Karl-Franzens Universität Graz
Institut für Physik

Projekt für das Institut für Weltraumforschung der Österreichischen Akademie der Wissenschaften
(Kontaktperson Jörg Weingrill)

Software PlugIn für das Programm `AstroArt 4.0`

Markus Hopfer
Andreas Windisch

Graz, am 27. Juli 2008

Inhaltsverzeichnis

1 Projektbeschreibung

1.1 Die Aufgabenstellung

Für die Software `AstroArt 4.0`*[1]* ist ein Plugin zu programmieren, vorzugsweise in Visual Basic, mit dem man die Bilder des SBIG Spektrographen*[2]* auswerten kann. Die Kamera des Spektrographen erzeugt dabei ein Bild mit 1600× 1200 Pixeln mit 16 bit. Das Spektrum wird über etwa 1000 Spalten aufgenommen. Je nach Objekt ist das Spektrum zwischen 10 und 100 Zeilen hoch.

Das Plugin sollte folgende Funktionen beherrschen:

- Mitteln mehrerer Zeilen um das Signal/Rausch Verhältnis zu verbessern
- Ausgabe als Liniengraph (Wellenlänge vs. Intensität)
- Einfache Kalibrierung durch Zuordnung einer Spalte einer bestimmten Wellenlänge

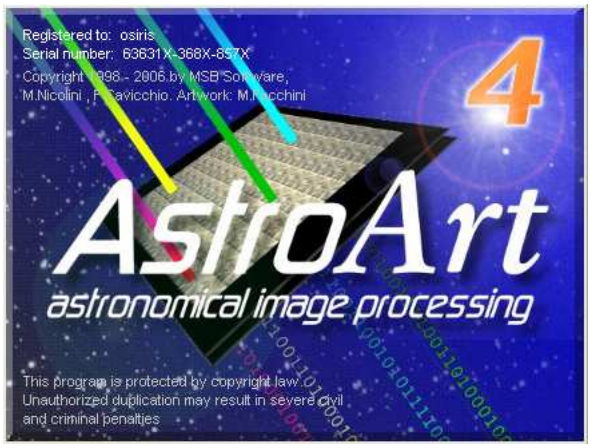
optional:

- Definition des Hintergrundes der dann vom Spektrum subtrahiert wird
- Export in eine Textdatei
- Absolute Kalibrierung anhand einer Planck-Kurve oder eines theoretischen Sternspektrums.

Das Plugin soll in Anlehnung an das Programm VSpec*[3]* entstehen. Es kann auch eine eigenständige Software geschrieben werden, allerdings vereinfacht das Plugin den Zugriff auf die Bilddateien im FITS-Format.
Die Software `AstroArt` sowie Testbilder werden zur Verfügung gestellt, außerdem wird Unterstützung für die Programmierung geboten.

1.2 Das Programm `AstroArt 4.0`

`AstroArt 4.0` ist ein Softwarepaket welches Routinen für Bildbearbeitung, Photometrie, Astrometrie, CCD Steuerung, etc anbietet.



Von `AstroArt` wird dem ambitionierten Softwareentwickler/Programmierer ein SDK (software development kit) zur Verfügung gestellt, welches es einem erlaubt, auf Funktionen des laufenden Programmes zuzugreifen. Hier soll im Folgenden ein kurzer Überblick darüber gegeben werden, ohne jedoch auf allfällige Details in zu hohem Maße einzugehen.
(Der Text wurde dabei aus dem den SDK beiliegenden html-File entnommen und übersetzt)

1.2.1 Zur Einführung

Ein PlugIn ist ein externes Modul (DLL) welches eine Anwendung erweitert. Für eine Software wie `AstroArt` sind PlugIns üblicherweise Filter, CCD-Control Elemente, Dateimanagement oder astronomische Berechnungen.

1.2.2 Anforderungen

Da es sich bei `AstroArt` PlugIns um Standard-Dlls handelt, kann für die Entwicklung derselben im Prinzip jeder Compiler herangezogen werden. Das SDK stellt einige grundlegende Beispiele zur Verfügung, welche in C bzw. Pascal verfasst, und mit Borland Compilern erfolgreich gebaut wurden. Um PlugIns mit Visual Basic zu erzeugen ist eine etwas unterschiedliche Vorgehensweise nötig, auf die wir später noch eingehen wollen.

1.2.3 Konvention

Man kann sich schnell einen Überblick über das Funktionsprinzip der PlugIn-Programmierung verschaffen, indem man einfach eines der Beispielprogramme kompiliert und wie folgt in das Programm einbaut: Es ist nicht notwendig die erzeugten PlugIns zu "installieren". Vielmehr wird jede DLL welche sich im `AstroArt`-Verzeichnis befindet beim Programmstart überprüft, und, wenn die Konvention `PI*.DLL, pi*.dll` (mit `*` als Platzhalter) erfüllt ist, installiert. Beispiele für gültige Dateinamen wären also etwa:
PISPECTRUM.DLL, pispectrum.dll, pifft.dll, etc.
Im Wesentlichen bedeutet dies, dass der Programmierer seine Software lediglich in das `AstroArt`-Verzeichnis kopieren muß. Ein automatischer Installations-Wizard kann zu diesem Zwecke geschrieben werden.

1.2.4 Exportierte Funktionen

Eine DLL (dynamic link library) ist eine Sammlung von Funktionen die an Applikationen exportiert werden, die diese DLL verwenden. `AstroArt` verlangt drei Funktionen von einem PlugIn:

- `pi_initialize`
- `pi_activate`
- `pi_finalize`

Ein Beispiel für eine Deklaration in C könnte etwa so aussehen:

```
int WINAPI pi_initialize(void *funcaddress, char *menuname, char *description, HWND
whandle)
void WINAPI pi_activate(void *pimage, HWND whandle)
void WINAPI pi_finalize(HWND whandle)
```

Die Funktionen im Detail:
`pi_initialize`:
Diese Funktion wird aufgerufen wenn `AstroArt` gestartet wird und nach PlugIns sucht. `menuname` und `description` sind Pointer, die auf nullterminierte Strings weisen, welche dann im Menü angezeigt werden. Dabei ist `menuname` der im Menü angezeigte Name, während `description` ein "hint" ist, der in der Statusbar von `AstroArt` angezeigt wird.

In dieser Abbildung sehen wir das Hauptmenü von `AstroArt`. Wir haben unser PlugIn "Spectrum" genannt, man sieht also im Menüpunkt "Plugin" unseren Eintrag.
`whandle` ist die Windowhandle des aktiven `AstroArt`-Fensters und kann ignoriert werden. `funcaddress` ist ein Pointer auf eine spezielle `AstroArt`-Funktion.

`pi_activate`:
Diese Funktion wird aufgerufen wenn der User das PugIn über das Menü aufruft. Dabei ist `pimage` der Pointer auf das momentan aktive Image. `whandle` ist die Windowhandle des aktiven MDI-Child Fensters.

`pi_finalize`:
Diese Funktion wird aufgerufen wenn `AstroArt` geschlossen wird. Sie kann verwendet werden um allokierten Speicher wieder freizugeben.

1.2.5 Typen und Befehle

Es gibt zwei spezielle Typen die ein PlugIn verwenden muss, wenn es mit `AstroArt` kommunizieren soll: pointer to image und pointer to callback. Beide werden durch eine 32-bit Variable repräsentiert, die wir `pimage` bzw. `pcallback` nennen.
`pimage` ist ein Pointer auf eine spezielle Struktur die ein Image repräsentiert. Das PlugIn benötigt keine weiteren Informationen darüber wie die Daten organisiert sind.
`pcallback` ist ein Pointer auf eine spezielle Funktion in `AstroArt` die vom PlugIn aufgerufen werden kann, um die ihm übertragene Aufgabe auszuführen. Dies nennt man eine "call-back"-Funktion, da sie vom PlugIn aufgerufen wird, während andere Funktionen (wie etwa `pi_activate` von `AstroArt` aufgerufen werden.
WICHTIG: Wenn das PlugIn threads verwendet, sollte nur der mainthread die call-back Funktion aufrufen. Da wir auf threading in unserem PlugIn verzichtet haben stellt dies also keine Bedrohung für uns dar.

1.2.6 Die call-back Funktion

Wieder geben wir ein Beispiel einer C-Deklaration:

```
typedef int WINAPI /*Tcallbackfunc) (Tcommand, void*, void*, void*);
```

Die call-back Funktion erwartet vier Parameter: Der erste ist das `command` (z.B. "erzeuge ein Image"), die anderen sind Parameter. Beispiel:

$myimage = callback(ac\_create, 400, 300, 0)$ erzeugt ein neues (Graustufen) Image in `AstroArt` mit 400 px Breite, 300 px Höhe, und gibt einen Pointer zurück der für weitere Bearbeitungen notwendig ist.

1.2.7 Befehle

```
enum Tcommand {ac_getbuffer, ac_getsize, ac_redraw, ac_close, ac_savefits, ac_copy,
ac_writeheader, ac_readheader, prepareundo, ac_thresholds, ac_transferfunc, ac_getselpoint,
ac_getselpoint2, ac_gethwnd, ac_create, ac_open, ac_getimage, ac_getversion, ac_rename,
ac_getname, ac_getra, ac_getdec, ac_getx, ac_gety, ac_modified};
```

Folgende Tabelle zeigt einige Funktionen die bei unserem Projekt zum Einsatz gekommen sind:

Name	**Parametertyp, Kommentar**	**gibt Zurück**
`ac_getbuffer`	`pimage, &buffer, colorplane` Diese Funktion besorgt einen Pointer der auf den Buffer weist, in dem das Image gespeichert ist. Falls ein Farb-Image vor liegt gibt `colorplane` an welche Farbe (R, G, od. B) betrachtet werden soll.	`0, 2`
`ac_getsize`	`pimage, &x, &y` Diese Funktion bekommt in x und y Breite und Höhe des Bildes `pimage`. Wenn die Funktion 0 liefert, ist `pimage` kein gültiger Pointer.	`0, 1`
`ac_getselpoint`	`pimage, &x, &y` Liefert die Koordinaten eines selektierten Punktes im Image.	`0, 1`
`ac_getselpoint2`	`pimage, &x, &y` Liefert die Koordinaten des zweiten selektierten Punktes im Image. So kann ein Selektionsrechteck erfasst werden.	`0, 1`
`ac_readheader`	`pimage, NULL, NULL` Liefert den Pointer zu einem null-terminierten String welcher den FITS header repräsentiert.	`Pheader`

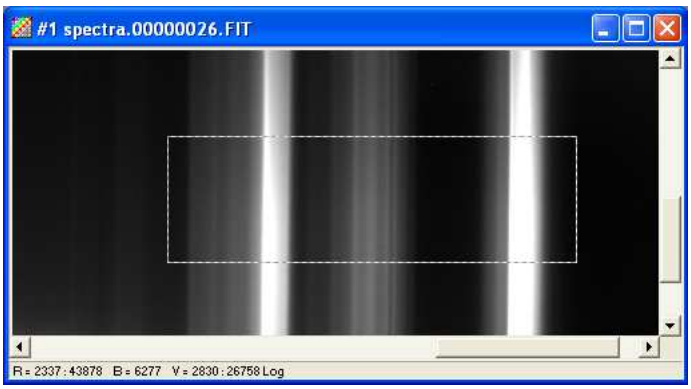
Wie die Liste am Beginn dieses Abschnittes zeigt gibt es noch viele weitere Befehle, auf die hier aber nicht weiter eingegangen wird. Das Prinzip sollte sich dennoch aus den genannten Beispielen erschließen.

Dies beendet unseren Einführenden Abschnitt in die von `AstroArt` definierte Schnittstelle an die unser PlugIn andockt.

1.3 Das FITS-Format

Es sollen noch ein paar Worte zu dem verwendeten FITS-Format*[4]* gesagt werden. Die Abkürzung "FITS" steht für "Flexible Image Transport System". Dieses Dateiformat, welches vorzugsweise

in der Astronomie zum Einsatz kommt, eignet sich besonders gut zur Aufnahme von Spektren. Folgendes Bild zeigt etwa ein solches FITS Image, geöffnet in `AstroArt`:



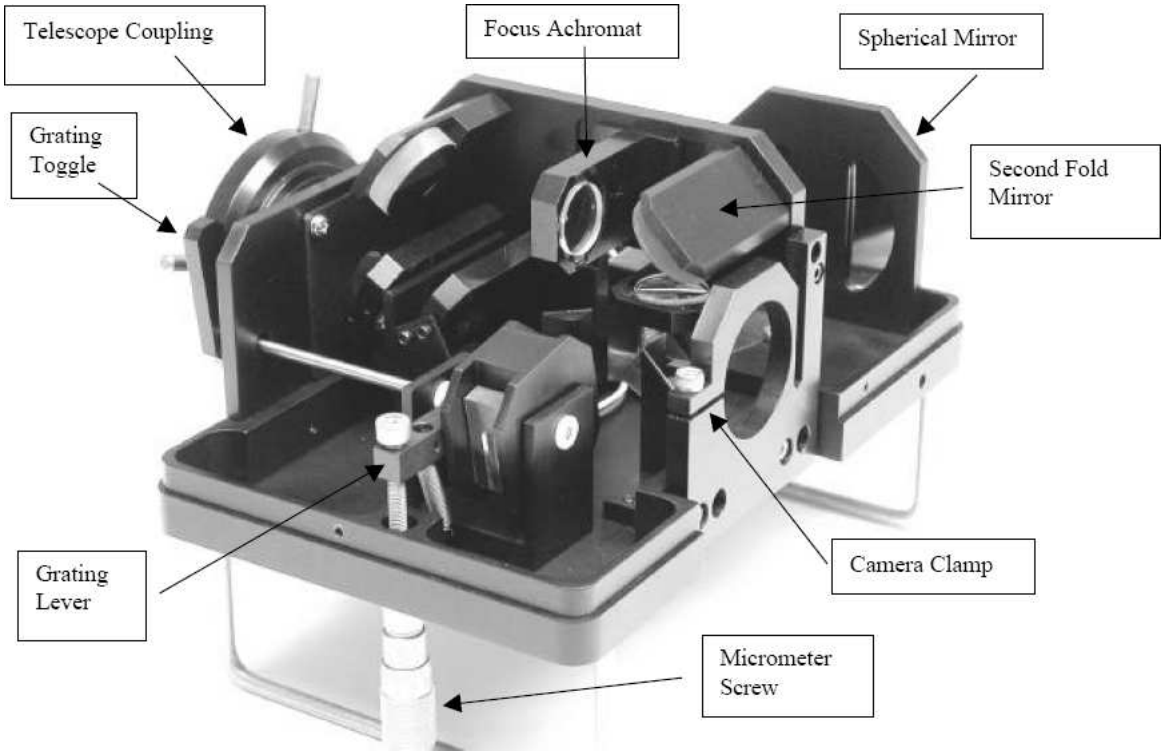
Die Intensitätsinformation liegt in der Graustufe der Darstellung, die Wellenlänge in der jeweiligen x-Position, also der Spalte.

1.4 Der SBIG Spektrograph

Die Images, die mit dem PlugIn ausgewertet werden sollen, sind mit dem SBIG Spektrographen in Verbindung mit einer CCD Kamera gewonnen worden. Aus diesem Grunde sollen hier auch die Spezifikationen des SBIG Spektrographen angeführt werden:

- Dispersion: 1.07 oder 4.3 $\mathring{A}$ per pixel
- Auflösung: Emissionslinie wird aufgenommen mit 2.2 oder 8 $\mathring{A}$ Halbwertsbreite
- Spektrale Abdeckung per Frame: 750 $\mathring{A}$ mit dem hochauflösenden Gitter bzw. 3200 $\mathring{A}$ mit dem niederauflösenden Gitter
- Zentralwellenlängenauswahl über kalibrierte Mikrometerschraube
- Empfindlichkeit: SNR 10:1 für einen 10 Mag Stern, 20 Minuten Belichtung mit einem ST-7E und einer 10" (25cm) Apertur im hochauflösenden Modus. Der niedrigauflösende Modus mit breitem Spalt ist um 2 Mag empfindlicher.
- Eingangsspalt: 18 μm, (2.3 Bogensekunden mit 63" (160cm) fokaler Länge)
- Akzeptanzwinkel: F/6.3 bis F/10. F/6.3 empfohlen für maximales Signal
- Abmessungen: 4× 5× 8"(10× 12× 20cm)
- Gewicht: Spektrograph plus ST-7: 5.2 pounds (2.4 kg)

Das folgende Bild zeigt den Spektrographen und seinen Aufbau:



2 Das PlugIn "Spectrum"

2.1 Wahl der Programmiersprache

Bei erster Betrachtung der Problemstellung schien es, als wäre Visual Basic 6.0 eine gute Wahl dieses Projekt umzusetzen. Dabei ist eine etwas differente Vorgehensweise notwendig als die in den vorherigen Abschnitten beschriebene, da mit VB6 nicht direkt eine DLL erzeugt wird, sondern eine von `AstroArt` bereitgestellte, "vermittelnde" DLL mit dem Namen pivblink.dll zum Einsatz kommt. Im Detail sieht das also so aus, dass die mitgelieferte pivblink.dll durch das Publishing des Projektes in der VB6 Entwicklungsumgebung mit den notwendigen Informationen versorgt wird. Da bereits ein Demoprojekt in VB vorhanden war sah es also so aus, als müsse man lediglich den eigenen Code in den vorgegebenen Rahmen einbetten, ohne dem Rest dabei Beachtung zu schenken. Dies funktionierte in den ersten Versuchen auch ganz gut, allerdings konnte ein Problem nicht in den Griff bekommen werden, was uns letztlich dazu bewog zu einer anderen Programmiersprache zu wechseln:
In Visual Basic gibt es grundsätzlich zwei Modi in denen eine Form geöffnet werden kann:

- `vbModal`
- `vbModeless`

Dabei kann eine Form die Modal geöffnet wird nicht mehr aus dem Fokus genommen werden, dh. es ist dann nicht mehr möglich auf ein anderes Programm im Hintergrund zuzugreifen. Ein Öffnen der Form im Modeless-Modus würde dieses Problem zwar beheben, jedoch ist es uns trotz intensiver Bemühungen nicht gelungen, dieses Problem zu beheben bzw. im Rahmen von VB zu umgehen. Dies hat uns letztlich dazu bewogen auf den Borland C++ Builder 6.0 umzusteigen, mit dem das Öffnen des PlugIns und Refokussieren der dahinterliegenden Applikation zu keinem

Zeitpunkt ein Problem darstellte. Auch für den Borland C++ Builder war bereits ein Demo PlugIn vorhanden, was uns den Einstieg in die uns neue Programmiersprache erheblich erleichterte.

2.2 Konzeption

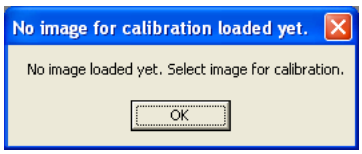
Es ist bei derlei Unterfangen nicht immer leicht und offensichtlich, wie eine Aufgabenteilung von Statten gehen sollte. Es gilt Schnittstellen zu definieren, unzählige Absprachen und Unterredungen zu führen und oftmaliges "Zuspielen" der Daten an den jeweilig anderen, um hier erfolgreich voranzukommen. Da aber das grundsätzliche Problem, den Fokus der Mutterapplikation zurückzugewinnen, hier nicht gegeben war, nahm das Projekt sehr schnell Form an und wuchs schließlich zur finalen Pracht heran.
Im Folgenden soll die Bedienung des PlugIns beschrieben werden, also der Ablauf in der Anwendung. Danach wollen wir uns die Lösungen im Detail ansehen, also in den Quellcode hineingehen und uns durch das Gehölz von Variablen und Funktionen schlagen, vielleicht sogar dem einen oder anderen Kommentar begegnen.

2.3 Die Bedienung des PlugIns "Spectrum"

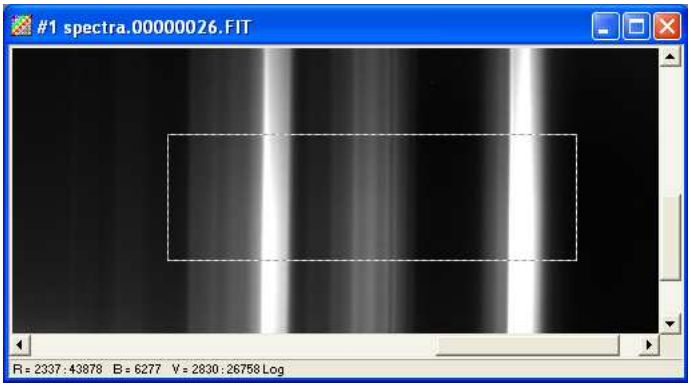
Nachdem das Programm `AstroArt` in englischer Sprache zur Verfügung steht haben wir uns entschieden, auch das PlugIn ausschließlich in englischer Sprache zu verfassen. (Ein Sprachenmix wäre einfach unschön). Wir wollen also den User begleiten, auf dem Wege der ihn zur Auswertung seiner Daten führt:
Folgendes Szenario sei gegeben:

- Ein Spektrum zur Kalibrierung des Spektrographen liegt vor (Wellenlänge bekannt)
- Ein Bild, welches Ausgewertet werden soll wurde unter denselben Bedingungen aufgenommen
- Das Plugin (die DLL) wurde in das `AstroArt`-Verzeichnis kopiert
- Der User ist geneigt mit unserem PlugIn an eine Ascii-Datei zu kommen, die Aufschluss über Wellenlänge und zugehöriger Intensität gibt.

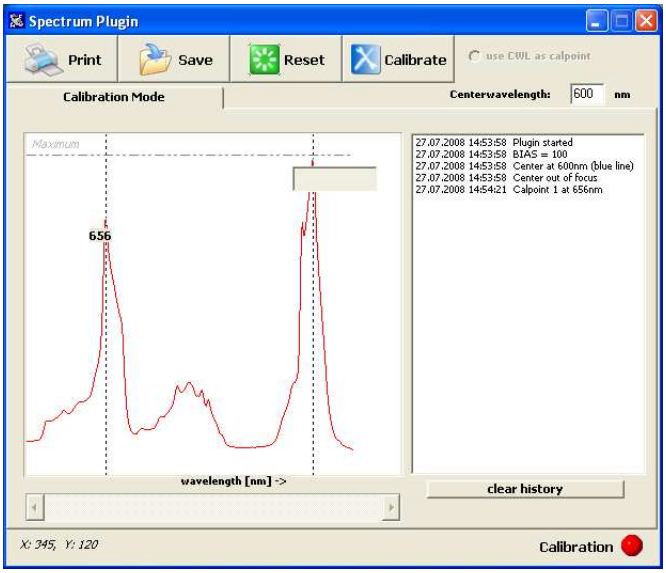
Der User, nennen wir ihn Schlicht U., öffnet auf seinem PC die Software `AstroArt`. Eine leere `AstroArt`-Umgebung begrüßt U. Dieser clickt auf Plugin→ Spectrum. Das Programm beginnt seinen Lauf: Da U. noch kein FITS Image geöffnet hat wird ihm dies erinnernd von dem PlugIn mitgeteilt, danach öffnet dieses einen "Open File Dialog" dessen Filter bereits auf die richtige Extension eingestellt sind.



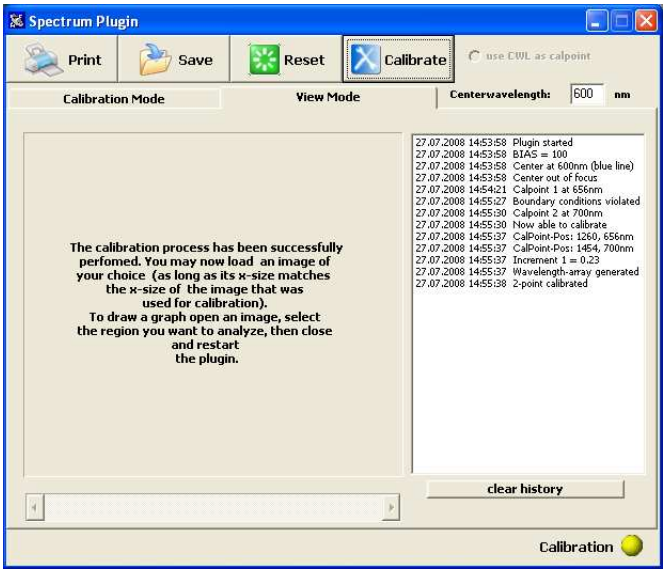
Er öffnet ein FITS Image und markiert einen interessanten Bereich, der etwa eine Emissionslinie mit bekannter Wellenlänge sein könnte.



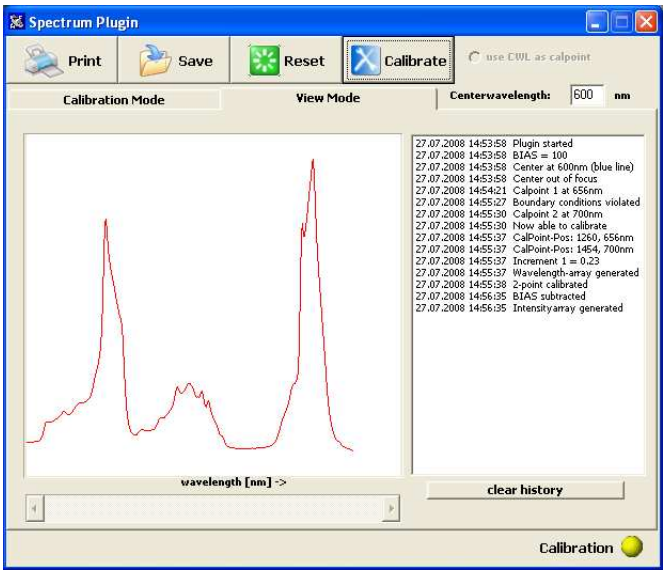
Nachdem er den Bereich markiert hat ruft U. das PlugIn neuerlich auf und befindet sich im Kalibriermodus.



Auf der Form des PlugIns sind nun eine Menge Informationen zu sehen. Wir sehen den Graph der Intensität als Funktion der Wellenlänge dargestellt. U. hat bereits offenbar eine H-α Linie identifiziert. Indem er mit der Maus ein Rechteck um den Peak gezogen hat, konnte er erfolgreich eine Wellenlänge zuordnen. Dabei hat das Plugin eine vertikale Linie eingezeichnet nachdem es das Maximum im selektierten Bereich gefunden hat, hat ein Eingabefenster geöffnet (ein solches ist gerade an dem anderen Peak geöffnet), und die dort eingegebene Wellenlänge in ein festes Label übernommen. Die rote LED im rechten unteren Eck verrät, dass noch zu wenig Informationen für eine erfolgreiche Kalibrierung eingegeben wurden. Ausserdem verfügt das PlugIn über eine History, welche, jeweils mit Datum und Zeitstempel, die Aktionen die vom User/vom PlugIn ausgeführt wurden zur Anzeige bringt. So wurden etwa das Bias, oder auch die Zentralwellenlänge aus dem FITS Header entnommen, falls diese Informationen dort verfügbar waren. Wurde auch die zweite Wellenlänge erfolgreich eingegeben, so kann der Button "Calibrate" gedrückt werden. Das PlugIn gibt die berechneten Stützstellen in der History aus und wechselt in den View-Mode, der vorher nicht verfügbar war.



Die LED ist jetzt gelb, dh. eine (suboptimale) Kalibrierung wurde durchgeführt. Die LED ist erst ab mindestens 3 Stützstellen grün. Außerdem wird U. auf die weitere Forgehensweise hingewiesen, dh. er kann nun ein Image öffnen und dieses analysieren:



U. hat jetzt die Möglichkeit z.B. den Graphen zu drucken (Button Print) oder aber auch die Werte (Wellenlänge und Intensität) in ein Ascii-File auszugeben (Button Save).
Kurz will ich auch noch den Radio-Button "Use CWL as Cal-Point" erwähnen:
Ist man bereits im Besitz eines Kalibrier-Images dessen Zentralwellenlänge genau bekannt ist, und wurde diese in den FITS Header eingegeben, so reicht es aus nur einen weiteren Punkt im Graphen zu markieren. Man clickt dann einfach auf den Button "Use CWL as Cal Point" und schon wird die Zentralwellenlänge mit dem aus dem Header extrahierten Wert als Stützstelle aufgenommen. Die History gibt übrigens darüber Auskunft wo sich die CWL befindet. Hat man einen Bereich markiert der die CWL beinhaltet, so wird diese als blaue, durchgezogene vertikale Linie im Plot

dargestellt. Falls sie nicht im gewählten Bereich liegt, so weist die History darauf hin (Center out of focus). Die CWL kann aber in jedem Falle als Cal-Point genutzt werden, solange sie nicht die Randbedingungen verletzt (Hinweis: Boundary conditions violated).
Wir haben jetzt also gesehen wie das PlugIn arbeitet. Nun wollen wir uns den Code im Detail ansehen.

2.4 Der Haupteinstiegspunkt des Plugins

Um das Plugin korrekt Öffnen zu können benötigt `AstroArt` drei Funktionen die von der DLL (dynamic link library) zur Verfügung gestellt werden müssen. Diese sind 'pi_initialize', 'pi_activate' und 'pi_finalize'.

Im Programm werden zunächst die Header-Dateien eingebunden sowie die Form definiert auf der letztlich die Ausgabe stattfindet. Die WINAPI 'DllEntryPoint' hängt mit dem Einstiegspunkt der DLL zusammen und ist vom Borland CBuilder generierter Code.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include <StrUtils.hpp>
//---------------------------------------------------------------------------
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
    return 1;
}
```

Im Folgenden werden die Callback-Funktionen definiert über die man mit `AstroArt` kommunizieren kann. Hier findet man ein Software-Development-Kit welches die Funktionsweise dieser Funktionen und der DLL im Speziellen anhand von Demoprogrammen verständlich macht.

```
//---------------------------------------------------------------------------
enum Tcommand {ac_getbuffer, ac_getsize, ac_redraw, ac_close, ac_savefits,
               ac_copy, ac_writeheader, ac_readheader, ac_prepareundo,
               ac_thresholds, ac_transferfunc, ac_getselpoint, ac_getselpoint2,
               ac_gethwnd, ac_create, ac_open, ac_getimage,
               ac_getversion, ac_rename, ac_getname, ac_getra,
               ac_getdec,ac_getx, ac_gety, ac_modified};

/* Commands: see docs

NAME              PARAMS TYPE                        RESULT

ac_getbuffer      (pimage, &buffer, colorplane)      0,2
ac_getsize        (pimage, &x, &y)                   0,1
ac_redraw         (pimage, calcstats, autovis)       0,1
ac_close          (pimage, askuser, NULL)            0,1
ac_savefits       (pimage, pfilename, NULL)          0,1
ac_copy           (pimage, NULL, NULL)               0,1
ac_writeheader    (pimage, pheader, NULL)            0,1
ac_readheader     (pimage, NULL, NULL)               pheader
ac_prepareundo    (pimage, NULL, NULL)               0,1
ac_thresholds     (pimage, min, max)                 0,1
ac_transferfunc   (pimage, trf, NULL)                0,1
ac_getselpoint    (pimage, &x, &y)                   0,1
ag_getselpoint2   (pimage, &x, &y)                   0,1
ac_gethwnd        (pimage, &HWND, NULL)              0,1
ac_create         (x,y,colored)                      pimage
ac_open           (pfilename, NULL, NULL)            pimage
ac_getimage       (selection, NULL, NULL)            pimage

ac_getversion     (NULL, NULL, NULL)                 300+
ac_rename         (pimage, name, path)               0,1
ac_getname        (pimage, keeppath, NULL)           pname
```

```
ac_getra          (pimage, x*100, y*100)            ra*100000
ac_getdec         (pimage, x*100, y*100)            dec*100000
ac_getx           (pimage, ra*1000000, dec*1000000) x*100
ac_gety           (pimage, ra*1000000, dec*1000000) y*100
ac_modified       (pimage, modified, nil)           0,1
*/
```

Das boolsche Variablenarray 'statusword' ist eine der zentralen Variablen im Programm. Es gibt Auskunft über den Programmstatus. Im Folgenden wird erklärt welche Bedeutung die Einträge mit dem jeweiligen Index haben.

```
////////////////////////////////////////////////////////////////////////////
//                                                                        //
//              statusword[10] ist ein boolean array der Klasse Form1     //
//       Das statusword gibt Auskunft ueber den momentanen Zustand des    //
//       Plugins. Dabei sieht die Aufteilung so aus:                      //
//                                                                        //
//           region of                                                    //
//           FITS image  calibration                                      //
//            selected    successful                                      //
//                |          |          |          |          |          //
//                |          |          |          |          |          //
//       ---------------------------------------------------------------  //
//      |     |     |     |     |     |     |     |     |     |     |    //
//      |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |    //
//      |     |     |     |     |     |     |     |     |     |     |    //
//       ---------------------------------------------------------------  //
//         |          |          |          |          |          |       //
//         |          |          |          |          |          |       //
//  calibration    cal-graph                                              //
//  image           drawn                                                 //
//  selected                                                              //
////////////////////////////////////////////////////////////////////////////
```

Um auf die Callback-Funktionen zugreifen bzw. Parameter übergeben zu können muss der folgende Codeabschnitt eingefügt werden:

```
typedef int WINAPI(*Tfcallback) (Tcommand, void*, void*, void*);

Tfcallback fcallback;
```

Nun können die einzelnen Callback-Routinen aufgerufen und parametrisiert werden. Ein Beispiel wäre:

```
MyImage = callback( ac_create,400,300,0 );
```

Hier wird ein FITS-Image erzeugt dessen Abmessung mit 400 Pixel Breite und 300 Pixel Höhe definiert ist. Zurückgegeben wird ein Pointer auf dieses Image.

Oft benutzte Werte werden als Konstanten definiert. Die globalen Variablen deklariert man danach.

```
int PACTIVEIMAGE = 4096;
int PSELECTIMAGE = 4097;
int PEQU_STD = 1000;
int PEQU_GAU = 1001;
int PCGREEN = 0;
int PCRED = 2;
int PCBLUE = 3;
//---------------------------------------------------------------------------
// global variables
//---------------------------------------------------------------------------
float *TempBuffer, *ImageBuffer;
int maxx, maxy, intensmax;
TRect MySourceRect;
TRect MyDestRect;
int gx1, gx2;
```

Um den Pixelwert an einer bestimmten Stelle im Image auslesen zu können wurde die Funktion 'getPixel' implementiert. Die if-Abfragen stellen sicher, dass die Werte in den jeweiligen Grenzen bleiben. Dann wird aus einem temporären Imagebuffer der jeweilige Pixelwert ausgelesen, welcher

die Intensität repräsentiert. Sie unterliegt einer Quantisierung von 2^{16}, kann also Werte zwischen 0 und 65535 annehmen. Der temporäre Buffer ermöglicht eine etwaige spätere Rücksetzung der durchgeführten Veränderungen am Image.

```
//---------------------------------------------------------------------------
// utility functions
//---------------------------------------------------------------------------
unsigned short getPixel(int x, int y)
{
    if ( x < 0 ) x = 0;
    if ( y < 0 ) y = 0;
    if ( x >= maxx ) x = maxx - 1;
    if ( y >= maxy ) y = maxy - 1;
    return TempBuffer[x+y*maxx];
}
```

Die Funktion 'Draw_Spektrum' ist verantwortlich für die Generierung der Plots. Es müssen die Koordinaten des Auswahlbereichs sowie das aus dem FITS-Header ermittelte Bias an die Funktion übergeben werden. Zunächst wird in einer if-Abfrage ermittelt welchen Status das Programm zurzeit einnimmt.
Ist das Programm bereits kalibriert kann das gewünschte Image gezeichnet werden. Die beiden Arrays für die Ascii-Ausgabe in die Datei werden zunächst auf Null gesetzt. Weiters wird das Metaimage 'Image4' weiß übermalen um sicherzustellen, dass keine Bildinformation eines alten Bildes auf das neue gezeichnet wird.
Nun wird in einer Schleife das Maximum der Intensität ermittelt. Dies dient der späteren Normierung. Für die Berechnung der Intensität wird über alle y-Werte summiert und anschließend gemittelt. Am y-Wert des Maximums wird zusätzlich noch eine horizontale Linie eingezeichnet. Sie zeigt an, wo sich das absolute Maximum des Plots befindet. Nun kann endlich mit der Zeichnung der eigentlichen Spektralkurve begonnen werden. Die Intensität wird wie gehabt berechnet und gleichzeitig in den Fensterbereich normiert. Hier erfolgt auch die Ausgabe in die Arrays für die Intensität und Wellenlänge. Der Plot wird mit der Funktion 'LineTo' in roter Farbe generiert. Danach erfolgt noch ein Eintrag in die History. Gezeichnet wird auf das Metaimage 'Image4', da dies für die Scrollfunktion des Plots nötig ist. Daher muß am Ende noch der jeweilige Bereich in das Ausgabefenster 'Image2' kopiert werden. Das erfolgt mit der Funktion 'CopyRect' wobei hierfür noch zwei Rechtecke definiert werden müssen.

```
void Draw_Spektrum( int x1, int x2, int y1, int y2, int Bias )
{
    int xAxesInc;
    double Intensity;

    if( Form1->statusword[3] == true )  //bereits kalibriert -> zeichne nun das Image
    {
        for( int i = 0; i < 1600; i++ )
        {
            Form1->intensityout[i] = 0;
            Form1->wavelengthout[i] = 0;
        }

        Form1->Image2->Canvas->Pen->Color = clRed;
        Form1->Image2->Width = 350;
        Form1->Image4->Width = 1600;
        Form1->imagewidth = abs(x2 - x1);
        Form1->Image4->Height = 320;

        // Image soll jedesmal geloescht werden, bevor ein neues gezeichnet wird
        Form1->Image4->Canvas->Brush->Color=clWhite;
        Form1->Image4->Canvas->FillRect( TRect(0,0,1600,Form1->Image4->Height) );

        xAxesInc = 0;
        //die erste Schleife berechnet den hoechsten mittelwert fuer die Normierung
        for( int x = x1; x <= x2; x++ )
        {
            for( int y = y1; y <= y2; y++ )
            {
                Intensity = Intensity + ( getPixel( x,y ) - Bias );
            }
            Intensity = Intensity /( y2 - y1 );
            if ( Intensity > intensmax) intensmax = Intensity;
```

```
        }

        Form1->Image4->Canvas->Pen->Color = clGray;
        Form1->Image4->Canvas->Pen->Style = psDashDotDot;
        Form1->Image4->Canvas->MoveTo( 0,20 );  // 320 - 300
        Form1->Image4->Canvas->LineTo( Form1->Image4->Width,20 );
        Form1->StaticTextMaximum->Visible = true;
        Form1->Image4->Canvas->Pen->Color = clRed;
        Form1->Image4->Canvas->Pen->Style = psSolid;

        for( int x = x1; x <= x2; x++ )
        {
            for( int y = y1; y <= y2; y++ )
            {
                // getPixel gives a value ranging from 0 - 65535.
                Intensity = Intensity + ( getPixel( x,y ) - Bias );
            }

            // Zuordnung der Intensitaet ohne Quanteneffizienz
            Form1->intensityout[x - x1] = Intensity / ( (y2 - y1)*(float)intensmax );
            //normierte Intesitaet, bias abgezogen
            Intensity = Intensity /( y2 - y1 )*( 300.0 / (float) intensmax );
            if( x == x1 ) Form1->Image4->Canvas->MoveTo( 0, 320.0 - Intensity );

            Form1->Image4->Canvas->LineTo( xAxesInc, 320.0 - Intensity );
            xAxesInc = xAxesInc++;
        }

        for( int i = 0; i <= Form1->imagewidth; i++)
        {
            // Wellenlaenge wird in nm ausgegeben
            Form1->wavelengthout[i] = Form1->xaxis[i + x1] + 100;
        }
        Form1->History("BIAS subtracted");
        Form1->History("Intensityarray generated");
        MySourceRect = Rect( 0, 0, Form1->Image2->Width, Form1->Image4->Height);
        MyDestRect = Rect( 0, 0, Form1->Image2->Width, Form1->Image2->Height );
        Form1->Image2->Canvas->CopyRect( MyDestRect,Form1->Image4->Canvas, MySourceRect );
    }
```

Ist das Plugin noch nicht kalibriert geht das Programm in den folgenden Codeabschnitt. An der Stelle an der die Position der CenterWaveLenght aus dem Header ermittelt wurde wird eine vertikale, blaue Linie gezeichnet. Anschließend wird - genau wie vorhin - der Plot gezeichnet. Nun allerdings auf 'Image3', welches das Metaimage für den Kalibriermodus darstellt. Am Ende wird das 'statusword[2]' auf true gesetzt und dem Plugin somit mitgeteilt, dass das Kalibrierspektrum soeben gezeichnet wurde. Falls der Benutzer das Plugin schließt und einen neuen Bereich auf dem FITS-Image markiert wird er darauf aufmerksam gemacht, dass schon ein Kalibrierimage geladen ist, falls er das Plugin wieder öffnet. Dies gewährleistet die else-Bedingung mit der Messagebox.

```
    else    //noch nicht kalibriert -> zeichnet das Kalibrierspektrum
    {
        if ( Form1->statusword[2] == false )
        {
            if ( (gx1 < 800) && (gx2 > 800) )
            {
                Form1->Image3->Canvas->Pen->Color = clBlue;
                Form1->Image3->Canvas->MoveTo(800 - gx1,0);
                Form1->Image3->Canvas->LineTo(800 - gx1,Form1->Image3->Height);
            }
            intensmax = 0;

            //die erste Schleife berechnet den hoechsten mittelwert fuer die normierung
            for( int x = x1; x <= x2; x++ )
            {
                for( int y = y1; y <= y2; y++ )
                {
                    Intensity = Intensity + ( getPixel( x,y ) - Bias );
                }
```

```
                Intensity = Intensity /( y2 - y1);
                if ( Intensity > intensmax) intensmax = Intensity;
            }

            Form1->Image3->Canvas->Pen->Color = clGray;
            Form1->Image3->Canvas->Pen->Style = psDashDotDot;
            Form1->Image3->Canvas->MoveTo( 0,20 );  // 320 - 300
            Form1->Image3->Canvas->LineTo( Form1->Image3->Width,20 );
            Form1->StaticTextMaximum->Visible = true;

            Form1->Image1->Canvas->Pen->Color = clRed;
            Form1->Image3->Canvas->Pen->Color = clRed;
            Form1->Image3->Canvas->Pen->Style = psSolid;
            Form1->Image3->Width = abs(x2 - x1);
            Form1->Image3->Height = 320;
            xAxesInc = 0;

            for( int x = x1; x <= x2; x++ )
            {
                for( int y = y1; y <= y2; y++ )
                {
                    Intensity = Intensity + ( getPixel( x,y ) - Bias );
                }

                Intensity = Intensity /( y2 - y1)*( 300.0 / (float)intensmax );

                if( x == x1 ) Form1->Image3->Canvas->MoveTo( 0   , 320.0 - Intensity );

                Form1->Image3->Canvas->LineTo( xAxesInc, 320.0 - Intensity );
                xAxesInc = xAxesInc ++;
            }
            MySourceRect = Rect( 0, 0, Form1->Image1->Width, Form1->Image3->Height );
            MyDestRect = Rect( 0, 0, Form1->Image1->Width, Form1->Image1->Height );
            Form1->Image1->Canvas->CopyRect( MyDestRect,Form1->Image3->Canvas, MySourceRect );
            Form1->statusword[2] = 1;
        }
        else
        {
            Application->MessageBox("Calibrationgraph already drawn.
            Use button ''Reset'' to recalibrate.","Info",MB_OK);
        }
    }
}
```

Die folgenden Funktionen zählen zu jenen Unterprogrammen im Plugin die von **AstroArt** aufgerufen werden. Zum einen handelt es sich um eine Initialisierungsfunktion zum anderen um eine Finalisierungsfunktion. **AstroArt** sucht bei jedem Start nach neuen DLLs im Installationsverzeichnis. Mit Hilfe der Funktion 'pi_initialize' kann der Name des Plugins - also wie es letztlich im Menü von **AstroArt** dargestellt wird - sowie eine kurze Beschreibung hinzugefügt werden.

```
//---------------------------------------------------------------------------
// AstroArt functions
//---------------------------------------------------------------------------
extern "C" __declspec(dllexport)
int WINAPI pi_initialize(Tfcallback funcaddress, char *menuname,char *description, HWND whandle)
{
    strcpy(menuname,"Spectrum");
    strcpy(description,"FITS Image Processing");
    fcallback = funcaddress;
    Form1 = NULL;
    return 1;
}
```

Wir betrachten nun die Funktion 'pi_activate'. Sie wird aufgerufen wenn das PlugIn gestartet wird. Zugegeben, die Funktion ist etwas länglich, aber ihre Arbeitsweise ist im Prinzip recht einfach. Sie orientiert sich im Wesentlichen an dem globalen Array statusword. Aber der Reihe nach: Zuerst werden Variablen für den Selektionsbereich definiert und genullt. Mittels callback-function wird

der im Image selektierte Punkt erhoben. Danach wird, ebenfalls mit callback-functions der FITS-Header ausgelesen, die CWL und das BIAS werden extrahiert.
Wir erreichen nun die erste Abfrage: Sie prüft ob bereits eine Form1 existiert. Außerdem sieht sie nach, ob wohl ein FITS Image in `AstroArt` geladen wurde. Falls nein öffnet sie einen Open File Dialog.

```
extern "C" __declspec(dllexport)
void WINAPI pi_activate(void *pimage, HWND whandle)
{
  int x1, x2, y1, y2, x1_old, Bias, center;
  char *PFITSHeader;

  x1 = x2 = y1 = y2 = maxx = maxy = 0;
  TempBuffer = 0;
  fcallback( ac_getselpoint ,pimage,&x1,&y1 );
  fcallback( ac_getselpoint2 ,pimage,&x2,&y2 );
  if ( x1 > 1600 || x2 > 1600 || y1 > 1200 || y2 > 1200 ) return;
  gx1 = x1;
  gx2 = x2;

  PFITSHeader = (char*) fcallback( ac_readheader ,pimage ,NULL,NULL );
  Bias = Form1->BiasSearch(PFITSHeader);
  center  = Form1->CenterSearch(PFITSHeader);

  if (Form1 == NULL) Form1 = new TForm1(Application);

  if ( pimage == NULL )
  {
    Application->MessageBox("No image loaded yet. Select image for calibration.",
    "No image for calibration loaded yet.", MB_OK);
    Form1->OpenDialog->Title = "Open Calibration Image...";
    Form1->OpenDialog->Filter = "FITS Image (*.fit; *.fts; *.fits)|
    *.fit;*.fts;*.fits|#
    All files (*.*)|*.*";
    Form1->OpenDialog->FilterIndex = 1;

    if( Form1->OpenDialog->Execute() )
    {
      pimage = (void*) fcallback( ac_open, Form1->OpenDialog->FileName.c_str() ,NULL,NULL );
      Form1->statusword[0] = 1;
    }
    return;
}
else
{

if( Form1->statusword[0] == 0)
{
 if( IDYES==Application->MessageBox("Do you want to use the active
 image for calibration?", "Calibration", MB_YESNO) )
 {
  Form1->statusword[0] = 1;
 }
 else
 {
 Form1->OpenDialog->Title = "Open Calibration Image...";
 Form1->OpenDialog->Filter = "FITS Image (*.fit; *.fts; *.fits)|
 *.fit;*.fts;*.fits|All files (*.*)|*.*";
 Form1->OpenDialog->FilterIndex = 1;
 if( Form1->OpenDialog->Execute() )
 {
  fcallback( ac_close, pimage, (void*)1,NULL );
  pimage = (void*) fcallback( ac_open, Form1->OpenDialog->FileName.c_str() ,NULL,NULL );
  Form1->statusword[0]=1;
 }
 return;
 }
}
```

```
}
```

Außerdem kann man sehen, dass falls schon ein Image geladen wurde, aber noch nicht kalibriert wurde der User gefragt wird, ob er das aktive Image zur Kalibrierung heranziehen möchte. Es folgt der nächste wichtige Abschnitt: Image geladen, Bereich selektiert aber noch nicht kalibriert.

```
if ( Form1->statusword[3] == false )     //noch nicht kalibriert
{
        Form1->Image3->Visible = false;
        if( abs(x1 - x2) == 0 )
        {
                Application->MessageBox("Select region within image for
    calibration and restart the plugin."
    ,"Select region",MB_OK);
                return;
        }
        Form1->statusword[1] = true;
        Form1->Image3->Width = abs(x2 - x1);
        fcallback( ac_getsize,pimage,&maxx,&maxy );
        fcallback( ac_getbuffer,pimage,&ImageBuffer,(void*)PCGREEN );
        Form1->calimagewidth = maxx;
        fcallback(ac_prepareundo,pimage,NULL,NULL);

        TempBuffer = (float*) SysGetMem( maxx*maxy*4 );
// temporary buffer to manipulate the image
        memcpy( TempBuffer,ImageBuffer,maxx*maxy*4 );

        if ( abs(x2 - x1) >= 350)
        {
                Form1->ScrollBar1->Max = abs(x2 - x1) - 350;
                Form1->ScrollBar1->Enabled = true;
        }
        else Form1->ScrollBar1->Enabled = false;

        Draw_Spektrum( x1,x2,y1,y2,Bias );
        Form1->TabSheetPicView->TabVisible = false;
        Form1->Image1->BringToFront();
        Form1->Visible = true;
        Form1->globalx1 = gx1;
        Form1->globalx2 = gx2;
        Form1->StaticText3->Parent = Form1->TabSheetCalibrate;
        Form1->StaticText3->Caption = "wavelength [nm] ->";
        Form1->StaticText3->Left = 160;
        Form1->StaticText3->Top = 340;
        Form1->Memo1->Text = "";
        Form1->Memo2->Text = "";
        Form1->StaticText3->Visible = true;
        Form1->StaticText3->BringToFront();
        Form1->History("Plugin started");
        Form1->History("BIAS = " + IntToStr(Bias));
        Form1->History("Center at " + IntToStr(center) + "nm (blue line)");

        if ( (gx1 > 800) || (gx2 < 800)) Form1->History("Center out of focus");

        Form1->Edit2->Text = IntToStr(center);
        Form1->Show();
}
```

Nach einigen Abfragen, ob Bereich selektiert oder nicht, ob Image größer als Darstellungsbereich oder nicht, wird die Funktion Draw_Spektrum aufgerufen. Sie zeichnet das Spektrum des selektierten Bereiches in ein Image. Außerdem wird die Form für die erste Anzeige konditioniert: Der View-Mode wird deaktiviert, die History mit den ersten Daten geladen. Letztlich wird die Form zur Anzeige gebracht.

```
if ( Form1->statusword[3] == 1 )   //bereits kalibriert -> zeichne nun das Image
{
        fcallback( ac_getselpoint,pimage,&x1,&y1 );
        fcallback( ac_getselpoint2,pimage,&x2,&y2 );
```

```
                Form1->Memo3->Visible = false;
                Form1->Image4->Visible = false;
                Form1->PageControl1->ActivePage->Enabled = true;
                if(  abs(x1 - x2) == 0 )
                {
                        Application->MessageBox("Select region within
            image and restart the plugin.",
            "Select region",MB_OK);
                        return;
                }

                if ( x1 > 1600 || x2 > 1600 || y1 > 1200 || y2 > 1200 ) return;

                fcallback( ac_getsize,pimage,&maxx,&maxy );
                if

 ( maxx == Form1->calimagewidth )
                {
                        fcallback( ac_getbuffer,pimage,&ImageBuffer,(void*)PCGREEN );
                        fcallback( ac_prepareundo,pimage,NULL,NULL );

                        TempBuffer = (float*) SysGetMem( maxx*maxy*4 );
            // temporary buffer to manipulate the image
                        memcpy( TempBuffer,ImageBuffer,maxx*maxy*4 );

                        if ( abs(x2 - x1) >= 350)
                        {
                                Form1->ScrollBar2->Max = abs(x2 - x1) - 350;
                                Form1->ScrollBar2->Enabled = true;
                        }
                        else Form1->ScrollBar2->Enabled = false;

                        Draw_Spektrum( x1,x2,y1,y2,Bias );
                        Form1->PageControl1->ActivePageIndex = 1;
                        Form1->Image2->Visible = true;
                        Form1->Visible = true;
                        Form1->StaticText5->Parent = Form1->TabSheetPicView;
                        Form1->StaticText5->Caption = "wavelength [nm] ->";
                        Form1->StaticText5->Left = 160;
                        Form1->StaticText5->Top = 340;
                        Form1->StaticText5->Visible = true;
                        Form1->StaticText5->BringToFront();
                        Form1->Show();
                }
                else Application->MessageBoxA("Image-width doesn't match
        width of calibration image!
        Select another image or recalibrate.", "info", MB_OK);
        }
}
```

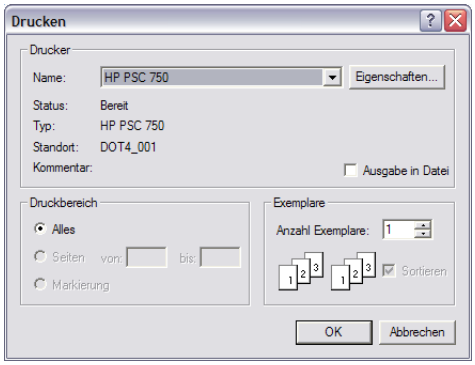
Dieser letzte Abschnitt wird angesprungen wenn nur noch das Auswertungsimage zu zeichnen ist.

Im Programm 'pi_finalize' kann man diverse Aktionen implementieren die ausgeführt werden wenn `AstroArt` schließt.

```
extern "C" __declspec(dllexport)
void WINAPI pi_finalize(HWND whandle)
{
    // optional clean-up
}
```

2.5 Die Print-Funktion

Um die generierten Plots ausdrucken zu können wurde eine Print-Funktion implementiert. Hierfür muss ein PrintDialog in der Form installiert werden. Beim Betätigen des Buttons "Print" wird nun die Unterfunktion "ButtonPrintClick" aufgerufen und ein Dialogfenster für den Drucker geöffnet. Man kann in diesem Dialogfenster die üblichen Einstellungen für den Drucker festlegen, d.h. z.B. Anzahl der Kopien oder Druckqualität.



In der Unterfunktion ist dafür der folgende Code zuständig:

```
/void __fastcall TForm1::ButtonPrintClick(TObject *Sender)
{
    TPrinter *MyPrinter = Printer();

    int copies; // number of copies of each page to print each time

    PrintDialog->Options << poPrintToFile;  // PrintToFile CheckBox active
    if ( !PrintDialog->Execute() ) return;  // user did not press OK

    copies = PrintDialog->Copies;

    // Title to appear in the print manager and network print banners
    MyPrinter->Title = "AstroArt 4.0 Spektrum Plugin";

    // Initialize print job
    MyPrinter->BeginDoc();
```

Zunächst wird ein Objekt 'MyPrinter' definiert. Diesem Objekt können später Eigenschaften zugewiesen werden. Weiters soll die Option 'PrintToFile' in PrintDialog aufscheinen. Wenn im Dialogfenster nicht auf 'OK' gedrückt wurde soll die Subroutine beendet werden. In die Variable 'copies' wird die Anzahl der ausgewählten Kopien geschrieben. Danach wird der Titel des Druckauftrags festgelegt und mit dem Druck begonnen.

Nun wird ein Druckbereich definiert in welchem später der Plot gedruckt wird. Dies geschieht mit dem Objekt 'MyRect', also einem Rechteck mit bestimmter Größe, welche abhängig vom eingestellten Druckformat ist.

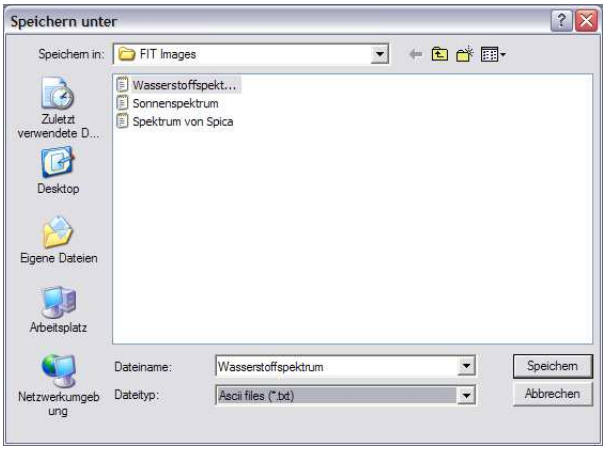
```
    TRect MyRect;
    if ( MyPrinter->Orientation == 0 )  // 0 = poPortrait
    {
        MyRect = Rect( MyPrinter->PageWidth/6, MyPrinter->PageHeight/4,
                       MyPrinter->PageWidth - MyPrinter->PageWidth/6,
                       MyPrinter->PageHeight - MyPrinter->PageHeight/4 );
    }
    if ( MyPrinter->Orientation == 1 )  // 1 = poLandscape
    {
        MyRect = Rect( MyPrinter->PageWidth/6, MyPrinter->PageHeight/6,
```

```
                MyPrinter->PageWidth - MyPrinter->PageWidth/6,
                MyPrinter->PageHeight - MyPrinter->PageHeight/6 );
    }

    for ( int j = 0; j < copies; j++ )
    {
        // print a banner
        MyPrinter->Canvas->Rectangle( 40,40,MyPrinter->PageWidth,
                        MyPrinter->PageHeight - 100 );
        MyPrinter->Canvas->MoveTo( 40,150 );
        MyPrinter->Canvas->LineTo( MyPrinter->PageWidth, 150 );

        if( Form1->PageControl1->ActivePageIndex == 0 )
        {
            MyPrinter->Canvas->StretchDraw( MyRect,Image3->Picture->Bitmap );
            MyPrinter->Canvas->TextOut( 100,80,"AstroArt 4.0 Spektrum Plugin,
                Date: " + Date() + ", Time: " + Time() + ", Calibration mode" );
            Form1->History("Printing calibration image");
        }
        if( Form1->PageControl1->ActivePageIndex == 1 )
        {
            MyPrinter->Canvas->StretchDraw( MyRect,Image4->Picture->Bitmap );
            MyPrinter->Canvas->TextOut( 100,80,"AstroArt 4.0 Spektrum Plugin,
                Date: " + Date() + ",Time: " + Time() + ", View mode" );
            Form1->History("Printing image");
        }
        MyPrinter->Canvas->DrawFocusRect(MyRect);
        MyPrinter->Canvas->TextOut( MyPrinter->PageWidth / 4,
        MyPrinter->PageHeight - MyPrinter->PageHeight/7,"Printing the standardized
            intensity I/Io [ ] as a function of the wavelenght" );
    }
    MyPrinter->EndDoc();
}
```

Der eigentliche Druckvorgang beginnt in der for-Schleife. Zur optischen Verschönerung wird ein Rahmen gezeichnet sowie ein Header implementiert, in welchem das Datum und die Uhrzeit mitausgegeben werden. Der Plot wird mittels der Funktion 'StrechDraw' in das oben definierte Rechteck eingepasst. Darunter wird eine Art Achsenbeschriftung ausgegeben sowie ein Hinweis auf den Druckvorgang an die History geschickt.

2.6 Die Save-Funktion

Die Save-Funktion dient zur Speicherung der Daten als Ascii-Code in einem Textfile. Wenn die Kalibrierung beendet ist und ein Image im 'View Mode' betrachtet wird können die Werte der Intensität und der zugehörigen Wellenlänge gespeichert werden.



Im Code wurde das mit einem SaveDialog realisiert, welcher in der Form installiert wird.

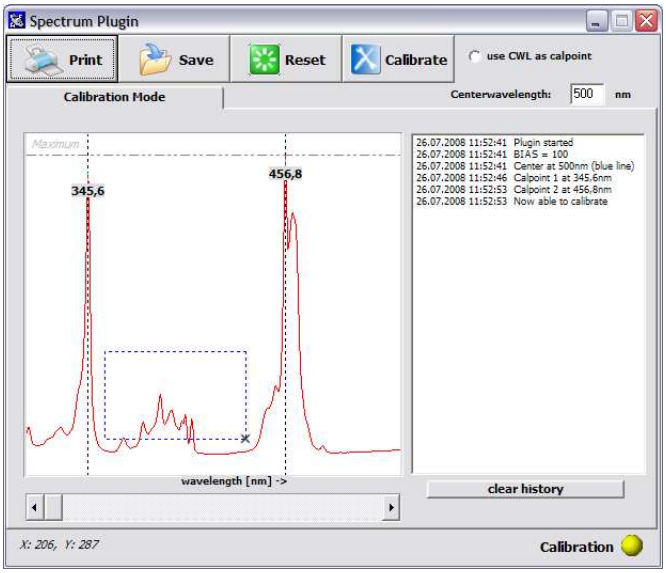
```
void __fastcall TForm1::ButtonSaveClick(TObject *Sender)
{
    SaveDialog->Options.Clear();
    SaveDialog->Filter = "Ascii files (*.txt)|*.txt|All files (*.*)|*.*";
    SaveDialog->DefaultExt = NULL;
    if ( SaveDialog->Execute() )
    {
        if ( SaveDialog->FilterIndex == 1 )
        {
            ofstream outfile;

            outfile.setf(ios_base::fixed);

            if ( !SaveDialog->Options.Contains(ofExtensionDifferent) )
                outfile.open( (SaveDialog->FileName + ".txt").c_str(),ios::out );
            else outfile.open( (SaveDialog->FileName).c_str(),ios::out );

            outfile << "_____________________________________________________________"
                << endl << endl;
            outfile << " AstroArt 4.0 Plugin" << endl << " Date: " <<
                DateToStr( Date() ).c_str() << ", Time: " << TimeToStr( Time() ).c_str()
                    << endl << endl << endl;
            outfile << "  Wavelenght [nm ]" << "       " << "Intensity I/Imax [ ]" << endl << endl;

            for( int i = 0; i < 1600; i++ )
            {
                outfile << setw(14) << setprecision(2) << Form1->wavelengthout[i]
                    << "    " << setw(20) << setprecision(5)
                    << Form1->intensityout[i] << endl;
            }
            outfile.close();
            Form1->History("Saved data in text file");
            return;
        }
    }
}
```

Durch den Filter wird sichergestellt, dass zunächst nur Textdateien dargestellt werden. Die Endung eines Dateinamens wird standardmäßig auf null gestellt. Wenn der SaveDialog ausgeführt wird, kann nun zuerst abgefragt werden welcher Filter eingestellt und ob die Endung des gewünschten Dateinamens eine andere als null ist. Ist das nicht gegeben wird die Datei mit der Endung '.txt' gespeichert. In jedem anderen Fall wird sie mit der jeweiligen Endung gespeichert. Dies garantiert, dass wenn die gewählte Datei schon die Endung '.txt' hat, nicht nocheinmal eine Endung hinzukommt. In das Objekt 'outfile' werden nun die Werte und eine kurze Info in die History

geschrieben.

2.7 Auswählen eines Bereichs im Spektrum

Um der jeweiligen Spektrallinie eine Wellenlänge zuordnen zu können, muss ein bestimmter Bereich markiert werden. Dies geschieht mit der Maus. Man bewegt den Zeiger an die gewünschte Stelle und zieht bei gedrückter Maustaste ein Rechteck über den Bereich. Nachdem man die Maustaste losgelassen hat sucht ein Alorithmus das Maximum im Spektrum und es erscheint eine Textbox in der man die Wellenlänge eingeben kann.



Dazu sind zunächst Einstellungen zu treffen, die gewährleisten das bestimmte Unterprogramme ausgeführt werden wenn sich der Mauszeiger über einem Objekt - hier 'Image1' - befindet und wenn dabei eine Maustaste gedrückt wird. Die dazu nötigen Subroutinen sind 'Image1MouseDown', 'Image1MouseMove' sowie 'Image1MouseUp'.

2.7.1 Image1MouseDown

Dieses Unterprogramm wird ausgeführt, wenn sich der Mauszeiger über dem Image1 befindet und eine Maustaste gedrückt wird. Der Quellcode hierfür sieht wie folgt aus:

```
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
    ButtonPressed = true;

    xCorner1 = X;
    yCorner1 = Y;
    X_start = xCorner1;
    Y_start = yCorner1;
    X_old = xCorner1;
    Y_old = yCorner1;
    Form1->Image1->Canvas->Pen->Mode = pmNotXor;
}
```

Zunächst wird die boolsche Variable 'ButtonPressed' auf true gesetzt. Diese legt fest ob eine Taste gedrückt wurde. Weiters werden die Koordinaten des Mauszeigers in Variablen geschrieben.

Zusätzlich wird in einen speziellen Zeichenmodus gewechselt. Wir wollen den ausgewählten Bereich im Spektrum mit einem Rechteck markieren. Der NotXOR-Modus stellt sicher, dass das alte Rechteck beim bewegen des Mauszeigers wieder gelöscht wird.

2.7.2 Image1MouseMove

Wenn sich der Mauszeiger über Image1 bewegt wird dieses Unterprogramm aufgerufen. Es gibt die Koordinaten des Zeigers an. Diese werden auf einem Label in der Form ausgegeben. Weiters wird abgefragt ob eine Taste gedrückt ist. Ist das der Fall wird von den Ursprungskoordinaten bis zur aktuellen Position ein gepunktetes, blaues Rechteck gezogen. Das vorige wird durch nochmaliges zeichnen eines Rechteckes derselben Farbe wieder gelöscht. Das funktioniert aber nur im NotXOR-Modus.

```
void __fastcall TForm1::Image1MouseMove(TObject *Sender,
        TShiftState Shift,int X, int Y)
{
    Form1->LabelStatus->Caption  =  AnsiString(" X: ") + X +
                                    AnsiString(",  Y: ") + Y;

    if( ButtonPressed )
    {
        Form1->Image1->Canvas->Pen->Style = psDot;
        Form1->Image1->Canvas->Pen->Color = clBlue;
        Form1->Image1->Canvas->Rectangle( X_start,Y_start,X_old,Y_old );
        Form1->Image1->Canvas->Rectangle( X_start,Y_start,X,Y );
        X_old = X;
        Y_old = Y;
    }
}
```

2.7.3 Image1MouseUp

Wenn die Maustaste wieder losgelassen wird erfolgt der Sprung in dieses Unterprogramm. Es werden die aktuellen Koordinaten des Mauszeigers übernommen sowie die Variable 'ButtonPressed' auf false gesetzt. Nun wird das Intensitätsmaximum im markierten Bereich gesucht. Dazu werden die Pixel im Bereich zeilenweise ausgelesen und ein neuer Wert für das Maximum festgelegt wenn die Farbe übereinstimmt und der Wert kleiner als das vorherige Maximum ist (kleiner deshalb weil von oben gezählt wird). Zusätzlich wird eine Variable 'OK' gesetzt, die angibt ob sich im betrachteten Bereich überhaupt etwas befindet.

```
void __fastcall TForm1::Image1MouseUp(TObject *Sender,
        TMouseButton Button,TShiftState Shift, int X, int Y)
{
    bool OK = false;
    ButtonPressed = false;
    xCorner2 = X;
    yCorner2 = Y;
    yMaximum = yCorner2;

    for( int i = xCorner1; i <= xCorner2; i++ )
    {
        for( int j = yCorner1; j <= yCorner2; j++ )
        {
            if( Form1->Image1->Canvas->Pixels[i][j] == clRed && j < yMaximum )
            {
                xMaximum = i;
                yMaximum = j;
                yCorner2 = yMaximum;      // Verbessert den Algorithmus
                OK = true;
            }
        }
    }
```

Nun wird das Rechteck das den Bereich markiert wieder entfernt indem es übermalt wird. Wenn nun 'OK' gesetzt ist, wird an der Stelle des Maximums eine vertikale Linie im Plot gezeichnet. Dies geschieht sowohl im Image1 als auch im Metaimage Image3. Nun erscheint eine Eingabebox in der man die Wellenlänge eingibt. Hier ist darauf zu achten, dass zusätzlich noch eine ScrollBar

implementiert wurde und der Wert derselbigen noch hinzuaddiert werden muss. Das Array 'yline' dient zur Speicherung der Position der Linie, damit später an der jeweiligen Stelle ein Label mit der Wellenlängeninformation eingefügt werden kann.

```
    Form1->Image1->Canvas->Rectangle( X_start, Y_start, X, Y );
    if( OK )
    {
        Form1->Image1->Canvas->Pen->Color = clBlack;
        Form1->Image3->Canvas->Pen->Color = clBlack;
        Form1->Image3->Canvas->Pen->Style = psDot;
        Form1->Image3->Canvas->Pen->Mode = pmNotXor;
        Form1->Image1->Canvas->MoveTo( xMaximum, Image1->Height );
        Form1->Image3->Canvas->MoveTo( xMaximum + Form1->ScrollBar1->Position, Image1->Height );
        Form1->Image1->Canvas->LineTo( xMaximum, 0 );
        Form1->Image3->Canvas->LineTo( xMaximum + Form1->ScrollBar1->Position, 0 );
        yline[calibrations] = xMaximum + Form1->Image3->Left +
        Form1->ScrollBar1->Position - Form1->ScrollBar1->Left;
        Edit1->Left = xMaximum;
        Edit1->Top = yCorner2 + 100;
        Edit1->BringToFront();
        Form1->RadioButton1->Enabled = false;
        Edit1->Visible = true;
        Edit1->SetFocus();
    }
    Form1->Image1->Canvas->Pen->Style = psSolid;
    Form1->Image1->Canvas->Pen->Mode = pmNot;
    Form1->Image3->Canvas->Pen->Style = psSolid;
    Form1->Image3->Canvas->Pen->Mode = pmNot;
}
```

Am Ende wird der Linienstil wieder zurückgesetzt.

2.8 Die Reset-Funktion

Sie dient zur Rücksetzung der Kalibrierung. Hier werden die Label für die Wellenlänge entfernt, die Bildinhalte gelöscht sowie alle beteiligten Variablen in ihrenGrundzustand versetzt.

```
void __fastcall TForm1::ButtonResetClick(TObject *Sender)
{
    for ( int i = 0; i < calibrations; i++)
    {
        mylabel[i]->Visible = false;
    }
    calibrations = 0;
    Form1->Image1->Picture = NULL;
    Form1->Image3->Picture = NULL;
    Form1->Image2->Picture = NULL;
    Form1->Image4->Picture = NULL;
    Form1->ScrollBar1->Position = 0;
    Form1->ImageProgress->Picture->LoadFromFile( ExtractFilePath(Application->ExeName)
        + "\\Pics\\ProgressRed.bmp" );
    Form1->statusword[2] = 0;
    Form1->statusword[3] = 0;
    Form1->TabSheetPicView->TabVisible = false;
    Form1->Image1->Canvas->Brush->Color=clWhite;
    Form1->Image1->Canvas->FillRect( TRect(0,0,Form1->Image1->Width,
        Form1->Image1->Height) );
    Form1->Image1->Enabled = true;
    Form1->RadioButton1->Enabled = true;
    Form1->RadioButton1->Checked = false;
    Form1->LabelStatus->Visible = true;
    Form1->StaticTextMaximum->Visible = false;
    cwlclicked = 0;

    //Nullsetzen aller Variablen:
    for ( int i = 0; i < 1600; i++)
    {
        xaxis[i] = 0;
```

```
        yaxis[i] = 0;
        wavelengthout[i] = 0;
        intensityout[i] = 0;
    }

    for ( int ii = 0; ii < 50; ii++)
    {
        calpoint[ii] = 0;
        valueposition[ii] = 0;
        increment[ii] = 0;
    }
    valuecounter = 0;
}
```

2.9 Auslesen des FITS-Headers

Um an bestimmte Information über das Image zu gelangen muß der jeweilige Header ausgelesen werden. Der Header ist ein genau definierter Bereich am Anfang des Bildes der Daten desselbige enthält. Das Auslesen geschieht mittels der zwei Unterprogramme 'BiasSearch' und 'CenterSearch'. Die beiden Programme erwarten einen Pointer auf einen String der die Headerinformationen enthält. Dieser Pointer heißt 'PFITSHeader' und wird beim erstmaligen Start des Plugins von der Hauptfunktion 'pi_activate' zur Verfügung gestellt.

Das Bias (oder Offset) der Bildinformation wird wie folgt ermittelt:

```
int __fastcall TForm1::BiasSearch( char *PFITSHeader )
{
    return StrToIntDef( MidStr(PFITSHeader,AnsiPos("BIAS", PFITSHeader)+10,20),0 );
}
```

Es wird die Position im Textstring ermittelt an der das Wort 'BIAS' steht. Dann werden ab dieser Position 10 Zeichen hinzugezählt und danach 20 Zeichen eingelesen. Aus diesen 20 Zeichen wird versucht einen Integer-Wert zu extrahieren. Ist dieser Versuch erfolglos gibt das Unterprogramm Null zurück, ansonsten wird das korrekt ausgelesene Bias ausgegeben.

Die CenterWaveLenght kann zur 'Standardkalibrierung' herangezogen werden. Sie definiert diejenige Wellenlänge - z.B. 500nm - die mit einem bestimmten Pixel auf der horizontalen Achse im FITS-Image korrespondiert.

```
int __fastcall TForm1::CenterSearch( char *PFITSHeader )
{
    int Position;
    Position = AnsiPos( "TELESCOP", PFITSHeader );
    for( int i = 0; i < 35; i++ )
    {
        if( StrToIntDef( MidStr(PFITSHeader,Position+i,1),0) != 0 )
        {
            return StrToIntDef( MidStr(PFITSHeader,Position+i,3),0 );
        }
    }
    return 0;
}
```

Zunächst wird wieder die Position eines bestimmten Wortes, nämlich 'TELESCOP' ermittelt. Ab dieser Position wird zeichenweise 35 Zeichen nach rechts gewandert und jedesmal geprüft ob die Funktion 'StrToIntDef' das Zeichen in eine Zahl umwandeln kann. Ist das der Fall, werden ab dieser Position drei Zeichen - die Centerwavelenght - eingelesen und auskodiert. Dieser Umweg ist nötig, weil die Position des Wertes veränderlich ist und 'StrToIntDef' nicht funktioniert wenn Zahlen und Buchstaben im String vermischt vorkommen. Hier möchte der Autor auf eine mögliche Fehlerquelle aufmerksam machen. Wenn der String nach dem Wort 'TELESCOP' schon vor dem Wellenlängenwert Zahlen enthält, werden schon ab dieser Position drei Zeichen ausgelesen.

2.10 Der Kalibrier-Button

Wenn die Statusanzeige der Kalibrierung grün oder zumindest gelb ist kann die Kalibrierung übernommen werden. Dazu klickt man auf den Button 'Calibrate'. Der Quellcode hinter diesem Button ist im Folgenden zu sehen:

```
void __fastcall TForm1::ButtonCalibrateClick(TObject *Sender)
{
    if ( Form1->statusword[3] == 1)
    {
        Application->MessageBoxA("Calibration already performed.
            Use Reset to recalibrate","info",MB_OK);
        return;
    }
    else
    {
        if( calibrations == 0 || calibrations == 1)
        {
            Application->MessageBoxA("More information needed.
                Assign at least one more wavelength.","Warning",MB_OK);
            return;
        }
        Form1->StaticTextMaximum->Visible = false;
        Form1->TabSheetPicView->TabVisible = true;
        Form1->statusword[3] = 1;
        Form1->PageControl1->ActivePageIndex = 1;
        Form1->RadioButton1->Enabled = false;
        Form1->ScrollBar2->Enabled = false;
        Form1->Memo3->Visible = true;
        Form1->Image1->Enabled = false;
        Form1->Calibrate();
        if( calibrations == 2)
        {
            Application->MessageBoxA("Warning! Only 2-point
                calibration performed!","Warning",MB_OK);
            Form1->History("2-point calibrated");
        }
        if( calibrations > 2)
        {
            Application->MessageBoxA("Calibration successful.", "Info", MB_OK);
            Form1->History(IntToStr(calibrations)+"-point calibrated");
        }
        Form1->LabelStatus->Visible = false;
    }
}
```

Immer wenn der Button betätigt wird erfolgt über eine Abfrage des Kalibrierstatus. Ist der Wert im 'statusword[3]' auf true dann ist die Kalibrierung zu einem früheren Zeitpunkt bereits vorgenommen worden und es erscheint eine MessageBox. Ansonsten wird nach der Anzahl der bisherigen Kalibrierungen gefragt. Ist diese kleiner als zwei wird ebenfalls eine Fehlermeldung ausgegeben. Nur im Fall von 'calibrate > 1' kann das eigentliche Unterprogramm 'Calibrate()', welches die Berechnung durchführt, gestartet werden. Wenn nur zwei Stützstellen benutzt wurden warnt das Programm vor etwaigen Ungenauigkeiten. Nicht mehr benötigte Objekte werden inaktiv geschalten sowie das Bit im 'statusword[3]' gesetzt.

2.11 Die Funktion Edit1KeyPress

Diese Funktion übernimmt eine sehr wichtige Rolle während des Kalibrierens: Ein Edit1 ist ein Eingabefeld, in unserem Falle eines in das der User die Wellenlängen eingibt um sie einem Maximum zuzuordnen. Eine Abfrage am Anfang verhindert, dass man Unsinn wie zwei Kommapunkte, Buchstaben oder dergleichen eingeben kann, was das Programm destabilisieren könnte. Danach wird der Key 13 abgefragt, die Enter-Taste. Wird diese gedrückt, so setzt sich zuerst ein Abfrageapparat in Bewegung, der checkt ob die Wellenlänge ausserhalb der Spezifikationen des Spektrographen liegen oder nicht, er überprüft ob die Randbedingungen verletzt wurden oder nicht (z.B. wenn zwischen zwei Stützstellen eine Wellenlänge platziert werden soll die dort nicht sein dürfte, weil die Wellenlängen am Rand dies verbieten). Hat die eingegebene Wellenlänge dieses Martyrium

überstanden, so wird aus einem Labelarray mylabel[50] ein Label zur Laufzeit generiert. Dieses Label wird mit dem Inhalt von Edit1 verziert, wird entsprechend eingerichtet so dass es am richtigen Ort steht und bekommt einen globalen Positionsmarker, so dass beim Scrollen die Position des Labels jederzeit aktualisiert werden kann. Die Variable calibrations wird am Ende dieser Funktion um 1 erhöht, falls der Kalibrierpunkt erfolgreich übernommen wurde. Außerdem wird die History mit einem entsprechenden Eintrag versehen, der über die globale Position der Stützstelle und die zugeordnete Wellenlänge berichtet.

```
void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
{
   Key = (((Key >= '0') && (Key <='9')) || (Key ==8) || (Key == 13)
   || (Key =='.') ) ? Key : 0;     // 0 - 9, BackSpace, Enter, "."
   if( Key == '.' ) DotCounter++;
    if( Key == 13 )
    {

     if( DotCounter >= 2 )    // no input like 3..567
     {
       Edit1->Text = "";
       DotCounter = 0;
       return;
     }
     DotCounter = 0;
     calpoint[calibrations] = xMaximum + Form1->ScrollBar1->Position;
     WaveLength[calibrations] = atof( Edit1->Text.c_str() );
     if (calibrations > 0)
     {
     for ( int iii = 1; iii <= calibrations; iii++)
      {
      if ((( calpoint[calibrations]  < calpoint[calibrations - iii])
      && ( WaveLength[calibrations] >= WaveLength[calibrations - iii]))
      ||  (( calpoint[calibrations]  > calpoint[calibrations - iii])
      && ( WaveLength[calibrations] <= WaveLength[calibrations - iii])))
         {
         Form1->History("Boundary conditions violated");
         Edit1->Text = "";
         return;
         }
      }
     }

     mylabel[calibrations] = new TLabel(this);
     mylabel[calibrations]->Parent = TabSheetCalibrate;
     if ( (WaveLength[calibrations] < 300.0) || (WaveLength[calibrations] > 755.0))
     {
       Form1->History("Value out of range");
       Edit1->Text = "";
       return;
      }

     mylabel[calibrations]->Caption = WaveLength[calibrations];
     if ( calibrations == 1 )
     {
     Form1->ImageProgress->Picture->LoadFromFile(ExtractFilePath(Application->ExeName)
     + "\\Pics\\ProgressYellow.bmp");
     }
     if ( calibrations == 2 )
     {
     Form1->ImageProgress->Picture->LoadFromFile(ExtractFilePath(Application->ExeName)
      + "\\Pics\\ProgressGreen.bmp");
     }

     mylabel[calibrations]->Left = xMaximum;
     mylabel[calibrations]->Top = yCorner2 + 30;
     mylabel[calibrations]->Visible = true;
     mylabel[calibrations]->BringToFront();
     Form1->History("Calpoint " + IntToStr(calibrations + 1)
     + " at " + mylabel[calibrations]->Caption + "nm" );
```

```
if ( calibrations > 0)
{
 Form1->History("Now able to calibrate");
}
WaveLength[calibrations] = atof( Edit1->Text.c_str() );
calibrations++;
if ( cwlclicked == 0)
{
 Form1->RadioButton1->Enabled = true;
}
Edit1->Visible = false;
}
```

2.12 Die Scrollbars

Die Scrollbar dient in unserem Falle folgendem Zweck: Das Fenster, in welchem der Graph (bei der Kalibrierung und bei der Analyse) gezeichnet wird, hat eine feste Größe. Das Image selbst aber kann sehr viel größer werden, da die Breite ja von dem selektierten Bereich abhängt. Gelöst wurde dieses Problem so, dass ein weiteres, verborgenes Image existiert, welches den eigentlichen Graphen beinhaltet. Ist der gewählte Bereich größer als das Fenster breit, so wird die ScrollBar aktiviert. Sie bewegt einen "virtuellen Rahmen" über das verborgene Image. Der Inhalt dieses Rahmens, der genauso breit ist wie das Fenster, wird ständig in das Anzeigefenster kopiert, so dass der Eindruck entsteht man würde durch das Image scrollen (in gewisser Weise tut man das ja auch).
Außerdem wird überprüft, ob die Label an einer Position stehen die sichtbar ist oder nicht. Entsprechend werden die Eigenschaften der Labels (visible = true oder false) gesetzt.

```
void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
 const TRect MySourceRect = Rect( Form1->ScrollBar1->Position, 0,
 Form1->ScrollBar1->Position + Form1->Image1->Width , Form1->Image1->Height );
 const TRect MyDestRect = Rect( 0, 0, Form1->Image1->Width, Form1->Image1->Height );
 Form1->Image1->Canvas->Brush->Color=clWhite;
 Form1->Image1->Canvas->FillRect(TRect(0,0,Form1->Image1->Width,Form1->Image1->Height));
 Form1->Image1->Canvas->CopyRect( MyDestRect,Form1->Image3->Canvas, MySourceRect );

 for( int x=0;x<calibrations;x++ )
 {
  mylabel[x]->Left =   yline[x] - Form1->ScrollBar1->Position;
  if ( mylabel[x]->Left <= Form1->Image1->Left + Form1->Image1->Width)
  {
   if ( mylabel[x]->Left >= Form1->Image1->Left) mylabel[x]->Visible = true;
  }
  else mylabel[x]->Visible = false;
 }
}
//---------------------------------------------------------------------------
```

2.13 Die History

Die History ist ein Feature, welches zu implementieren mir besonders am Herzen lag. Unelegant zeigt sie dass sie aus zwei reduntanten Memo-Boxen besteht, die aus "gewachsenen" Gründen existieren. Es wäre am Ende zu Umständlich gewesen den Code und die Boxen auf eine zu reduzieren. Dennoch ist die History-Funktion eine sehr wichtige: wann immer ein lästiges Debuggen notwendig ist eignet sich die History hervorragend um Variableninhalte zu visualisieren, etc. Außerdem besitzt sie einen Datums und Zeitstempel und klärt den User über den Status bzw. den Zustand der Analyse auf.

```
void __fastcall TForm1::History(AnsiString text)
{
 Form1->Memo1->Text = Form1->Memo1->Text + DateTimeToStr(Date()) + " "
 + Time() + "  " +  text + "\r\n";
 Form1->Memo2->Text = Form1->Memo2->Text + DateTimeToStr(Date()) + " "
 + Time() + "  " +  text + "\r\n";
}
//---------------------------------------------------------------------------
```

2.14 Radio-Button CWL as Calpoint

Der Radio Button Use CWL as Calpoint gibt wie ja bereits erwähnt dem User die Möglichkeit die Zentralwellenlänge (am 800 px) als Stützstelle zu verwenden. Im Prinzip benimmt sich diese Funktion sehr ähnlich wie Edit1KeyPress, nur dass hier eben keine Werteingabe per Hand erfolgt, sondern dass der Wert aus dem Header übernommen wurde. Es werden dieselben globalen Variablen manipuliert, aus diesem Grund ist der Radio Button auch inaktiv wenn gerade ein Edit1 geöffnet ist. Es bestünde die Gefahr dass es zu undefinierten Zuständen käme, da ja beide Funktionen mit den gleichen Variablen arbeiten. Ist der Radio Button einmal gedrückt worden, und wurde die Stützstelle akzeptiert, so ist er für die laufende Analyse nicht mehr verfügbar.

```
void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
        calpoint[calibrations] = 800 - Form1->globalx1;
        WaveLength[calibrations] = atof( Edit2->Text.c_str() );
        if (calibrations > 0)
        {
                for ( int ii = 1; ii <= calibrations; ii++)
                {
                 if (( (calpoint[calibrations] < calpoint[calibrations - ii])
       && ( WaveLength[calibrations] >= WaveLength[calibrations - ii]))
                 ||  ( (calpoint[calibrations] > calpoint[calibrations - ii])
       && ( WaveLength[calibrations] <= WaveLength[calibrations - ii])))
                 {
                  Form1->History("Boundary conditions violated");
                  Form1->RadioButton1->Checked = false;
                  return;
                 }
                 }
        }

        mylabel[calibrations] = new TLabel(this);
        mylabel[calibrations]->Parent = TabSheetCalibrate;
        mylabel[calibrations]->Caption = WaveLength[calibrations];
        mylabel[calibrations]->Left = 800 - Form1->globalx1 - Form1->ScrollBar1->Position;
        mylabel[calibrations]->Top = yCorner2 + 30;


        if ( (800 - ScrollBar1->Position) > 0 )      //zur Zeit sichtbar
        {
                mylabel[calibrations]->Visible = true;
                mylabel[calibrations]->BringToFront();
        }
        else //zur zeit nicht sichtbar
        {
        }

        if ( calibrations > 0) Form1->History("Now able to calibrate");

        xMaximum = 800 - Form1->globalx1;
        yline[calibrations] = xMaximum;
        RadioButton1->Enabled = false;

        if ( calibrations == 1 )
        {
        Form1->ImageProgress->Picture->LoadFromFile(ExtractFilePath(Application->ExeName)
    + "\\Pics\\ProgressYellow.bmp");
        }
        if ( calibrations == 2 )
        {
        Form1->ImageProgress->Picture->LoadFromFile(ExtractFilePath(Application->ExeName)
    + "\\Pics\\ProgressGreen.bmp");
        }
        Form1->History("Using CWL as calpoint " + IntToStr(calibrations + 1));
        Form1->History("Calpoint " + IntToStr(calibrations + 1) + " at "
    + Form1->Edit2->Text + "nm" );
        calibrations++;
        cwlclicked = 1;
```

```
}
//---------------------------------------------------------------------------
```

2.15 Die Kalibrierung

Diese Funktion berechnet aus den gegebenen Daten die Kalibrierung, dh. sie erstellt ein globales Array, das jeder globalen Pixelposition eine Wellenlänge zuordnet. Dazu werden eine Menge for-Schleifen benötigt. Zuerst einmal fährt eine for-Schleife die globale Achse ab und zählt alle Einträge. Sie merkt sich die Position der Stützstellen, und auch deren Wellenlängen-Werte. Danach werden die Inkremente berechnet. Dabei wird so vorgegangen, dass immer n-1 Inkremente zu n Stützstellen berechnet werden. Dh. dass z.B. zwei Stützstellen ein Inkrement implizieren, usf. Stehen die Inkremente fest, so werden die Randbereiche links und rechts aufgefüllt. Dabei wird jeweils das Inkrement benutzt, dass die äußersten Stützstellen produziert haben. Danach wird der Rest befüllt. All dies geschieht durch lineare Interpolation, dh. es wird nur linear genähert.

```
void __fastcall TForm1::Calibrate(void)
{
valuecounter = 0;
for ( int l = 0; l < 50; l++)
 {
   increment[l]  = 0;
   valueposition[l] = 0;
 }

for( int i = 0; i < calibrations; i++)
//Befuellen des Arrays an den bekannten Stellen
 {
   int sum = globalx1 + calpoint[i];
   xaxis[sum] = WaveLength[i];
 }

for ( int ii = 0; ii < 1600; ii++)
//Pruefen wieviele Werte gesetzt wurden und welche die aeusseren sind
 {
   if ( xaxis[ii] != 0)
   {
   valueposition[valuecounter] = ii;
   Form1->History("CalPoint-Pos: " + IntToStr(ii)
   + ", " + FloatToStr(xaxis[ii])
   + "nm");
   //Ausgabe fuer Ueberpruefung der Werte
   valuecounter++;
   }
  }

for ( int iii = 0; iii < (valuecounter - 1); iii++)   //Berechnet die Incremente
 {
   increment[iii] = (xaxis[valueposition[iii + 1]]
   - xaxis[valueposition[iii]])/(valueposition[iii + 1]
   - valueposition[iii]);
   Form1->History("Increment "
   + IntToStr(iii + 1) + " = "
   + FloatToStr(increment[iii]).sprintf("%.2f",increment[iii]));
  //Ausgabe fuer Ueberpruefung der Werte
 }

for ( int v = valueposition[0]; v > 0; v--)
//linke aeussere Werte einfuellen
 {
   xaxis[v-1]= xaxis[v] - increment[0]
  //Form1->History("Value at "
  + IntToStr(valueposition[0]/2) + ": "
  + FloatToStr(xaxis[valueposition[0]/2]));
  //Ausgabe fuer Ueberpruefung der Werte

for ( int vi = valueposition[valuecounter - 1]; vi < Form1->calimagewidth; vi++)
 //rechte aeussere Werte einfuellen
```

```
 {
   xaxis[vi+1] = xaxis[vi] + increment[0];
 }
  //Form1->History("Value at "
  + IntToStr(valueposition[valuecounter - 1]
  +(Form1->calimagewidth
  - valueposition[valuecounter - 1])/2)
  +": " + FloatToStr(xaxis[valueposition[valuecounter - 1]
  +(Form1->calimagewidth - valueposition[valuecounter - 1])/2]));
 //Ausgabe fuer Ueberpruefung der Werte

for( int vii = 0; vii < valuecounter - 1; vii++)
//auffuellen der zwischenraeume
 {
   for ( int viii = valueposition[vii]; viii < valueposition[vii + 1]; viii++)
   {
    xaxis[viii+1] = xaxis[viii]
    + increment[vii];
   }
   //Form1->History("Value at: "
   + FloatToStr(valueposition[vii]
   + (valueposition[vii+1]
   - valueposition[vii])/2 )
   + " = " +  FloatToStr(xaxis[valueposition[vii]
   + (valueposition[vii+1] - valueposition[vii])/2 ]));
   //Ausgabe fuer Ueberpruefung der Werte
 }

 Form1->History("Wavelength-array generated");
}
//---------------------------------------------------------------------------
```

3 Quellenangaben

[1] **http://www.msb-astroart.com**
[2] **http://www.sbig.com/sbwhtmls/spectrometer2.htm**
[3] **http://www.astrosurf.com/vdesnoux/index.html**
[4] **http://fits.gsfc.nasa.gov**

4 Appendix A: Sourcecode `pispectrum.cpp`

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include <StrUtils.hpp>

//---------------------------------------------------------------------------
USEFORM("Unit1.cpp", Form1);
//---------------------------------------------------------------------------
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{
    return 1;
}
//---------------------------------------------------------------------------

enum Tcommand {ac_getbuffer, ac_getsize, ac_redraw, ac_close, ac_savefits,
               ac_copy, ac_writeheader, ac_readheader, ac_prepareundo,
               ac_thresholds, ac_transferfunc, ac_getselpoint, ac_getselpoint2,
               ac_gethwnd, ac_create, ac_open, ac_getimage,
               ac_getversion, ac_rename, ac_getname, ac_getra,
               ac_getdec,ac_getx, ac_gety, ac_modified};

/* Commands: see docs

NAME              PARAMS TYPE                       RESULT

ac_getbuffer     (pimage, &buffer, colorplane)     0,2
ac_getsize       (pimage, &x, &y)                  0,1
ac_redraw        (pimage, calcstats, autovis)      0,1
ac_close         (pimage, askuser, NULL)           0,1
ac_savefits      (pimage, pfilename, NULL)         0,1
ac_copy          (pimage, NULL, NULL)              0,1
ac_writeheader   (pimage, pheader, NULL)           0,1
ac_readheader    (pimage, NULL, NULL)              pheader
ac_prepareundo   (pimage, NULL, NULL)              0,1
ac_thresholds    (pimage, min, max)                0,1
ac_transferfunc  (pimage, trf, NULL)               0,1
ac_getselpoint   (pimage, &x, &y)                  0,1
ag_getselpoint2  (pimage, &x, &y)                  0,1
ac_gethwnd       (pimage, &HWND, NULL)             0,1
ac_create        (x,y,colored)                     pimage
ac_open          (pfilename, NULL, NULL)           pimage
ac_getimage      (selection, NULL, NULL)           pimage

ac_getversion    (NULL, NULL, NULL)                300+
ac_rename        (pimage, name, path)              0,1
ac_getname       (pimage, keeppath, NULL)          pname
ac_getra         (pimage, x*100, y*100)            ra*100000
ac_getdec        (pimage, x*100, y*100)            dec*100000
ac_getx          (pimage, ra*1000000, dec*1000000) x*100
ac_gety          (pimage, ra*1000000, dec*1000000) y*100
ac_modified      (pimage, modified, nil)           0,1

////////////////////////////////////////////////////////////////////////////
//                                                                        //
//              statusword[10] ist ein boolean array der Klasse Form1     //
//      Das statusword gibt Auskunft ueber den momentanen Zustand des     //
//      Plugins. Dabei sieht die Aufteilung so aus:                       //
//                                                                        //
//          region of                                                     //
//          FITS image  calibration                                       //
//            selected    successful                                      //
//               |          |          |          |          |           //
//               |          |          |          |          |           //
//       ------------------------------------------------------------     //
```

```
//      |     |     |     |     |     |     |     |     |     |     |  //
//      |  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |  //
//      |     |     |     |     |     |     |     |     |     |     |  //
//       ------------------------------------------------------------   //
//         |           |           |           |           |           //
//         |           |           |           |           |           //
//  calibration    cal-graph                                           //
//  image           drawn                                              //
//  selected                                                           //
////////////////////////////////////////////////////////////////////////

*/


typedef int WINAPI(*Tfcallback) (Tcommand, void*, void*, void*);

Tfcallback fcallback;

int PACTIVEIMAGE = 4096;
int PSELECTIMAGE = 4097;
int PEQU_STD = 1000;
int PEQU_GAU = 1001;
int PCGREEN = 0;
int PCRED = 2;
int PCBLUE = 3;

//---------------------------------------------------------------------------
// global variables
//---------------------------------------------------------------------------
float *TempBuffer, *ImageBuffer;
int maxx, maxy, intensmax;
TRect MySourceRect;
TRect MyDestRect;
int gx1, gx2;


//---------------------------------------------------------------------------
// utility functions
//---------------------------------------------------------------------------
unsigned short getPixel(int x, int y)
{
        if ( x < 0 ) x = 0;
        if ( y < 0 ) y = 0;
        if ( x >= maxx ) x = maxx - 1;
        if ( y >= maxy ) y = maxy - 1;
        return TempBuffer[x+y*maxx];
}

void Draw_Spektrum( int x1, int x2, int y1, int y2, int Bias )
{
        int xAxesInc;
        double Intensity;

        if( Form1->statusword[3] == true )
    //bereits kalibriert -> zeichne nun das Image
        {
                for( int i = 0; i < 1600; i++ )
                {
                        Form1->intensityout[i] = 0;
                        Form1->wavelengthout[i] = 0;
                }

                Form1->Image2->Canvas->Pen->Color = clRed;
                Form1->Image2->Width = 350;
                Form1->Image4->Width = 1600;
                Form1->imagewidth = abs(x2 - x1);
                Form1->Image4->Height = 320;
```

```
                Form1->Image4->Canvas->Brush->Color=clWhite;
    // Image soll jedesmal gelscht werden, bevor ein neues gezeichnet wird
                Form1->Image4->Canvas->
         FillRect( TRect(0,0,1600,Form1->Image4->Height) );

                xAxesInc = 0;
                for( int x = x1; x <= x2; x++ )
 //die erste Schleife berechnet den hoechsten mittelwert fuer die normierung
                {
                        for( int y = y1; y <= y2; y++ )
                        {
                                Intensity = Intensity + ( getPixel( x,y ) - Bias );
                        }

                        Intensity = Intensity /( y2 - y1 );
                        if ( Intensity > intensmax) intensmax = Intensity;
                }

                Form1->Image4->Canvas->Pen->Color = clGray;
                Form1->Image4->Canvas->Pen->Style = psDashDotDot;
                Form1->Image4->Canvas->MoveTo( 0,20 );  // 320 - 300
                Form1->Image4->Canvas->LineTo( Form1->Image4->Width,20 );
                Form1->StaticTextMaximum->Visible = true;
                Form1->Image4->Canvas->Pen->Color = clRed;
                Form1->Image4->Canvas->Pen->Style = psSolid;

                for( int x = x1; x <= x2; x++ )
                {
                  for( int y = y1; y <= y2; y++ )
                  {
                   Intensity = Intensity + ( getPixel( x,y ) - Bias );

    // getPixel gives a value ranging from 0 - 65535.
                  }

         Form1->intensityout[x - x1] =
     Intensity / ( (y2 - y1)*(float)intensmax );
//zuordnung der Intensitaet ohne Quanteneffizienz
         Intensity = Intensity /( y2 - y1 )
     *( 300.0 / (float) intensmax );
//normierte Intesitaet, bias abgezogen
         if( x == x1 ) Form1->Image4->Canvas->
     MoveTo( 0, 320.0 - Intensity );

         Form1->Image4->Canvas->
     LineTo( xAxesInc, 320.0 - Intensity );
         xAxesInc = xAxesInc++;
         }

         for( int i = 0; i <= Form1->imagewidth; i++)
         {
         Form1->wavelengthout[i] = Form1->xaxis[i + x1] + 100;
 // Wellenlnge wird in nm ausgegeben
         }
         Form1->History("BIAS subtracted");
         Form1->History("Intensityarray generated");
         MySourceRect = Rect( 0, 0, Form1->Image2->Width, Form1->Image4->Height);
         MyDestRect = Rect( 0, 0, Form1->Image2->Width, Form1->Image2->Height );
         Form1->Image2->Canvas->
     CopyRect( MyDestRect,Form1->Image4->Canvas, MySourceRect );
         }
         else
         {       //noch nicht kalibriert -> zeichnet das Kalibrierspektrum
            if ( Form1->statusword[2] == false )
            {
               if ( (gx1 < 800) && (gx2 > 800) )
```

```
                {
                 Form1->Image3->Canvas->Pen->Color = clBlue;
                 Form1->Image3->Canvas->MoveTo(800 - gx1,0);
                 Form1->Image3->Canvas->LineTo(800 - gx1,Form1->Image3->Height);
                }
                intensmax = 0;

              for( int x = x1; x <= x2; x++ )
//die erste Schleife berechnet den hoechsten mittelwert fuer die normierung
              {
                for( int y = y1; y <= y2; y++ )
                {
                 Intensity = Intensity
        + ( getPixel( x,y ) - Bias );
                }

                Intensity = Intensity /( y2 - y1);
                if ( Intensity > intensmax) intensmax = Intensity;
                 }

                Form1->Image3->Canvas->Pen->Color = clGray;
                Form1->Image3->Canvas->Pen->Style = psDashDotDot;
                Form1->Image3->Canvas->MoveTo( 0,20 );  // 320 - 300
                Form1->Image3->Canvas->LineTo( Form1->Image3->Width,20 );
                Form1->StaticTextMaximum->Visible = true;

                Form1->Image1->Canvas->Pen->Color = clRed;
                Form1->Image3->Canvas->Pen->Color = clRed;
                Form1->Image3->Canvas->Pen->Style = psSolid;
                Form1->Image3->Width = abs(x2 - x1);
                Form1->Image3->Height = 320;
                xAxesInc = 0;

                for( int x = x1; x <= x2; x++ )
                {
                 for( int y = y1; y <= y2; y++ )
                 {
                  Intensity = Intensity + ( getPixel( x,y ) - Bias );
        // getPixel gives a value ranging from 0 - 65535.
                 }

                Intensity = Intensity /( y2 - y1)*( 300.0 / (float)intensmax );

                if( x == x1 ) Form1->Image3->Canvas->MoveTo( 0   , 320.0 - Intensity );

                 Form1->Image3->Canvas->LineTo( xAxesInc, 320.0 - Intensity );
                 xAxesInc = xAxesInc ++;
                 }
                MySourceRect = Rect( 0, 0, Form1->Image1->Width, Form1->Image3->Height );
                MyDestRect = Rect( 0, 0, Form1->Image1->Width, Form1->Image1->Height );
                Form1->Image1->Canvas->
          CopyRect( MyDestRect,Form1->Image3->Canvas, MySourceRect );
                Form1->statusword[2] = 1;
               }
               else
               {
                Application->MessageBox("Calibrationgraph already drawn.
         Use button ''Reset'' to recalibrate.","Info",MB_OK);
               }
         }
}

//---------------------------------------------------------------------------
// AstroArt functions
//---------------------------------------------------------------------------
extern "C" __declspec(dllexport)
int WINAPI pi_initialize(Tfcallback funcaddress, char *menuname,
```

```
char *description, HWND whandle)
{
        strcpy(menuname,"Spectrum");
        strcpy(description,"FITS Image Processing");
        fcallback = funcaddress;
        Form1 = NULL;
        return 1;
}

extern "C" __declspec(dllexport)
void WINAPI pi_activate(void *pimage, HWND whandle)
{
        int x1, x2, y1, y2, x1_old, Bias, center;
        char *PFITSHeader;

        x1 = x2 = y1 = y2 = maxx = maxy = 0;
        TempBuffer = 0;
        fcallback( ac_getselpoint ,pimage,&x1,&y1 );
        fcallback( ac_getselpoint2 ,pimage,&x2,&y2 );
        if ( x1 > 1600 || x2 > 1600 || y1 > 1200 || y2 > 1200 ) return;
        gx1 = x1;
        gx2 = x2;

        PFITSHeader = (char*) fcallback( ac_readheader ,pimage,NULL,NULL );
        Bias = Form1->BiasSearch(PFITSHeader);
        center  = Form1->CenterSearch(PFITSHeader);

        if (Form1 == NULL) Form1 = new TForm1(Application);

        if ( pimage == NULL )
        {
                Application->MessageBox("No image loaded yet.
 Select image for calibration.", "No image for calibration loaded yet.", MB_OK);
                Form1->OpenDialog->Title = "Open Calibration Image...";
                Form1->OpenDialog->Filter = "FITS Image (*.fit; *.fts; *.fits)|
        *.fit;*.fts;*.fits|All files (*.*)|*.*";
                Form1->OpenDialog->FilterIndex = 1;

                if( Form1->OpenDialog->Execute() )
                {
                        pimage = (void*) fcallback( ac_open, Form1->OpenDialog->
            FileName.c_str() ,NULL,NULL );
                        Form1->statusword[0] = 1;
                }
                return;
        }
        else
        {

                if( Form1->statusword[0] == 0)
                {
                        if( IDYES==Application->MessageBox("Do you want to use the
                active image for calibration?", "Calibration", MB_YESNO) )
                        {
                                Form1->statusword[0] = 1;
                        }
                        else
                        {
                                Form1->OpenDialog->Title = "Open Calibration Image...";
                                Form1->OpenDialog->Filter =
"FITS Image (*.fit; *.fts; *.fits)|*.fit;*.fts;*.fits|All files (*.*)|*.*";
                                Form1->OpenDialog->FilterIndex = 1;
                                if( Form1->OpenDialog->Execute() )
                                {
                                        fcallback( ac_close, pimage, (void*)1,NULL );
                                        pimage = (void*) fcallback( ac_open, Form1->
                        OpenDialog->FileName.c_str() ,NULL,NULL );
```

```
                                        Form1->statusword[0]=1;
                                }
                                return;
                        }
                }

        }

        if ( Form1->statusword[3] == false )     //noch nicht kalibriert
        {
                Form1->Image3->Visible = false;
                if( abs(x1 - x2) == 0 )
                {
                        Application->MessageBox("Select region within image for
    calibration and restart the plugin.","Select region",MB_OK);
                        return;
                }
                Form1->statusword[1] = true;
                Form1->Image3->Width = abs(x2 - x1);
                fcallback( ac_getsize,pimage,&maxx,&maxy );
                fcallback( ac_getbuffer,pimage,&ImageBuffer,(void*)PCGREEN );
                Form1->calimagewidth = maxx;
                fcallback(ac_prepareundo,pimage,NULL,NULL);

                TempBuffer = (float*) SysGetMem( maxx*maxy*4 );
// temporary buffer to manipulate the image
                memcpy( TempBuffer,ImageBuffer,maxx*maxy*4 );

                if ( abs(x2 - x1) >= 350)
                {
                        Form1->ScrollBar1->Max = abs(x2 - x1) - 350;
                        Form1->ScrollBar1->Enabled = true;
                }
                else Form1->ScrollBar1->Enabled = false;

                Draw_Spektrum( x1,x2,y1,y2,Bias );
                Form1->TabSheetPicView->TabVisible = false;
                Form1->Image1->BringToFront();
                Form1->Visible = true;
                Form1->globalx1 = gx1;
                Form1->globalx2 = gx2;
                Form1->StaticText3->Parent = Form1->TabSheetCalibrate;
                Form1->StaticText3->Caption = "wavelength [nm] ->";
                Form1->StaticText3->Left = 160;
                Form1->StaticText3->Top = 340;
                Form1->Memo1->Text = "";
                Form1->Memo2->Text = "";
                Form1->StaticText3->Visible = true;
                Form1->StaticText3->BringToFront();
                Form1->History("Plugin started");
                Form1->History("BIAS = " + IntToStr(Bias));
                Form1->History("Center at " + IntToStr(center) + "nm (blue line)");

                if ( (gx1 > 800) || (gx2 < 800)) Form1->History("Center out of focus");

                Form1->Edit2->Text = IntToStr(center);
                Form1->Show();
        }

        if ( Form1->statusword[3] == 1 )   //bereits kalibriert -> zeichne nun das Image
        {
                fcallback( ac_getselpoint,pimage,&x1,&y1 );
                fcallback( ac_getselpoint2,pimage,&x2,&y2 );

                Form1->Memo3->Visible = false;
                Form1->Image4->Visible = false;
                Form1->PageControl1->ActivePage->Enabled = true;
```

```
                if(  abs(x1 - x2) == 0 )
                {
                        Application->MessageBox("Select region within
        image and restart the plugin.","Select region",MB_OK);
                        return;
                }

                if ( x1 > 1600 || x2 > 1600 || y1 > 1200 || y2 > 1200 ) return;

                fcallback( ac_getsize,pimage,&maxx,&maxy );
                if ( maxx == Form1->calimagewidth )
                {
                        fcallback( ac_getbuffer,pimage,&ImageBuffer,(void*)PCGREEN );
                        fcallback( ac_prepareundo,pimage,NULL,NULL );


                        TempBuffer = (float*) SysGetMem( maxx*maxy*4 );
// temporary buffer to manipulate the image
                        memcpy( TempBuffer,ImageBuffer,maxx*maxy*4 );

                        if ( abs(x2 - x1) >= 350)
                        {
                                Form1->ScrollBar2->Max = abs(x2 - x1) - 350;
                                Form1->ScrollBar2->Enabled = true;
                        }
                        else Form1->ScrollBar2->Enabled = false;

                        Draw_Spektrum( x1,x2,y1,y2,Bias );
                        Form1->PageControl1->ActivePageIndex = 1;
                        Form1->Image2->Visible = true;
                        Form1->Visible = true;
                        Form1->StaticText5->Parent = Form1->TabSheetPicView;
                        Form1->StaticText5->Caption = "wavelength [nm] ->";
                        Form1->StaticText5->Left = 160;
                        Form1->StaticText5->Top = 340;
                        Form1->StaticText5->Visible = true;
                        Form1->StaticText5->BringToFront();
                        Form1->Show();
                }
                else Application->MessageBoxA("Image-width doesn't match
width of calibration image! Select another image or recalibrate.", "info", MB_OK);
        }
}

extern "C" __declspec(dllexport)
void WINAPI pi_finalize(HWND whandle)
{
        // optional clean-up
}
```

5 Appendix B: `Unit1.h`

```
//---------------------------------------------------------------------------
#ifndef Unit1H
#define Unit1H
#include <Buttons.hpp>
#include <Classes.hpp>
#include <ComCtrls.hpp>
#include <Controls.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <StdCtrls.hpp>
#include <Graphics.hpp>
#include <StrUtils.hpp>


//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:    // IDE-managed Components
        TPrintDialog *PrintDialog;
        TSaveDialog *SaveDialog;
        TOpenDialog *OpenDialog;
        TImage *Image1;
        TImage *Image2;
        TBitBtn *ButtonPrint;
        TBitBtn *ButtonSave;
        TBitBtn *ButtonReset;
        TStaticText *StaticText1;
        TPageControl *PageControl1;
        TTabSheet *TabSheetCalibrate;
        TTabSheet *TabSheetPicView;
        TEdit *Edit1;
        TBevel *Bevel2;
        TLabel *Label1;
        TLabel *Label2;
        TLabel *Label3;
        TBevel *BevelStatus;
        TLabel *LabelStatus;
        TImage *ImageProgress;
        TEdit *Edit2;
        TScrollBar *ScrollBar1;
        TImage *Image3;
        TStaticText *StaticText3;
        TStaticText *StaticText4;
        TStaticText *StaticText5;
        TBevel *Bevel1;
        TImage *Image4;
        TMemo *Memo3;
        TScrollBar *ScrollBar2;
        TMemo *Memo1;
        TMemo *Memo2;
        TButton *Button2;
        TButton *Button3;
        TStaticText *StaticText6;
        TStaticText *StaticText7;
        TRadioButton *RadioButton1;
        TBitBtn *ButtonCalibrate;
        TStaticText *StaticTextMaximum;
        void __fastcall ButtonPrintClick(TObject *Sender);
        void __fastcall ButtonSaveClick(TObject *Sender);
        void __fastcall ButtonResetClick(TObject *Sender);
        void __fastcall Image1MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y);
        void __fastcall Image1MouseMove(TObject *Sender,
    TShiftState Shift,int X, int Y);
        void __fastcall Image1MouseUp(TObject *Sender,
```

```
    TMouseButton Button, TShiftState Shift, int X, int Y);
        void __fastcall Edit1KeyPress(TObject *Sender, char &Key);
        void __fastcall TabSheetPicViewShow(TObject *Sender);
        void __fastcall TabSheetCalibrateShow(TObject *Sender);
        void __fastcall ButtonCalibrateClick(TObject *Sender);
        void __fastcall ScrollBar1Change(TObject *Sender);
        void __fastcall ScrollBar2Change(TObject *Sender);
        void __fastcFrom1->statusword[2] = 1;
                }all History(AnsiString text);
        void __fastcall Button3Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
         int __fastcall BiasSearch( char *PFITSHeader );
         int __fastcall CenterSearch( char *PFITSHeader );
        void __fastcall RadioButton1Click(TObject *Sender);
        void __fastcall Calibrate(void);
        void __fastcall TabSheetCalibrateHide(TObject *Sender);

private:     // User declarations
public:      // User declarations
    __fastcall TForm1(TComponent* Owner);
    bool statusword[10];
    int calimagewidth;
    int imagewidth;
    double xaxis[1600];
//Wellenlaengenwerte zu jedem Pixel 0...1599 -> Resultat der Kalibrierung
    double yaxis[1600];
//Intensitaetswerte zu jedem Oixel 0...1599 -> Resultat der Intensitaetsberechnung
    double wavelengthout[1600];
//Endgueltiger Bereich des Betrachteten Bildes -> auf File schreiben
    double intensityout[1600];
//Endgueltige y-Werte des auszuwertenden Bildes -> auf File schreiben
    int cwl;
//CenterWaveLength
    int calpoint[50];
//Lokale Position eines Kalibrierpunktes
    int globalx1;
//Globaloffset des Nullpunktes
    int globalx2;
//Globalkoordinate des rechten Randes
    double increment[50];
// Variablen fuer die Kalibrierung
    int valuecounter;
// Variablen fuer die Kalibrierung
    int valueposition[50];
// Variablen fuer die Kalibrierung


};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

6 Appendix C: Unit1.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <math.h>
#include <dstring.h>
#include <stdio.h>
#include <fstream>
#include <iostream.h>
#include <iomanip.h>
#pragma hdrstop

#include "Unit1.h"

//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;

//---------------------------------------------------------------------------
// global variables
//---------------------------------------------------------------------------
bool ButtonPressed;
int calibrations = 0, DotCounter = 0;
int xCorner1, yCorner1, xCorner2, yCorner2, X_old, Y_old,
X_start, Y_start, xMaximum, yMaximum;
double WaveLength[50] = {0};
TLabel *mylabel[50] = {0};
TLabel *scale[50] = {0};
int mylabeldist[50] = {0};
int yline[50] = {0};
bool cwlclicked = 0;

//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{

}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonPrintClick(TObject *Sender)
{
        TPrinter *MyPrinter = Printer();

        int copies;      // number of copies of each page to print each time

        PrintDialog->Options << poPrintToFile;  // PrintToFile CheckBox active
        if ( !PrintDialog->Execute() ) return;     // user did not press OK

        copies = PrintDialog->Copies;

        // Title to appear in the print manager and network print banners
        MyPrinter->Title = "AstroArt 4.0 Spektrum Plugin";

        // Initialize print job
        MyPrinter->BeginDoc();

        TRect MyRect;
        if ( MyPrinter->Orientation == 0 )       // 0 = poPortrait
        {
                MyRect = Rect( MyPrinter->PageWidth/6,MyPrinter->PageHeight/4,
        MyPrinter->PageWidth - MyPrinter->PageWidth/6,
        MyPrinter->PageHeight - MyPrinter->PageHeight/4 );
        }
        if ( MyPrinter->Orientation == 1 )        // 1 = poLandscape
        {
                MyRect = Rect( MyPrinter->PageWidth/6,MyPrinter->PageHeight/6,
```

```
        MyPrinter->PageWidth - MyPrinter->PageWidth/6,
    MyPrinter->PageHeight - MyPrinter->PageHeight/6 );
    }

    for ( int j = 0; j < copies; j++ )
    {
        // print a banner
        MyPrinter->Canvas->Rectangle( 40,40,
    MyPrinter->PageWidth,MyPrinter->PageHeight - 100 );
        MyPrinter->Canvas->MoveTo( 40,150 );
        MyPrinter->Canvas->LineTo( MyPrinter->PageWidth, 150 );

        if( Form1->PageControl1->ActivePageIndex == 0 )
        {
                MyPrinter->Canvas->
    StretchDraw( MyRect,Image3->Picture->Bitmap );
                MyPrinter->Canvas->
    TextOut( 100,80,"AstroArt 4.0 Spektrum Plugin,
      Date: " + Date() + ", Time: " + Time() + ",
     Calibration mode" );
                Form1->History("Printing calibration image");
        }
        if( Form1->PageControl1->ActivePageIndex == 1 )
        {
                MyPrinter->Canvas->
    StretchDraw( MyRect,Image4->Picture->Bitmap );
                MyPrinter->Canvas->
    TextOut( 100,80,"AstroArt 4.0 Spektrum Plugin,
    Date: " + Date() + ", Time: " + Time() + ", View mode" );
                Form1->History("Printing image");
        }
        MyPrinter->Canvas->DrawFocusRect(MyRect);
        MyPrinter->Canvas->TextOut( MyPrinter->PageWidth / 4,
    MyPrinter->PageHeight - MyPrinter->PageHeight/7,
    "Printing the standardized intensity I/Io [ ]
     as a function of the wavelenght" );
    }
    MyPrinter->EndDoc();
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonSaveClick(TObject *Sender)
{
    SaveDialog->Options.Clear();
    SaveDialog->Filter = "Ascii files (*.txt)|*.txt|All files (*.*)|*.*";
    SaveDialog->DefaultExt = NULL;
    if ( SaveDialog->Execute() )
    {
      if ( SaveDialog->FilterIndex == 1 )
         {
          ofstream outfile;

          outfile.setf(ios_base::fixed);

          if ( !SaveDialog->Options.Contains(ofExtensionDifferent) )
      outfile.open( (SaveDialog->FileName + ".txt").c_str(),ios::out );
          else outfile.open( (SaveDialog->FileName).c_str(),ios::out );

          outfile << "-------------------------------------------------------------"
          << endl << endl;
          outfile << " AstroArt 4.0 Plugin" << endl << " Date: "
          << DateToStr( Date() ).c_str() << ", Time: "
          << TimeToStr( Time() ).c_str() << endl << endl << endl;
          outfile << "  Wavelenght [nm ]" << "       "
          << "Intensity I/Imax [ ]" << endl << endl;
                for( int i = 0; i < 1600; i++ )
                {
```

```
                outfile << setw(14)
                << setprecision(2) << Form1->wavelengthout[i]
                << "    " << setw(20) << setprecision(5)
                << Form1->intensityout[i] << endl;
                          }
                outfile.close();
                Form1->History("Saved data in text file");
                return;
                  }
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1MouseDown(TObject *Sender,
TMouseButton Button, TShiftState Shift, int X, int Y)
{
        ButtonPressed = true;

        xCorner1 = X;
        yCorner1 = Y;
        X_start = xCorner1;
        Y_start = yCorner1;
        X_old = xCorner1;
        Y_old = yCorner1;
        Form1->Image1->Canvas->Pen->Mode = pmNotXor;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseMove(
TObject *Sender, TShiftState Shift,int X, int Y)
{
        Form1->LabelStatus->Caption =
    AnsiString(" X: ") + X + AnsiString(",  Y: ") + Y;

        if( ButtonPressed )
        {
                Form1->Image1->Canvas->Pen->Style = psDot;
                Form1->Image1->Canvas->Pen->Color = clBlue;
                Form1->Image1->Canvas->
        Rectangle( X_start,Y_start,X_old,Y_old );
                Form1->Image1->Canvas->
        Rectangle( X_start,Y_start,X,Y );
                X_old = X;
                Y_old = Y;
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1MouseUp(TObject *Sender,
 TMouseButton Button,TShiftState Shift, int X, int Y)
{
        bool OK = false;
        ButtonPressed = false;
        xCorner2 = X;
        yCorner2 = Y;
        yMaximum = yCorner2;

        for( int i = xCorner1; i <= xCorner2; i++ )
        {
          for( int j = yCorner1; j <= yCorner2; j++ )
           {
             if( Form1->Image1->Canvas->Pixels[i][j] == clRed && j < yMaximum )
             {
              xMaximum = i;
              yMaximum = j;
              yCorner2 = yMaximum;
        // Verbessert den Algorithmus.
              OK = true;
```

```
        // Eine Variable die angibt, ob sich
        //im Auswahlbereich berhaupt etwas befindet.
                        }
                }
        }
        Form1->Image1->Canvas->Rectangle( X_start,Y_start,X,Y );
        if( OK )
        {
                Form1->Image1->Canvas->Pen->Color = clBlack;
                Form1->Image3->Canvas->Pen->Color = clBlack;
                Form1->Image3->Canvas->Pen->Style = psDot;
                Form1->Image3->Canvas->Pen->Mode = pmNotXor;
                Form1->Image1->Canvas->
        MoveTo(xMaximum, Image1->Height );
                Form1->Image3->Canvas->
        MoveTo(xMaximum + Form1->ScrollBar1->Position, Image1->Height );
                Form1->Image1->Canvas->
        LineTo(xMaximum, 0 );
                Form1->Image3->Canvas->
        LineTo(xMaximum + Form1->ScrollBar1->Position, 0 );
                yline[calibrations] = xMaximum
        + Form1->Image3->Left
        + Form1->ScrollBar1->Position - Form1->ScrollBar1->Left;
                Edit1->Left = xMaximum;
                Edit1->Top = yCorner2 + 100;
                Edit1->BringToFront();
                Form1->RadioButton1->Enabled = false;
                Edit1->Visible = true;
                Edit1->SetFocus();
        }
        Form1->Image1->Canvas->Pen->Style = psSolid;
        Form1->Image1->Canvas->Pen->Mode = pmNot;
        Form1->Image3->Canvas->Pen->Style = psSolid;
        Form1->Image3->Canvas->Pen->Mode = pmNot;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)
{
        Key = (((Key >= '0') && (Key <='9')) || (Key ==8)
        || (Key == 13) || (Key =='.') ) ? Key : 0;
    // 0 - 9, BackSpace, Enter, "."
        if( Key == '.' ) DotCounter++;
        if( Key == 13 )
        {

         if( DotCounter >= 2 )    // no input like 3..567
         {
         Edit1->Text = "";
         DotCounter = 0;
         return;
         }
         DotCounter = 0;
         calpoint[calibrations] = xMaximum + Form1->ScrollBar1->Position;
         WaveLength[calibrations] = atof( Edit1->Text.c_str() );
         if (calibrations > 0)
          {
          for ( int iii = 1; iii <= calibrations; iii++)
          {
           if ((( calpoint[calibrations]  < calpoint[calibrations - iii])
    && ( WaveLength[calibrations] >= WaveLength[calibrations - iii]))
          ||  (( calpoint[calibrations]  > calpoint[calibrations - iii])
    && ( WaveLength[calibrations] <= WaveLength[calibrations - iii])))
          {
          Form1->History("Boundary conditions violated");
          Edit1->Text = "";
         return;
```

```
            }
            }
            }

            mylabel[calibrations] = new TLabel(this);
            mylabel[calibrations]->Parent = TabSheetCalibrate;
            if ( (WaveLength[calibrations] < 300.0)
        || (WaveLength[calibrations] > 755.0))
            {
            Form1->History("Value out of range");
            Edit1->Text = "";
            return;
            }

            mylabel[calibrations]->Caption = WaveLength[calibrations];
            if ( calibrations == 1 )
            {
            Form1->ImageProgress->Picture->LoadFromFile(
       ExtractFilePath(Application->ExeName) + "\\Pics\\ProgressYellow.bmp");
            }
            if ( calibrations == 2 )
            {
            Form1->ImageProgress->Picture->LoadFromFile(
            ExtractFilePath(Application->ExeName) + "\\Pics\\ProgressGreen.bmp");
            }

            mylabel[calibrations]->Left = xMaximum;
            mylabel[calibrations]->Top = yCorner2 + 30;
            mylabel[calibrations]->Visible = true;
            mylabel[calibrations]->BringToFront();
            Form1->History("Calpoint " + IntToStr(calibrations + 1)
            + " at " + mylabel[calibrations]->Caption + "nm" );
            if ( calibrations > 0)
            {
             Form1->History("Now able to calibrate");
            }
            WaveLength[calibrations] = atof( Edit1->Text.c_str() );
            calibrations++;
            if ( cwlclicked == 0)
            {
            Form1->RadioButton1->Enabled = true;
            }
            Edit1->Visible = false;
          }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonResetClick(TObject *Sender)
{
          for ( int i = 0; i < calibrations; i++)
          {
                  mylabel[i]->Visible = false;
          }
          calibrations = 0;
          Form1->Image1->Picture = NULL;
          Form1->Image3->Picture = NULL;
          Form1->Image2->Picture = NULL;
          Form1->Image4->Picture = NULL;
          Form1->ScrollBar1->Position = 0;
          Form1->ImageProgress->Picture->LoadFromFile(
     ExtractFilePath(Application->ExeName) + "\\Pics\\ProgressRed.bmp");
          Form1->statusword[2] = 0;
          Form1->statusword[3] = 0;
          Form1->TabSheetPicView->TabVisible = false;
          Form1->Image1->Canvas->Brush->Color=clWhite;
          Form1->Image1->Canvas->FillRect(
     TRect(0,0,Form1->Image1->Width,Form1->Image1->Height));
```

```
        Form1->Image1->Enabled = true;
        Form1->RadioButton1->Enabled = true;
        Form1->RadioButton1->Checked = false;
        Form1->LabelStatus->Visible = true;
        Form1->StaticTextMaximum->Visible = false;
        cwlclicked = 0;

        //Nullsetzen aller Variablen:
        for ( int i = 0; i < 1600; i++)
        {
         xaxis[i] = 0;
         yaxis[i] = 0;
         wavelengthout[i] = 0;
         intensityout[i] = 0;
        }

        for ( int ii = 0; ii < 50; ii++)
        {
         calpoint[ii] = 0;
         valueposition[ii] = 0;
         increment[ii] = 0;
        }

        valuecounter = 0;
}

//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheetPicViewShow(TObject *Sender)
{
        Form1->ScrollBar1->Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheetCalibrateHide(TObject *Sender)
{
        Form1->StaticTextMaximum->Visible = false;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::TabSheetCalibrateShow(TObject *Sender)
{
        Form1->ScrollBar1->Visible = true;
        Form1->StaticTextMaximum->Visible = true;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ButtonCalibrateClick(TObject *Sender)
{
        if ( Form1->statusword[3] == 1)
        {
                Application->MessageBoxA("Calibration already performed.
         Use Reset to recalibrate","info",MB_OK);
                return;
        }
        else
        {
                if( calibrations == 0 || calibrations == 1)
                {
                        Application->MessageBoxA("More information needed.
         Assign at least one more wavelength.","Warning",MB_OK);
                        return;
                }
                Form1->StaticTextMaximum->Visible = false;
                Form1->TabSheetPicView->TabVisible = true;
                Form1->statusword[3] = 1;
                Form1->PageControl1->ActivePageIndex = 1;
```

```
                Form1->RadioButton1->Enabled = false;
                Form1->ScrollBar2->Enabled = false;
                Form1->Memo3->Visible = true;
                Form1->Image1->Enabled = false;
                Form1->Calibrate();
                if( calibrations == 2)
                {
                        Application->MessageBoxA("Warning! Only 2-point
            calibration performed!","Warning",MB_OK);
                        Form1->History("2-point calibrated");
                }
                if( calibrations > 2)
                {
                        Application->MessageBoxA("Calibration successful.",
            "Info", MB_OK);
                        Form1->History(IntToStr(calibrations)+"-point calibrated");
                }
                Form1->LabelStatus->Visible = false;
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar1Change(TObject *Sender)
{
        const TRect MySourceRect = Rect( Form1->ScrollBar1->Position,
     0,Form1->ScrollBar1->Position + Form1->Image1->Width ,
     Form1->Image1->Height );
        const TRect MyDestRect = Rect( 0, 0, Form1->Image1->Width,
     Form1->Image1->Height );
        Form1->Image1->Canvas->Brush->Color=clWhite;
        Form1->Image1->Canvas->FillRect(TRect(0,0,Form1->Image1->Width,
    Form1->Image1->Height));
        Form1->Image1->Canvas->CopyRect( MyDestRect,Form1->Image3->Canvas,
     MySourceRect );

        for( int x=0;x<calibrations;x++ )
        {
        mylabel[x]->Left =    yline[x] - Form1->ScrollBar1->Position;
        if ( mylabel[x]->Left <= Form1->Image1->Left + Form1->Image1->Width)
        {
        if ( mylabel[x]->Left >= Form1->Image1->Left) mylabel[x]->Visible = true;
        }
        else mylabel[x]->Visible = false;
        }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ScrollBar2Change(TObject *Sender)
{
        TRect MySourceRect = Rect( Form1->ScrollBar2->Position, 0,
    Form1->ScrollBar2->Position + Form1->Image2->Width , Form1->Image2->Height );
        TRect MyDestRect = Rect( 0, 0, Form1->Image2->Width, Form1->Image2->Height );
        Form1->Image2->Canvas->Brush->Color=clWhite;
        Form1->Image2->Canvas->FillRect(TRect(0,0,
    Form1->Image2->Width,Form1->Image2->Height));
        Form1->Image2->Canvas->CopyRect( MyDestRect,
    Form1->Image4->Canvas, MySourceRect );
}
//---------------------------------------------------------------------------

void __fastcall TForm1::History(AnsiString text)
{
        Form1->Memo1->Text = Form1->Memo1->Text +
    DateTimeToStr(Date()) + " " + Time() + "   " +  text + "\r\n";
        Form1->Memo2->Text = Form1->Memo2->Text +
    DateTimeToStr(Date()) + " " + Time() + "   " +  text + "\r\n";
}
```

```
//---------------------------------------------------------------------------

void __fastcall TForm1::Button3Click(TObject *Sender)
{
        Form1->Memo1->Text = "";
        Form1->Memo2->Text = "";
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Button2Click(TObject *Sender)
{
        Form1->Memo1->Text = "";
        Form1->Memo2->Text = "";
}
//---------------------------------------------------------------------------

int __fastcall TForm1::BiasSearch( char *PFITSHeader )
{
       return StrToIntDef( MidStr(PFITSHeader,
    AnsiPos( "BIAS", PFITSHeader)+10, 20),0 );
}
//---------------------------------------------------------------------------

int __fastcall TForm1::CenterSearch( char *PFITSHeader )
{
        int Position;
        Position = AnsiPos( "TELESCOP", PFITSHeader );
        for( int i = 0; i < 35; i++ )
        {
         if( StrToIntDef( MidStr(PFITSHeader, Position+i, 1), 0) != 0 )
         {
          return StrToIntDef( MidStr(PFITSHeader,Position+i, 3),0 );
         }
        }
       return 0;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
        calpoint[calibrations] = 800 - Form1->globalx1;
        WaveLength[calibrations] = atof( Edit2->Text.c_str() );
        if (calibrations > 0)
        {
          for ( int ii = 1; ii <= calibrations; ii++)
          {
            if (( (calpoint[calibrations] < calpoint[calibrations - ii])
       && ( WaveLength[calibrations] >= WaveLength[calibrations - ii]))
            ||  ( (calpoint[calibrations] > calpoint[calibrations - ii])
       && ( WaveLength[calibrations] <= WaveLength[calibrations - ii])))
            {
             Form1->History("Boundary conditions violated");
             Form1->RadioButton1->Checked = false;
             return;
            }
           }
        }

        mylabel[calibrations] = new TLabel(this);
        mylabel[calibrations]->Parent = TabSheetCalibrate;
        mylabel[calibrations]->Caption = WaveLength[calibrations];
        mylabel[calibrations]->Left = 800 - Form1->globalx1 -
    Form1->ScrollBar1->Position;
        mylabel[calibrations]->Top = yCorner2 + 30;


        if ( (800 - ScrollBar1->Position) > 0 )      //zur Zeit sichtbar
```

```
        {
                mylabel[calibrations]->Visible = true;
                mylabel[calibrations]->BringToFront();
        }
        else //zur zeit nicht sichtbar
        {
        }

        if ( calibrations > 0) Form1->History("Now able to calibrate");

        xMaximum = 800 - Form1->globalx1;
        yline[calibrations] = xMaximum;
        RadioButton1->Enabled = false;

        if ( calibrations == 1 )
        {
        Form1->ImageProgress->Picture->LoadFromFile(
    ExtractFilePath(Application->ExeName) + "\\Pics\\ProgressYellow.bmp");
        }
        if ( calibrations == 2 )
        {
        Form1->ImageProgress->Picture->LoadFromFile(
    ExtractFilePath(Application->ExeName) + "\\Pics\\ProgressGreen.bmp");
        }
        Form1->History("Using CWL as calpoint " + IntToStr(calibrations + 1));
        Form1->History("Calpoint " + IntToStr(calibrations + 1) + " at " +
    Form1->Edit2->Text + "nm" );
        calibrations++;
        cwlclicked = 1;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Calibrate(void)
{
valuecounter = 0;
    for ( int l = 0; l < 50; l++)
    {
        increment[l]  = 0;
        valueposition[l] = 0;
     }

        for( int i = 0; i < calibrations; i++)
    //Befuellen des Arrays an den bekannten Stellen
        {
                int sum = globalx1 + calpoint[i];
                xaxis[sum] = WaveLength[i];
        }

        for ( int ii = 0; ii < 1600; ii++)
       //Pruefen wieviele Werte gesetzt wurden und welche die aeusseren sind
        {
         if ( xaxis[ii] != 0)
         {
          valueposition[valuecounter] = ii;
          Form1->History("CalPoint-Pos: "
    + IntToStr(ii) + ", " + FloatToStr(xaxis[ii])
    + "nm"); //Ausgabe fuer Ueberpruefung der Werte
          valuecounter++;
         }
        }

        for ( int iii = 0; iii < (valuecounter - 1); iii++)
    //Berechnet die Incremente
        {
         increment[iii] = (xaxis[valueposition[iii + 1]]
    - xaxis[valueposition[iii]])/(valueposition[iii + 1]
    - valueposition[iii]);
```

```
        Form1->History("Increment " + IntToStr(iii + 1)
    + " = "
    + FloatToStr(increment[iii]).sprintf("%.2f",increment[iii]));
    //Ausgabe fuer Ueberpruefung der Werte
        }

        for ( int v = valueposition[0]; v > 0; v--)
    //linke aeussere Werte einfuellen
        {
         xaxis[v-1]= xaxis[v] - increment[0];
        }
        //Form1->History("Value at " + IntToStr(valueposition[0]/2)
    + ": " + FloatToStr(xaxis[valueposition[0]/2]));
    //Ausgabe fuer Ueberpruefung der Werte

        for ( int vi = valueposition[valuecounter - 1];
     vi < Form1->calimagewidth; vi++)
    //rechte aeussere Werte einfuellen
        {
        xaxis[vi+1] = xaxis[vi] + increment[0];
        }
        //Form1->History("Value at "
     + IntToStr(valueposition[valuecounter - 1]
     +(Form1->calimagewidth - valueposition[valuecounter - 1])/2)
     +": " + FloatToStr(xaxis[valueposition[valuecounter - 1]
     +(Form1->calimagewidth - valueposition[valuecounter - 1])/2]));
      //Ausgabe fuer Ueberpruefung der Werte

        for( int vii = 0; vii < valuecounter - 1; vii++)
    //auffuellen der zwischenraeume
        {
         for ( int viii = valueposition[vii]; viii < valueposition[vii + 1]; viii++)
          {
          xaxis[viii+1] = xaxis[viii] + increment[vii];
          }
      //Form1->History("Value at: " + FloatToStr(valueposition[vii]
    + (valueposition[vii+1] - valueposition[vii])/2 ) + " = "
    +  FloatToStr(xaxis[valueposition[vii] + (valueposition[vii+1]
    - valueposition[vii])/2 ]));
    //Ausgabe fuer Ueberpruefung der Werte
        }

        Form1->History("Wavelength-array generated");
}
//---------------------------------------------------------------------------
```