionality is sometimes missing in the available device drivers. For example, the Ocean Optics USB device driver is only capable to handle single threaded requests. Unfortunately this means that only one device can be accessed at the same time although the USB communication would allow to work with several devices at once.

However, DOASIS still offers a multithreaded, multi device access although the driver is not capable of such a feature. Achieved is this kind of access type through a trick of the driver model: It creates multiple copies of the driver image for each device used. Since the operating system distinguishes different driver images by its file name, it creates several copies of the device driver images in separate memory spaces [Custer, 1993]. Therefore these copies act like different device drivers. The device is then accessed through the proxy objects provided by DOASIS which delegate the requests to the correct driver image. Figure 5.5 illustrates the dependencies between the different modules.

This little trick allows the DOASIS framework to work with several devices at once even though the manufacturer does only provide a single device access. Of course the framework needs to take care about multithreaded accesses to the same device. It needs to redirect these requests to the correct driver image and needs to synchronize their execution. Otherwise two different driver images would try to access the same device without any knowledge of the other driver which will lead to a crash for sure.
Figure 5.5: DOASIS overcomes limitations of device drivers that are not reentrant and can only access one device at the same time. This is accomplished by dynamically creating new images of the single threaded device driver. For each device a driver image is created and loaded into separate ranges of the application’s memory space. Driver proxies ensure to synchronize and delegate calls to the device driver instances properly.
6 Automation

Manual measurements are only used to setup a measurement site and validate the correct operation of the connected devices and detectors. The complete measurement procedure was described in section 3. Doing each step from recording a spectrum to evaluating it is a very time-consuming task. Doing long term measurements where a large number of spectra is recorded over a long period of time cannot be done in manual mode. An automation mechanism is required.

Automation has other advantages, too: A user can customize the behavior of the DOASIS user interface using little macros that do a specific task.

But how should the automation support be added? First we have a look at the other DOAS applications available. MFC and XDOAS [Grassi et al., 2001] are the only applications that allow automation by using user defined scripts. The scripts contain a sequence of commands which will get executed one after another by the applications script interpreter [Goos and Zimmermann, 2002]. Since XDOAS is based on MFC it supports the same scripting commands. A closer look at the MFC scripting support should be done.

The commands in a MFC script are the same as the commands used to interactively control the application. MFC is a command based application which is controlled by entering specific command codes and the appropriate parameters [Gomer et al., 1995]. The command codes were defined in a way that a command can be represented by a single character. The j for example denotes the smoothing operator. Furthermore a command gets immediately executed as soon as the command character was entered. So the command’s parameters need to be entered before the command symbol. This limitation also forbids the usage of a command character in a parameter definition.

A MFC script is a simple sequence of commands. Additionally support for a limited conditional program flow can be added using if...goto statements. An example MFC script is given below.

```
13n
1.1615N
14n
1.1400N
13,14n
11n 1u 2u Y 1n 2# 2u 1; 12u 5j
11n
%'
10 '2 j
```

! Change to Region 13
! Load dark current
! Do offset correction
! Change to Region 11
! Fraunhoferspectrum
! Underground correction
! Calculate a ring spectrum
! Change to region Region 11
! Build logarithm
! Smooth spectrum
Of course, the MFC script interpreter is quite simple: Read a line from the script and use this line as a direct input to the command parser. Anything else is done by the command parser like executing a manually entered command.

The syntax of the MFC scripts is not easy to understand, although it is very efficient when the the commands are well known. Even experienced users need to have a closer look at a script to understand its functionality.

In DOASIS the automation support should be more intuitive. The scripts should be easy to read and a clear definition of the program flow and the commands is required. Fortunately there are already several scripting languages available. One of the most featured scripting language is the JavaScript language which is used to get dynamic content into HTML pages.

JavaScript was developed in 1997 by Netscape [Netscape Communications Corporation, 1997]. The basic idea behind JavaScript was to have a scripting language that gives the user a wide range of possibilities but still having a syntax that is easy to read and understand. The language syntax of JavaScript is based on the Java language, which is also well known to many developers creating web based applications. The control structures of Java were more or less directly included in JavaScript, but, for example, variable declaration was simplified and a functional program structure is used.

The JavaScript language seemed to be a suitable choice to integrate automation support into the DOASIS framework. To do this either an own JavaScript interpreter needs to be developed and some additional syntax must be defined to access the DOASIS functionality from a script or one of the already existing script engines could be used. As the major part of DOASIS is based on the .NET framework, it was consequential to use the scripting support of the .NET framework.

6.1 JScript .NET

The .NET framework does not directly contain support for a script interpreter. But several compilers are included in the framework. Currently a VisualBasic, a C# and a JScript compiler are present.
The JScript compiler creates .NET assemblies from JScript code which is quite similar to the JavaScript language. JScript actually is an extended JavaScript language [Microsoft Corporation, 2000a,b]. While JavaScript was developed to make web pages more dynamical, JScript’s design goal was to have a full featured programming language. Besides the functionality of JavaScript JScript has additional support for creating objects and other more complex structures. Access to several system functions is directly included. Furthermore, since the JScript compiler generates .NET assemblies, the full range of .NET functionality can be used in a JScript. A quick reference guide of the JScript .NET language can be found in appendix D.

```javascript
// define a function that prepares a spectrum
function PrepareSpectrum(spec)
{
    // correct the light source
    SpecMath.Div(spec, lightSource);
    // do a high pass filter on the spectrum
    SpecMath.HighPassBinomial(spec, 100);
    // build the logarithm
    SpecMath.Log(spec);
    // do a low pass filter
    SpecMath.LowPassBinomial(spec, 5);
    // return the prepared spectrum
    return spec;
}

// create a new spectrum object
var spec = Specbar.GetSpectrum("Test");

// record a spectrum
Spectrograph.Scan(spec, 1, 1000);

// prepare the spectrum
spec = PrepareSpectrum(spec);
```

This little example demonstrates the power of the JScript syntax. Compared to the MFC script even an unexperienced user can directly understand what operations are carried out. Also, since JScript has the complete range of control structures of Java, even very complex functions can be defined. Even large and very complex measurement procedures are written easily in a JScript script. Defining functions and objects make the language even more powerful.

Additionally the JScript code gets executed very fast. Instead of an interpreter a compiler is used. The code generated by the compiler runs at full application speed. Of course the script code is not generally as fast as ‘normal’ program code. The simplifications in regard to variable handling and other syntactical changes require often additional code that slows down
execution a little bit. But in general the compiled script code is far faster than an interpreted code.

Since the script gets compiled into .NET binary code, all advantages from .NET apply to the script code as well like running the script in a managed environment. But the code generated internally always uses a late binding style for method and property invocation. The script compiler does not know the type of a variable's content by default. So any method or property call applied to an untyped object does not get syntax checked when compiled. It inserts code that generates the method call during runtime. Therefore any syntax error or type mismatches will first occur when the faulty code gets executed. This late binding behavior slows down execution speed as well. To avoid this problem it's possible to declare a variable as a typed variable. Thus the compiler knows the type of its content and can directly check its syntax.

6.1.1 Debugging JScripts

Development of scripts, especially of very complex scripts, gets faster if scripts can be debugged. The .NET framework’s SDK already contains a very powerful debugger. Debugging scripts using this debugger is possible, since they get compiled into a .NET assembly which can be handled by the debugger. Besides the shipped .NET debugger there are already a lot of other graphical debuggers available.

And how can JScript being debugged? A simple insertion of a code line that contains the `debugger` statement launches an installed JIT-debugger. Otherwise the debugger can just attach itself to the DOASIS application and break into the current script execution. In both cases as soon as the debugger is attached to the application, scripts can be processed step by step and even the content of variables can be examined or modified.

These features make the JScript compiler a very powerful tool to add automation support to an application using scripts based on the JScript-/JavaScript syntax.
6.2 Running Scripts

While the .NET framework does provide a very powerful JScript compiler, DOASIS must provide the middleware layer to run user defined scripts. DOASIS must control the state of a script like starting and stopping it but also needs to create the required compiler and generate the code to be executed.

A JScript gets compiled by first creating an instance of the JScriptCodeProvider. The JScriptCodeProvider object then will create the necessary compiler. Now the compiler parameters need to be defined and finally compilation of the script can be initiated.

```csharp
// create the code provider
CodeDomProvider cdpProvider = new Microsoft.JScript.JScriptCodeProvider();

// get the compiler
ICodeCompiler ccCompiler = cdpProvider.CreateCompiler();

// setup the compiler parameters
CompilerParameters cpParams = new CompilerParameters();
cpParams.GenerateInMemory = false;
cpParams.GenerateExecutable = true;
cpParams.CompilerOptions = "/debug";

// compile the JScript code
CompilerResults crResults = ccCompiler.CompileAssemblyFromSource(cpParams, szMiniScript);
```

If the compilation was successful, the compiler generates an assembly in memory that contains the compiled script code.

Executing the script is done by starting the default entry point of the assembly, which is the JScript.Global method by default. Of course Reflection can be used to find a special script function and it is possible to use such a special function as starting point as well. But it’s necessary to know that all global script code, which includes all special initialization code of global variables, is not executed in such a case.

Compiling and starting a script is done very quickly. Stopping a running script gets more complicated. Since the script assembly is a simple .NET assembly with objects and methods, the application cannot run the script code in another way than normal application program code is executed. So calling the entry point of the script assembly means to leave the control flow inside the script until it terminates somehow.

DOASIS therefore runs such scripts in a separate thread. The script thread can then be controlled using the Threading namespace of the .NET framework. The script thread can be suspended, resumed or terminated.
Terminating a script using the system’s methods to force a thread to be stopped, unfortunately means that the script would get aborted most likely while doing some sort of command. As long as the command does not access some hardware or interact with some other critical system resource, termination is not harmful. The system will free all resources allocated by the thread and just stop its current execution.

During access to critical system resources termination of a thread is not applicable. The hardware accessed or the device driver may get confused if the communication with it aborts abruptly. Often in such a case the system gets instable or further access to the resource is no longer possible.

To avoid this, DOASIS signals a script to terminate as soon as possible. A script should therefore check at regular intervals if the framework requests an abortion. If such an abort condition is detected, the script should finish the current command and stop as soon as possible. The script gets terminated softly and there should be no influences on the system stability or hardware used.

Scripts somehow are not always cooperative in such a way. DOASIS still needs to be able to force termination of a script if it does not react on a stop request. As already mentioned, terminating a script that does not access a critical resource can be done by forcing the termination of the script’s thread. If such a termination occurs during a critical access, the script thread would still get terminated, but the system may also abort the whole application as all system resources are allocated by the application and not only by a single thread. The forcefully release of a critical section may cause serious application error and the system therefore terminates the whole process that caused the problem.

When DOASIS forces the termination of a script such a behavior is not acceptable. Therefore a separate application domain is created in which the script will be started. The new application domain gets threaded by the system like a new separate process. A uncooperative script gets terminated by killing the process it runs in respectively unloading the appropriate application domain. Killing the whole application domain has another useful side effect: Since the system treats the application domain like a new process, unloading it forces the system to cleanup all resources allocated by the script as well as bringing the device drivers into an initial state (this does not include file and resource handles, since they belong to the main
process). The script termination does not have any effect on the behavior of the current DOASIS application at all.

6.3 Extending JScript

Now that the scripting support is included in DOASIS it is still not clear how to use other modules and extension form a script. By default JScript only supports a very limited set of functions which include basic mathematical support and little I/O functionality. The first extension is included by using JScript .NET. The JScript .NET compiler includes the complete .NET framework which has a huge set of predefined classes. It’s even possible to create user defined dialogs from a script, access network resources and do complex file operations.

Additionally COM or ActiveX controls are easy to use. The intrinsic object type `ActiveXObject` encapsulates a COM object and represents a transparent proxy to access an associated object. Although the .NET framework already provides support to interoperate with COM objects, the specialized `ActiveXObject` object makes their usage even easier.

6.3.1 DOASIS JScript Extensions

At last all DOASIS objects and functions should be available in an automation script. Actually an automation script will use the objects and methods of the DOASIS framework mostly.

The DOASIS namespaces are available by default to all scripts that get started from the DOASIS framework. DOASIS ensures that all required assemblies are loaded into the scripts application domain. A script can than directly use all of the DOASIS Framework like it uses any other .NET assembly.

Creating user defined objects and functions in a script even allows to register them to DOASIS. So even the user interface can be extended from within a script besides the possibility to add further evaluation and data processing algorithms.

Using the JScript command-line compiler this mechanism can be used to quickly add new functionality to DOASIS: First develop a JScript that
interoperates with DOASIS. If everything works as expected, the script can be compiled into an assembly which will be added into DOASIS as an external resource.
Part III

Analysis
7 DOAS Evaluation

In a DOAS evaluation it is tried to find the best fit of a set of reference spectra to the measured data. The model function given in equation (3.11) unfortunately introduced several nonlinear parameters to the evaluation model. Finding the best fit of the reference spectra therefore needs some sort of nonlinear solution method.

7.1 Levenberg-Marquardt Nonlinear Fitting

One well established method to numerically solve nonlinear problems is the Levenberg-Marquardt algorithm [Levenberg, 1944; Marquardt, 1963]. It is a combination of the Inverse Hessian method and the Steepest Decent method. Since there is no linear solution for the model’s parameters this algorithm uses an iterative approach that tries to get an better estimate for the model’s parameter with each step until the optimal solution is found. But first of all the problem to be solved needs to be defined.

Given a model function \( f(x; a) \) with a set of nonlinear parameters \( a = a_0, a_1, a_2, \ldots, a_k \) and a set of measurement data \( y = y_0, y_1, \ldots, y_n \), the optimal solution for the model parameter \( a \) would look like:

\[
  f(x_i; a) = y_i
\]  

(7.1)

The error between the model function \( f(x; a) \) and the real data set \( y \) is defined by the merit function \( \chi^2 \).

\[
  \chi^2 = \sum_{i=0}^{n} \frac{y_i - f(x_i; a)}{\sigma_i}
\]  

(7.2)

The evaluation starts with an initial guess of the parameter set and tries to improve these parameters \( a_n \) in each iteration step. The gradient of \( \chi^2 \) in regard to the parameter set \( a_n \) defines the direction in which the parameter set must be altered to become the optimal solution, thus making the gradient \( \nabla \chi^2(a_n) \) zero.
\[ a_{n+1} = a_n - c \times \nabla \chi^2(a_n) \quad (7.3) \]

Where \( c \) must be a constant small enough to not exhaust over the optimal solution of \( a \). This method is called the Steepest Decent method.

The Steepest Decent method searches the minimum of \( \chi^2 \) by doing many little steps inside the solution space even if there would be faster ways to find it. Suppose that the \( \chi^2 \) function can be approximated by a quadratic function with a \( k \)-dimensional vector \( d \) and a \( k \times k \) matrix \( D \):

\[ \chi^2(a_n) \approx y_i - d \cdot a_n + \frac{1}{2} a_n \cdot D \cdot a_n \quad (7.4) \]

The optimal solution for \( a \) is then given by

\[ a_{\text{min}} = a_n + D^{-1} \cdot (-\nabla \chi^2(a_n)) \quad (7.5) \]

All we need is the matrix \( D \) which is the second derivative of \( \chi^2 \) or also called the Hessian Matrix. Putting things together equation (7.4) can be rewritten as a set of linear equations:

\[ \sum_{l=1}^{k} \alpha_{ml} \Delta a_l = \beta_m \quad (7.6) \]

with

\[ \alpha_{ml} = \sum_{i=1}^{n} \frac{1}{\sigma_i^2} \left[ \frac{\partial f(x_i; a_n)}{\partial a_m} \frac{\partial f(x_i; a_n)}{\partial a_l} \right] \quad (7.7) \]

\[ \beta_m = -\frac{1}{2} \frac{\partial \chi^2(a_n)}{\partial a_m} \quad (7.8) \]
Using the $\Delta a = (a_0, a_1, \ldots, a_k$ as increments to $a_n$ will give a good but not optimal approximation of the correct solution. Since $\chi^2$ will most likely not be of perfect quadratic shape, we still have to repeat (7.6) over and over again and use the resulting $\Delta a$ as increments to the last parameter set $a$ until the result is close enough to the minimum of $\chi^2$. The advantage of the Inverse Hessian method is that the $\chi^2$ will be most likely nearly of quadratic shape close to its minimum. Thus this method converges faster to the optimal solution when $a_n$ is already sufficiently close enough to the minimum of $\chi^2$.

The idea of Levenberg was to combine these two methods. The Inverse Hessian method works well nearby the model’s solution while the Steepest Decent method is more applicable when the current set of parameters is still far away from the optimal parameter set.

Combining these two methods results in:

$$ \sum_{l=1}^{k} \alpha_{ml} \Delta a_l = \beta_m $$

(7.9)

with

$$ \alpha'_{ml} = \begin{cases} \alpha_{ml} (1 + \lambda) & (m = l) \\ \alpha_{ml} & (m \neq l) \end{cases} \quad \text{(7.10)} $$

In the case of $\lambda = 0$ equation (7.9) equals (7.6). Increasing $\lambda$ emphasizes the diagonal elements resulting in

$$ \Delta a_l = \frac{1}{\lambda \cdot \alpha_{ll}} \beta_l $$

(7.11)

Because $\beta$ is the gradient of $\chi^2(a)$, this is again the Steepest Decent method. Therefore the parameter $\lambda$ can be used to switch between the two methods. The basic iteration step now looks like this:

1. Compute $\chi^2(a_n)$ and set $\lambda$ to an adequate start value (eg. 0.001).
2. Determine $\Delta a$ and prepare $a_{n+1} = a_n + \Delta a$.

3. If $\chi^2(a_{n+1}) < \chi^2(a_n)$ do
   - $\lambda = \lambda \cdot 0.1$
   - $a_n = a_{n+1}$
   - Goto step 2.

4. If $\chi^2(a_{n+1}) \geq \chi^2(a_n)$ do
   - $\lambda = \lambda \cdot 10$
   - Goto step 2.

If the new estimates for $a$ get worse, $\lambda$ is increased to prefer the Steppes Decent method and with each further worse estimation the step size gets larger to look for a better solution further away from the current point. On the other hand, if the new estimate is better than the old one, $\lambda$ gets decreased assuming the Inverse Hessian method is suited better since the estimate gets closer to the minimum of $\chi^2$.

To speed up this iterative approach the model’s parameters are split up into a set of linear and a set of nonlinear parameters. The linear parameters are determined using a simple least squares method keeping the nonlinear parameters untouched. After the linear parameters have been calculated one step of the iterative algorithm for the nonlinear parameters is done. The linear parameters are kept constant now. The evaluation goes further and the linear parameters are calculated again using the new set of nonlinear parameters and so on. The complete workflow is shown in figure 7.1.

Of course a terminal condition must be defined to stop the iterative evaluation process. First of all a maximum number of iteration steps can be defined. This upper limit of iterations is useful when the algorithm jumps around the solution and does not improve the result anymore. On the other side it would be useful to abort as soon as the result is estimated ‘well enough’. The second terminal condition therefore is defined by

$$\frac{\chi^2(a_n) - \chi^2(a_{n+1})}{\chi^2(a_n)} < \epsilon$$

(7.12)

A result is considered stable as soon as the model’s error function $\chi^2$ does not improve the result by at least $\epsilon \cdot \chi^2(a_n)$. In practice a suitable value for $\epsilon$ is $10^{-5}$. 
Figure 7.1: The Levenberg-Marquardt nonlinear optimization algorithm. In each iteration loop the non-linear parameters are estimated by Levenberg-Marquardt and the linear parameters are determined based on the current non-linear parameter set.

[Stutz and Platt, 1996] show how to estimate the error of this nonlinear evaluation method, which is more likely to be overestimated by the statistical error due to artifacts in the residuals.
7.1.1 Further Optimizations to Levenberg-Marquardt

The Levenberg-Marquardt algorithm hardly relies on the calculations of the $D$ matrix which is the Hessian matrix of the $\chi^2$ merit function. Most implementations use the simple differential quotient method to determine the derivatives of $\chi^2$ in regard to the different model parameters $a$.

The derivatives are given by slightly modifying a single parameter $a_i$ and then reevaluate the whole model function with the modified parameter. The difference between these two results determines the derivation in regard to parameter $a_i$. Doing this for each parameter requires to evaluation the model function at least $k + 1$ times for $k$ parameters.

[Leue, 1999] introduced a Spline interpolation to be used in the DOAS model function. This has several advantages. Besides the need to interpolate the spectra data for shift and squeeze operations, the spline can be used to directly determine the necessary derivatives. The derivatives of the DOAS model function can be calculated by doing a precalculation step where the derivative of each reference spectrum is build from the spline’s derivative. Calculation of the model function’s derivative then reduces to several additions and multiplications. The overall performance gain is at about 30-60% for the whole evaluation procedure.

Furthermore the DOAS model does not take into account dependencies between the parameters of the different reference spectra. Often the nonlinear fitting produces false results by shifting or squeezing a reference spectrum into the wrong direction. This happens especially when a absorption cross-section has a periodic structure. Often the reference spectrum is then shifted at about half of the period’s width and vertically flipped thus resulting in a negative concentration. To avoid such results its convenient to use the shift and squeeze parameters of cross-sections with very strong absorption structures for several or all other cross-section as well. Also the cross-section are prepared in the same way making it senseless to allow different shifts and squeezes. Using the same values for each cross-section is more applicable in this case. This is done by linking the shift or squeeze parameter of a cross-section the the ones of another cross-section. [Leue, 1999] already introduced a link matrix to solve this problem. Linking parameters also reduces the number of parameters to be determined in the fitting process. Less parameters result in a faster and more stable evaluation.
The link matrix $L$, which is of dimension $n \times k$, maps the parameter vector $a_f$ to the model functions parameter set $a_m$. The fit parameter set $a_f$ contains $n$ parameters where $n \leq k$ and $k$ is the number of parameter of the model function.

Besides linking the different parameters the concentrations of some absorption cross-sections are already known before evaluation. It makes sense to still use this cross-sections during fitting. For example to determine the shift and squeeze parameters of an spectrum recorded by the GOME satellite, the Fraunhofer cross-section is still included in the evaluation although its concentration is known quite well (most likely it should be at about one). But the Fraunhofer cross-section has the strongest absorptions in the recorded data, so the shift and squeeze values from it are used as appropriate shift and squeeze parameters for all other cross-sections. Still using all already known absorbers in the fitting algorithm therefore stabilizes the results and reduces the residual's structures which is used to estimate the results quality. The parameters of such an well known absorption cross-section must be fixed to a certain value to get the DOAS model function behave in the describes way.

While linking and fixing a parameter sets it to a certain value, thus making it constant during evaluation, setting limits to a parameter still keeps it within the set of model parameters. But the range of the parameter is cut down to a defined interval within the solution space. Like linking the shift and/or squeeze parameters of cross-sections with periodic or other 'problematic' structures, a limitation is used so get similar effects. Linking makes a parameter dependent from another cross-section. Sometimes this behavior is not appropriate, for example, when the preparation and calibration of the different cross-sections do not match well enough. In such a case all parameters of all cross-section should be still available as free parameters during evaluation, but to avoid problems with too big shifts or squeezes, only a certain range is allowed.

The limitation of a parameter can be done using a penalty function that increases the $\chi^2$ error of a parameter set by a certain amount. Since the evaluation algorithm relies on the $\chi^2$ value, it discards a parameter set, if
the $\chi^2$ is higher than the previous one. By defining the following penalty function a limitation can be realized:

$$\chi^2_p(a_n) = \chi^2(a_n) + \sum_{i=1}^{k} P(a_i)$$

(7.14)

$$P(a_i) = \begin{cases} 
0 & (l_i \leq a_i \leq h_i) \\
\exp|a_i - l_i|p_i & (a_i < l_i) \\
\exp|a_i - h_i|p_i & (a_i > h_i)
\end{cases}$$

The $\chi^2$ error is only increased when the parameter $a_i$ gets outside its bounds defined by $l_i$ and $h_i$. The amount of penalty added depends on how far away the parameter is from its boundaries. We chose an exponential function which can be controlled by the correction factor $p_i$. The correction factor is used to define how hard the current error of a parameter in regard to its boundaries gets involved in the penalty function $P(a_i)$. In fact, the parameter $p_i$ gives the user the possibility to either still allow parameters outside their boundaries if the overall result fits better or strictly forcing a parameter to be inside its valid range. Very small values of $p_i$ make it more likely that a parameter can exceed its limits, while setting it to $+\infty$ will always set $\chi^2_p(a_n)$ to $+\infty$ as well in case of a boundary violation. The fit algorithm will always discard any parameter set with a $\chi^2_p$ of $+\infty$, since at least the initial guess has a valid non-infinite result.

### 7.2 Fit Framework

Besides the basic evaluation algorithm and the extensions presented in the previous section, further algorithmic extensions are already discussed in the DOAS community. Different model functions should be used like doing an intensity fitting instead using the logarithm space or including an additional non-logarithmical polynomial to cover further unwanted structures that arise from the detector or other physical processes. Also other fitting methods should be available to cross-validate the results. All these changes should be available, but still including all the functionality that is currently present.
Since DOASIS is already a widespread, flexible framework for handling spectra data, a subframework was included covering the mathematical part of the evaluation. The major goal was to have a framework that can handle different fitting algorithms as well as giving the user the ability to change the model function used to describe the DOAS problem.

Since there are already several enhancements present like the spline interpolation it would be nice if these enhancements are still used when possible but without the need to have the user to include these features himself in his new evaluation process.

The new approach to handle such an evaluation problem is based on the separation of the fitting algorithms and the representation of the model function that describes the problem. The basic idea is always to find an abstract representation of the model function. Other frameworks do this by using to called Functors [Silicon Graphics, Inc., 1994; Köthe, 1999, 2000]. A functor is an object that represents a mathematical operation or function. A minimal functor takes a set of input parameters and calculates an output value depending on the function it implements.

The fitting algorithm is represented by a separate object that gets a functor as parameter whereas the functor is the model function to be fitted to a data set.

DOASIS extends the idea of using functors is several ways:

- A functor should not only make the function itself available, but also its derivatives. A functor can contain an analytical derivative to speedup evaluation and improve accuracy. If no specialized derivation is given, it will default to a numerical method. Providing an analytical method to calculate the derivative of a function also makes the results numerically more stable, especially when function values are near the machine’s numerical resolution limit.

- A parametrized functor, which is a functor with additional function parameters, should have separate sets for the linear and nonlinear parameters. The evaluation algorithm can decide whether to use the linear or the nonlinear parameters or a combination of both. For example, if a fitting algorithm can only handle linear parameters, it will only operate on the linear ones. On the other side a nonlinear algorithm can optimize the linear parameters, too, and therefore should operate on both parameter sets.
• The functor should take care about the extended parameter features like linking, fixing etc. of a parameter automatically. So neither the user nor a fitting algorithm needs to know which parameter is fixed or linked. Of course, in mathematical terms this always is the same as using a matrix that maps the model’s parameters to a list of parameters that get optimized by the fitting process. What we want to avoid is to setup this mapping manually every time the model function changes.

But not only a complete change of the model function requires a new mapping matrix, also, for example, fixing a parameter has an effect to the mapping matrix. The functor should therefore itself take care about its parameters and only propagate the parameters involved in the optimization process. The fitting algorithm therefore does not need to know anything about linking etc. and can operate on the functor’s parameters without any limitations.

• A functor should provide a penalty method that gives a measure for the accuracy of the current parameter set. The default penalty method will be equation (7.14).

A model function should be build up by a tree of several functors where each functor describes a basic mathematical operation. For example, there should be a functor that does a summation of all operands, or another functor represents a spline interpolated data set. The evaluation model function can be setup by combining the different atomic functors where a functor can get other functors as parameters. The resulting tree of functor objects allows evaluations in a way similar to Divide-and-Conquer algorithms [Bentley, 1999]: A functor determines its current state by querying its parameters respectively its subfunctors when required. This increases numerical stability, since analytical methods are used where possible instead of numerical estimations, and can improve performance, when subfunctors provide optimized evaluation algorithms themselves.

The optimization algorithm should be exchangeable, too. In general an optimizer tries to minimize the error between a model function and a given data set. The error is determined using a specific merit function. To be able to use different merit functions for fitting, first of all the optimization algorithm no longer evaluates the error itself. The optimizer will get a functor and try to minimize the results of the functor itself \(|f(x; a)| \rightarrow 0\), thus the optimizing object becomes a minimizer actually. The merit functions are represented as functors as well and will get the model to be optimized and
the target data set. The model’s parameters are transparently propagated through the merit functor.

A change of the optimization method does not need any modification to the current model function or the merit function used. Also minimizers may be combined like functors to have a minimizer that is actually build up of a linear minimizer and a nonlinear one.

Of course the performance of the complete evaluation framework should be at least as fast as in the already existing implementations [Leue, 1999; Van Roozendael and Fayt, 2001], but still be more structured and more flexible.

7.2.1 Class Structure

DOASIS offers its evaluation methods in the DoasCore.Math.Fit namespace. Basically each functor is derived from DataSet which is just a representation of a wavelength-intensity mapping of the sampled spectrum. The DataSet is extended by the Function interface, which is the basic interface of a functor. It offers all the functionality of a function including calculation of its derivative. The ParamFunction is a further extension of the Function interface. It includes the complete handling of linear and nonlinear function parameters through a special ParameterVector class, which will be explained later. Additionally the derivatives in regard to the function’s linear and nonlinear parameters are calculated by this interface, too. So using either the Function or ParamFunction interface provides anything required to represent a functor.

Starting from these interfaces several predefined atomic function objects are available to provide basic mathematical operations. For example a object to sum up several other functors, a multiplication of two functors or a negation of a single functor is available. Additionally functors that interpolate the given data are present using different kinds of interpolation methods like linear interpolation and cubic spline interpolation.

Also more complex functions like the Gaussian function and a convolution can be used. To make evaluation of DOAS spectra easier a ReferenceSpectrumFunction is present. It already includes support to handle the parameters of a reference spectrum as well as including spline interpolation and
the optimized calculation of the reference spectrum derivative in regard to a parameter.

Currently there is only one merit function available namely the $L_2$ merit function.

So far everything necessary to build a model function and determine the model's error is available. The minimizers need to implement the Minimizer interface. It expects a `ParamFunction` object as parameter and will optimize the parameters of this object by minimizing the functions results. Currently a simple least squares minimizer and a Levenberg-Marquardt minimizer is available. The `StandardFit` class itself just uses the least squares and the Levenberg-Marquardt minimizers to do the combined evaluation procedure explained in section 7.1.

![Diagram](image.png)

**Figure 7.2:** The object structure of the DOASIS Fit Framework. A model function is made up of several functors like the `ReferenceSpectrumFunction` and the `PolynomialFunction` classes and combined to a DOAS model function. This model function is used by the Minimizer to determine the parameters of the model function.

The `ParameterVector` class keeps track of a specific set of parameters of a functor. So the `ParamFunction` class contains two parameter sets. One for its linear parameters and one for its nonlinear parameters. The `ParameterVector` is responsible to handle all special operations of a parameter. Besides the possibility to set a parameter to a fixed value, thus making it 'invisible' to the minimizer, linking and limitation through a penalty function is included. Inside the `ParameterVector` a self organizing structure keeps references to all functors that have parameters linked.
to the ParameterVector’s parent functor. Linked parameters are removed from the exported set of function parameters and any modification of a link source is propagated to all attached function objects. Its even possible to have a cascading link structure. Of course, all function objects contain an analytical method to get a functions derivative. Special care needs to be taken about linked and fixed function parameters, since this may changes the function’s derivative.

The framework provides an easy-to-use way to handle all the needs of parameter management. While all other evaluation procedures require a manual mapping between the parameters used by an optimizer and the model’s parameters, DOASIS takes care about this problem automatically. But also changes to the model function or the optimizer get simpler. An exchange of one or more functors change the model function in the desired way or new optimization strategies are used by exchanging the minimizer object during evaluation.

7.2.2 Modular Extensions

Further functors that either provide simple atomic or complex operations can be included by implementing the required interface ParamFunction. The same applies to new merit functions or optimizers which need to inherit the Minimizer definition. But this does not necessarily require to change DOASIS internally. The interoperability of the .NET framework makes it possible to define the new objects from a JScript and use this scripting object during evaluation.

Of course an evaluation using user supplied functors or optimizers requires a far more complex automation script than using the already provided objects. But to test and validate new approaches in DOAS analysis, DOASIS provides a way to do this with a minimal effort.

7.2.3 Performance Considerations

Although the DOASIS fitting framework uses an object oriented approach to represent fitting algorithms as well as model functions, its performance should be comparable to present fit implementations using a straight forward functional model. Avoiding the bottlenecks of an object oriented
environment like object creations and strict use of by-reference techniques are the design goal of the fit framework in regard to performance.

Extensive object creation, for example, is a real performance killer. Each object that gets created needs to be allocated on the application’s heap and the garbage collector has to add it into his tracking structures to be able to determine whether the object is still in use or if it can be disposed. Additionally, besides the performance penalty when creating an object, large numbers of objects that are only used for a short period of time causes the garbage collector to start cleaning up the heap and relocating still used objects.

Additionally using by-value parameters has the same effect like creating large sets of small objects. Each time a by-value parameter is passed to a method, the runtime needs to create a copy of the object.

To avoid these problems, the fitting framework uses its build in Vector and Matrix classes to focus on reusing existing data objects. Data arrays are tried to only get allocated once and will be reused by other Vector or Matrix objects by simply referencing to the same data array but using different access semantics like defining an area of interest or use different subsets of the data object.

Besides object reuse, special care was taken to only use the intrinsic data types and basic array types instead of (more convenient) lists or collections. Using arrays with intrinsic data types gives us the benefit of highest performance. The intrinsic data types are directly mapped to data types of the underlying processor and arrays will be allocated in continuing data blocks. Thus an array access gets reduced to a simple indexed read or write operation of the processor.

And, of course, common code optimizations like loop unrolling are used as well [Bentley, 1999; Odersky and Philippsen, 1995].

7.2.4 Numerical Considerations

DOASIS internally uses the IEEE 754 Double Precision format. This floating-point format was chosen to get the best compromise between performance and accuracy. Modern FPUs include direct support for this kind of data type, thus no software based mathematical operations are required and DOASIS will gain the full power of the processor(s).
The IEEE 754 Double Precision format is a 64-bit floating point format that uses 53-bit for precision and a 11-bit exponent. Due to the nature of DOAS, reference spectra only contain very small values, e.g. a $SO_2$ cross-section is in the range of $10^{-19}$. Several operations during a DOAS evaluation scale such a cross-reference by very large values or even worse it is necessary to build the exponential of it like for the Intensity Fitting, which is described in section 7.2.6.

These kinds of operations quickly run into numerical problems. The IEEE 754 Double Precision format can store a precision of up to $2^{-56}$. But, for example, the exponential of $10^{-19}$ needs a precision of at least $2^{-63}$ to keep the necessary decimal parts. Therefore building the exponential of $10^{-19}$ using the double precision format results in exactly 1 without any decimal places. So any further operation with such a data set includes a significant error.

To minimize the effects of numerical round-off errors, DOASIS internally allows to rescale any reference spectra into a range between 0 and 1 to ensure all reference spectra are in a comparable value range and have about the same precision. Additionally in critical fitting operations, like the Intensity Fitting, a pre-evaluation step is included that estimates the fit coefficients of each reference spectra. This scales the reference spectra into a numerical more reasonable range and stabilizes evaluation as well. [Goldberg, 1991] explains the numerical problems that arise in scientific applications on computers. Of course, internally several well known numerical methods are used to avoid numerical problems. For example, to solve linear equation systems the LU-Decomposition or the Gauss-Jordan Elimination with Full Pivoting is used as described in [Press et al., 1995].

A possible solution for these problems would be to use the System.Decimal or decimal data type of the .NET framework. This 128-bit data type uses 96-bit precision and unlike the IEEE 754 format the basis of the mantissa is 10 and not 2 what resolves some number representation problems as well. However, there is currently no hardware support available for this data type, thus, evaluation performance would drop dramatically.

Let’s see, if the fitting framework and its optimizations produce comparable results in regard to accuracy and performance. But first the general workflow about how to use the fit framework will be described.
7.2.5 Evaluation Setup

How is the current DOAS evaluation setup using the new evaluation framework? Figure 7.2 shows the object dependency tree and figure 7.3 the general process flow to setup an evaluation.

First we build up the DOAS model function starting with a set of ReferenceSpectrumFunction objects. Each of these objects represent a single cross-section. Linking, fixing and limiting a parameter will be applied to the appropriate instances of this class. Further more the current DOAS model includes a polynomial of a certain order $m$. So an object of type PolynomialFunction is required. All these objects are put together using the summation object. The summation object represents the root instance of the DOAS model function. All linear and nonlinear parameters of the different sub-objects are propagated appropriately through the summation object.

![Diagram of Model Function and Fitting](image)

**Figure 7.3:** General process flow to use the fit framework. First create the model function based on the given atomic functors or by creating a custom model function object. Second create the fit object. Putting the model function object into the fit object starts the evaluation process.

The next step is to setup the merit function. The measured data and therefore the target function of the model will be represented by a Discrete-
Function object, which actually represents a set of data points as a function without any interpolation. The merit is build using the StandardMetricFunction object which builds the $L_2$ merit between the model function and the measured data.

As a last step the minimizer must be setup. An instance of StandardFit contains the optimization explained in section 7.1. The minimizer gets the merit function object as the only parameter.

The minimizer can now start its optimization strategy. At the end of the optimization all function objects contain the result’s parameter set. Of course, each function object only contains the results of the parameter that belong to it. So each ReferenceSpectrumFunction object contains its VCD, shift and squeeze parameter while the PolynomialFunction object only holds the polynomials coefficients. All statistical errors and correlation informations are available by the function objects as well.

### 7.2.6 Performance Analysis and Comparison

One of the goals in developing the evaluation framework was to get an object orientated environment that should be as fast as the already present fitting implementations. This was a hard task, since there were several additions to the evaluation procedure that make the problem even more complex like the several modification methods of parameters. But also more flexible interpolation methods should be used that do no longer rely on data sets with equidistant samples. Namely it should be possible to use a wavelength scale instead of spectrometer channels, since most devices do not have a linear wavelength dispersion. A spline interpolation that operates on any dispersion must be used and special care needs to be taken to gain the same performance as when using spline optimizations on equidistant grids. At the end the new framework got comparable results in regard to the previous implementations from [Leue, 1999] and [Van Roozendael and Fayt, 2001], which will be shown in the following sections.

**DOASIS Fitting Methods**

DOASIS offers two basic types of fitting methods: The Optical Density Fitting and the Intensity Fitting where these parameters are available:
1. The Optical Density Fitting

This evaluation method will later be referred to as DOAS Fit. Basically it implements the well known DOAS evaluation procedure that fits several reference spectra against a measured spectrum in logarithmic space. Additionally an offset polynomial that resides in the intensity space can be added. The actual equation used for evaluation is this:

\[
\ln I(\lambda) = \sum c_i \sigma_i (s_i + t_i \lambda) + P(\lambda) + \frac{O(\lambda)}{I(\lambda)}
\]  

(7.15)

where \(O(\lambda)\) is an additional offset polynomial in the intensity space to accommodate broad band structures in this domain.

In order to increase numerical stability of the offset polynomial, the above equation uses the approximation

\[
\ln \left( 1 + \frac{O(\lambda)}{I(\lambda)} \right) \approx \frac{O(\lambda)}{I(\lambda)}
\]  

(7.16)

for small \(O(\lambda)\). Only the shift and squeeze parameters have to be evaluated with a non-linear Levenberg-Marquardt algorithm. All other parameters are solved using a Least Squares linear fit algorithm.

2. The Intensity Fitting

Intensity Fitting is done by evaluating the Lambert-Beer’s Law in its original equation where all references are in the exponent of the light source’s intensity:

\[
I(\lambda) = e^{\sum c_i \sigma_i (s_i + t_i \lambda) + P(\lambda)} + O(\lambda)
\]  

(7.17)

Almost all parameters are non-linear except the coefficients of the offset polynomial. Since the Intensity fit quickly gets numerical unstable, a pre-least squares fit is done to get proper starting values for the cross-section coefficients.

DOASIS allows the user to link, fix or limit any of the available parameters. Also take into account that the equations above given are missing the \(I_0(\lambda)\). DOASIS expects the user to provide the light source spectrum as an additional cross-section reference spectrum in the fit setup. Since mass evaluations in DOASIS are done with the use of JScripts, all reference spec-
tra can be modified before and after each evaluation of a single spectrum. Therefore most of the functionality present in other DOAS evaluation applications can be done in these scripts as well. This offers a very flexible way for any kind of DOAS evaluation.

**Other DOAS Fit Implementations**

The DOASIS fit implementation is compared against two other DOAS evaluation applications.

The first reference fit implementation is the formerly used DOAS fit used in the satellite group of the IUP to evaluate GOME spectra done by [Leue, 1999]. This implementation only provides the DOAS Fit method without any additional offset polynomial. Also only the shift and squeeze parameters can be linked to another reference spectrum and it is not possible to fix any of them. The aim of this implementation was to get a stable and fast evaluation of large data sets. The implementation was successfully compared to the MFC DOAS evaluation and to the NO$_2$ data provided by the DLR and showed a correlation coefficient of about 0.994 to these other two DOAS evaluations [Leue, 1999].

Second the DOASIS Fit is compared to the WinDOAS evaluation methods. WinDOAS offers a Optical Density Fitting, an Intensity Fitting with an multiplicative DOAS polynomial and a fully non-linear Intensity Fitting. The basic equation used in WinDOAS is a little bit different to the one used in DOASIS:

\[
I(\lambda) = I_0(\lambda) e^{\sum c_i \sigma_i(s_i + t_i \lambda) + P(\lambda)} + Offset(\lambda) + U(\lambda)
\]  

(7.18)

where

- \( Offset(\lambda) \) : Instrumental offset between the spectrum to analyse and the control spectrum.
- \( U(\lambda) \) : Undersampling cross-section.

The offset is the same as the offset in DOASIS: \( Offset(\lambda) = \frac{O(\lambda)}{I(\lambda)} \).
**Evaluation Setup**

To compare the results of the different implementations, a $NO_2$ analysis of the spectra of one GOME orbit was done with each application. The orbit file is called 90101010.lv1 and contains 2233 spectra. The range between 430-450nm is used for the $NO_2$ evaluation.

As reference spectra the Fraunhofer spectrum contained in the orbit file is used. Additionally the Ring spectrum, an $O_3$, the $NO_2$ and the $H_2O$ reference spectra are fitted against the measured spectrum. All shift and squeeze parameters are linked against the Fraunhofer spectrum. The DOAS polynomial fitted has an order of 3 and where possible an offset polynomial of order 2 is included.

While the Satellite Fit and DOASIS directly read the data from the GOME orbit file and calculate their own Ring reference, the necessary spectra and references are provided to WinDOAS in the STD-file format.

Application versions used:

- DOASIS 3.2.1906
- WinDOAS 2.10
- Satellite The latest implementation from the SourceSafe database

In WinDOAS no further settings are used than adding the reference spectra and the two types of polynomials to ensure a proper comparison.

**Results of the Comparison**

All results will be compared to the results of the DOASIS DOAS Fit without any additional offset polynomial. Only the $NO_2$ VCDs are taken into account here.

At first the different fit methods available in DOASIS itself are compared against each other to validate them.

Figure 7.4 compares the DOASIS Fit against the DOASIS Fit with an offset polynomial. The correlation coefficient is 0.96. The discrepancy of 4% is caused by the second polynomial added.

The results of the DOASIS Intensity Fit against the DOASIS DOAS Fit are shown in figure 7.5. Here we reach a correlation factor of 0.98. So the Intensity Fit matches pretty much the results of the numerical more stable DOASIS Fit.
Also the DOASIS Intensity Fit with additional offset polynomial reaches a correlation factor of 0.99 as shown in figure 7.6.

All results so far have a slightly offset compared to the DOASIS DOAS Fit results. As soon as the offset polynomial is added the NO$_2$ concentration decreases. But that is what can be expected if another absorber is added.

A nearly perfect match is achieved with the original DOAS Fit of the Satellite Group at the IUP. The correlation factor in figure 7.7 is 0.99.

Using the DOAS Fit (WinDOAS calls this the Optical Density Fit), DOASIS and WinDOAS produce the same results. The correlation factor is 0.99979 and an offset below a factor of 0.001 can be found (figure 7.8).

Adding the offset polynomial in WinDOAS shows a little bit more mismatches in the evaluation. In figure 7.9 slightly higher discrepancies for very low or very high values can be seen, especially for small concentrations where differences in the evaluation and implementation have a higher impact. The correlation factor drops to 0.95, which is still in the range of the statistical errors of the evaluation.
Unfortunately WinDOAS was not able to evaluate any of the spectra using the Intensity Fit regardless of using the Full-Marquardt or the Marquardt-SVD evaluation. However, it was not clear, if the setup was correct or if WinDOAS encounters numerical problems in these cases. Changing the fit setup resolved the problem in WinDOAS, but since the fit setups do not match anymore in this case the results were not taken into account during this comparison.

**Evaluation Speed**

Besides the results of the NO$_2$ analysis also the time was taken into account that the different implementations needed to find a stable result. Figure 7.10 compares the evaluation times each application needed for evaluating the test GOME orbit.

DOASIS showed good results here as well. Its C++ implementation, which is used in the Satellite group now, was about twice as fast as the original one used. And even the C# implementation is slightly faster than the original
C++ code although everything is done in an object orientated way under the managed environment of .NET.

In the current evaluation setup WinDOAS has one big disadvantage. All measurement spectra need to be present in separate ASCII based files. Reading the content of ASCII files is a very expensive operation while DOASIS preloads all spectra of an orbit into memory before starting evaluation. The assumption here is that most time is required to load the spectra. Comparing the evaluation speeds therefore is a bit unfair to WinDOAS.

One of the reasons why the DOASIS fit is faster than the other fits is the fact that it needs less non-linear iteration steps to find a stable solution. This is achieved by the way the DOASIS Fit Framework is implemented. Each atomic evaluation function element offers an analytical derivation of itself in regard to the various kinds of parameters present. Having an analytical derivation available increases the accuracy of it and eliminates the need to evaluate the whole model function twice or more times to get its derivation through a differential quotient algorithm. Higher accuracy of the gradient
vector used to estimate the next non-linear solution reduces the number of iteration steps necessary to find the optimal solution.

Conclusions

DOASIS shows very good correlations with the existing DOAS evaluation implementations. However, there still will be situations where DOASIS will fail to produce the same results as WinDOAS. Especially in cases where the maximum numerical resolution of the system is reached round-off errors can have a major influence [Überhuber, 2002]. This can quickly happen in the case of Intensity Fitting.

When comparing the results it has to be ensured that all evaluation parameters are the same. WinDOAS offers a wide range of preprocessing methods which are not automatically included in DOASIS like the Undersampling Correction and many other things. But most of the precalculations can be accomplished with small JScript applications in DOASIS as well.
Figure 7.8: Comparison of the DOASIS DOAS Fit against the WinDOAS DOAS Fit.
**Figure 7.9:** Comparison of the DOASIS DOAS Fit against the WinDOAS DOAS Fit with additional offset polynomial.

**Figure 7.10:** Evaluation times needed by the different fit implementations to process 2233 spectra of one GOME orbit on a 1.5GHz Pentium M processor.


8 Multiscale Analysis

The basic idea behind DOAS is the separation of the measured signal into parts that contain the desired information and a part that contains unwanted distortions. The current evaluation procedure requires a manual separation of these two parts by applying high and low pass filters with a user defined frequency separation to the measured data. Remaining structures are tried to be covered by a polynomial of a certain degree.

The parameters chosen for the high pass filter and the polynomial can have an effect on the evaluation result. A polynomial with a too high order can overlap with absorption structures. The high and low pass filtering is done heuristically. The filters are chosen to remove most of the unwanted broad band structures and the high frequent noise. But there is no defined way to specify the border of a useful spectral range.

Similar problems are commonly known as Homomorphic Filters [Oppenheim and Schafer, 1975]. These filters also split the incoming signal in a high frequency and low frequency part. Depending on the algorithm’s needs further operations are applied either to the high or low frequency part. Homomorphic Filter are used in the signal processing environment to extract information from a certain frequency band while keeping the rest of the signal untouched.

To get rid of the polynomial, thus the need to specify the polynomial’s order manually, an appropriate high pass filter is needed. On the other hand an adequate low pass filter should be applied to remove noise in the signal. An optimal band pass filter is supposed to result in a spectrum that contains the required absorption structures only.

8.1 Laplace Pyramid

A commonly used way to build a set of band pass filtered spectra is the Laplace Pyramid. An example Laplace Pyramid of a spectrum can be seen in figure 8.1. The Laplace Pyramid itself is based on the Gauss Pyramid which is a set of low pass filtered spectra [Jähne, 1997; Kraus et al., 2001].

The Gauss Pyramid is build starting from the source spectrum doing the following sequence:
1. Apply a smoothing filter to the spectrum so that the result only contains the lower half of the source data’s frequency range. Commonly this is done using a binomial filter.

2. Downsample the low pass filtered data to half of the resolution of the original data set.

3. Start over with the downsampled data set at point 1., if the data set is still sufficiently large enough. Commonly the downsampling can be aborted, if the result does only contain 32 or 16 data samples.

The low pass filtering can be done using a binomial filter mask. The transfer function of a binomial mask of width 16 ($B^{16}$) is close to a low pass filter that cuts off all frequencies that are present in the upper half of the spectral range of the incoming signal. [Jähne, 1997; Gröning, 1996] shows that an even faster low pass filtering can be achieved by first using a cascade of smaller binomial masks. Actually it’s enough to first use a $B^4$ mask. The result gets downsampled using the reduction operation $R$ and again a filter $B^2$ is applied. The result is the same as using the $B^{16}$ mask but with less operations.

Figure 8.1: Example Laplace evaluation of a GOME spectrum. Each Laplace level contains the upper half of the frequency range of the previous level.
The Laplace Pyramid algorithm. In each iteration a downsamples and low pass filtered spectrum of the previously created spectrum is calculated. The new spectrum will be subtracted from the previous one so at the end a set of high pass filtered spectra is created.

The Laplace Pyramid uses the low pass filtered data from the Gauss pyramid. To get a set of band pass filtered data, the downsampled levels are expanded again and subtracted from their preceding level. The different data sets $L_n$ of a Laplace Pyramid are determined from the data set of the Gauss Pyramid $G_n$ and the expanded data set of the next Gauss Pyramid level $G_{n+1}$. This is repeated until the last pyramid level $h$ is reached. Figure 8.2 shows the workflow to generate the Laplace Pyramid.
\[ L_n = \begin{cases} 
  G_n - R^{-1} G_{n+1} & (n < h) \\
  G_n & (n = h) 
\end{cases} \quad (8.2) \]

Since in the Gauss Pyramid the next level \( G_{n+1} \) contains the lower half of the frequency range of the current data set \( G_n \), the resulting Laplace data set \( L_n \) will contain the upper half of the frequency range of \( G_n \). Each level \( L_n \) therefore contains the data of a band pass filter, where the width of the filter gets halved with each further pyramid level. The last level of the Laplace Pyramid will be the same as the one of the Gauss Pyramid.

If we do a frequency analysis of the given data, the *Nyquist-Shannon* theorem defines that the maximum frequency contained in our data set can be \( \frac{N}{2} \) where \( N \) is the number of samples. Given this, the Laplace pyramid produces a set of spectra that contain the frequency range between

\[ \frac{N}{2 \times (l + 1)} < f < \frac{N}{2 \times l} \quad (8.3) \]

for a given pyramid level \( l \).

### 8.1.1 Evaluation Analysis

The results of the multiscale analysis have been compared to existing DOAS evaluations. In particular the same evaluation setup as for the validation of the results of the DOASIS Fit Framework was used (see section 7.2.6). Again a \( NO_2 \) evaluation was done using the same GOME orbits and the same DOAS fitting parameters which include a DOAS polynomial of order 2.

Of course, the DOAS evaluation for the Laplace pyramid results was a little bit different compared to the standard DOAS model function used: Since we want to use the Laplace pyramid to find an optimal frequency range for evaluation, no DOAS polynomial or offset polynomial was used.
Results

Following the results of the \( \text{NO}_2 \) standard DOAS evaluation will be compared to the results of the DOAS analysis with Laplace pyramid filtered spectra. Figure 8.3 and 8.4 show the correlation plots when level three and two of the Laplace pyramid where used for evaluation.

![Figure 8.3: Comparison of the DOASIS DOAS Fit against level three of the Laplace Analysis.](image)

The results for these two filter ranges are disappointing. Figure 8.3 shows that the evaluation failed at all for all spectra. The evaluation of Laplace pyramid level two reaches an average correlation factor of 1.75567 but the single results have high errors and the results can’t be really treated as matches.

If we go one level up in the Laplace pyramid, respectively using level one which represents a frequency filter of the range \( \frac{N}{8} < f < \frac{N}{4} \), the results get better. The correlation factor of the plot in figure 8.5 is 0.5797, but the average error is till around 0.01768. Especially for small concentrations of \( \text{NO}_2 \) the evaluation seems to fail.

Using the upper half of the frequency range of the spectrum, we can increase the correlation of the results to 0.99793 (figure 8.6). Although this is very
close to an optimal match of the evaluation results, there is still an average variation of the results of about 0.002 while the correlation of the different fitting methods in section 7.2.6 was about a factor of approx. 100 better.

So simply using the results of the Laplace pyramid did not show the expected results within an acceptable error range. But the results of figure 8.6 show that the necessary information about the relevant spectral structures is contained in the frequency range $N/4 < f < N/2$. This frequency range also contains all the noise. The idea now was to reduce the noise level in this frequency range and repeat the evaluation. Therefore a second bandwidth filter was applied that cuts the frequency range to $N/4 < f < 3N/8$.

This time the correlation factor went down to 0.98824, but the variance of the correlation factors was reduced to 9.06513·$10^{-4}$ which is in the same range as the correlations of the standard DOAS evaluations have. Figure 8.7 clarifies these results.

Unfortunately the frequency range used for the last DOAS-Laplace analysis cannot be directly retrieved from the Laplace pyramid. But using the
knowledge of the results previously presented, a single filter can be applied to get the optimal frequency range in a single preprocessing step without the need to create the full Laplace pyramid. Although the filter algorithm used for the Laplace pyramid which is based on simple binomial filters still can be used as a basis for a fast bandwidth filter.

Conclusions

The Multiscale Analysis using a Laplace pyramid seems to be a promising approach to overcome some of the uncertainties in the DOAS analysis. Good correlations between existing results and multiscale evaluations without the need to use a polynomial of a certain order. Also the relevant frequency window for the \( \text{NO}_2 \) evaluation could be determined. This avoids the heuristical high and low pass filtering that is normally used. The optimal frequency range is between \( \frac{N}{4} < f < \frac{3}{8} N \).

Unfortunately no performance improvement could be achieved. The optimal frequency window still requires the same resolution as the original spectrum.
Figure 8.6: Comparison of the DOASIS DOAS Fit against level zero of the Laplace Analysis.

has. The performance gain due to the missing polynomial during the fitting process is outweighed by the necessary preprocessing.

The results presented here, of course, are only valid for the given NO$_2$ evaluation in the range of 430-450nm. For other spectral ranges or species, the optimal analysis window must be determined as well separately.
Figure 8.7: Comparison of the DOASIS DOAS Fit against level zero of the Laplace Analysis with an additional binomial low pass filter.
Multiscale Analysis
9 Summary And Outlook

Since DOAS is a very powerful tool to measure trace gas concentrations even over large distances, the DOAS analysis gets more and more important in monitoring our environment and identify pollutants or identifying complex processes in our atmosphere.

Tools are necessary to make the DOAS measurements and evaluation as simple as possible to provide this method to a wider range of users and applications. DOASIS, of course, does not claim to be the all-in-one tool for DOAS, but the DOASIS framework can be used as basis for measurement and spectrum post-processing. An easy to use user interface is included with an open structure to be ready for future needs. DOASIS offers a framework to get the necessary hardware support for measurements, the data management and the evaluation of the recorded data from one hand and through a common interface structure. Not only developers of components for DOAS measurements and evaluations benefit from the framework. Users also do not have to handle different tools for different processing steps, thus the possibility for evaluation errors gets reduced. Support for various kinds of devices can be controlled through a common interface including a unique solution to overcome parallel processing of non reentrant device drivers.

The necessary DOAS evaluation tools are available in a flexible fitting framework that improves evaluation performance and tries to make modifications in the evaluation process easy. All the components of DOASIS can be used in an automation environment that is based on the JScript language which is easy to learn and understand even by non software developers.

The primary goal of the DOASIS framework design was not to develop a monolithic set of classes. An easy way for other developers is offered that makes integration of third-party components a simple task: Write a component that offers the necessary interfaces and add it to the configuration. DOASIS takes care of creating the necessary runtime instances as well as offering its functionality to the UI. External modules can be integrated through the .NET interoperability infrastructure that offers a wide range of connection points to other components. So even unexperienced developers can write new (UI-)components.

However, sometimes an external component is not enough and the framework itself needs to be extended or necessary maintenance tasks should
be done. Due to the complexity of the framework, which up to now has grown to about 100,000 lines of code and 60,000 lines of documentation, and its unique object oriented approach, special care needs to be taken to not break any of the existing interactions or introduce performance penalties or dead locks. For example, the event model was introduced to notify an UI attached to the framework about changes automatically. Easily small modifications can lead to bouncing events and dead locks.

Using JScript for automation made writing user specific measurement or evaluation routines simple and allows even non software developers to quickly get started and do their first measurements. Nevertheless, experience shows that many users are overstrained by the huge set of objects and methods available. Also users need to get used to think in the "object oriented" way to fully understand how a script works. More common tasks need to be directly integrated into the framework and the UI to increase usability in such a manner that only a limited set of functions is made available to non-experienced users. But DOASIS is still a framework to fulfill scientific needs, thus its flexibility and capabilities should not be limited to only provide a few standardized evaluation methods.

Additionally a new DOAS analysis method was evaluated: The Laplace pyramid. The Laplace pyramid was used to find the optimal bandwidth filter for NO\textsubscript{2} evaluations in the spectra range between 430-450nm. It was shown that using a bandpass filter overcomes the need to approximate broad band structures using a polynomial and to pre-process the measured spectra using low- and high-pass filters. The results of this method are quite promising and further investigations need to be done to find the optimal bandwidth filter for other spectra ranges and species.

Fortunately the DOASIS user community is constantly growing and it its already used in various work-groups across the world. It is used in training courses for students as well as in sophisticated research projects, which demonstrates its flexibility and usability. Even cross-platform support is working in its first stage as appendix F shows. Of course, development of DOASIS is not complete yet, but the framework introduced in this work offers a feasible basis that can be extended to meet future requirements.
Part IV

Appendix
A Application Description

This chapter will give a short overview about the user interfaces that are available.

A.1 Console Application

The DoasConsole application offers only very limited functionality in regard to user interaction. Basically it was designed to run DOASIS JScripts without the need to have the resource expensive GUI running. Therefore it only expects the name of a script given during startup (figure A.1). It will only run the JScript and display its console output in a simple command window. Also proper mapping of several UI components are transparently added to still enable display of progress bars etc.

Figure A.1: Command window of the DoasConsole component.

A.2 Graphical User Interface

DoasUI is the main component that offers a rich graphical user interface. Like most modern UI applications, tabbed MDI displays combined with self adapting menu and toolbar components have been integrated.

All additional components are attached to the main window as docking toolboxes. The toolboxes can be placed or docked anywhere in the application window. So the user can adapt the whole workspace to his needs.
A.2.1 The Main Window

The main window component is used to display the currently selected spectrum. The spectrum display can be configured in several ways. Besides simple operations like zooming or panning, overlays or multi chart displays can be activated to get a quick visual feedback on evaluation results (figure A.2).

![Figure A.2: The main component of the DoasUI. The spectra are available through a tabbed MDI user interface.](image)

Menus and Toolbars

The menus and toolbars are directly connected to each other. Selected menu items are available as toolbar buttons as well. So quick access to the most commonly used operations can be realized through the toolbars (figure A.3).

A.2.2 Properties and Statistics

All properties that are associated with a spectrum are displayed in the properties toolbox window as shown in figure A.4 a). But the properties control not only displays the content of each property. Also read-write properties can be edited directly. So changing, for example, the name of a spectrum just requires to enter the new name in the proper edit box. Event collections like the calibration polynomial coefficients or the intensity values are available and allow to change, add or remove single values.
Figure A.3: Example of the menu structure. Selected math operations are available in the toolbar as well. These items have the same icons. By default only items that are frequently used will be displayed to increase usability of the menus.

Each spectrum already comes with a huge set of predefined properties. If additional properties are required, the user can simply add them through this dialog and they will become immediately available and are even accessible in JScripts through the Properties collection.

Additional views of the properties can be selected. The properties can be organized in categories or simply sorted in alphabetic order. The subset of the statistical properties are displayed in a separate window as well to provide a quick overview about these values.

A.2.3 Specbar

The Specbar is the central spectra management component in DOASIS. Internally the Specbar keeps track of all spectra that have been loaded into DOASIS and allows easy access to them through a collection indexer. The Specbar component of the UI is the graphical part of this management tool.
Application Description

Figure A.4: The properties of the currently active spectrum as displayed in a). Read-Write properties can be modified directly and new user defined properties can be added. b) The statistics subset of the spectrum properties is displayed in a separate window to get a quick overview about the characteristics of the spectrum.

The list of all spectra is displayed where thumbnails give a quick overview of the content of a spectrum as shown in figure A.5.

A.2.4 Spectrographs and Devices

Spectrographs and other devices that are manageable by DOASIS, are controlled by the three UI components shown in figure A.6. These components display the properties and functions of a device in a generic manner so that different kinds of devices are presented in a common way.

Figure A.6 b) shows a device that offers several device interfaces. For each kind of interface a generic component is added that is available through the tabs on top of the device properties. Device settings are changed by simply editing the properties in the properties lists. Changes are directly reflected to the device.
Figure A.5: The specbar lists all spectra that are currently loaded into DOASIS. The specbar is the central spectrum management tool.

DoasUI also periodically checks the health state of a device and provides a quick user feedback through a status icon in the right corner of the status bar.

A.2.5 Console Output

The console toolbar is a simple text based input/output window where any kind of information can be displayed (figure A.7). Mostly JScripts will make use of this window to display information to the user.

A.2.6 JScripts, MiniScripts and QuickScripts

The JScript engine of DOASIS can be used to run little automation routines. Besides the basic functionality to select a script, start and stop it (figure A.8a), DoasUI offers support for MiniScripts (figure A.8b) and QuickScripts (figure A.8c). A MiniScript is a very small JScript. You can use the MiniScript toolbox to directly enter the JScript code and run it. There’s no need to create a separate script file.
Figure A.6: These three tool windows make up the components to control spectrographs and devices. a) is the main control panel to record a spectrum through a connected spectrograph. b) is the main control panel for all other kinds of devices like the MiniDOAS USB controllers. Depending on the device’s capabilities each tab will allow access to a specific device module. c) shows as an example the control panel of the ADC module of the MiniDOAS device.

Figure A.7: The console input/output window. All text input and outputs of scripts are redirected into this window.
QuickScripts associates selected JScripts with hot-keys. The script will be executed whenever the hotkey is pressed. Automation scripts can so be used for special operations that will occur frequently and are not handled by one of the standard UI components.

**Figure A.8:** DoasUI offers several ways to handle JScripts. a) is the main component to control JScripts. A script can be selected, started and stopped. b) can be used to directly enter small JScripts without the need to create a separate script file. Also it is useful to assign scripts to certain hotkeys which can be done in c).

### A.2.7 Fitting

The fitting toolbox offers everything to configure a DOAS evaluation. Selection of the necessary reference spectra as well as defining the additional fit parameters is done through this dialog shown in figure A.9. The result of an evaluation gets shown in the main window as a multi-chart view that shows each reference spectrum and the additional evaluation components like the
polynomial. This visual display gives a quick feedback on the quality of the evaluation results.

**Figure A.9:** The main control dialog to setup the evaluation parameters and start a spectrum evaluation. a) is available as one of the docked control panel so fitting parameters can be quickly changed and an evaluation started by just two clicks. b) shows the detailed fitting parameters that control the evaluation like the selection of the fitting model used.
B  Extend DOASIS

In chapter 5 the internal structures and work flows of DOASIS are described. One of the major topics during development was to keep DOASIS as open as possible to allow other developers or users to extend the functionality without the need to modify the core application. Most of the components that are available in DOASIS can be configured from outside and new components can be plugged in by adding only a few changes to the application configuration.

The next sections will describe the available components and how to configure user created components so they will be accessible in DOASIS. All configuration settings defined here are stored in the system registry database under the following main key:

\texttt{HKEY\_LOCAL\_MACHINE\Software\Institute \ of \ Environmental \ Physics}

On non-Microsoft Windows systems, simple text-based property files are used. Subkeys are separated by a dot instead of a backslash to meet the properties notation. The naming convention of these property files is \\
\texttt{<Assembly Name>\_Machine.props} where Assembly Name is either DoasCore or DoasUI.

B.1  Menu Structure

The most common way to present the functionality to a user through a UI is to add a menu item to the application. In DOASIS the whole menu structure is defined in the application’s configuration under DoasUI\Menu. Each menu item is defined by a separate subkey. The hierarchical structure of the menu item subkeys directly reflects the menu hierarchy.

The hierarchical structure of a menu requires two kinds of menu items:

B.1.1  Parent Menu Items

A menu parent item is the root for a whole set of menu items that will be presented as a submenu below the parent node. The whole DOASIS menu
Extend DOASIS

Table B.1: Configuration settings to define a parent menu item.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MenuText</td>
<td>The text that will be displayed for the menu item. Can include the &amp; character to define shortcut highlights.</td>
</tr>
<tr>
<td>Description</td>
<td>A short description of the menu item. This description will be displayed in the status bar when the mouse is moved over the menu item.</td>
</tr>
<tr>
<td>SubMenus</td>
<td>Lists the names of the subkeys that will make up the submenu. The order in which the keys are listed, they will appear in the menu. Separators are added by inserting a ' - ' character.</td>
</tr>
</tbody>
</table>

has the root parent node DoasUI\Menu defined. All subkeys of the root node define the major menu items. Subkeys in the main menus define submenus.

To define a parent menu item, only a few settings are required. In each parent key, the multi-string value SubMenus defines the order in which the subkeys will represent the menu items. Menu separators can be added here as well. The complete list of configuration values of a parent menu item is given in table B.1.

B.1.2 Standard Menu Item

A standard menu item defines several things. First, of course, it defines a menu item in the menu tree of DOASIS. The menu item can be decorated with icons or check marks as needed. Besides the look of the menu item, the event handler that is called when the item is selected must be defined. Table B.2 shows all settings available for a menu item.

Each menu item can have a shortcut key defined. The shortcut allows the user to select a menu item without the need to click through the menu structure. The menu item is selected by pressing the shortcut key combination.

Additionally an optional icon can be defined that will be displayed in front of the menu item. If the menu item is assigned to a toolbar as well, the same icon will appear in the toolbar as a shortcut.

Some methods need some user input to get started. DOASIS also offers a set of methods to do the parameter handling automatically for the developer so
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MenuText</td>
<td>The text that will be displayed for the menu item. Can include the &amp; character to define shortcut highlights.</td>
</tr>
<tr>
<td>Description</td>
<td>A short description of the menu item.</td>
</tr>
<tr>
<td>Type</td>
<td>The type name of the class that defines the event handler.</td>
</tr>
<tr>
<td>Method</td>
<td>The event handler method that will be called when the menu is selected. The method must have the signature of the EventHandler delegate.</td>
</tr>
<tr>
<td>Assembly</td>
<td>The assembly that holds the class type defined by Type. If this value is not set, the local assemblies are searched for the given class type.</td>
</tr>
<tr>
<td>Shortcut</td>
<td>The shortcut keys that can be used to allow the user a quick selection of the menu item. Example: &quot;CtrlC&quot;</td>
</tr>
<tr>
<td>ImageList</td>
<td>The image list that contains the icon to be displayed in the menu and in the toolbar.</td>
</tr>
<tr>
<td>ImageIndex</td>
<td>The index of the icon in the image list.</td>
</tr>
<tr>
<td>Toolbar</td>
<td>The name of the toolbar into which this menu item will be inserted with the defined icon.</td>
</tr>
<tr>
<td>AlwaysAskParameter</td>
<td>If set to True, DOASIS will always represent the user a parameter input dialog regardless if the menu item was called via the menu tree, a shortcut key or the toolbar.</td>
</tr>
<tr>
<td>ParameterID&lt;N&gt;</td>
<td>Defines the internal name of parameter &lt;N&gt;.</td>
</tr>
<tr>
<td>ParameterType&lt;N&gt;</td>
<td>Defines the type of parameter &lt;N&gt;. Can be any intrinsic type or Spectrum.</td>
</tr>
<tr>
<td>ParameterDescription&lt;N&gt;</td>
<td>A short description of this parameter.</td>
</tr>
<tr>
<td>ParameterMin&lt;N&gt;</td>
<td>In case of a numeric parameter this defines the minimum value that is valid.</td>
</tr>
<tr>
<td>ParameterMax&lt;N&gt;</td>
<td>In case of a numeric parameter this defines the maximum value that is valid.</td>
</tr>
<tr>
<td>ParameterTitle</td>
<td>Defines the title that the parameter input dialog will have.</td>
</tr>
<tr>
<td>OptionID</td>
<td>The parameter identifier, if the parameter is selected from a set of available options.</td>
</tr>
<tr>
<td>Options</td>
<td>A list of strings the define the possible values for the optional parameter.</td>
</tr>
</tbody>
</table>

*Table B.2: Configuration settings to define a menu item.*
Extend DOASIS

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filter</td>
<td>Defines the file type filter that will be used to decide whether a file will be processed by this file type or not. Additionally the filter definition will be added to the file type selection list in the file open/save dialogs.</td>
</tr>
<tr>
<td>Type</td>
<td>The type name of the class that represents the file type. DOASIS will create an instance of this class to read or write a file of this type. This class must implement the IFormatter interface.</td>
</tr>
<tr>
<td>Assembly</td>
<td>The assembly that holds the class type defined by Type. If this value is not set, the local assemblies are searched for the given class type.</td>
</tr>
</tbody>
</table>

Table B.3: Configuration settings to define a file type.

there is no need to create custom GUI components for each menu handler. The generic parameter query framework can manage parameters of numeric or string type as well as spectrum objects or optional selections. For each parameter to be queried, its ID and its type must be defined. A dialog will be created that requests the user to input data of the defined type. In case of a spectrum selection, the list of available spectra will be presented.

DOASIS will only request user input, if either the menu item was selected from the menu tree or if no default values for the parameters are present. If the menu item is selected through a shortcut or a toolbar, the default values will be used immediately without any further user interaction. This enables the user to set parameters once and repeat the same action over and over again without having to enter the same parameters more than once.

B.2 File Types

Although DOASIS already comes with a huge list of supported file types, users may need to add other import or export filters. DOASIS uses the serialization technique to read or write files. Any custom file type handler therefore must implement the IFormatter interface that defines the necessary methods to serialize or deserialize a data stream into a spectrum object.

A custom file type filter can be added by creating a new subkey in the application configuration under the key DoasCore\FileTypes. Each file
type filter is defined by the values given in table B.3. Special care needs to be taken by the definition of the file type filter. An example file type filter definition looks like this:

**Doasis Version 1 (*.spe)|*.spe**

The file type filter is made up of two sections which are separated by the | character. The first part defines the name and the text that will be displayed when ever a file type needs to be selected by the user. The second part defines the file name pattern that is used to match a file name against the list of available filters. The pattern can contain the well known wildcard characters ? and *. If the file name pattern matches a given file name, DOASIS will create an instance of the filter class and start the serialization process.

To include a new file type, the new filter needs to be added to the list of file types defined in the value `DoasCore\FileTypes\PriorityOrder`. In this list each line must contain the exact name of the subkey of a file type definition. The order in which the file type definitions appear defines the order in which the file name patterns will be matched. If multiple file type filters match a file name, the first match will be used.

### B.3 Spectrometers And Other Devices

DOASIS is capable to work with different kinds of spectrometers, imaging devices and other device types. In general DOASIS distinguishes two kinds of devices:

- **Spectrographs**
  These devices are used to record spectral data. They offer support to control the wavelength range that will be recorded and various other kinds of settings.

- **Support Devices**
  These devices offer general support to control motors, use AD converters or control analogue or digital output channels. These devices do not directly interact with the spectrograph, but are needed to, for example, control the temperature of the spectrograph or move the whole device to another viewing angle.
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The display name of the spectrograph.</td>
</tr>
<tr>
<td>Description</td>
<td>An optional short description of the spectrograph.</td>
</tr>
<tr>
<td>Type</td>
<td>The type name of the class that represents the spectrograph. DOASIS will create an instance of this class to access the spectrograph. This class must implement the DoasCore.Device.ISpectrograph interface.</td>
</tr>
<tr>
<td>Assembly</td>
<td>The assembly that holds the class type defined by Type. If this value is not set, the local assemblies are searched for the given class type.</td>
</tr>
</tbody>
</table>

Table B.4: Configuration values to define a new spectrograph device.

### B.3.1 Spectrographs

To make DOASIS aware of the available spectrograph modules, once again the configuration is inspected. This time DoasCore\Spectrograph is the configuration key to be used. For each spectrograph that will be available, a subkey exists. Each subkey must contain the values defined in table B.4.

When the user selects a specific spectrograph, DOASIS will create an instance of the given class to access the spectrograph. Also any reference of the type DoasCore.Device.Spectrograph will delegate to the selected device object. The user can therefore create JScripts that are either directly bound to a specific type of spectrograph or use the common class Spectrograph. The second method has the advantage that nothing needs to be changed in a custom script, if the spectrograph is changed.

Additionally the DOASIS UI will create a proper dialog to give the user access to all properties that the spectrograph object offers.

### B.3.2 Devices

The installed device types must be defined under the configuration key DoasCore\Devices. Table B.5 defines the necessary configuration values to specify a new device. Like for spectrographs each device definition is stored in a separate subkey. DOASIS will then automatically enumerate all subkeys to get the list of installed modules.

The UI creates a dialog for the currently selected device. Since these generic devices can offer various kinds of functionality, a set of interfaces is avail-
<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>The display name of the spectrograph.</td>
</tr>
<tr>
<td>Description</td>
<td>An optional short description of the spectrograph.</td>
</tr>
<tr>
<td>Type</td>
<td>The type name of the class that represents the device. DOASIS will create an instance of this class to access the device.</td>
</tr>
<tr>
<td>Assembly</td>
<td>The assembly that holds the class type defined by Type. If this value is not set, the local assemblies are searched for the given class type.</td>
</tr>
</tbody>
</table>

*Table B.5: Configuration values to define a new device.*

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IADC</td>
<td>Interface to make ADC input channels available to the user.</td>
</tr>
<tr>
<td>IFlags</td>
<td>Allows the user to control digital output signals.</td>
</tr>
<tr>
<td>ITemperature</td>
<td>Realizes functions to control temperature settings.</td>
</tr>
<tr>
<td>IMotor</td>
<td>Control a set of stepper or servo motors.</td>
</tr>
</tbody>
</table>

*Table B.6: Possible interfaces to define functional blocks of a device object.*

able to allow the UI to create special components for the different options available. The list of device interfaces is given in table B.6. Besides the interfaces for the functional blocks, each device class must be inherited from `DoasCore.Device.Device`. This base class is required to do basic condition tests on the device to detect disconnects.

## B.4 Adding User Extension Assemblies

The previous sections defined the different configuration settings necessary to setup new menu items, file types or devices. The default installation of DOASIS automatically configures DOASIS to use the already internally present extensions. The user can add his own extension with three little steps:

1. Create an assembly that contains the new functionality that should be added to DOASIS. Make sure to implement the necessary interfaces and expose all required properties.
2. Copy the assembly created to a location where DOASIS can find the assembly. Normally this is either the installation path of DOASIS (preferred location) or the default system folder.

3. Add the necessary values to the configuration.

If everything is configured correctly, the next time DOASIS starts, the extension will appear in the proper locations and can be used further.

B.5 COM/ActiveX Extensions

Since .NET offers a very flexible support to directly access COM/ActiveX objects, user created COM objects can be used in DOASIS as well. To access COM objects in user scripts, a simple command is required:

```
new ActiveXObject("ProgID")
```

This will create an instance of the COM/ActiveX object with the given ProgID. However, COM objects can only be valid menu handlers. They cannot be used as spectrograph device or file type handler due to the lack of the necessary inheritance of .NET interfaces.
C File Formats

DOASIS has support to read and write several file formats. This is necessary to both interface with other applications like Windows or MFC and to import existing data sets that have been recorded with various other kinds of tools.

Due to the extensible nature of DOASIS input and output filter can be added to accommodate the needs of each user. Thus this chapter cannot cover all possible in- and output formats and will only describe the most common file formats.

C.1 ASCII Encoding

There is a lot of confusion caused by the different encoding formats used in ASCII based files. Unfortunately there is no standard that defines how numbers and dates are encoded in an ASCII file. To avoid these problems DOASIS always uses the English encoding for numbers and dates, which are defined by:

- **Numbers**
  The decimal separator will be the dot (‘.’).

- **Dates**
  The date format is defined by MM/DD/YYYY where MM is the month, DD the day of the month and YYYY the year. The year may be defined only using two digits.

The system’s locale settings are not taken into account when exporting ASCII data. Files imported into DOASIS that do not have the correct encoding may produce strange results.

C.2 DOASIS Version 2 Format (.sp2)

The DOASIS Version 2 Format is a binary representation of a Spectrum-object. This binary representation is defined by the BinaryFormatter-class of the .NET framework. In detail this binary representation is a serialization of the complete object tree that defines a spectrum. In the data stream first
the object tree is stored by using reference numbers for each object or value present in the tree. The tree definition also includes the type specifiers for each element.

After the tree definition the content of each object and value follows.


However, this file format is only suitable in an object oriented environment. Although the stream can be deserialized manually, it primarily targets on .NET applications.

C.3 XML SOAP Format (.xml)

The SOAP format is similar to the binary serialization described in the previous section. The SOAP format is a self-describing XML file that contains a complete object tree of a Spectrum object. Like in the binary format first the object tree is defined followed by the content of each contained object.

A detailed description of the SOAP format is given by [Box et al., 2000].

C.4 MFC Format

The MFC file format is used by the MFC application [Gomer et al., 1995]. The MFC file format can be separated into two parts. The first part is a binary dump of a spectrum structure of the MFC application. The header is defined as follows:

```c
#pragma pack(push, MFCHeader, 1)
typedef struct
{
    char szVersion[20];       // version
    int iNChannel;             // number of channels
    void* pData;
    char szSpecName[20];      // spectrum name
    char szSite[20];          // site location
    char szSpectrometer[20];  // spectrometer name
```
char szDevice[20];  // device name
char szFirstLine[80];
char szLater[18];
int iSpecInBlock;
int iNumSpecOfBlock;
double fDispersionHighDensity[3];  // wavelength calibration
double fWavelength;  // wavel. of chan. N-1
float fAverage;  // intensity average
short iODLeft;  // OD marker left
short iODCenter;  // OD marker center
short iODRight;  // OD marker right
float fOptDens;  // OD
float fOptWavelength;  // OD wavelength
int iType;
char szDate[9];  // start date
char szStartTime[9];  // start time
char szStopTime[9];  // stop time
char cFillDummy1;
int iMathLow;  // math low
int iMathHigh;  // math high
int iMinChannel;  // zoom low
int iMaxChannel;  // zoom high
int iMarker;  // marker
int iNoScans;  // number of scans
float fExposureTime;  // exposure time
float fLatitude;  // site’s latitude
float fLongitude;  // site’s longitude
int iNoPeaks;
int iNoBands;
float fMin;  // intensity minimum
float fMax;  // intensity maximum
float fYScale;
float fOffsetScale;
float fWavelength1;
float fAverageOld;
float fDispersion[3];  // wavelength calibration
float fOptDen;  // OD
int iMode;
int iSmooth;
After the header the spectrum’s intensity data is available. For each channel defined by the number of channels in the header, a 32 bit float values is appended after the header.

### C.5 Extended Standard Format (.std)

The *Extended Standard Format* is based on the *Standard Format* ASCII-based file format introduced by MFC [Gomer et al., 1995]. For interoperability issues an ASCII encoding was chosen to avoid byte swapping problems between different system types.

This file format has been extended in a way to accommodate the needs of the new spectrum data that needs to be stored. Compatibility with older applications is ensured by providing the same header as the *Standard Format* used to. This default header is defined by these values:

```
GDBGMNUP       // magic key
VERSION        // version
1024           // the number of channels
CHANNEL0       // the intensity values. each value on
......          // a single line up to the
......          // number of channels defined
CHANNEL_N-1    // spectrum name
NAME           // spectrum name
```
After this standard file compatible header, any properties of a spectrum object that have not been defined yet are appended in an ASCII representation. Each property is defined by its name followed by an equal sign. Next to the equal sign optionally the type of the data can be defined enclosed in round brackets. Array types use the additional array definition using square brackets.

Intrinsic data types like integer or float values are not prefixed by their type definition. This reduces the file size and such values can easily be converted to the required target types.

Here are a few examples of such extended property definitions:

```
OpticalDensity = 0.837559474420517
OpticalDensityCenter = 1024
OpticalDensityLeft = 102
OpticalDensityRight = 1945
Pressure = 0
Remark = ""
CalibPolynomial = (System.Double[3])298.729595247891 0.12 0.0
```

The order of these extended properties is not important.
C.6 ASCII and Cross-Section Format (.txt, .csv, .xs)

All ASCII based formats only contain the intensity values of a spectrum. Optionally the wavelength calibration data can be included. No additional information is stored in such kinds of files.

These ASCII files are made up of rows. Each row represents a single intensity value. Several values can be stored in a single row by separating them either by a tabulator or a space character. Each of the values within a row make up a column. Thus the number of lines/rows defines the number of channels of the spectrum. The number of column defines whether only intensity data is present, a wavelength calibration is included or even a two-dimensional spectrum is given. The decision what kind of data is available is made by these three simple rules:

- **One column**: Only the intensity values of a single spectrum is present.
- **Two columns**: The intensity and wavelength values of a spectrum are available. The first column defines the wavelength calibration. The second one the intensity values.
- **Three or more columns**: A two dimensional spectrum is given. The first columns contains the wavelength calibration. Starting at the second column each further column contains the spectra data of a row within the two-dimensional spectrum.

C.7 GOME Level 1 Files

DOASIS also has support to directly import GOME Level 1 file as a two-dimensional spectrum. Each data row in the spectrum represents the spectra data of one pixel contained in the orbit. During importing the GOME Level 1 data, the desired spectral range and the kind of data to be read needs to be defined.

The level 1 files are the preprocessed spectra data provided by the ESA. The algorithms applied to generate the level 1 data are explained in [ESA, 1995b] and [ESA, 1995c].
D JScript .NET Quick Guide

This chapter will give a brief overview about the JScript language and show how to write JScript scripts that can be run inside DOASIS. In depth description and the JScript language can be found here [Microsoft Corporation, 2000b] and here [Microsoft Corporation, 2000a]. Additionally the complete .NET framework documentation is given here [Microsoft Corporation, 2000c].

A JScript is a simple text file that contains a sequence of commands that should be processed. The commands will be executed in the same sequence as they are defined in the script. Additionally using control structures (D.5) and functions (D.6) the processing order can be changed while the script is executed.

D.1 Imports And Namespaces

The first part of each script is defined by the import-statements. These import statements are used to define which assemblies will be used by this script. At least the System-assembly should always be included, since this is the central library that contains all .NET classes.

Defining only a import-statement for the assembly name, ensures that the assembly is available in the script. However, to access any of the classes contained in the assembly, the full namespace hierarchy has to be defined:

```javascript
DoasCore.Math.SpecMath.Add(spec1, spec2);
```

For convenience subnamespaces can be completely included to avoid the need of a full namespace declaration. If a subnamespace is imported, any class in it can be directly used.

```javascript
// import the math subnamespace
import DoasCore.Math;
SpecMath.Add(spec1, spec2);
```

In cases where including different namespaces that contain classes with the same names, the compiler will complain about ambiguities that cannot be resolved. In such a case the user needs to define the full namespace declaration of the class regardless whether the namespace was included or not.
Within a script it is often required to store some data and read, write or modify it. To identify the stored data an identifier is required. Using this identifier the stored data can be accessed. Such identifiers are called variables. To define a variable the keyword `var` is used.

```javascript
// generic variable declaration
var <identifier> [= <initial value>];
```  

```javascript
// define the empty variable "test"
var test;
```  

```javascript
// define the variable "test2" and assign the integer value 2 to it
var test2 = 2;
```  

In JScript a variable can hold any kind of data. If different kinds of data are involved in an operation, the JScript compiler tries to determine which data type the result should have. Here are a few examples:

```javascript
// create an integer variable
var val1 = 1;
```  

```javascript
// create a floating point variable
var val2 = 3.14;
```  

```javascript
// create a string variable
var val3 = "54";
```  

```javascript
// add the integer and the floating point value.
// the result will be a floating point value because
// an integer could not correctly hold the result
var res1 = val1 + val2; // = 4.14
```  

```javascript
// add the float value to the string. The result will be a string, since
// the first operand is the string that cannot be converted to a number
// and any following operand will be appended
var res2 = "" + val3 + val2; // = 549.14
```  

```javascript
// if you do the previous operation with exchanged operands, the result
// will be a floating point value. Because the first operand is a float
// and the string can be converted to a number as well
var res3 = val2 + val3; // = 57.14
```  

When several operands are involved in an operation, the first operand is always the dominant type and defines what the resulting data type will be. However, the compiler only distinguishes between numeric types and non-numeric types. If different numeric types are involved, the result will be of that numeric data type that has the highest precision to hold the result.

### D.2.1 Typed Variables

To avoid ambiguities and to clearly define the data types to be used, a typed variable can be defined. Unlike the untyped variable, the user explicitly defines what kind of data will be stored in this variable. Any attempt to store some other data in it will fail with an error.
To define the type of a variable, the type is defined after the variable identifier.

```javascript
// general declaration of a typed variable
var <identifier> : <type> [= <initial value>];

// define an integer variable
var test : int = 5;

// define a float variable
var test2 : double = 5;
```

Sometimes it is required to define a typed variable. Since the JScript compiler tries to automatically convert a variable to a suitable type that fits the needs of the current operation, ambiguities can happen. For example, an operation requires either an integer, a float or a string as a parameter and should be called with a variable that contains a number. Now the compiler cannot decide which data type to used because the number-type can be converted into any of the possible types. To resolve this problem, the type of a variable must be defined and the compiler can choose the correct method call.

**Data Types**

Although it is not required to define the type of a variable, every JScript user should be aware of the available data types. The JScript engine determines the data type required to store a value automatically depending on how the value is defined and what operation will be done. Table D.1 shows how the different data types are distinguished.

**D.3 Operators**

In a script it is not only necessary to store values in variables, but also the content of variables may need to be modified or combined in certain ways. In particular the most common operations are the arithmetic operations of numerical values and to do conditional tests against variables. Table D.2 shows most common operators available in JScript.

**D.4 Blocks**

Although a script can simply be a sequential series of commands, for certain operations it is necessary and useful to group some commands into an
### Data Type Description

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>char</strong></td>
<td>A single character enclosed in single quotation marks. Example: 'f'</td>
</tr>
<tr>
<td><strong>string</strong></td>
<td>A sequence of text enclosed in double quotation marks (&quot;). Example: &quot;Hello&quot;</td>
</tr>
<tr>
<td><strong>int</strong></td>
<td>A 32-bit integer value. Example: −15</td>
</tr>
<tr>
<td><strong>bool</strong></td>
<td>A boolean value that can either be true or false. Example: false</td>
</tr>
<tr>
<td><strong>double</strong></td>
<td>A 64-bit floating point value. Example: 3.14152965</td>
</tr>
</tbody>
</table>

**Number**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Number</strong></td>
<td>Any kind of numerical value. This kind of data ensure to automatically choose the highest possible accuracy required to store the actual data. Heavy usage of this type will slow down execution because this data type is realized in software and cannot be handled by current processors.</td>
</tr>
</tbody>
</table>

**Date**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Date</strong></td>
<td>An object representing a date and time definition. Example: &quot;10/08/2003 15:45&quot;</td>
</tr>
</tbody>
</table>

**Array**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array</strong></td>
<td>An array is a list of elements. An array has a fixed length and can store any kind of data. For dynamic expanding lists and maps have a look at the System.Collections namespace of the .NET framework.</td>
</tr>
</tbody>
</table>

**object**

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>object</strong></td>
<td>Any kind of data used in a JScript is based on the object data type (except the intrinsic ones). Thus the object type is a placeholder for any kind of user defined objects.</td>
</tr>
</tbody>
</table>

**Table D.1**: Available intrinsic data types in JScript.

---

execution block. Such an execution block is enclosed in curly brackets `{}`. For example, an if-statement will skip the next command that follows the if-statement depending on the condition. If multiple commands should be executed depending on the same condition, they can be grouped into an execution block.

Example:

```csharp
// check, if the value is larger than 3
if (x > 3)
{
    // all commands in this block will be run when x > 3 is true.
    Console.WriteLine("Value is larger than 3");
    x = 2;
}
```
Operator Description

+, -, *, / Defines an arithmetic operation of the two operants. Additionally the +-operator can be used to concatenate string.
Example: 3 + 4

!, &, &&, | | Defines the boolean operations "not", "and" and "or". The result will always be a boolean value.
Example: (x > 3) & & (x < 10)

!, ^, &, | | Bit operators that do a "negate", "xor", "and" or "or" operation.
Example: x & 3

==, !=, <, > Comparison operators that do an "equals", "not equals", "smaller" and "greater" operation.
Example: 5 != 4

Table D.2: Available operators in JScript.

D.5 Control Structures

JScript provides several kinds of control structures to allow either the execution of some commands depending on a certain condition or to do loops for repeating operations.

D.5.1 if-else-Statement

This statement is used for conditional execution of either the if- or the else-block. If the condition is true, the if-block gets executed. Of the condition fails, the optional else-block will be run.

Example:

```javascript
// get user input
var input = Console.ReadLine();

// print some response depending on the input
if (input == "hello")
    Console.WriteLine("Hello world!");
else
    Console.WriteLine("What do you want?");
```
D.5.2 Loops

Repeating a single operation or a complete execution block is often required especially to record or evaluate time series. JScript offers three types of loop operations.

**for-Loop**

A for-loop is used when the number of repetitions is a fixed number. The exact definition looks like this:

```javascript
for(init; condition; loop-operation)
{
  execution block
}
```

The loop gets processed in this order:

1. Run the init-command.
2. Repeat the execution block until the condition is false. After each time the execution block was run, process the loop-operation-command.

This sounds complicated, but makes life easy. The standard use-case for a for loop is to repeat an operation x-times. So first initialize a counter variable. Define a condition that meets the number of times the operation should be repeated. Increment the counter each time by one.

Example

```javascript
var i;
// repeat 3 times
for(i = 0; i < 3; i++)
{
  Console.WriteLine(i);
}
```

**while- and do-while-Loops**

while-loops are used when the number of iterations is not exactly known or when a loop should be prematurely aborted in case of some asynchronous operation. The general workflow is that the execution block attached to a while- or do-while-statement will be repeated until the condition will be false.

Example:

```javascript
// repeat the following loop until the user pressed the Stop button
while(!Scripts.StopAllScripts)
{
  Console.WriteLine("Script still running");
}
```
The difference between the while- and the do-while-loop is the minimum number of times the loop is done. A while-loop first checks the condition and then runs the block. So if the condition is false from the beginning, the block will never get executed. A do-while-loop first checks the condition after the execution block was processed, thus, even if the condition is false, the loop will run at least once.

D.6 Functions

Reoccurring code sequences that use some input data to produce a certain result or to do a certain command sequence should not be copied multiple times into a script. Having several copies of the same code is catastrophic for maintenance and error analysis, because any change to the command sequence requires to update all occurrences of it in a script. Also the readability is highly affected and it is awful to go through some code that always has the same sequences pasted in.

Therefore such common command sequences can be grouped into a function. When a function is defined, the command sequence only has to be defined once and everytime it is required somewhere in the script, it just can be referenced by its function name. Optionally a function can expect multiple parameters and can return a result, thus, the function can change its behavior depending on the input values and return the result of its operation.

```csharp
// generic declaration of a function
function <funcname>(({param1, ...})
{
    function body

// define a function that always writes the text
// "Hello world!" on the screen
function hello()
{
    Console.WriteLine("Hello world!");
}

// define a function that determines the average of two values
function average(value1, value2)
{
    return (value1 + value2) / 2;
}

// execute the hello-function
hello();

// determine the average of two numbers
```
It is important to know that the code that defines a function will not get executed when the function is declared. Only calls to the function will instruct the JScript compiler to switch to the function’s body and execute the commands listed there. If a function is just declared but never called, it will never be executed!

D.7 Accessing DOASIS Functions

Any method or object DOASIS exposes is located in the DoasCore namespace. Thus, to access them always the DoasCore namespace must be included.

Now any object or method can be directly accessed or use additional import statements to get shortcuts to their namespaces. Since .NET is an object oriented environment, all methods and properties are bound to an object.

An object is a collection of properties that describe the state of it and a set of methods that can read, write or modify an object’s content in some way. To access an object’s properties or methods, you always need an instance of a class. The class is like a building plan for an object that describes what data is stored and what the different methods will do. To create an object the new operator is required:

```javascript
// create an instance of the DoasFit class
var fit = new DoasCore.Math.DoasFit();
```

This particular instance of the class DoasFit, which is the object we want, is stored in the variable fit. Now it is possible to access the properties and methods of this object using the following syntax:

```javascript
// set the max fit steps property
fit.MaxFitSteps = 100;

// call the DoFit method
fit.DoFit(mySpec);
```

There is one exception for this procedure: static-methods and -properties. Static methods are identified by the keyword static that is included in the property or method definition. Static members of a class do not require

```javascript
var param1 = 3;
var param2 = 3.14;
var avg2 = average(param1, param2);
```
an object to be created. They can be directly accessed through the class’s name:

```csharp
// import the necessary namespaces
import Dosacore.Math;
import Dosacore.Scripts;

// check whether the user pressed the Stop button
if(Scripts.StopAllScripts)
    Console.WriteLine("User pressed stop button!");

// do a convolution
SpecMath.Convolute(mySpec, kernel);
```

## D.8 Error Handling

Several errors can cause a script to abort. Critical errors are reported as *exceptions*. An *exception* is a special state that causes the current script to immediately abort. Without any special exception handling the script just stops and the error message including some information about where that error occurred will be shown to the user.

However, when a script gets aborted in such a way, for example, files that have been opened or other system resources will not get closed automatically. It is the user’s responsibility to ensure any resources that are allocated outside the .NET environment are released in case of an error. The JScript language allows to catch exception and to execute additional code.

Exceptions are caught using a *try-catch*-block. A *try-catch*-block defines a sequence of commands that potentially cause an exception. If no error occurs, nothing happens except that the commands in the *try*-block get executed. But if an error occurs, the *try*-block gets immediately aborted and the *catch*-block gets executed.

```csharp
// place the required variables outside the try-block
var myFile = null;

// start the try block, since one of the next statements may fail
try
{
    // open the file for reading
    myFile = System.IO.File.OpenText("test.txt");

    // read the first line of text and print it
    Console.WriteLine(myFile.ReadLine());

    // we do NOT close the file here, since an error
    // while reading the first line may cause this block to
    // abort prematurely!!!
}

catch(e)
{
    // this block gets executed when an exception occurs.
    // simply display the error to the user and rethrow
    // the exception to abort the whole script
    Console.WriteLine("Exception: " + e.Message);
    throw e;
}
```
An optional `finally`-block will always be executed regardless whether an exception occurred or not. Information about an exception is only available in the `catch`-block where the parameter will hold the exception object that describes what happened.

### D.9 Example

The following example demonstrates how to evaluate a series of already recorded spectra. At the beginning of the script the required namespaces are imported, so the necessary assemblies get loaded and we do not have to use the full namespace declaration when accessing objects.

The first object we create is a `AutoFileName` object which is used to load a set of spectrum files one after another. After an instance of this object has been created, we have to define the necessary properties.

Next the fit scenario is opened. The user already set up the required fit parameters like fit range and reference spectra and stored it into a scenario file. In an evaluation script such a scenario can simply be loaded to make all cross-section spectra available.

Now a loop starts to open one spectrum file after another from the list of available files defined in the `AutoFileName` object. Additionally the loop condition checks if the script should stop due to a user request.

Inside the `while`-loop the spectrum gets analyzed using the predefined fit scenario. If the result of the fit is valid, the result is written into a text based result file that can be imported into other applications for further processing.
import System;
// use the DoasCore library
import DoasCore;
// use the Device namespace
import DoasCore.Device;
// use the Input-/Output namespace
import DoasCore.IO;
// use the Script namespace
import DoasCore.Script;
// use the Spectra namespace
import DoasCore.Spectra;

var afnFile = new AutoFileName();

// define the base path where the files can be found
afnFile.BasePath = "C:\\scanTest";

// define the prefix of the file names
afnFile.Prefix = "a";

// define the extension of the file names
afnFile.Suffix = ".std";

// the files are grouped into subfolders of 100 files each
afnFile.FoldersPerFolder = 100;

// the file name contains 5 digits
afnFile.NumberOfDigits = 5;

// look for the first existing file
afnFile.FindFirstIndex();

// create the spectrum object that will receive the loaded data
var specEval = Specbar.GetSpectrum("Eval");

// open the fit scenario
// in this case we need to define the complete namespace
// for hierarchy, since another Math namespace exists in the System-library, too!
var fit = DoasCore.Math.DoasFit.Open("c:\\scanTest\\EvaluateExample.fs");

// repeat until the stop button was pressed and a file could be loaded
while(!Script.StopAllScripts & afnFile.Open(specEval))
{
    // run the fit
    if(fit.DoFit(specEval))
    {
        Console.WriteLine("Fit successful for spectrum " + specEval.FileName);
        fit.AppendResultToFile("C:\\scanTest\\EvaluateResult.txt");
    }
    else
    {
        Console.WriteLine("Fit failed for spectrum " + specEval.FileName);
    }
}

Console.WriteLine("Evaluate stopped.");

This example could be extended in several ways. For example, the results of a certain or all species could be made directly visible to the user. Simply create a new Spectrum-object that has the drawing style set to plot dots instead of lines and put the desired values into such an object.

Also the fit scenario object does not have to be fixed during a series evaluation. It would be possible to add, remove or change one or more of the reference spectra depending on the currently loaded measurement spectrum. Simply the appropriate reference spectrum in the ReferencesInfo-property has to be exchanged with a new one.
E DOASIS Documentation

Not only the design of a framework and its applications is important. Also it is essential to document all interfaces and define what’s going on inside a function. Of course, due to the huge set of already present methods and objects, including a complete documentation of them would exceed the extends of this work. Especially when developing either JScripts or DOASIS extensions, a useful online reference documentation is the preferred way to look for additional information.

Taking this into account, a online documentation was created that contains additional background information, quick guides and a complete reference description. This online documentation gets automatically installed with each DOASIS installation and can be accessed through the Help menu.

The documentation also contains information about JScript, how to work with project files and how JScripts can be debugged. Thus this online documentation should always be the first place to look for information what is available and how to access it.

Like other components, DOASIS gets shipped with an IntelliSense compatible description of its methods, objects and parameters that allows suitable source editors like the Visual Studio Editor or the SharpDevelop Editor to show in-place context sensitive information.

E.1 Getting Started

The next sections give an overview about where in the complete DOASIS Programmer Documentation more information about a specific topic can be found. Most of the details can be found in the Symbol Reference section of the online documentation which lists all Objects and methods DOASIS exports.

E.1.1 Mathematical Operations

All mathematical operations are located in the DoasCore.Math namespace of the Doasis core library. There you’ll find special classes that implement different kind of functionality like:
• SpecMath: basic mathematical methods
• DoasFit: fitting methods
• Scattering: Ring and Raman spectrum calculation
• ScanGeometry: SZA support

E.1.2 Hardware Support

The DoasCore.Device namespace contains all classes for each of the supported devices. Using the generic class Spectrograph allows to scan without the need to know the exact hardware used.

E.1.3 Spectrum File Support

Using the DoasCore.IO.SpectrumFile class in the DoasCore.IO namespace allows you to open and save a spectrum. For a set of spectra the DoasCore.IO.AutoFileName class should fit your needs. Also the Spectrum class contains methods to open and save the spectrum directly without the need to use the DoasCore.IO.SpectrumFile class.

E.1.4 Spectrum Management

The DoasCore.Spectra namespace contains everything to manage a set of spectra. The following can be found there:

• Spectrum: Basic object that represents a spectrum within DOASIS
• Specbar: A collection of spectra that will be visible in the Specbar of the graphical user interface

E.1.5 User Interaction DOASIS

Doasis offers a namespace called DoasCore.HMI (Human Machine Interface). The following features are available there:

• UserInterfaceWriter: Basic text output to the user
• UserInterfaceReader: basic text input from the user
• ProgressBar: Display the progress within your algorithm
Additional DOASIS redirects any in- or output operations to the default `System.Console` object to the currently active UI.

### E.1.6 User Interaction .NET

The .NET framework offers several methods to display messages to the user:

- **`System.Console`**: Use this class to write or read text messages from the user. In Doasis these messages will be shown in the Output window.

- **`System.Windows.Forms`**: This namespace contains everything to generate and display any kind of windows to the user e.g. message boxes.

### E.1.7 Basic File Support

The `System.IO` namespace of the .NET framework offers everything to work with files. Either text or binary files.
F Linux Support

Today the primary platform to run .NET applications on is Microsoft Windows. However, due to the nature of the .NET concept, it would be desirable to get Linux support as well. Two projects try to achieve this goal:

- **MONO**
  The MONO projects is sponsored by Novell. Besides the goal to create a portable implementation of the core .NET class for Linux, MONO also offers compilers and further development tools as well as its own extension like the Gtk# package to ease creation of portable X-Windows applications [Novell, 2002].

- **DotGNU**
  This project is similar to MONO, but focuses more on the web based services of the .NET framework [Free Software Foundation, 2003].

Although both projects achieved very good results so far in porting the .NET framework to Linux, DOASIS cannot simply be started on this kind of frameworks. What are the road blocks?

- **The GUI**
  Both projects have not fully completed the implementation on the Win-Form package. Basic forms and dialogs will already work, but other fancy UI components like the window docking manager currently still refuses to run under MONO or DotGNU. A higher compatibility level is tried to be reached by using the Windows emulator WINE to simply delegate WinForm calls directly to their native Windows based implementation. Also MONO has plans to reach full compatibility of the System.Windows.Forms-namespace in version 1.2.

- **JScript**
  The MONO project has started working on JScript support. A command line JScript compiler is already available. However, a code provider to compile JScripts at runtime is still missing.

- **COM/ActiveX Extensions**
  Several legacy objects in DOASIS are still present as native COM objects. The reason for not having all modules available as .NET assemblies only is for compatibility reasons. Users tend to refuse to rewrite existing code.
• Native System Access

Some modules in DOASIS require direct access to native system methods. One of these modules, for example, is the USB and serial communication library. Since .NET does not offer support for direct hardware access, these modules must directly use the control functions of the underlying operating system. These method calls will never work on a Linux system. Separate native libraries for the different kinds of operating systems are required.

Besides the problems above, it is possible to run JScripts on a Linux system. Of course, only scripts that make use of the algorithmic functionality of DOASIS will work. As mentioned above, a code provider to compile JScripts at runtime is not available in MONO yet. But there is other ways to run JScripts. Both, the MONO framework as well as the .NET framework offer a JScript compiler that compiles a JScript into a native .NET assembly. Such a compiled JScript directly can be run in the .NET environment without the need of one of the DOASIS applications. Only the DOASIS core assembly DoasCore and its dependent assemblies are required.

But first of all a few oddities have to be avoided. The JScript support in MONO is not quite as flexible as the .NET implementation yet. Problems can occur, if untyped variables are used as object containers. Hence, all variables that will not contain an intrinsic data type should be declared as a typed variable. Besides getting MONO compatible JScripts, this also improves type checking during compile time and avoids runtime errors due to type mismatches. How to define a typed variable can be found in appendix D.2.1.

The next step is to compile a JScript at design time into a .NET assembly to avoid the need of a runtime JScript code provider. Using the .NET framework, a JScript can be compiled into an assembly using the following command:

\texttt{jsc \textless JScript\textgreater}

The \texttt{jsc}-JScript compiler can be found in the .NET installation directory. Each .NET version is shipped with its own JScript compiler. The result of the compilation will be a standalone .NET application. Finally, to run a script on Linux, install the MONO framework, copy the JScript assembly,
the DoasCore assembly and all other assemblies, configuration files and properties files to the Linux system. To start the script enter:

```mono <JScript Assembly.exe>
```

This will start the compiled JScript within the MONO framework. Figure F.1 shows the output of the evaluation script used to calculate the results shown in section 7 running on a RedHat Fedora Core 4 system using the MONO 1.1.13 framework.

![Screenshot of a DOASIS 3.2.2209 evaluation script running on a Fedora Core 4 system using the MONO 1.1.13 framework.](image)

**Figure F.1:** Screenshot of a DOASIS 3.2.2209 evaluation script running on a Fedora Core 4 system using the MONO 1.1.13 framework.
Linux Support
Bibliography


ESA. *GOME level 0 to 1 algorithms description*. DLR, 1995b. ER-TN-DLR-GO-0022.
ESA. *GOME level 1 to 2 algorithms description*. DLR, 1995c. ER-TN-DLR-GO-0025.


T. Wagner, F. Erle, L. Marquard, C. Otten, K. Pfeilsticker, T. Senne, J. Stutz, and U. Platt. Cloudy sky photon path lengths as derived from DOAS obser-


