Gxsm and Xxsm User/Referenz Manual and Hackerguide

Andreas Klust and Percy Zahl

Insitut für Festkörperphysik
Universität Hannover

May 8, 2002

Vorwort / Preface

Das Jahr 2000 ist vorraussichtlich mein letztes am Institut für Festkörperphysik der Universität Hannover. In diesem Vorwort möchte ich die Historie der ehemals "'nur"' STM-Software festhalten. Ich habe die letzten fünf Jahre, die Zeit vor meiner Diplomarbeit eingeschlossen, dort verbracht und in der Arbeitsgruppe von Michael Horn-von-Hoegen begonnen, mich mit der Tunnelmikroskopie und den zugehörigen Techniken zu beschäftigen. Im Sommer 1995 zeigte mir A.Meier ein kleines, noch unberührtes, unter einer Schutzhaube verborgenes Gerät – ein Micro STM. Wir besaßen nur dieses Gerät zusammen mit einem Tunnelstromverstärker, einer Signalprozessor gesteuerte Meßkarte (PC31) – und – dem Wissen, daß damit ein gewisser G.Meyer in Berlin erfolgreich ein Gerät gleichen Prinzips bedient. Ich machte es mir zur Aufgabe, der ganzen Sache Leben einzuhauchen ...

Von G.Meyer hatte ich unterdessen einige Software zusammengesammelt und wir versuchten, die diversen DOS/Pascal Programme auf unserem hochmodernen P90 zum Laufen zu bringen. Währenddessen vertiefte ich mich in die Geheimnisse der DSP-Programmierung und der Kommunikation zwischen Host-PC und DSP.

Des Fortschritts wegen beschloß ich, ein bereits vorhandenes OS/2-Programm namens PMSTM weiter zu entwickeln. PMSTM basierte auf einer schon erheblich älteren OS/2 Version vonL.Anderson und wurde von H.Bethge zum Messen eingesetzt.

Gleichzeitig produzierte R.Kumpe einige Assemblerroutinen zur Ansteuerung seiner Datatranslation-Karte unter OS/2 – welche ebenfalls einen spezial STM-Eigenbau, jedoch noch mit analogem Regler, bedienen sollte.

Parallel wurden mit H.Pietsch die notwendigen Zusätze für AFM implementiert.

Es verging einige Zeit, aber dann war es soweit: Die neue OS/2 Software konnte das Gerät steuern und Daten aufnehmen und anzeigen. An Luft wurden erste Versuche mit Goldproben und abgeknipster Wolframspitze erfolgreich abgeschlossen. Weitere Verbesserungen in nahezu jeder Hinsicht konnte ich im Verlauf meiner Diplomarbeit einbringen – das Gerät war unterdessen im UHV der Maschine "'Quantum"' im Einsatz.

Die Geschichte des Micro STM´s wurde im weiteren wesentlich von R.Hild bestimmt, der das höchst empfindliche Gerät in einem an Federn aufgehängten Kupferblock mit 3D-Wirbelstromdämpfung versenkte und somit zu Höchtstleistung brachte.

Die Zeit war gegen das wunderbar stabile OS/2, es wurde zum Außenseiter und eigentlich benötigte man es nur noch zum Messen ...
Windows erschien mir als durchaus erfahrenem und gebrantmarktem DOS/Win3.X Programmierer als völlig ungeeinete Platform, so hatte ich doch die Vorzüge einer stabilen Betriebssystemplatform mit OS/2 zu schätzen gelernt. Ich konnte jedoch mit meinen Unix/Linux Grundkenntnissen schnell eine zukunftssichere Alternative finden und entschied, erste Versuchte mit einer neuen Software für mein SPA–LEED zu unternehmen. Das SPA–LEED sollte nämlich ebenfalls mit einer Signalprozessorkarte gesteuert werden, um später ggf. ohne zusätzlichen Aufwandt ein STM nachrüsten zu können.
Das Resultat war ein Programm namens xspa, welches unter Verwendung der xforms Libary unter X11 lief.
Aus diesen Erfahrungen schöpfend entwickelte ich ein völlig neues Konzept für eine grundlegend neue Struktur eines neuen STM-Programmes – der alte Code war zu steif und beinhaltete viel zu viele globale Variablen. Auch das fixe Datenformat dat wurde als historisch abgelegt und es fand ein Übergang zu dem NetCDF-Format statt. Das neue Konzept ist objektorientiert und ermöglicht erstmals eine flexible Mehrkanal- Verwaltung und -Messung (dies wurde in PMSTM nur per Trick in unflexibler Weise zur Datenaufnahme hineigebastelt, da das AFM die gleichzeitige Aufzeichnung von Kraft und Reibungssignal ermöglicht).
Es entstand xxsm, welches strukturell die Grundlage des heutigen gxsm darstellt. Dieses Programm entstand im Verlauf der Diplomarbeit von R.Hild, der die Entwicklung life mit seinem STM verfolgen mußte bzw. durfte :=)
Auch unser neues Spielzeug, ein Luft-AFM, wurde von xxsm gesteuert sowie eine Streulichapparatur (SARLS).
Die Computertechnologie schreitet unaufhaltsam vorann und es ist zu befürchten, daß bald keine ISA-Slots mehr verfügbar sind. So entschieden wir für neue Geräte, eine PCI-Version der DSP-Karte (PCI32) zu kaufen. Diese Karte schien der PC31, mal von dem PCI-Bus abgesehen, doch sehr vergleichbar. Es gab jedoch einige Unwegsamkeiten, die mich einige Nerven kosteten...
Jedenfalls entwickelte ich ein Kernelmodul für diese Karte und später eine Variation für die alte PC31, denn auf User-Space IO konnte wegen PCI Konfiguration, etc. nicht mehr ausgewichen werden. Zusätzlich mußten alle Utilities (Loader, Terminal) an die neue Karte angepasst werden. Eine neue Library machte das anfängliche Chaos komplett – die Hoffnung, das DSP- Programm nicht umschreiben zu müssen, war vergebens. Jedoch konnte mit einigen Tricks weiterhin eine, zwar neue, aber gemeinsame Version erhalten bleiben.
Die alte xforms Libary ist zwar extrem effizient, insbesondere meine sehr schnellen MIT-SHM Bilddarstellungsroutinen, aber die Oberfläche und Menüdarstellung lassen einige Wünsche offen. Eine modernes Toolkit Namens Gtk+/Gnome weckte mein Interesse im Sinne des Fortbestandes und der Weiterentwicklung dieser unterdessen mächtigen Mikroskopie- Software. Ein Kraftakt von diversen Nächten zwischen Juni

und Dezember 1999 brachte xxsm im neuen Gewand als gnomified xxsm - kurz gxsm hervor.
Das objektorientierte Konzept von Gtk+/Gnome vereinfachte es auch erheblich, die SPA–LEED-Ansteuerung in gxsm mitaufzunehmen. Darüberhinaus entstanden gleichzeitig einige Tools wie Gfit, Goszi und dsp-applet.
Dieses Werk soll im weiteren all denjenigen helfen, die einerseits mit gxsm arbeiten, aber auch etwas mehr über die Internas erfahren wollen. So gliedert sich das folgende Manual in

- einen anwendungsbezogenen Teil
- nützliche Tips (HOWTOs zu STM und AFM) und
- einen Versuch das Programm-Konzept zu erläutern.

Ich möchte hier meinen Dank an alle aussprechen, die zu meiner Arbeit beigetragen haben, insbesondere jedoch:
M.Henzler für das immer gute Instituts-Klima und die schönen "'Almen"'.
M.Horn-von-Hoegen dafür, daß er mir die Zeit zum ständigen Arbeiten an diesem Projekt gelassen hat.
G.Meyer für seine Starthilfe bei der DSP Programmierung und diversen Gesprächen.
H.Bethge für Ihre Gedult mit mir im Keller.
R.Kumpe für seine Mitarbeit an PMSTM und Diskussionen.
H.Pietsch für sein Mitwirken an PMSTM.
L.Anderson, den ich leider nie persönlich kennengelernt habe, für seine Arbeit an PMSTM.
U.Köhler und seine Truppe für einige Diskussionen.
A.Meier für seine endlose Gedult mit mir ...
F.J.Meier zu Heringdorf für immerwieder freundlich und fröhliche BS Discussions mit diversen Guinness.
R.Hild und M.Bierkandt für deren Ausdauer mit den ewig neuen Versionen.
A.Klust für alle Beiträge zu diesem Projekt.
H.Goldbach für SPA–LEED Discussions und für den (noch nicht) herausgesuchten BF krams.
Heilo für die Gewährung einer Mehraufwandzulage.

Negenborn Januar 2000 Percy Zahl

. . . Gxsm has been licensed as GPL and goes to SourceForge.net . . .

Es ist Sonntag, der 21.1.2001, es schneit draußen und die Zeit ist mal wieder ein Jahr fortgeschritten; Ich stecke mitten in meinen Vorbeitungen zum Ortswechsel von Negenborn/Hannover nach Denver/Colorado und möchte ein paar Bemerkungen zum Stand des Gxsm-Projekts festhalten, nachdem meine Dissertation mit der Veröffentlichung meines Werkes zum Stress auf Oberflächen abgeschlossen ist.
Das Gxsm-Projekt ist seit Herbst 2000 offiziell als Open-Source auf http://sourceforge.net via Web-Interface und CVS-Access verfügbar. Die Verzeichnishirarchie wurde überarbeitet und "Gxsm" ist nun das Projekthauptverzeichnis.
Im Dezember 2000 wurde ein flexibles Plug-In Interface für Gxsm entworfen und damit begonnen alle speziellen Erweiterungen in Plug-Ins auszulagern. Damit wird eine erhebliche Flexibilität erzielt. Im Januar 2001 sind ein SPA–LEED Simulationsmodul, Peak-Finder, Fokus-Tool und Oszi-Plugin hinzugekommen. Ein Support für die alte Burr-Brown Karte für SPA–LEED ist ebenfalls verfügbar und bereits im Einsatz.

Negenborn Januar 2001 Percy Zahl

. . . Gxsm goes international. . .

Just a few up-to-date remarks:
As far as I know Gxsm is used now in more than four countries around the globe.
Kernel 2.4.x support was implemented and is proofed to work well.
More and more Plug-Ins are added. . .
A Gnome Druid to guide the new Gxsm user along all most important settings was added.

Golden, Colorado USA, August 2001 Percy Zahl

Contents

1 Introduction

Gxsm[1] – Gnome X Scanning Microscopy – is a powerful graphical interface for any kind of 2D and 3D (multilayered 2D mode) data acquisition methods, especially designed for SPM and SPA-LEED, which are used in surface science. It includes methods for 2D data (of various types: byte, short, long, double) visualization and manipulation. We are currently using it for STM, AFM, SARLS, and SPA-LEED. The project was founded at the Institute for Solid State Physics [2] Leading developer of Gxsm is Percy Zahl[3]. The program was developed using the GNU Public License under Linux with X-Window-support.
A example: The STM Image at the right shows a atomically resolved surface of a with about 2 Monolayers of Germanium covered Silicon 100 crystal surface. Brighter mean the "atoms" are at a higher layer. The structure made of overlapping "balls" looking like a mesh represents the local density of states – or just density of the electrons seen by the STM tip at the surface. The maxima can be interpreted as the positions of the atoms of the top layer... Gxsm is powerful, it is easy to extend via plug-ins, it is unlimited in data size... and it is a free and open source software hosted by SourceForge! It is based on the Linux Gtk+/Gnome development environment. And it's free: Gxsm is licensed under the terms of the GNU General Public License (GPL).

1.1 Gxsm at a glance

- Support for STM, AFM, SPA-LEED, SARLS, and any other 2D (2D layered and multi-channel) data-acquisition method can be supported by Gxsm.

- The Gxsm core handles multiple channels of 2D (layered) data fields of arb. type and unlimited size and a grey-scale or false color view of 2D data image, using "on the fly" data transformation as there are:

[1]The Project can be found in the Internet at http://gxsm.sf.net
[2]Institut für Festkörperphysik, Universität Hannover, Appelstraße 2, D-30167 Hannover, Germany
www.fkp.uni-hannover.de
[3]e-mail: zahl@users.sourceforge.net

  - "Quick": a line regression and subtraction is performed on each scan-line
  - "Direct": only contrast and brightness adjustments
  - "Logarith.": logarithmic scaling mode, almost used by diffraction methods
  - "Horizontal": automatic line average subtraction
  - "Diff.": view of X-derivative, $[Z(X+1) - Z(X)]$
  - "Periodic": like "Direct" Mode, but grey-scale is applied modulo #grey-levels
  - More sophisticated background correction and data analysis methods are implemented as filters.

- Data presentation is by default a (grey or false color) image but it can be switchted to a profile view (1d), profile extraction on the fly... Or you can use a 3D shaded view (using MesaGL) which now offers a sophistacated scene setup.

- The "high-level" scan controller is now separated from the Gxsm core and is build as PlugIn, while the real-time "low-level" scanning process, data-acquisition and feedback loop (if needed), runs on the DSP – if present, else a dummy image is produced. The current scan-line, marked in red, can be viewed simultaneously as profile. (View-¿red Profile)

- Extremely flexible configuration of user settings and data acquisition and probe modes.

  - Easy to extend by Plug-ins, some examples of existing Plug-ins:
    * Math operations on 2D data:
    * Background correction methods
    * Image filtering 1D and 2D, including several Fourier transformation methods
    * Image analysis/statistics: histogram, step analysis, ...
    * Geometric transformations: scaling, rotation, affine, ...
    * and more, write and contribute your favorite math Plug-in for Gxsm! Don't be afraid, there is a step by step instruction tutorial and a math Plug-in generator, all you need to do is to add your math code!
  - Special datafile/formats import/export filters, (easy to extend via Plug-Ins):
    * a set of simple raw formats (.byt,.sht,.flt,...) see docu for details!
    * Grey Images in .pgm format (P5 type)

    - digital NanoScope data import (some versions)
    - Omicron Scala
    - WxSM (in preparation)
    - Park Scientific (AFM, basic import support)
    - Targa (.tga) export, in 8/16/24bits color depth
    - any NetCDF file containing a 2D data array
  - Special instrument control Plug-Ins:
    - a Scan Control Panel
    - SPM: DSP Control: Feedback and Scan Characteristics (Speed, ...)
    - SPM: Mover/slider and auto approach controls
    - SPM: flexible Probing: Spectroscopy (STS), Force-Distance Curves (AFM), using the DSP also Digital LockIn Probing (e.g. dI/dU (U)) is possible without any additional hardware! But not only SPM, I already ran our Quadrupole Mass Analyzer with it!
    - SPA-LEED: Peak Finder, Focus-Tool, Rate-Meter, Peak Intensity Monitor
    - SPA-LEED Control (used with SPA-LEED test module)
    - CCD control (very special)
  - NanoPlotter Plug-in: reads simple HPGL files and moves along the plot path using predefined DSP settings (U, I, Feedback Parameters, Speed, ...) for "Pen-Up" and "Pen-Down" movements. This was in principle already possible via the remote control and a script, but now it's much more convenient and user friendly! Even the path is shown ... can be modified, saved and re plotted!
  - A Plug-in categorizing mechanism automatically only loads the required Plug-Ins for the actual setup: E.g. no Hardware Control Plug-ins are loaded in "offline" Data Analysis Mode.
  - At the time there are more than 41 Plug-ins.

- Gxsm itself is hardware independent, that means a predefined command-like instruction/data exchange mechanism has to be served by a device driver (loadable kernel module). See hardware section for details!

- Scan parameter changing on the fly – you can modify the feedback parameters, change the scan speed or switch the tunneling voltage while scanning in between scan lines. For example: Imagine you are scanning a STM topography and current Image, the surface looks flat, then just change the feedback parameters to CP=0 and CI=1e-5 (something small) and now you are in constant height mode!

- On the fly, even while scanning is in progress, you can view profiles, extract data parts, re-scale – just do all you like!
- Remote Control Interface: The Gxsm scanning progress is fully scriptable by a special fifo-command interface: So you may use any language you like, I like perl, to put commands to Gxsm. This is extremely useful to program tricky and or long SPA-LEED scans.
- Cross Platform: works on Linux i386, PPC, others may work too.
- Gxsm takes full advantage of the NetCDF data format.
- Scan auto saving, session logging, Plug-in details browser, NC-View, PS-Printing and a Icon generator are available too.
- Included Add Ons:
- a digital real-time scope "Goszi"
- a free Linux COFF loader to launch the DSP program on PC31 and PCI32 cards
- the required kernel modules
- DSP binary (precompiled) COFF files for STM and AFM, ask for more!
- A DSP status monitor, called "lcd" and as Gnome Applet "dsp_applet".

1.2 Supported Instruments

In August 2001 Gxsm supported the following seven instruments:

1. Scanning Tunneling Microscopy (STM)
2. Atomic Force Microscopy (AFM)
3. Scanning Angle Resolved Light Scattering (SARLS)
4. CCD-cameras (Percy's telescope)
5. Spot-Profile-Analysis of Low-Energy Electron Diffraction (SPA–LEED)
6. any similar method will work too

Computer-hardware support was originally limited to two signalprocessor-interfaceboards (DSP-boards) of the types PC31 and PCI32 manufactured by Innovative Integration. But the internal structure of Gxsm allowed for easy extendability to other hardware.
Knowing this, a new kernelmodule with a DSP-emulation thread was developed in January 2001 which allowes the use of non-DSP-interfaces. A hardware-free SPA–LEED simulation modules and a module for the Burr-Brown-Series were written.
The CCD-camera is controlled by a simple kernelmodule, which drives some hardware connected to the parallel port.
All hardware control is done via the `/dev/pcdsp` device (in Gxsm this devicename is configurable). Read/Write and IOControl operations control the opened device. This functionality was originally used fpr communication with the DSP-card. The driver (kernelmodule) has a high-level interface, including commands for autonomously scanning lines with successing data delivery.
The first part of this document describes the usage of the software. The last part describes its internal structure and step-by-step instructions for your own modifications of the X/Gxsm software.
Hint: In SPA–LEED mode we have a modified main window, displaying the SPA–LEED typical parameters and options.

2 Gxsm Project Installation

2.1 System requirements

Gxsm need a machine running Linux. At least you will need a Pentium Class CPU P133 (a K6-200 or better will be fine too) with a minimum of 64MB ram. Gxsm is now proofed to run on a PowerPC/G4, using Debian/PPC-unstable, but I don't know about using PCI hardware, because I've a laptop used for data analysis only. To compile and run Gxsm the Gnome/Gtk+ Libs and some additional libs are required. To be precise here is a list of actually required libs of my version running on Debian Linux 2.2 + Gnome-Helixcode update from, simply add to you Debian's /etc/apt/sources.list a line like "deb http://spidermonkey.helixcode.com /distributions/debian unstable main" and run a update. All needed libs are available as Debian Packets, but Gxsm was build on a Suse 7.0 System + Gnome Helixcode Update as well and should be working an any recent linux system!
A call to "ldd gxsm" shows the following depends of gxsm:

```
pzahl@charon:~$ ldd Gxsm/src/gxsm
        libm.so.6 => /lib/libm.so.6 (0x6ff68000)
        libglib-1.2.so.0 => /usr/lib/libglib-1.2.so.0 (0x6ff17000)
        libdl.so.2 => /lib/libdl.so.2 (0x6fef4000)
        libgmodule-1.2.so.0 => /usr/lib/libgmodule-1.2.so.0 (0x6fed1000)
        libXi.so.6 => /usr/X11R6/lib/libXi.so.6 (0x6fea8000)
        libXext.so.6 => /usr/X11R6/lib/libXext.so.6 (0x6fe77000)
        libX11.so.6 => /usr/X11R6/lib/libX11.so.6 (0x6fd55000)
        libz.so.1 => /usr/lib/libz.so.1 (0x6fd25000)
        libfftw.so.2 => /usr/lib/libfftw.so.2 (0x6fcdb000)
        librfftw.so.2 => /usr/lib/librfftw.so.2 (0x6fc98000)
        libnetcdf_c++.so.3 => /usr/lib/libnetcdf_c++.so.3 (0x6fc4f000)
        libnetcdf.so.3 => /usr/lib/libnetcdf.so.3 (0x6fc02000)
        libgdk_pixbuf.so.2 => /usr/lib/libgdk_pixbuf.so.2 (0x6fbc8000)
        libgnomecanvaspixbuf.so.1 => /usr/lib/libgnomecanvaspixbuf.so.1 (0x6fba400
        libgnomeui.so.32 => /usr/lib/libgnomeui.so.32 (0x6fa7c000)
        libart_lgpl.so.2 => /usr/lib/libart_lgpl.so.2 (0x6fa4d000)
        libgdk_imlib.so.1 => /usr/lib/libgdk_imlib.so.1 (0x6fa05000)
```

```
libSM.so.6 => /usr/X11R6/lib/libSM.so.6 (0x6f9db000)
libICE.so.6 => /usr/X11R6/lib/libICE.so.6 (0x6f9a3000)
libgtk-1.2.so.0 => /usr/lib/libgtk-1.2.so.0 (0x6f7e1000)
libgdk-1.2.so.0 => /usr/lib/libgdk-1.2.so.0 (0x6f77f000)
libgnome.so.32 => /usr/lib/libgnome.so.32 (0x6f743000)
libgnomesupport.so.0 => /usr/lib/libgnomesupport.so.0 (0x6f71e000)
libesd.so.0 => /usr/lib/libesd.so.0 (0x6f6f7000)
libaudiofile.so.0 => /usr/lib/libaudiofile.so.0 (0x6f6b3000)
libdb3.so.3 => /usr/lib/libdb3.so.3 (0x6f5b9000)
libgtkgl.so.5 => /usr/lib/libgtkgl.so.5 (0x6f58b000)
libGL.so.1 => /usr/lib/libGL.so.1 (0x6f40b000)
libglut.so.3 => /usr/lib/libglut.so.3 (0x6f3a6000)
libGLU.so.1 => /usr/lib/libGLU.so.1 (0x6f371000)
libXmu.so.6 => /usr/X11R6/lib/libXmu.so.6 (0x6f339000)
libgtkdatabox-0.1.13.so.0 => /usr/lib/libgtkdatabox-0.1.13.so.0 (0x6f30f000)
libstdc++-libc6.2-2.so.3 => /usr/lib/libstdc++-libc6.2-2.so.3 (0x6f298000)
libc.so.6 => /lib/libc.so.6 (0x6f144000)
libpthread.so.0 => /lib/libpthread.so.0 (0x6f10d000)
libXt.so.6 => /usr/X11R6/lib/libXt.so.6 (0x6f088000)
/lib/ld.so.1 => /lib/ld.so.1 (0x30000000)
```

2.1.1 2.2.x kernel

The included kernel modules are testet with linux kernel 2.2.14-18, support of older (2.0.X kernel) was removed. To compile the modules you will need to have the kernel properly installed in "/usr/src/linux" and you should have the symlinks to the actual kernelcode from
"/usr/include/linux − > /usr/src/linux/include/linux"
and
"/usr/include/asm − > /usr/src/linux/include/asm".
The running kernel should be consistent with your kernel source!

2.1.2 2.4.x kernel and DevFs

The linux 2.4.x kernel generation is now supported. The updated modules are now located in "Gxsm/plug-ins/hard/modules", while the 2.2.x version modules are frozen and untouched residing in "Gxsm/pci32/modules". The 2.4.x modules are designed to work with DevFs (Device File System), so I strongly recommend to have a 2.4.x kernel supporting DevFs!

2.2 Getting Gxsm via CVS

Go the the Gxsm Project's home page at
http://sourceforge.net/projects/gxsm
Follow the link at the bottom "CVS Repository" and have a look at the instructions there:
Anonymous CVS Access
This project's SourceForge CVS repository can be checked out through anonymous (pserver) CVS with the following instruction set. The module you wish to check out must be specified as the modulename. When prompted for a password for anonymous, simply press the Enter key.
cvs -d:pserver:anonymous@cvs.gxsm.sourceforge.net:/cvsroot/gxsm login
cvs -z3 -d:pserver:anonymous@cvs.gxsm.sourceforge.net:/cvsroot/gxsm co Gxsm
This will checkout (co) the most recent Gxsm source code tree in a directory tree starting with "Gxsm".

2.3 Making all of Gxsm

2.3.1 Configuration

Change into the Gxsm directory and type (if you have the "stow" utility installed and like to sort your local soft enter the optional prefix like below in brackets):
./autogen [–prefix=/usr/local/stow/gxsm]

2.3.2 Compilation

Type make in the Gxsm dir:
make

2.3.3 Installation

Switch to root and run "make install":
make install (do this with root privilegs)
(if you are using stow, then call stow in /usr/local/stow: "stow gxsm".)
If there are any errors, in most cases there are missing development packages, have a look at the fisrt occuring error and finger out what's missing.

2.4 Kernel Modules

2.4.1 Compiling 2.2.x kernel version modules

If you are using a 2.4.x kernel, go to next section!
Change to Gxsm/pci32/modules:
./configure
make
The modules pc31.o and pci32.o should be there now.

2.4.2 Compiling 2.4.x kernel version modules

You will need a kernel and you system with DevFs support!
Change dir to "Gxsm/plug-ins/hard/modules":
./configure
make
There is no need to make devices anymore! But remember: The devices are automatically generated: "/dev/pcdsps/tms320"

2.4.3 some helping scripts

All file paths in the here provided scrips are handhacked, so please adjust to you needs!
It's handy to install a small "rc.boot script" like:

```
percy@nano:~$ cat /etc/rc.boot/pci32

#!/bin/sh
echo Loading PCI32 module...
insmod pci32
echo Done.
echo Loading an starting SPM DSP program...
# replace the paths and xafm.out file locations by your needs
/usr/local/bin/loadpci -d /dev/pcdsps/tms320 \
    /usr/local/stow/gxsm/share/gxsm/pci32/tms320-htstm.out
echo Done.
```

And for (emergency) DSP restart a small script like may be handy:

```
percy@nano:~$ cat /usr/local/stow/gxsm/share/script/dsp-reset.sh

#!/bin/sh
sync
echo ----------------------------------------------------------------------
echo DSP emergency restart, be prepared!
```

```
echo ------------------------------------------------------------------
echo Killing all possibly DSP using apps next.
echo Ready to go? I will kill all DSP Apps: goszi, gxsm and lcd!!!
echo ""
echo hit return to go
read
echo ------------------------------------------------------------------
sync
echo killing goszi
killall -9 goszi
echo killing gxsm
killall -9 gxsm
echo killing lcd
killall -9 lcd
echo waiting 2s....
sleep 2
echo Done.
echo ------------------------------------------------------------------
echo Syncing Disks for emergency
sync
echo ------------------------------------------------------------------
echo "Resetting DSP, Loading an (re)starting SPM DSP program..."
echo ------------------------------------------------------------------
sync
# adjust paths and .out file name to you needs!
loadpci -d /dev/pcdsps/tms320 /usr/local/stow/gxsm/share/gxsm/pci32/tms320-htstm
.out
echo Done.
echo ------------------------------------------------------------------
echo Have fun...
echo ""
echo hit return
read
```

2.4.4 Installing

The reqired module can now be loaded using insmod.
The to access the driver a char device "/dev/pcdsp" is needed (only 2.2.x kernel):
mknod /dev/pcdsp c 254 0 -m 666 (as root)
will create this.
Check if everythin is right: Have a look at "/var/log/mesages" while doing insmod / rmmod:

tail -f /var/log/messages (as root)
The Major-Number reportet at insmod should be consistent with the device major 254!

2.5 DSP: PC31, PCI32 Card and Tools

The DSP Card needs to be feed with the DSP program, therefore we need some tools:
Change to "Gxsm/pci32/load"
do there a
xmkmf
make

2.6 BurrBrown Card (SPALEED)

2.6.1 BurrBrown Module

Erstellung der Kernel Module:
In Gxsm/plug-ins/hard/modules/ ein make ausführen. (dazu ist eine Installierter Kernel 2.4.x/DevFs + Source/Header notwendig)
Dann sind zu finden:
Gxsm/plug-ins/hard/modules/:
Makefile

dspemu.c	→ Basis DSP Emulation
spaleed_emu.c	→ SPA-LEED Simulations Modul
spaleed_bb.c	→ SPA-LEED BurrBrown Modul
	(Base Adr: 0xd0000 ist vorgegeben, Editor!!)
dspbbspa.o	→ mit insmod ladbares Modul für BB
dspspaemu.o	→ Simulationsmodul

2.6.2 Gxsm Configuration for SPA–LEED

Gxsm Settings to run SPA–LEED (Preference Folder):

Hardware/Card:	"spa"
DataAq/PIDSrcA1:	"Counts" , alles anderen Kanäle auf "-"
Instrument/Type:	"SPALEED"

Instrument/Name:	“Name der Maschine”
Instrument/XCalib, YCalib	
Instrument/EnergyCalibVeV,	Sensitivity ggf anpassen.
User/Unit:	“V”

3 Program Start

3.1 Command line parameters

Gxsm uses the gtk/glib libraries. This offers a simplified and flexible use of command line parameters. `gxsm --help` shows a list of options with only the last section (gxsm-options) being relevant for Gxsm configuration.

```
zahl@uranus:~/C/Gxsm/src> ./gxsm --help

gxsm options
  -h, --Hardware-Card=default
    Hardware Card: no | spm | spa | ccd

  -u, --User-Unit=default
    XYZ Unit: AA | nm | um | mm | BZ | sec | V | 1

  --disable-plugins
    disable default plugin loading on startup

  --force-configure
    force to reconfigure Gxsm on startup
```

Right after the options shown above you can list files for immediate opening to the free channels.
All known formats (7.1) are autodetected.

3.2 First start of Gxsm

Starting with gxsm version 1.1.13 a new gxsm user is guided via a Configuration Druid to get the most essential settings right 3.1. I expect this Druids to be well self-documented.

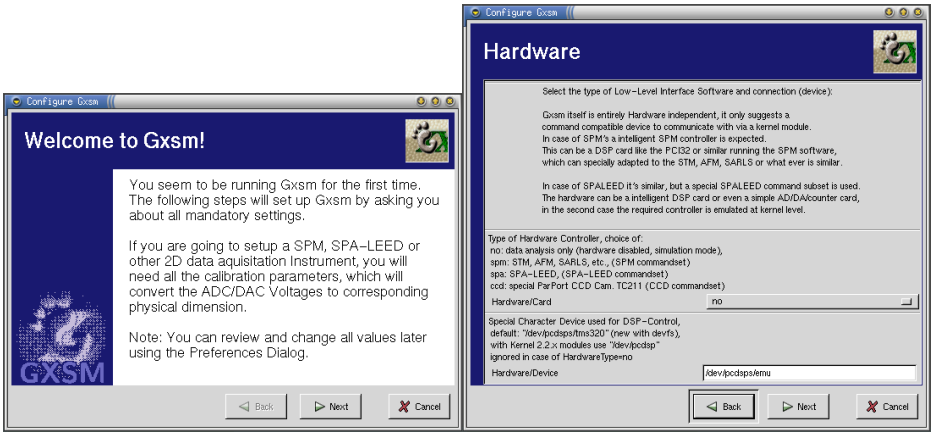


Figure 3.1: First two pages of Gxsm Configuration Druid.

3.3 Mainwindow

After startup, the main window appears, if you saved a window configuration earlier, subsequently those windows reappear.

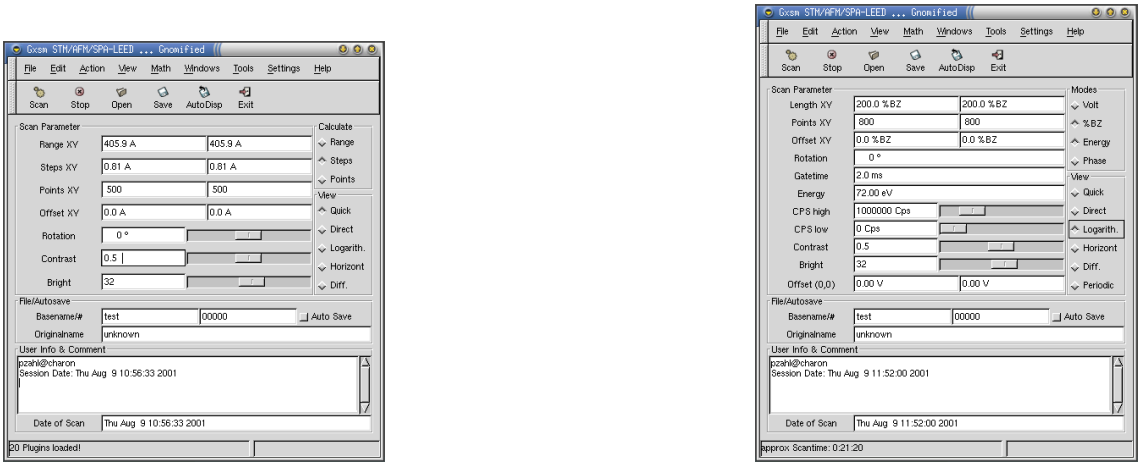


Figure 3.2: Gxsm Main Window, `SPM`-variant left, `SPALEED`-variant right.

3.4 Gnome-Resources

All your settings are saved in the file `~/.gnome/gxsm`. The configuration druid uses this file and the save-values command will rewrite it, just like the `Accept` and `OK` button in the preferences.
With special care this textfile can be edited, to set defaultvalues. To restore a totally screwed configuration simply remove it. The next time you start gxsm the configuration wizard will pop up to create your settings from the defaults.
With Settings/Preferences (dt: Einstellungen/Einstellungen) Gxsm can be configured during runtime, the changes are (as seen above) saved in `~/.gnome/gxsm`. This file can be copied to any new user, who wants to share the same instrument.
Any of the settings provides a small help text for your information.

3.4.1 Folder Hardware

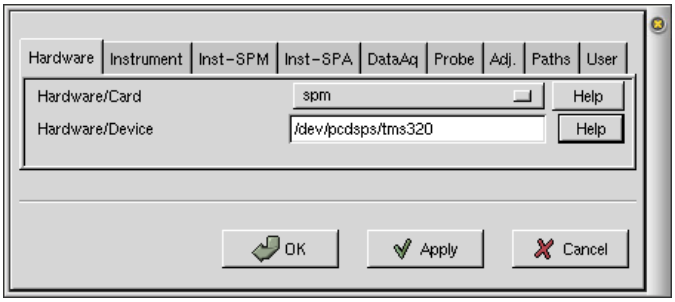


Figure 3.3: Gxsm Preferences Folder, Hardware Page

Hardware/Card Choose hardware:

`no`: no hardware connected, internal dummy-mode.
`spm`: intelligent Scanning Probe Hardware Module expected at device. (pci32.o, pc31.o at /dev/pcdsp (see later) + running xafm.out on DSP interface board)
`spa`: intelligent SPA–LEED Hardware Module expected at device. (pci32.o, pc31.o, dspbbspa.o, dspspaemu.o /dev/pcdsp (see later) and running spa.out on DSP interface board in the case the DSP version is running)
`ccd`: for CCD-Modul (ccd.o)

Hardware/Device path to the device mentioned above (typ. `/dev/pcdsp`).

3.4.2 Folder Instrument

Instrument/Type One of STM, AFM, SARLS, SPALEED, CCD.

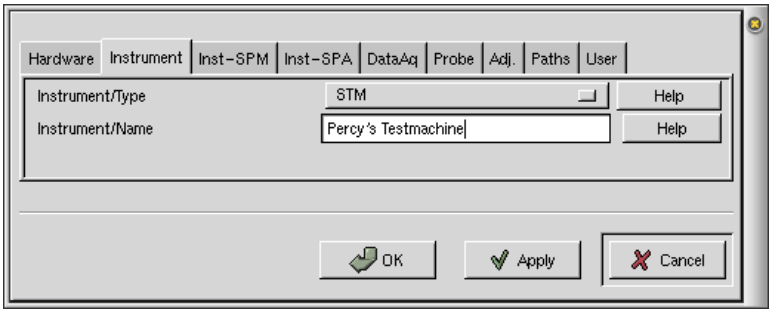


Figure 3.4: Gxsm Preferences Folder, Instrument Page

Instrument/Name Arbitrary name, max. 30 characters.

3.4.3 Folder Inst-SPM

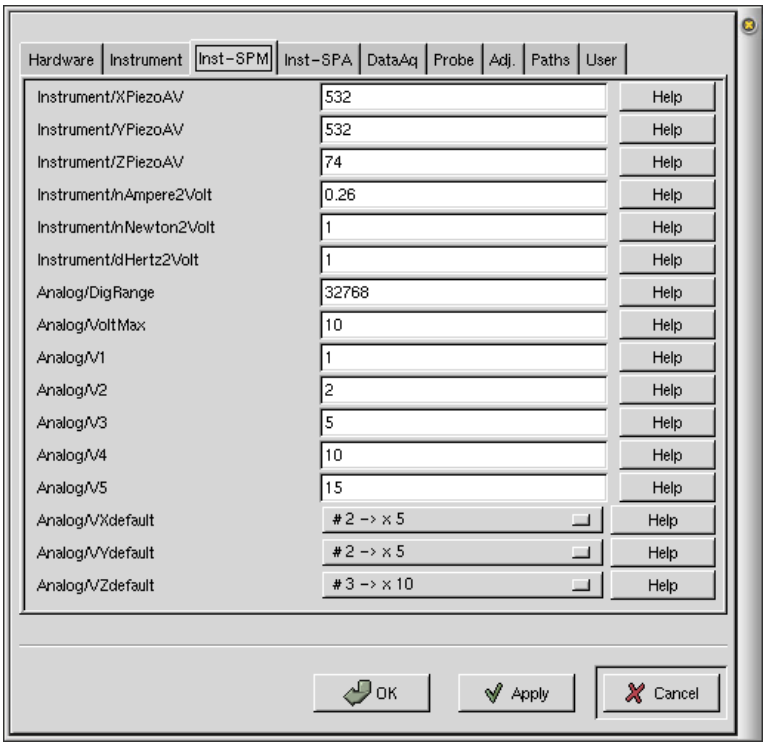


Figure 3.5: Gxsm Preferences Folder, Instrument SPM Page

Instrument/X,Y,ZPiezoAV SPM Piezo sensitivity in A/V.

Instrument/nAmpere2Volt Tunnel amplifier sensitivity: `factor*Volt=nA`

Instrument/nNewton2Volt for AFM Setpoint in nN, 1 for 1nN = 1V

Instrument/dHertz2Volt NC AFM Setpoint

Analog/DigRange positiv maximum value for X,Y,Z DA-conversion with respect to VoltMax. The converters have to be bipolar (e.g. ±10 V).

Analog/VoltMax DA Voltage corresponding to DigRange.

Analog/V1-V5 Piezoamplifier Settings, typically: 1,2,5,10,15.

Analog/VX/Y/Zdefault Preferences at programm startup, valid indexvalues are 0 . . . 4 relativ to V1-V5.

3.4.4 Folder Inst-SPA

Instrument/X,YCalibV SPALEED: X,Y calibration factor, 1V at DA → ±10 V resp. 15 V at octopol front/back.

Instrument/EnergyCalibVeV factor*Volt = energy in eV

Instrument/Sensitivity BZ = U · Sensitivity/$\sqrt{\text{Energy[eV]}}$

Instrument/ThetaChGunInt Half angle between channeltron and elektron gun (intern).

Instrument/ThetaChGunExt unused.

Sample/LayerDist Atom layer distance of the sample in Ångstroem, used for calculation of phase (energy in `S`).

Sample/UnitLen unused.

3.4.5 Folder DataAq

See also section 4.2 for Input-Konfiguration.
Task MB to do more!!!!

3.4.6 Folder Probe

Task MB to do!!!!

3.4.7 Folder Adj.

Configuration of Value-Slider:
Adjustments.Name/min,max for Area (please care for min < max !!), Adjustments.Name/step,page for stepwidth at Cursor, Click.

3.4.8 Folder Paths

Path/Logfiles

Path/Data

Path/RemoteFifo path to remote fifo (ro by Gxsm)

Path/RemoteFifoOut path to control fifo `Echo` (wo by Gxsm)

Path/Plugins Additional Plugin searchpath

3.4.9 Folder User

User/SaveWindowGeometry Always `false`, for future use...

User/Unit XYZ-Einheit: Choice of AA, nm, um, mm, BZ, sec, V, 1

User/HiLoDelta Checking distance of array for calculation of Min/Max for Autodisplay, 1 = all points visited.

User/FileType nc (dat possible, but out of date.)

User/NameConvention digit: Auto enumeration with 001, 002, ..., alpha: Auto enumeration with aaa, aab, ...

User/SliderControlType slider: Omicron Slider Control, mover: Besocke

User/Palette path to palette image (may be altered at runtime) See also ...

Parameter	Description
-h, –Hardware-Card=*type*	set up type of hardware
no	no hardware
spm	PC31/PCI32 via /dev/pcdsp, STM/AFM mode
spa	PC31/PCI32 via /dev/pcdsp, SPA–LEED mode
ccd	TC211 CCD Mode (user-space-io)
-u, –User-Unit=*unit*	XYZ unit
AA	Å, $1e-10$ m
nm	nm, $1e-9$ m
–disable-plugins	Disables the loading of plugins on startup mainly for debugging.
–force-configure	force to reconfigure Gxsm on startup (via Druids)

Table 3.1: commando line parameters for calling Gxsm

4 Channels

One of the most important features of Gxsm is the multichannel-capability. Multichannel-capability describes the simultaneous data acquisition and display from different sources. You may e.g. at the same time measure topography and friction-forces with the AFM.

4.1 The channel dialog

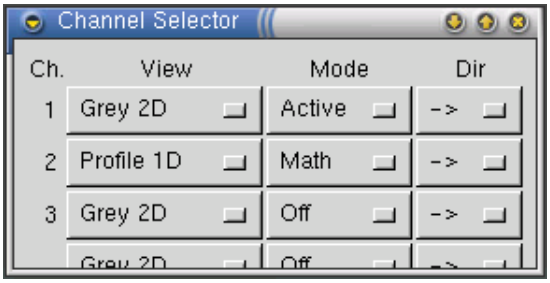


Figure 4.1: Channel Selector Window

The channel dialog pops up selecting Ch. Sel. in the main window. You can use it, to select the displaymode (View) and the port for data acquisition (Mode) for any channel.
One of the following modes can be chosen:

1. No During data aquisition no data are displayed.
2. Grey 2D The data are displayed as a grayscaled image in an own window.
3. Surface 3D Three dimensional view of your data in an own window.

The display options are described in section (9)
For acquisition of data several modes are available:

1. Off or On. Off deletes the channel.

2. Active. The most important mode, it sets the active channel. All image manipulation is done using this channel. Only one channel can be Active at a time.

3. Math Channel, which stores the result of the last operation is automatically called Math.

4. X Needed for several math/image-manipulation, that need more than one source. See section (13.4.5).

5. *+- Scan Xsm:* This channel will be used as a target for a scan. Which data source is used to fill this channel is defined by the following indicator (e.g. topography).

   The toggle +- Scan defines the scanning direction at which data are collected. Thus `+Topo` means measurement of your topography during movement of your scanhead in +X direction. The `+- Scan` button toggles this direction.

   Gxsm: In Gxsm the third row has a new button, which indicates the direction, with which data are acquised.

6. Inputchannel, whose names are given by the local configuration (See 4.2). Different data sources are selected here.

4.2 Input configuration

In this section the assignment of data sources to the Gxsm channels is explained. This configuratioin is done using the Preferences dialog - section DataAq. The particular configuration depends heavily on your data acquisition hardware and is unique for each instrument.
To give a symbolic name to your input channels you use the x-resources. E.g. `Xsm*DataSrcA1: ZForce` means, that your first multiplex-(MUX)-channel is given the name `ZForce` in an AFM-environment. Depending on your needs and hardware, several channels are available.
The PCI32 has four A/D-converters with four input-lines directly connected. The older PC31 has only two A/D-connectors, but four input-lines are adressed via a 4x-multiplexer, which results in eight usable analog input lines. Name convention in the x-resources is A and B for the converters and 1 to 4 for the four multiplexer input lines (PC31).
The PCI32 has four converters which are called A1, A2, A3 and A4.
[1]

[1]The sources called C and D are optional yet and unused.

Scanning Probe Microscopy (SPM) usually has one input variable which is controlled to be constant by the control-loop on the DSP.
Through setting and holding the normal force (`ZForce`, see above) you gain the topography (`Topo`) signal in contact-mode-AFM operation.
This name is configurable through the x-resources as well. The entry `Xsm*PIDSrcA1: Topo` means, that the signal which is acquired from and set to the first multiplexer-input line at the AD-converter A is called `Topo`.
You can set default multiplex-channels to be active on program start. Simply add `, *` at the end of any resource entry. [2]
The first three letters of your channel names must be non-ambiguous, because they will be used in your automatically generated filenames. (See also section 7.1).

[2]The comma is mandatory.

5 Control – Scan and Sample

In Gxsm the DSP-Control is implemented as a plug-in, loaded only for spm-instruments and spm-hardware. You can open the dialog via the `Window`-menu.

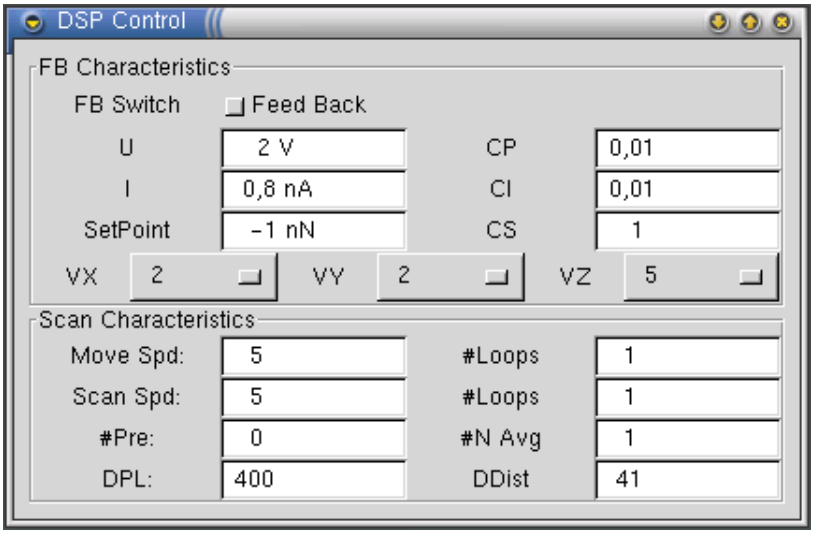


Figure 5.1: DSP Control plug-in window

5.1 Proportional Integral Regler

All parameters which control the details of the scanning procedure are set in the dialog DSP Control In STM-mode U is the tunnel voltage and I the target tunnel current. In AFM-mode U is the Set-Point – that is the voltage, onto which the z-force-signal coming from the PSD shall be driven to –, I is without function.
The control behaviour – FeedBack - Characteristics – is set thorugh **CP** (proportional part) and **CI** (integral part).
Recipe if no sensible values are known:

1. Set CP to zero, CI to 0.01. If you now change the z-offset, the control should follow slowly.

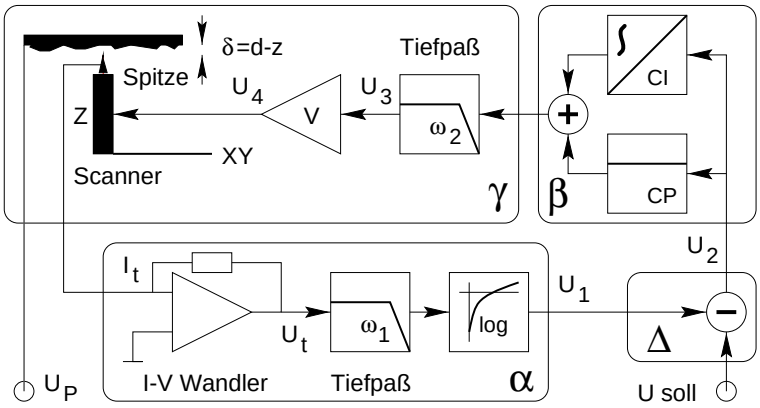


Figure 5.2: Control circuit - scheme (STM)

2. Increase CI slowly until the control behaviour begins to become unstable. When you abruptly change z-offset the control oscillates.

3. Now decrease CI again until all oscillations are gone.

4. Set CP to 100% to 200% of CI to stabilize the control.

You may increase CI and CP experimentally for quicker response.
CS is a slope correction in the case, that your sample is tilted towards the intended optimal foces plane. CS=1 results in the addition of the mean slope to lighten the load off the z-signal-correction. CS=0 prevents this slope correction, CS=0.5 add only the half of the extrapolated height, CS>1 add accordingly more than the average slope would suggest.

5.2 Scan Characteristics

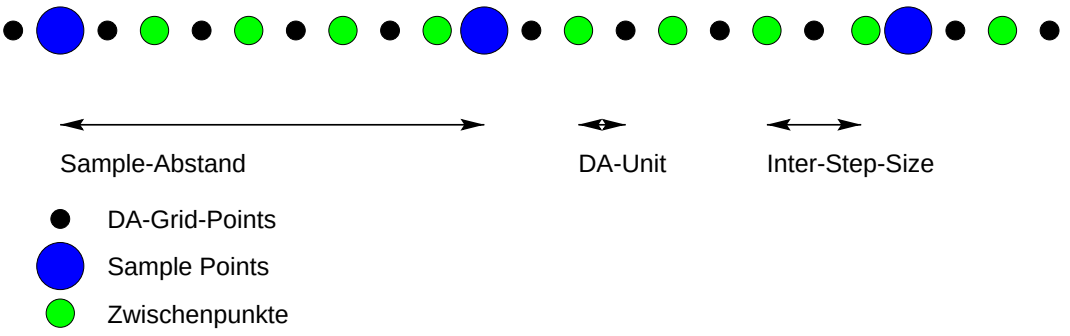


Figure 5.3: Linescan execution

The `Control` dialog allows the user to set the following scan properties:

A 2D-Scan is defined by some parameters: A size (range X, Y), number of points in X and Y (points x, y). With this information one can calculate the sample dot distance (dx, dy). This dx and dy have to be a multiple of the smallest possible stepwidth (`DA-unit`) and, as a consequence, the total range has to be a multiple of the small stepwidth (dx, resp. dy).
If the stepwidth is large, it leads to a smoother movement, not to jump from sample point to sample point, but to define some intermediate steps (Inter-steps).
Inter-steps contains the stepwidth of intermediate steps in multiples of DA-units. An entry of 1 means, that the range between two sample points is covered via steps of the length 1xDA-unit. A smaller value is not possible.
During a scan for position changes the `Line` entry is used, otherwise the `Move` entry Additionaly it is possible to stay for some time at any sample point. The textfield `FB-Loops` contains the number of feedback-loops which halts the scanhead for a limited time. The loop runs with 50 kHz.
The parameter `N` sets the how often at any position a value is sampled. After these values are measured. The result is their average.
`Pre` defined the number of steps, which are scanned before any data acquisition takes place.
`DPL` (dots per line) and `dos distance` (in DA-units) cannot be set. They are FYI only and set at scan-start

6 `Mover` – Fine-tuning

ToDo by Markus :=)

6.1 Modus: `mover` oder `slider`

6.2 `Auto`: automatic approach

6.3 Mover control

Lens, Rot, ... – ToDo by AK :=)

6.4 Slider control

7 File I/O

7.1 Load and Save

Several fileformats are understood by Gxsm. Filetype is determined by the filename, so please care for correct naming of your data. (You can use `mmv` if you have to rename larger amounts of files.)
As time of this writing the following fileformats are known by Gxsm. Some of them (e.g. NetCDF) are implemented in the Gxsm core, others are added via plug-ins (e.g. the SCALA SPM format is handled by the `omicron_io` plug-in).

Suffix	description
.nc	NetCDF Format, default, see section 16.4.3 additionally to nc-files produced by Gxsm it handles all NetCDF-files, which contain 2D of type `Byte`
.dat	(old) STM data format, see section 16.4.1
.d2d	(old) SPA–LEED data format
.hdf	Park Scientific Instr. HDF, read only, see section 16.4.4
.sdf	Surface data format, read only, not all features used yet.
.tf?, .tb?	SPM file format used by Omicron's SCALA software for topographic data, currently read-only.
.sf?, .sb?	SPM file format used by Omicron's SCALA software for gridded spectroscopy data, currently read-only.
.*.gz	data in GNU zip format. (* nc, dat or hdf)
.*.bz2	data in bzip2 format. (* nc, dat or hdf)

The default fileformat for saving SPM data (either NetCDF or dat) can be set in the Preferences Dialog - section User, FileType.
If a channel is active, the opened file is using it, otherwise the first free channel is occupied.
`Save As` pops up a dialog window, where you can enter a filename and browse the directory structure, while `Save` uses an autogenerated filename from a basename, an ID-name for the channel and a running number which is increased after a scan has finished. The numbering can also be switched to alphanumeric counting in the Preferences Dialog - section User, FileNameConvention.

`Browse` is comparable to `Load`, only the `Open` dialog is not closed after loading a file; for quick review of files.

7.2 Autosave

The best program may crash due to unforeseen conditions. To prevent data loss in this case an `Autosave` options is built in.[1] You have to distinguish two sorts of autosave:

1. Autosave at the end of a scan.

2. Intermediate Autosaves for backup reasons.

While the first one is simply a *Save*-command at the end of scan (or a moviescan) for convenience. The second one is a bit more sophisticated.
The preferences have three entries in the User-section to control autosave behaviour. These are AutosaveUnit, AutosaveValue and AutosaveOverwritemode. AutosaveUnit can be one of percent, lines or seconds and define the repetition unit of your autosaves. AutosaveValue defines is value. If you choose for example `AutosaveUnit=seconds` and `AutosaveValue=100`, then every 100 seconds a file will be written to your harddisk, with a special name, that indicates its nature.
X/Gxsm normally asks before a file is overwritten, to prevent deletion of important datafiles. Anyway, if you have a scan running and it is suddenly interrupted by the Filedialog, your scan may be ruined. As a consequence, `AutosaveOverwritemode=true` offers you to force the writing of intermediate autosave files. Let me repeat this: Only the intermediate autosaves are allowed to be overwritten automatically using this flag. The autosave at the end of scan will again ask for permission to overwrite any file.
To gain maximum security for your data, none of the intermediate saves are overwritten during one scan. Instead, a second counter ('subcounter') is used, to signify the ongoing autosave process.
Example: Your basename is picard, your counter shows 00000. If you hit *Save*, your file will be saved as `picard001XYp.nc`. If you hit it again, you have `picard001XYp.nc`. If you start a scan with Autosave enabled, at the end of a scan the file `picard002XYp.nc` will be written, no matter, if you created intermediate autosaves or not. If you now decide to create intermediate backups of your scan every 30 seconds, `AutosaveUnit=seconds`, `AutosaveValue=30`, the first autosave file will be after 30 seconds named `picard003_001XY.autosav`
Half a minute later `picard003_002XY.autosave.nc` will be written, followed by `picard003_003XY.autosave.nc`. When your scan is finished, `picard003XY.nc`, will be written. (No subcounter and no `autosave`). If you reach the end of your scan, all

[1] It replaces the quick-and-dirty approach of autosave.perl.

files `picard003_*XY.autosave.nc` can be deleted. (To do: Allow automatic deletion of temporarily saved scans)
If you want to use the autosave feature at the end of a scan, but no backup saves, simply choose a combination in the preferences, that will never be reached, e.g. 110 percent.
Note: This feature was added for slow scans. If your scan is very fast, and you entered a high rate of saves to do, some saves may be skipped. (Like some percentages or seconds may be skipped in the statusbar.)

7.3 `Export` and `Import`

`Import` and `Export` are for loading and saving the `Active` Scans in one of the so called `GNU`-formats.[2] [3] The filetype is determined by the filenames extension. SPA–LEED images in the locally common D2D-format are also read. For more instructions see the Hacker-Guide (16.4.2).
One of the drawbacks of the simpler file-formats is the lack of any additional scaling or other meta-information. Gxsm allows to save this scaling-information in separate files which can be loaded automatically from the directory which keeps your gnu-file. When you load a gnu-file (e.g. with suffix .flt for floats) a file with the additional suffix .sklinfo is searched for. If this file does not exist, Gxsm tries to load the scaling information from `/tmp/labinfo`.
Example: The file `n15_333d_log.flt` shall be read with meta-information. You have to prepare the file `n15_333d_log.flt.sklinfo` in the same directory.
The `.sklinfo`-file may look like this:

```
PSD-Channel:../../flt/n15_333d.flt
0
500
20
x
3.1
2.8
20
../../flt/n15_333d.flt
```

This brief information-file contains at least nine lines with the following data:

[2] This has nothing to do with the 'Gnu's not Unix-project', but is more or less historically.
[3] The formats .byt, .sht, .flt, .dbl, .cpx, .pgm are supported.

Line	Value	contents from example
1	Label für x axis	PSD-Channel:../../flt/n15_333d.flt
2	X-min	0
3	X-max	500
4	number of ticmarks	20
5	label for y axis	x
6	Y-min t	3.1
7	Y-Max	2.8
8	number of Tickmarks	20
9	title	../../flt/n15_333d.flt

This information is particularly useful to give a scale to a grayshaded printout.

7.4 Drag and Drop

Gxsm accepts all loadable files via 'drag and drop', e.g. from the Gnome-midnight-commander, Nautilus, or Netscape Navigator. Even dragging URL's pointing to loadable files on the web is possible.

If you drop a file on a channel-window, it is loaded into that channel. To create a new window with a new channel bound to it, drop the file above the main window.

7.5 `Make Icons`

`Make Icons` allows the print-out of larger series of images for browsing and quick survey. You enter a regular expression in `Mask` and a path in `Path` and all files fitting into this expression are printed into a postscript-file `Icon File` for later (or immediate) printing.

Gxsm can produce icons in several layouts. You may select `2x3`, `4x6` or `6x9` per page resulting in less and larger or more and bigger icons of your scans.

The image resolution is scaled to printer-appropriate values to decrease render time for the printer and to minimize to risk of memory shortage for your page-printer.

A number of options can be selected to improve your print-out. Play with the possibilites to find your favorite combination.

Option	
—	keine
E-Reg 30	Subtraction of a regression plane, 30% of border excluded
E-Reg 5	Subtraction of a regression plane, 5% of border excluded
Lo-Res	for 600dpi printer
Hi-Res	for 1200dpi printer
default-Skt	Uses the settings from the `Display`
auto-Skl	Automatic rescaling
Line Reg	Uses the settings from the `Quick`-display
—	none

There are some more undocumented yet.

7.6 Print PS

`Print PS` saves single images (the active scan) in EPS[4]-format
There are several undocumented options.

7.7 Save Geometry

To save the location and sizes of your windows for later restore, choose `Save Geometry`. A file `.xsmgeometry` is created in your home-directory to save the information. If you reopen windows their position is restored from the file.

7.8 Save Values

The most important parameters[5] can also be saved when you choose `Save Values`. This information is also saved in your home directory in a file called `.xsmvalues`. On program startup this file is read. For further information see 16.5.2 in the hacker-guide.

[4] encapsulated postscript
[5] scan size, etc.

8 Monitor – Eventlogging

In the current directory a log-file is created for every session. The filename is named `EV_MMMYYYY.log`, with MMM being the month and YYYY the year of the monitoring. This file is opened in append-mode and not overwritten if it already exists on startup.
The following actions are logged (examples):

Time Stamp	Action	Comment	value 1 . . . 3
Wed Feb 2 12:26:23 2000	Monitor :	startup :	
Wed Feb 2 12:26:24 2000	Gxsm :	startup :	
Wed Feb 2 12:26:39 2000	*StartScan :	XpSRCS: 1 XmSRCS: 1 :	
Wed Feb 2 12:26:45 2000	*EndOfScan :	interrupted :	
Wed Feb 2 13:22:01 2000	*StartScan :	XpSRCS: 1 XmSRCS: 1 :	
Wed Feb 2 13:24:17 2000	*EndOfScan :	OK :	
Wed Feb 2 13:24:17 2000	*Save :	./zahl_200muU_R_002CPp.nc :	

The log-file entries since the beginning of the last session can be watched in Window/Monitor (Window/Display).
The field Value1 is unused. [1].

[1] You can create an entry in this file with the method gapp-¿monitorcontrol-¿LogEvent("*Save", fname);

9 Data representation

For visual representation of your acquired data several methods and modes are at hand.
Default is a two dimensional grayscaled image. The following modes are available for display via the Channelselector-¿View.

no There is no no window open, data can be acquired anyway. Saves space and performance.

Grey 2D Standard grayshade image. Several functions to modify.

Surface 3D 3D surface-rendering, uses MESA GL, Hardware accelleration is recommended. Several functions and options via popup menus.

Profile 1D Display of horizontal lineprofiles. Use popup menu to browse through them.

9.1 Displaymodes

`Quick`, `Direct`, `Logarith.`, `Horizontal Diff`, `Periodic`
In `Direct` mode are displayed 'as is'. Only contrast and brightness are regarded (see 9.3.1)
In scanning probe microscopy two sorts of artefacts are common, which prohibit the use of the direct mode. First of all, the sample is commonly slightly tilted with respect to the scan head; secondly, from line to line small 'jumps' may occure. To meet this shortcomiings, the `Quick` Modus was introduced. Only for display a line regression for any line profile is carried out. The source data are not modified.
`Horizontal` shifts the lines, so that their average is zero.
`Diff.` displays the lines first derivation. Calculated through:

$$H(x=0)=0, H(x, \text{for} x=1,2,3,\ldots N-1)=H(x)-H(x-1)$$

9.2 Logarithmic Scale

Logarithmic scaling is chosen by the `Logarith` Button in the main window. It's purpose is the correct (appropriate) representation of 2D-SPA–LEED data. The translation happens in the following way:

$$Grey \propto \log\left(1 + |Data - Min| \times Contrast + Bright\right),$$

with Min being the smallest value.

9.3 Display

9.3.1 Helligkeit und Kontrast

The parameters `Bright` and `Contrast` control the conversion of the data in grayshade- or color-values:

$$Grey = Data \times Contrast + Bright$$

9.3.2 SPA–LEED: CPS High – Low

In SPA–LEED mode the scaling of CPS-High/Low can be set.

9.3.3 Gamma Korrektur

Die Gammakorrektur wird nicht `online` verwendet, sonder funktioniert wie ein Filter (Background-Gamma). Dabei werden the Z-Werte nach folgender Rechnung transformiert:

$$Z_{new} = (Z_{hi} - Z_{lo})\frac{(Z_{hi} - Z_{lo})^{\gamma}}{(Z - Z_{lo})^{\gamma}}$$

9.4 Color mode, Xxsm only

Xxsm offers to display a color image of your data, using a mapping algorithm, which maps the data onto a part of the color circle. You can also use a new color palette.

9.4.1 Falschfarbendarstellung

Choose View-Use Color HUE (Xxsm only) to activate the color mode. Use Autodisp to initialize the display[1].

[1] and to extend the data area to 1024

The slice of the color circle you wish to map can be chosen through HUE `Start-Winkel` to `Ende-Winkel`.
Start at black ... HUE-Start ... HUE-Ende ... white.
Use Display-Contrast und Display-Bright you can select a distint portion of the generated palette.

9.4.2 Own palettes

The color palette is simply a one dimensional PNM-bitmap file, that can be selected through an entry in the resources, or in the Preferences. Click View-Use Palette to apply it. Check Autodisp again
Here a short description of the PNM file format:

```
P3
# CREATOR: The GIMP's PNM Filter Version 1.0
1024 1
255
R
G
B
R
G
B
...
```

The file is expected to have 1024 RGB-entries (R, G, B values in the range 0 ... 255) just following the header. Use `gimp` to generate this conveniently. [2]
Gxsm can load a new palette at any time. Choose a new one in Settings/Preferences/User/Palette and press AutoDisp.

[2] 1. Palette Editor, 2. fill the picture with 1024x1 pixel, 3. save as `meinepalette.pnm` (ascii-type).

10 Programming via Remote-Control

The remote control allows to automatically run scan procedures. A fifo-device[1] is a dummy device through which you can send command sequences to X/Gxsm. Default names are `remote` and `remoteecho`. `remoteecho` is for receiving information from the process.

The simplest way to use this feature is to create a pipe that passes all commands and arguments straightaway into the fifo.

`cat mycommandfile > remote`

or if you want to enter your commands interactively

`cat > remote`

You will find the file `rscript.perl` as an example in the source-tree.

For testing purposes have a look at `remote.sh` (in `Gxsm/src`).

The backchannel must be opened for reading (`cat remoteecho`). The scripts care for that.

Note: Before interrupting the fifo you have to enter `cmd quit` or gxsm fails to notice its stop and waits forever.

The remote control can be called via Tools. Pressing Run blocks Gxsm, until *both* fifos are opened. Interrupt the remote control with Stop and the remote command `cmd quit`.

10.1 Gxsm Base Command Set

The remote control understands the following commands:

[1] Fifo=first in, first out, named pipe, see `man mkfifo`. Create with `mkfifo remote remoteecho`.

Symbols:

X:	real Value
N:	int Value
S:	String

Commands:

Cmd	Arg.	Values		description
set	RangeX	X		set X-Range to Value X
set	RangeY	X		...
set	StepsX	X		
set	StepsY	X		
set	PointsX	N		
set	PointsY	N		
set	OffsetX	X		
set	OffsetY	X		
set	Rotation	X		
set	Display	X		
set	Contrast	X		
set	...	X		
set	LengthX	X		set X-Length to Value X (SPALEED)
set	Energy	X		set Energy to Value X (SPALEED)
set	...	X		
scan	start			Start Area-Scan
scan	init			Init Area-Scan only
scan	update			Update all Hardpars.
scan	line	N		Scan Line N
scan	stop			Stop Area-Scan
file	save			auto save to File **Note:** Channelindex starts with N=0!
file	saveas	S	N	channel N saved to File=SS
file	load	S	N	load to Channel N, File=SS
gnu	import	S	N	import to Channel N, GnuFile=SS
gnu	export	S	N	export from Channel N, GnuFile=SS

Commands (continued):

Cmd	Arg.	Values	Beschreibung
cmd	autodisp		autodisp on active channel
cmd	chmodeA	N	channel N to mode Active
cmd	chmodeX	N	channel N to mode entryX
cmd	chmodeM	N	channel N to mode Math
cmd	quick		Quick-View Active Ch
cmd	direct		Direct-View Active Ch
cmd	log		Log.-View Active Ch
cmd	sleepms	N	sleep for N ms
cmd	quit		close remote fifo
cmd	unitbz		set Length in %BZ
cmd	unitvolt		set Length in Volt
cmd	uniteV		set E in eV
cmd	unitS		set E as S (Phase)
cmd	echo	IamReady	Echo Meldung (Sync Script with Gxsm)
cmd	logev	String	Write `String` to Log-File
analog	da0	X	set analog values [V]...
menupath	S		Activate Menupath=S
action	X		Action X, used by plug-ins see plug-in

10.2 DSP Peak Find Plugin Commandset

Commands Plugin `DSP Peak Find`:

Cmd	Arg.	Values	Beschreibung
action	DSPPeakFind_XY0_1		Get fitted XY Position
action	DSPPeakFind_OffsetFromMain_1		Get Offset from Main
action	DSPPeakFind_OffsetToMain_1		Put Offset to Main
action	DSPPeakFind_EfromMain_1		Get Energy from Main
action	DSPPeakFind_RunPF_1		Run Peak Finder
action	DSPPeakFind_XXX_N		run action XXX (see above) on PF Folder N

10.3 More on remote control

10.3.1 Explicit channel selection via index

The channelindex begins with 0, the first channel is activated with `cmd chmodeA 0` and can be autoscaled with `cmd autodisp`.

10.3.2 Menu-path execution

You can choose any action, that can be reached through menus. This allows the execution of any math- or plug-in-operation.

```
menupath Math/Background/Line\ Regress
```

Spaces have to be escaped with a backslash.

10.3.3 Synchronisation

The echo-command can be used to syncronize a scan and the script
`cmd echo Hallo`

10.3.4 Hacking info

List of all field can be grep'ped from the source.

```
grep RemoteList *.C:

U.AddEntry2RemoteList(          "DSP_U", *RemoteEntryList);
Cp.AddEntry2RemoteList(         "DSP_CP", *RemoteEntryList);
I.AddEntry2RemoteList(          "DSP_I", *RemoteEntryList);
Ci.AddEntry2RemoteList(         "DSP_CI", *RemoteEntryList);
Sp.AddEntry2RemoteList(         "DSP_SetPoint", *RemoteEntryList);
Mvs.AddEntry2RemoteList(        "DSP_MoveSpd", *RemoteEntryList);
Mvn.AddEntry2RemoteList(        "DSP_MoveLoops", *RemoteEntryList);
LSs.AddEntry2RemoteList(        "DSP_ScanSpd", *RemoteEntryList);
LSn.AddEntry2RemoteList(        "DSP_ScanLoops", *RemoteEntryList);
Pre.AddEntry2RemoteList(        "DSP_Pre", *RemoteEntryList);
NAv.AddEntry2RemoteList(        "DSP_NAvg", *RemoteEntryList);
Rx.AddEntry2RemoteList(         "RangeX", RemoteEntryList);
Ry.AddEntry2RemoteList(         "RangeY", RemoteEntryList);
Stx.AddEntry2RemoteList(        "StepsX", RemoteEntryList);
Sty.AddEntry2RemoteList(        "StepsY", RemoteEntryList);
Pnx.AddEntry2RemoteList(        "PointsX", RemoteEntryList);
Pny.AddEntry2RemoteList(        "PointsY", RemoteEntryList);
```

```
Offx.AddEntry2RemoteList(       "OffsetX", RemoteEntryList);
Offy.AddEntry2RemoteList(       "OffsetY", RemoteEntryList);
Rot.AddEntry2RemoteList(        "Rotation", RemoteEntryList);
Contrast.AddEntry2RemoteList(   "Contrast", RemoteEntryList);
Bright.AddEntry2RemoteList(     "Bright", RemoteEntryList);
Rx.AddEntry2RemoteList(         "LengthX", RemoteEntryList);
Ry.AddEntry2RemoteList(         "LengthY", RemoteEntryList);
Pnx.AddEntry2RemoteList(        "PointsX", RemoteEntryList);
Pny.AddEntry2RemoteList(        "PointsY", RemoteEntryList);
Offx.AddEntry2RemoteList(       "OffsetX", RemoteEntryList);
Offy.AddEntry2RemoteList(       "OffsetY", RemoteEntryList);
Rot.AddEntry2RemoteList(        "Rotation", RemoteEntryList);
Gate.AddEntry2RemoteList(       "Gatetime", RemoteEntryList);
Energy.AddEntry2RemoteList(     "Energy", RemoteEntryList);
CPShigh.AddEntry2RemoteList(    "CPShigh", RemoteEntryList);
CPSlow.AddEntry2RemoteList(     "CPSlow", RemoteEntryList);
Off00x.AddEntry2RemoteList(     "Offset00X", RemoteEntryList);
Off00y.AddEntry2RemoteList(     "Offset00Y", RemoteEntryList);
Counter.AddEntry2RemoteList(    "Counter", RemoteEntryList);
```

10.4 Sample Script für `Heringplot`

More examples are in Gxsm/src/*.perl
This script offers several helpful functions, like R(), Set() or Echo():

```
#!/usr/bin/perl

# Example Script for programing Xxsm (C) PZ 1998-12

use Time::Local;
use IO::Handle;

# if you have no fifo, then
# create fifo before with "mkfifo remote"
$remotefifo = "remote";
$remotefifoecho = "remoteecho";

# now open the fifo used for remotecontrol
open(REMOTE, "> $remotefifo\0")
or the "sysopen $remotefifo: $!";
```

```
# use autoflush !!
REMOTE->autoflush(1);

# Open Echo-Fifo for status return
open(REMOTEECHO, "< $remotefifoecho\0")
or the "sysopen $remotefifoecho: $!";

# --------------- Users program starts here ------------------------
# setup scan
R("cmd chmodeA 0");
R("cmd log");
R("cmd unitS");
R("cmd unitbz");
$px=600;
Set("PointsX", $px);
Set("PointsY", 200);
$bzx=240;
Set("LengthX", $bzx);
Set("LengthY", 0);
$Xoff=0;
$Yoff=0;
Set("OffsetX",0);
Set("Offset00X",$Xoff);
Set("Offset00Y",$Yoff);
R("scan init");
$Gate=2;
Set("Gatetime",$Gate);

# now go from S=3.3 to S=7.2 !
for($l=0, $S=3.3; $S<7.2; $S+=0.02, $l++){
# Set Phase (Energy is in S-Mode)
   Set("Energy", $S);
# do some Offset correction...
   Set("Offset00Y",$Yoff+($S-3.3)*0.3/4);
# Set Length to force recalculation with new Energy
   Set("LengthX", $bzx);
   Set("LengthY", 0);
   Echo();
# wait until Energy has stabilized
   sleep(1);
# start scanline
   Scan("line", $l);
```

```
# wait for scanline done
   Echo();
}
# end scan mode
R("scan stop");

# --------------- Users program ends here -----------------------

# say goodbye
R("cmd quit");
R("byebye");
# and close remote session
close(REMOTE);

# ---------------- Program End ---------------------------------

# some usefull subs

# Send Remote Commands in some ways...
# Syntax Example: Set("energy",72,0);

# Syntax Example: R("set energy 72");
sub R {
    REMOTE->printf("$_[0]\n");
    printf("$_[0]\n");
}

# Syntax Example: Set("energy",72);
sub Set{
    REMOTE->printf("set $_[0] %f\n",$_[1]);
    printf("set $_[0] %f\n",$_[1]);
}

# Syntax Example: Scan("line",133);
sub Scan {
    REMOTE->printf("scan $_[0] $_[1]\n");
    printf("scan $_[0] $_[1]\n");
}

# Syntax Example: Rx("file","name.nc",0);
sub Rx {
    REMOTE->printf("$_\n");
```

```
    printf("$_\n");
}

# wait for echo
sub Echo {
   $dummy="";
   REMOTE->printf("cmd echo ready-echo\n");
   printf("ping send !\n");
   REMOTEECHO->read($dummy, 11);
   printf("got echo !\n");
}
```

11 Plug-ins

Plug-ins are shared libary objects with self describing control-structures, that allow a program to use them, load and unload them at runtime.
A plug-in can access all resources created by Gxsm and perform any manipulation with your scans. It is the proposed way to write new algorithms to share with the Gxsm community. See the Plug-in hacking sections in the Hacker guide (16.9)

11.1 Tools/Plugin Details

Plugin Configurator/Viewer.

11.2 Tools/NC Edit

Generic NC-File Viewer.

11.3 Window/DSP Probecontrol

Implementation of a generic `Probe-Method`. No 2D data acquisition, but `probing` at a single point.

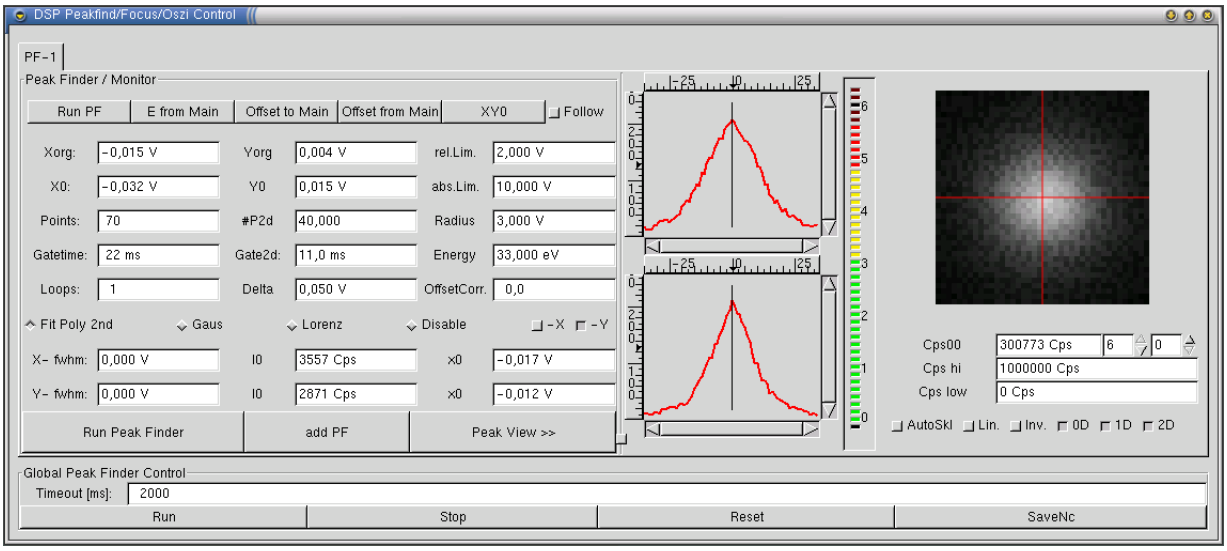
11.3.1 Probing: Force-Distance-Curves

For AFM...

11.3.2 Probing: Spectroscopy, STS

For AFM and STM...

11.3.3 DSP PeakFindControl



For SPA–LEED/SARLS.

12 Tools

12.1 DSP-Program Loader

For writing the DSP-program to the DSP a loader is needed. `loadpci` offers this functionality and can additionally output information from the DPRAM (`DPRAM viewer`) and is also a terminal to the DSP.
Note: Set -d option for the device path when using a 2.4.x kernel.

```
zahl@uranus:~/C/Gxsm/src> loadpci --help
loadpci -[bcCdmqvxrlth] file.out
 -d: path to device (/dev/pcdsp is default),
     devfs: use -d /dev/pcdsps/tms320
 -b: clear_bss
 -c: ULONG-Dump -C C Dump -d: +debugging code
 -m: DPRAM Memorydump
 -q: quite
 -v: verbose
 -x: Prog. Entry Point
 -r: relocate 0xNNNNNN
 -l: load loader/talker and check
 -t: terminal
```

12.2 DSP-Monitor

To survey the DSP there is a monitoring tool (`dsp-applet`) Eight LED inform you about the status of the DSP, an LCD is also supplied to display numerical information.

12.2.1 lcd

xforms DSP monitor variant. TODO.

12.2.2 dsp-applet

Gnome DSP monitor variante as applet. TODO.

12.3 Software oszillograph

Goszi simulates a two-channel memory-oscilloscope with FFT functions, for data acquisition for SSIOD, for calibration of SSIOD, as a focussing tool forr SPA–LEED and for gaining information about quarz counter.
Note: You have to set the correct device.

```
zahl@uranus:~/C++/dsp/Documentation> goszi --help

goszi options
  -s, --DspSoft=spm           Dsp Soft Modes:   'spm'  : assumes afm.out
                              running on DSP, access via /dev/pcdsp (default),
                              'spa'  : assumes spa.out running on DSP, access
                              via /dev/pcdsp,   'dummy': use simulation
                              hardware (osc. sinwave)
  -c, --Ctrl=oszi             Control Modes:   'oszi'    : Oszillocope + FFT
                              Mode,   'dataaq'  : Data-Aquisitaions Mode,
                              'ssiod'   : SSIOD Calibration Mode,   'focus'
                              : SPA-LEED focussing tool Mode,  'counter' :
                              GPIB Counter tool Mode,
```

12.4 Fit frontent for gir

Gfit is a frontend für the script oriented fit-program `gir`.
It is a tool for analysis of large amount of data sets.

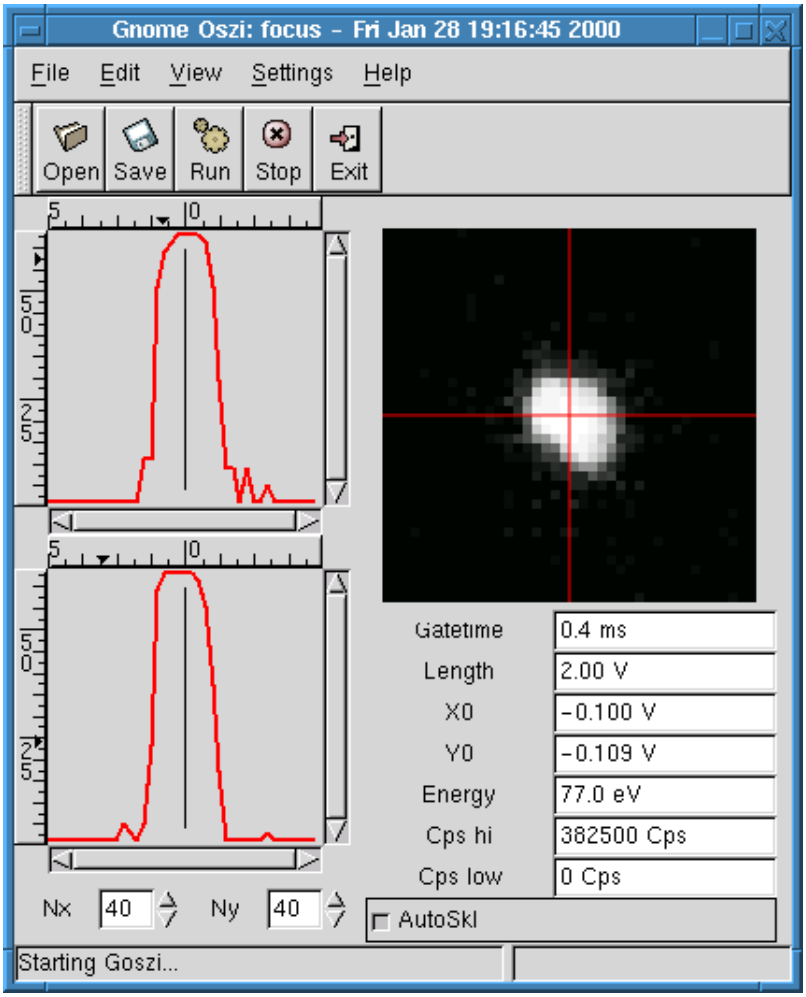


Figure 12.1: Goszi as SPA–LEED focus tool

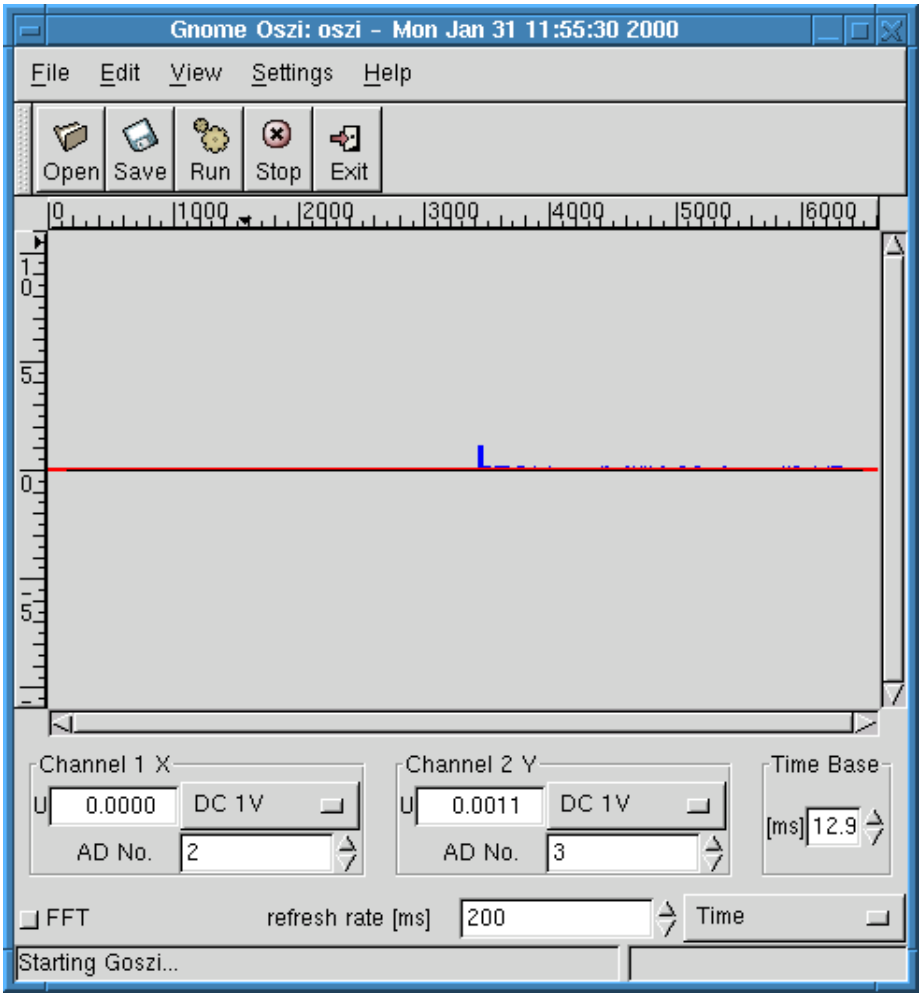


Figure 12.2: Goszi as oszilloscope tool

13 Bildbearbeitung

13.1 Menu Edit

Copy legt eine Kopie des "'Active-Channel"' an.
Crop schneidet aus dem "'Active-Channel"' ein Rechteck oder einen Kreis, je nach Selektion, aus und erzeugt damit einen neuen Channel. Bei Kreis-Crop wird der Bereicht ausserhalb des Kreises mit dem Background Value = [RadiusField] gefüllt.

13.2 Untergrund entfernen: Math/Background

In diesem Abschnitt werden Filterfunktionen zum entfernen von diversen "'Untergrundabarten"' beschrieben. Alle Filter dieser Gruppe sind über den Background-Dialog zugänglich.
Zusätzlich können hier mathematische Operationen mit zwei Scans durchgeführt werden. Z.B. kann der "'Active-Channel"' mit "'X-Channel"' addiert, subtrahiert, multipilziert oder dividiert werden.

13.2.1 Linear Regression 1D

Diese Funktion dient zum entfernen eines geneigten Untergrundes und von Sprüngen zwischen den einzelnen Scanzeilen. Der Filter ist in der Funktion BgLin1DScan() in xsmmath.C implementiert.
BgLin1DScan() zieht von jeder Scanzeile eine Gerade ab, die durch Regression ermittelt wird. Dazu wird der automatische Geradenabzug aus den Darstellungsroutinen verwendet.

13.2.2 Parabolic 1D

Dieser Untergrundabzug dient zum entfernen eines gekrümmten Untergrundes, wie er z.B. bei großen Scanbereichen durch Nichtlinearitäten der Scanpiezos entsteht. Der Filter ist in der Funktion BgPoly2nd1D() in xsmmath.C implementiert.

BgPoly2nd1D() fittet an jede Scanzeile eine Parabel an, die dann von den Daten abgezogen wird. Evtl. Krümmungen senkrecht zur Scanrichtung werden durch den Zeile für Zeile unterschiedlichen Offset der Parabel ebenfalls entfernt.

13.2.3 E Regression

Abzug einer Ausgleichsebene, nur der mit einem Rechteck markierte Bereich dient als Referenz.

13.2.4 Z Drift corr.

Es wird eine Zeilenweise Z-Drift Korrektur ausgeführt. Dafür kann ein X-Intervall (aus rechteckiger Markierung) als Referenz verwendet. In diesen X-Intervall werden alle Z-Werte gemittelt und durch die gewonnene Z(y)-Kurve ein Least-Square-Polynomfit N-ten Grades (bisher Eingabe der Ordnung im Terminal) durchgeführt. Die Ausgleichsfunktion wird zur Zeilenweisen Z-Korrektur verwendet.

13.3 Math/Filter 1D

In diesem Abschnitt werden die 1D Filterfunktionen beschrieben. Diese Funktionen werden Zeile für Zeile auf die daten angewandt und berücksichtigen damit keinerlei Korrelationen zwischen den Scanzeilen. Alle 1D Filter sind über den 1D Filter-Dialog zugänglich.

13.3.1 Invert

Dieser Filter invertiert die Daten, d.h. er führt die Konversion $z \rightarrow -z$ durch.

13.3.2 Phase FFT

Der Name trügt, es verbirgt sich ein Tool um benachbarte Spalten miteinander zu vertauschen – dies ist nur eine Bug-Fix für .dat Dateien vor dem 15.7.1998 die mit xxsm und quantum gescannt wurden !
Aufruf von ColSwp()

13.3.3 Powerspectrum (log PowerSpec)

Diese Funktion berechnet von jeder Zeile das Powerspektrum (Betragsquadrat der Fouriertransformation) und stellt es dann in logarithmischer Skalierung dar. Das Berechnen des Powerspektrums ist in der Funktion F1D_LogPowerSpec in xsm-math.C implementiert. Mit dem entsprechenden 2D Filter kann das Ganze auch in zwei Dimensionen durchgeführt werden.

13.3.4 Lowpass

Führt eine Gauss-Faltung auf jede Zeile aus. Die Breite ist in Pixeln vorwählbar.

13.3.5 Lin. Stat. Diff

Kantenherforhebung durch Differentaion nach folgender Formel: [1]

$$I_i = \frac{1}{9} \sum_{k=i-4}^{i+4} s_k$$

$$t_i = \frac{1}{4} \frac{s_i - I_i}{\sqrt{\frac{1}{2} \sum_{k=i-4}^{i+4} (s_k - I_k)^2}} + \frac{I_i}{2}$$

13.3.6 Koehler

$$L_i = 0.92 s_{i-1} + 0.08 s_i$$

$$R_i = 0.92 L_{i+1} + 0.08 L_i$$

$$t_i = s_i - R_i$$

13.4 Math/Filter 2D

Im folgenden werden die 2D Filter beschrieben. Alle 2D Filter sind über den 2D Filter-Dialog zugänglich.

13.4.1 Despike

Herausfiltern von Spikesartigen Störungen. Funktioniert sehr effektiv ! Filter arbeitet Spaltenweise – d.h. senkrecht zur Scanrichtung. Algorithmus: siehe F1D_Despike() in xsmmath.C – wer mag kann sich das ja ansehen, die Korrektur wird irgendwie aus der lokalen n-fachen Ableitung gewonnen . . . !

[1] Verwendete Symbole:

- s_i: Source = Urbilddatenarray einer Zeile, i: Pixelindex
- t_i: Target = Zielbilddatenarray einer Zeile, i: Pixelindex

13.4.2 3x3 Convolution

Dieser Filter führt eine diskrete Faltung der Daten mit einer 3x3 Matrix durch. Die einzelnen Matrixelemente können in einem Dialogfenster angegeben werden. Der Filter ist über die Klasse F2D_Conv3x3 und die Funktion F2D_Conv3x3_FD() in xsmmath.C implementiert. Die Klasse F2D_Conv3x3 ist in xsmmath.h definiert.
Die eigentliche Faltung wird durch folgende Rechnung durchgeführt:

$$z'(n,m) = \sum_{-1\le i\le 1} \sum_{-1\le j\le 1} z(n+i, m+j) \cdot K_{ij},$$

wobei durch K_{ij} die Elemente der 3x3 Matrix des Faltungskerns beschrieben werden und z bzw. z' für die Daten vor und nach der Filterung stehen. Um Randeffekte zu vermeiden, werden bei der Faltung die erste und letzte Scanzeile und die erste und letzte Spalte abgeschnitten.

13.4.3 logarithmisches Powerspektrum

Berechnet das Powerspektrum, d.h. das Betragsquadrat der Fouriertransformierten, des Scans. Das Powerspektrum wird in logarithmischer Skalierung dargestellt, damit man die Details besser erkennen kann. Die Darstellung erfolgt so, daß die Raumfrequenz (0,0) in der Bildmitte dargestellt wird. Dieser Filter ist in F2D_PowerSpec() in xsmmath.C implementiert.

13.4.4 Allgemeiner Fourierfilter (IFT(X*FT()))

Diese Funktion erlaubt es, beliebige Teile aus der Fouriertransformierten eines Scans zu entfernen. Dazu muß zunächst das Powerspektrum (s. 13.4.3) des Scans berechnet werden (s.o.). Danach kann ein Bereich des Powerspektrums mit Hilfe der Bearbeitungstools ausgewählt werden. Dieser Bereich kann dann mit Hilfe des "Stop"-Buttons im Background Menü entfernt (auf ZEROVALUE gesetzt) werden. Alternativ dazu kann mit dem "Pass"-Button auch alles außerhalb des gewählten Bereiches gelöscht werden. Der so bearbeitete Scan sollte auf Kanal X gesetzt werden (Channel Selector). Die Funktion IFT(X*FT()) berechnet dann die Fouriertransformierte des aktiven Scans, schneidet alle Bereiche heraus, die im Scan X auf ZEROVALUE gesetzt wurden, und transformiert das so modifizierte Forierspektrum zurück. Damit steht ein im Fourierraum arbeitender Filter zur Verfügung, der in der Funktion F2D_iftxft() in xsmmath.C implementiert ist.

13.4.5 Gauß-Stop und Gauß-Pass Fourierfilter

Die Funktionen "FT Gauss Stop" und "FT Gauss Pass" funktionieren ähnlich wie der "IFT(X*FT())" Filter. Sie sind in F2D_FT_GaussStop und F2D_FT_GaussPass

in xsmmath.C implementiert. Bei Ihnen wird das Fourierspektrum des aktiven Scans mit der Funktion $exp(-(r-r_0)^2/R^2)$ (Gauss Pass) bzw. $1-exp(-(r-r_0)^2/R^2)$ (Gauss Stop) multipliziert und anschließend zurücktransformiert. Die Halbwertsbreite und Mittenposition der Gaußfunktionen muß vor Aufruf der Funktionen ausgewählt werden. Dazu sollte zuerst mit der "lg PowerSpec" Funktion das Powerspektrum des Scans berechnet werden. Anschließend kann mit Hilfe des Kreis-Tools die Mittenposition und die Halbwertsbreite der Gaußfkt. markiert werden. Der Radius des Kreises entspricht dann dem R in den obigen Definitionen. Dem Kanal mit dem Powerspektrum sollte danach der Name X zugeordnet werden (mit dem Channel Selector) und der Kanal mit dem Ausgangsscan wieder aktiviert werden. Jetzt muß nur noch der gewünschte Fourierfilter ausgeführt werden.

13.4.6 allgemeine 2D Convolution mit speziellen Kernels

Alle Convolutionsfilter Falten das Bild mittels einer Kernelmatrix der Mindestgröße $2R+1$. Die Mindestgröße wird ggf. automatisch soweit vergrößert, bis Matrixelemente ungleich Null entstehen.
Der allgemeine Convolutionsalgorithmus ist in mem2d.C: MemDigiFilter::Convolve : public Mem2d untergebracht. Die Kernels selbst sind abgeleitete Objekte von MemDigiFilter und stellen die ehemals rein virtuelle Funktion CalcKernel() bereit. Die Filter werden in xsmmath.C ausgeführt, d.h. der entsprechende Kernel wird erzeugt und dann auf die Quelle Src− >mem2d angewendet und das Resultat im Ziel Dest− >mem2d abgelegt. Siehe z.B. F2D_Smooth() in xsmmath.C.
Die 2-D Convolutionskernel stammen noch aus alten Zeiten, d.h. pmstm - fancyfil.c von L.Anderson bzw. seinem Vorgänger ... Wer mag soll sich die aktuellen Kernel in mem2d.C genauer ansehen !

Kernel Smooth

Es wird eine Matrix mit einer Gauß Funktion erzeugt: Tiefpaß

$$K_{ij} = 4 * e^{-\frac{i^2+j^2}{r^2}}$$

Kernfkt: MemSmoothKrn()

Kernel Stat. Diff.

Berechnet im wesentlichen die Ableitung in Scanrichtung. Dies geschieht durch die Faltung mit der folgenden Fkt. (in MemDeriveXKrn::CalcKernel() in mem2d.C implementiert):

$$K_{ij} = C \cdot j \cdot e^{-\frac{i^2}{r_y}-\frac{j^2}{r_x}} \tag{13.1}$$

Kernel Tderive

1-d second derivative, curvature ? [Tderiv_kernel]
Kernfkt: MemTderiveKrn()

Kernel Local Heigth

Es wird ein lokaler Hintergrundabzug vorgenommen. Die Größe der betrachteten Umgebung ist R. Also eine Art Hochpassfilter.
Kernfkt: MemLclhtKrn()

Kernel Curvature

use curvature (second derivative) as height [lderiv_kernel]
Kernfkt: MemCurvatureKrn()

13.5 Math/Transformationen

13.5.1 Rotate (PlugIn)

Needs Rectangle Selection.

13.5.2 Affine (PlugIn)

Needs Koord Obj.

13.5.3 Shear X bzw Y (PlugIn)

13.5.4 Quench Scan

Halbiert Scangröße mittels Zusammenfassung von je 2x2 Pixeln (Mittelwert).

13.5.5 Scale Scan

Scan beliebig in X bzw. Y skalieren.

13.5.6 Mirror X, Mirror Y und diagonal

Spiegelung des Bildes um die x-/ y-Achse bzw. Diagonale.

13.5.7 Add, Sub, Mul, Div X

Math Ops using Active- and X-Channel ...

13.5.8 Merge H, V

Zusammensetzen von Scans in horizontaler bzw. vertikaler Richtung.

13.6 Math/Statistik

13.6.1 Histogramm

Scan Höhenverteilung bestimmen, Resultat erscheint in neuem Math Kanal, zur Ansicht View-Mode auf "Profile 1D" stellen!

13.6.2 HistoHOP: Auswertung von Stufenfolgen

Der histoHOP Filter dient zur Vorbereitung der Auswertung von Stufenhöhen und Terrassenlängen. Der Filter sucht Zeile für Zeile Stufen und berechnet die Länge und Höhe der dazugehörigen Facette sowie die Länge der anschließenden Terrasse. Dieser Filter wurde von Holger Pietsch entwickelt[2]. Der Filter ist in xsmmath.C über die Klasse F2D_HistoHOP implementiert.
Die Detektion der Stufen geschieht durch eine Faltung mit der Funktion

$$K_{ij} = C \cdot j \cdot e^{-\frac{i^2}{r_y} - \frac{j^2}{r_x}} \tag{13.2}$$

(s.a. 13.4.6, Stat. Diff. Kernel Faltung). Nach der Faltung werden alle Bereiche, die den Schwellwert threshold überschreiten als Facetten, die anderen als Terrassen behandelt. Wenn die Filteroption "step detect only" gewählt ist, wird ein binäres Bild erzeugt, in dem dei Facettenbereiche den Wert 1 und die Terrassenbereiche den Wert 0 haben.
Durch den oben beschriebenen Filterprozeß kann das Bild in eine Folge von Stufenhöhen-Terrassenlängen Daten zerlegt werden. Der Zerlegungsprozeß wird durchgeführt, wenn die "step detect only" Option *nicht* ausgewählt ist. Die Daten geben jeweils die Länge und Höhe einer Facette sowie die Länge und Höhe der darauffolgenden Terrasse an. Aus diesen Daten wird das ursprüngliche Bild "rekonstruiert". Durch den Vergleich des Originals mir der Rekonstruktion kann die Qualität der Filterung abgeschätzt werden.
Der Filter an sich bietet keine weiteren Möglichkeiten zur Auswertung der Daten, speichert diese jedoch in einer Datei im ASCII-Format ab. Diese Daten können dann mit externen Programmen ausgewertet werden. Dazu stehen einige kleine Perl-Skripte zur Verfügung:

[2]H. Pietsch, PhD Thesis, Hannover 1997

Skript	Beschreibung
filterhig.pl	Filtert Stufen deren Höhe unter h liegt heraus.
stepheightavg.pl	Berechnet die mittlere Stufenhöhe.
stepheighthist.pl	Erstellt ein Stufenhöhenhistogramm.
terlenavg.pl	Berechnet die mittlere Terrassenlänge.
terlenhist.pl	Erstellt ein Terrassenlängenhistogramm.

Table 13.1: Perl-Skripte zur Auswertung der mit dem HistoHOP Filter erzeugten Daten

13.6.3 Math/Baseinfo

Das Plugin Baseinfo berechnet einige oft gebrauchte Scanparameter. Es muß ein Rechteck oder nichts markiert sein.
Berechnet werden:

- Das Minimum (z-Wert) und seine Position,
- das Maximum (z-Wert) und seine Position,
- die Summe über alle Werte ($\sum z$),
- die Summe über die Quadrate aller Werte ($\sum z^2$)
- der Mittelwert aller Werte

$$\overline{z} = \frac{1}{A} \sum_{x,y} z(x,y) \text{ mit } A := x \cdot y$$

- die rms-Rauhigkeit

$$\begin{aligned} \sigma^2 &= \frac{1}{A} \int_A d\vec{x} \, [(z(x,y) - \overline{z})^2] \\ &= \frac{1}{A} [\sum_{x,y} z(x,y)^2 - 2\overline{z} \sum_{x,y} z(x,y) + \sum_{x,y} \overline{z}^2] \\ &= \frac{1}{A} [\sum z(x,y)^2 - 2\overline{z} \sum z + \overline{z}^2 \sum 1] \\ &= \frac{1}{A} [\sum z(x,y)^2 - 2\overline{z} \sum z + \overline{z}^2 A] \end{aligned}$$

Bei der Begutachtung von σ ist auf ein gutmütiges Verhalten des Mittelwertes zu achten. Auf der Standardausgabe werden die ermittelten Werte ausgegeben, um ein Markieren mit der Maus zu ermöglichen. Der Pixelmode ist z.Z. nicht implementiert.

13.6.4 Math/Step counter

Dieses Plugin zählt die Anzahl der Stufen, die größer als 256 a.u. sind. An jedem Ort (x, y) wird der z-Wert mit dem rechten Nachbarn verglichen. Falls Differenz größer 256, wird dies als Aufwärts-, bzw. Abwärtsstufe gezählt. Das Ergebnis landet auf der Konsole.

13.7 Math/Misc

Diverses...

13.8 Hinzufügen von Math Ops

Neue Math Operationen können via PlugIns auf sehr einfache Weise hinzugefügt werden. Siehe Abschnitt PlugIns im Hacking-Kapitel 16!

14 STM-HOWTO (Quantum-Pott)

14.1 Scanner abnehmen und Transfer

1. Probenhalter und Probe aus dem STM entfernen
2. Scanner halb hochfahren und mit Wobbelstick vorsichtig abziehen (Magnethalterung)
3. Scanner auf Transferblock (Pinöppel zeigt nach oben, in der Schleuse zum Transferstab hin) setzten.
4. Scanner in die Schleuse bringen, beim Durchgang von dem Ventil zur Schleuse die Transfergabel etwa 30° kippen.

14.2 Ätzaufbau in Betrieb nehmen und Spitze ätzen

14.2.1 Ätzgerät Aufbau, Plan

1. Beide VA-Ringe in die Ätzschale einsetzen
2. Ätzschale zu $\frac{2}{3}$ mit 2n-Natronlauge füllen

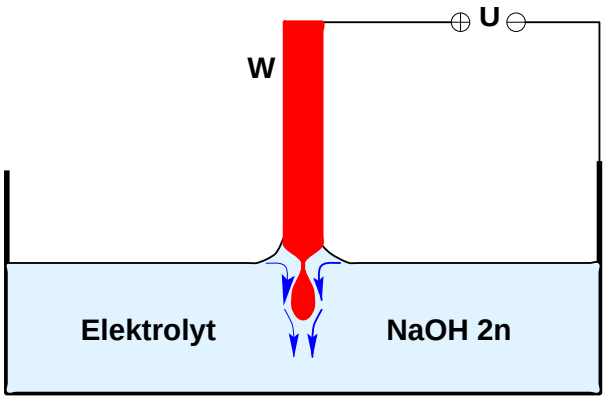


Figure 14.1: Ätzaufbau, siehe Diplomarbeit P.Zahl

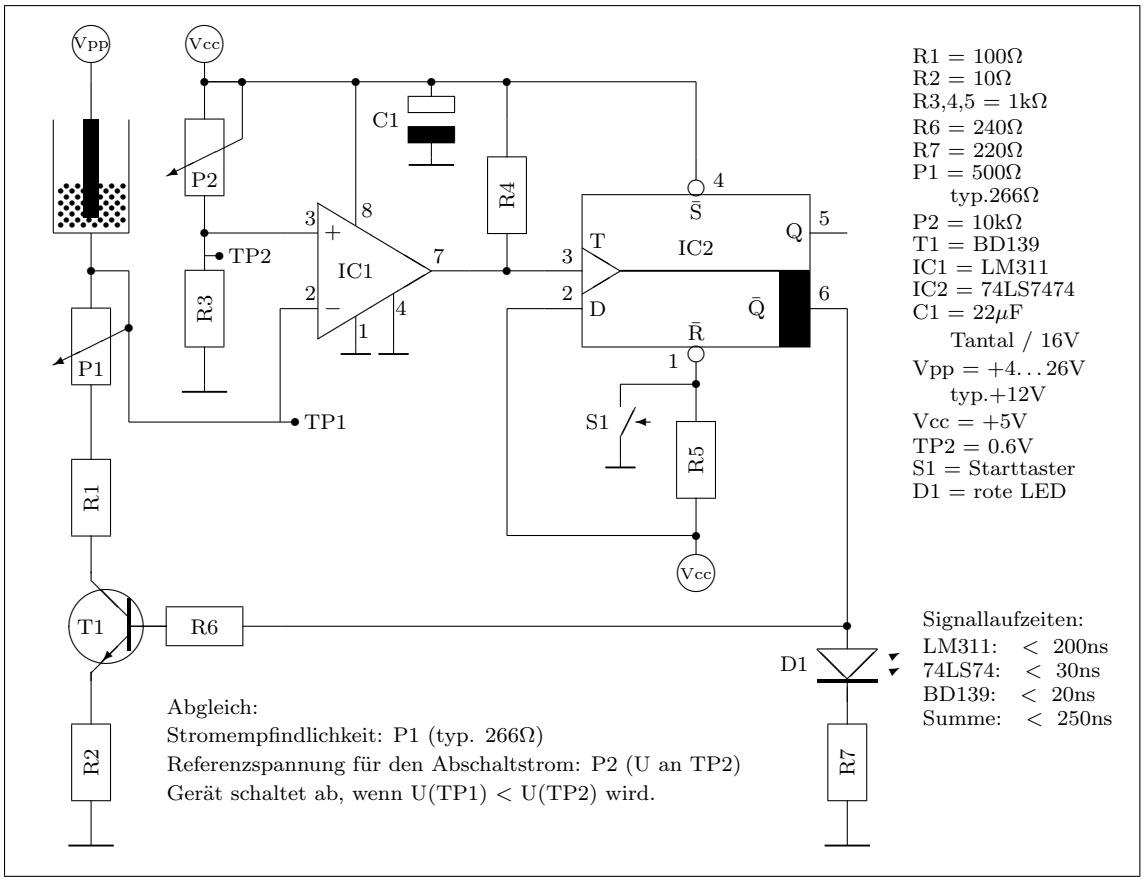


Figure 14.2: Spitzenätzschaltung, Version vom 15.8.1996

3. Elektronik anschließen: (meine freifliegende Platine + Netzteil + Ätzaufbau)

   - Verkabelung prüfen:
   - Blaues Kabel mit Krokoklemme an Minus-Elektrode (Blechring) anschließen.
   - Rotes Kabel (Plus-Pol vom Netzgerät) an den Ätzaufbau anschließen. (ist mit dem Spitzehalter (VA-Kapilare) verbunden)
   - +5V an dem schwarzen Kästchen prüfen:

     – Zuerst unbedingt die Kabel zur Ätzschaltung abziehen !
     – Stromversorgung über Batterieklip (Klip an Klip...) zum Netztgerät (rot: Plus, blau: Minus)
     – Netzgerät auf 12V stellen.
     – Rote Buchse: +5V
     – Gelbe Buchse: 0V

   - ggf 5V neu einstellen (mit Drehknopf)
   - Kabel zur Ätzkiste wieder anschließen.

4. 0,25mm Wolframdraht (ca. 20cm Länge) mit Propanol abwischen und in Ätzhalter (VA-Kapillare) einführen.

5. Den Draht etwa 6mm aus der Kapillare herausstehen lassen und ausrichten (parallel zur Kapillare, **sehr wichtig !**).

6. Den Draht ca. 4mm in die Lauge tauchen und kurz anätzen. (wird blank)
   Dazu:

   - Netzgerät (NG) auf 20V stellen
   - Start Taste (gelbe Taste auf der Ätzplatine) drücken (rote LED geht an)
   - Anätzen beobachten (Blasen am Draht)
   - NG auf 12V stellen, dabei sollte die LED auch ausgehen ! (falls nicht: NG abstellen)

7. Drahtende auf Oberflächenkontakt einstellen und dann genau 2mm eintauchen

8. Bei typ. 12V am NG den Starttaster drücken (ggf. Staubdach aufsetzen) und warten bis die LED ausgeht.
   (Keine Vibration, kein Luftzug etc. während des Ätzens !!!)

9. Sichtkontrolle: Spitzenrest abgefallen und Spitze OK ?
   Falls nicht: Etwas Draht nachschieben und erneut ...

10. Spitze mit Halter ausbauen

11. Spitze unter sehr flachem Winkel in dest. Wasser tauchen und vorsichtig schwenken

12. Mikroskop: Spitze im Halter lassen und Spitze begutachten.
    Es sollten keine Absätze, Knubel, etc. im Profil zu sehen sein. Es darf nur ein gleichförmig auf die Spitze zulaufender sauberer Bogen erscheinen, an der Spitze sollte die Auflösung des Mikroskops scheitern und es dürfen nur Interferenzen zu sehen sein.

13. Spitze OK ? Falls ja: Spitze mit einer Pinzette halten und auf etwa 10mm Länge abschneiden.
    Falls nicht: Etwas Draht nachschieben und erneut ...

14. Spitze in Spitzenhalteblock abstellen – fertig.

14.3 Glühen der Spitze

1. Schleuse mit N_2 belüften
2. Spitze mit einer Pinzette in die Heizvorrichtung (Mo-Blech Wendel in der Probengarage) einlegen. Dabei die Spitze nach außen hervorstehen lassen und beim Ablegen darauf achten, daß die Spitze nicht herunterfällt. (etwas knifflig)
3. Schleuse abpumpen, warten bis $p < 10^{-5}$ mbar
4. Verkabelung zum 30A Centro prüfen !
   (**zwei Plus Leitungen mit zwei Fliegenbeinen an Pin 4 (P1) der Probengaragendurchführung und eine dicke Masseleitung**)
5. Die Wendel mit ca. 10 . . . 15A zur Rotgluht bringen und abstellen. Die Spitze sollte kurz nachleuchten.
6. Schleuse mit N_2 belüften
7. Die Spitze nochmals im Mikroskop begutachten.

14.4 Einsetzen der Spitze in den Scanner

1. Scanner (mit Transferblock) aus der Schleuse holen und auf Papier absetzten.
2. Alte Spitze aus dem Scanner entfernen, dazu mit kleinem Schraubendreher die Halteschraube lösen.
3. Neue Spitze auf Länge schneiden (ca. 5mm) und einsetzen, kleine Schraube leicht anziehen.
4. Scanner in die Schleusengabel setzten.
5. Schleuse abpumpen, warten bis $p < 10^{-5}$ mbar
6. Scanner mit der Gabel über einem der Heizstäbe der Probengarage positionieren
7. Heizung anstellen (ca. 40W)
8. Mindestens 5h heizen
9. Scanner zum STM transferieren und einsetzten (siehe Scanner-Ausbau).

Probe präparieren etc., und in STM setzten. Dazu die Spitze **ganz nach unten** fahren !

14.5 STM Programm starten und Spitze annähern

1. Oszi: TimeBase=20ms, CH1=U_z, 5 oder 2V/Div, CH2=I_{Tunnel} , 50mV/Div
2. (old: nur OS/2) Den DSP-Ordner auf der OS/2 Arbeitsoberfläche öffnen.
3. (old: nur OS/2) Programm-Symbol DSP-Start aufrufen.
   Meldung: Programm läuft, Taste drücken.
   OK: DSP-Programm läuft jetzt und an der Koppelbox blinkt die LED "'Blink"'
4. xxsm Programm mit Hardwaresupport starten, (ggf. mit Script stm, htstm, ...)
5. DSP-Annäherungsparameter: vier Werte prüfen (Richtwerte: 18; 0,08s; 0,05%; 0,1%)
6. DSP-Regelparameter: (Richtwerte: 2V; 0,4nA; CP=0,01; CI=0,012)
7. DSP-Komandopanel: Regler Start, U_z läuft gegen Min-Wert (Piezo ausgestreckt)
8. Manuelle Annäherung mit visueller Kontrolle im Mikroskop
   - Drehknopf (Poti) an der Piezo-Handsteuerung auf 10 (schnell)
   - Mikroskop auf kleinste Vergrößereung stellen, ggf. Beleuchtung optimieren
   - Probe etwa mittig unter der Spitze plazieren.
   - **Achtung hohe Spannung:**
     auf keinem Fall die Piezoantriebe der Probenverschiebeeinrichtung mit dem Probenhalter berühren !
   - Spitze an die Probe grob heranfahren (Spiegelbild wird sichtbar)
   - Poti auf 3,1 stellen
   - Mikroskop auf maximale Vergrößerung stellen
   - Spitze bis auf einen virtuellen Abstand von 1mm annähern (ca. 1/120mm)
   - Lampe ausschalten, Luftfüße auf Funktion überprüfen, alle unnötigen Bodenkontakte vermeiden.
9. Drehknopf (Poti) an der Piezo-Handsteuerung auf 3,16 (Quantum)
10. Piezoverstärker Einstellungen überprüfen: X,Y=2, Z=5, Z-Offset=5,0 (Mitte)
11. DSP-Komandopanel: Spitze nähern starten
12. Oszi beobachten: 18 Pulse, Regler ein: Spitze nähert sich, Spitze zurück, etc.

13. Bei Erfolg: Automatischer Übergang in den Betriebsbereiten Zustand, Regler aktiv
14. Bei Vorzeitigem Abbruch (Störung wurde als Tunnelstrom gedeutet) gibt es keine Tunnelstromanzeige, dann das Annähern erneut starten !
15. Das wars !
16. viele Bilder aufnehmen ...
17. Hinweis: Die Annäherungsstelle ist oft schlecht, X,Y-Offset etwas verstellen !
18. Sicherheitshinweise:
    - Z-Piezospannung immer positiv (typ: 0...+80V) (Dann kein Crash bei Stromausfall !)
    - U_{Z-DA} (Oszi CH1) im Mittel immer um Null herum, nie größer 4V ! (ab +5V Crashgefahr !!!!)
    - Bei starker thermischer Drift immer U_Z beobachten, ggf. Offset korrigieren. Sonst Crashgefahr !!!
    - Abs. Tunnelspannung nie größer 8V ! (Ab besteht 10V Gefahr den Tunnelstromverstärker zu killen !!!)
    - Nie die X/Y Sliderpiezos mit der Probe / dem Probenhalter berühren, Sicherheitsabstand von 1mm einhalten !! (Hohe Spannungen beim ansteuern der Piezos killen den Bias Voltage Amplifier)
    - Beim tunneln nie die STM-Anlage berühren, Crash-Gefahr !
    - Den X,Y,Z-Offset nie schlagartig verstellen, Crash-Gefahr !

14.6 Spitze zurückziehen

1. Z-Offset auf 0,0 (U=-150V)
2. DSP-Komandopanel: Spitze entfernen, Poti auf 10 stellen
3. Warten bis Spitze ganz unten angekommen ist
4. Poti auf 1 (zur Sicherheit)

15 AFM-HOWTO (allgemein)

15.1 AFM – Justage

1. Lever einbauen, grob justieren
2. Laser auf Lever focussieren
3. Laser auf PSD-Mitte zentrieren

15.2 Lever-Annäherung

1. von Hand Probe kurz vor den Lever bringen
2. mit Auge / Camera / Mikroskop via Slider annähern
3. Z-Kraf Meßsignal beobachten und vorsichtig weiter annähern (langsam !!)
4. Regelparameter optimieren
5. Kraft-Abstandskurve / -Verhalten studieren ...
6. messen, messen, messen

15.3 Meß-Modi

15.3.1 Contact-Mode

15.3.2 non-Mode

16 Internals: Xxsm and Gxsm Hacker Guide

16.1 Software Requirements

Xxsm and its descendant Gxsm were developed using “Debian/GNU Linux 2.2”. Though all included software should be portable to any fairly recent distribution.

16.1.1 Compiler

For compilation any ANSI C++ Compiler should be vaild. Program development was done using the GNU C++ compiler, Gnome, the GTK+ toolset together with the Helixcode (now Ximian)-extensions (see http://www.ximian.com) in the following versions.
g++ –version
2.95.2
gtk-config –version
1.2.8
gnome-config –version
gnome-libs 1.2.8
If you use Debian, too, you may use this entry in your `/etc/apt/sources.list` to obtain the necessary development packages:
deb http://spidermonkey.helixcode.com/distributions/debian unstable main

16.1.2 Libraries

The old and old-fashioned Xxsm version uses the following libraries:
libX11 (tested with version 6.1 XFree86), libXext (tested with version 6.3 XFree86) and the ANSI C math-libraries libm (tested with version 5.0.9). Both common C-libraries libc5 and libc6 (glibc2) can be used. So far no special libc6 functions are used.
The graphical user interface uses the XForms-library in version 0.88. For creation of the interface is done with the accompanied form designer ’fdesign’.

Gxsm needs the complete Gnome/Gtk+ development packages. Right now (2001) we are using the Helixcode/Ximian-Gnome-development-packages.
With Xxsm a new dataformat was introduced, known as NetCDF. As a consequence, we need its libraries newer or equal to version 3.4 (libnetcdf and libnetcdf_c++ for the C++-frontend). Documentation can be found at http://unidata.ucar.edu /packages/netcdf/index.html.
Math routines, which use Fourier transforms, additionally neet the fftw-libraries libfftw and librfftw (¿=2.0). These routines provide algorithms for the fast calculation of complex and real Fourier-transforms. Xxsm can be compiled with fftw-support disabled. If you compile without the preprocessor directive HAS_LIBFFTW, all functions using fftws functions will not be executed. More documentation about fftw can be found at http://theory.lcs.mit.edu/˜ fftw/.

Hint: All development-packages and sources for successful Gxsm-compilation are available as Debian

The so called rem Remote-Control-Option (see below) needs the the PThread Libary (Linux-threads) for background tasks; Debian has this packed with the standard C-libraries.
For control of external devices which have a GPIB-bus (IEEE-488) the libgpib Libary (linux-GPIB) is needed (Xspa, goszi):

```
Title = The Linux GPIB-Package
Version = 2.02
Desc1 = Driver Module for National Instruments GPIB (IEEE488) boards,
Desc2 = pcIIa (alpha), and HP82335 HPIB boards, C-Interface library,
Desc3 = Tcl/Tk frontend.
Author = Claus Schroeter
AuthorEmail = clausi@chemie.fu-berlin.de
Maintainer = Claus Schroeter
MaintEmail = clausi@chemie.fu-berlin.de
Site1 = ftp.llp.fu-berlin.de
```

(Hint: Not yet available for 2.2.X kernel!? Please check.)
For three dimensional display the OpenGL library, resp. its free clone MesaGL is used. This is also part of the standard Debian distribution.

16.2 Internal structure of Gxsm/Xxsm

Several objects are defined globally, you can access them from any part of the program. Those objects are defined in glbvars.h.

16.2.1 XSM_Instrument class

The XSM_Instrument class provides some instrument-specific functions.

There is only one object of this class (XSM_Intrument *Inst, see glbvars.h) which can be accessed globally.

The main purpose of XSM_Instrument is the definition of unit conversion from internal units to SI-units. Examples are conversion from digital to Ångstrom (fct. double Dig2XA(long dig)) or vice versa (fct. double XA2Dig(double ang)). Another task is the calculation of D/A-resolutions (fct. double XResolution()).

16.2.2 Gxsm Klassenhirarchie

handy Application Base Class and Tools				
gapp_service.h:	class	MyGnomeTools		
gapp_service.h:	class	AppBase	: public	MyGnomeTools
gapp_service.h:	class	DlgBase	: public	MyGnomeTools
gapp_service.h:	class	GnomeAppService	: public	AppBase

Main Internal Class and derivation				
xsm.h:	class	Xsm		
surface.h:	class	Surface	: public	Xsm

Instrument properties, holded by main object				
instrument.h:	class	XSM_Instrument		
instrument.h:	class	STM_Instrument	: public	XSM_Instrument
instrument.h:	class	AFM_Instrument	: public	XSM_Instrument
instrument.h:	class	SPALEED_Instrument	: public	XSM_Instrument

Strucured Data Classes		
xsmtypes.h:	class	XsmRescourceManager
xsmtypes.h:	class	Display_Param
xsmtypes.h:	class	Scan_UserInfo
xsmtypes.h:	class	SCAN_DATA
xsmtypes.h:	class	MkIconsData
xsmtypes.h:	class	PrintPSData

Main Scan Object and specializings				
scan.h:	class	Scan		
scan.h:	class	TopoGraphicScan	: public	Scan
scan.h:	class	SpaScan	: public	Scan

Main Application Object, holds “surface” object				
gxsm_app.h:	class	App	: public	GnomeAppService

sub GUI Objects, holded by App				
gxsm_app.h:	class	ChannelSelector	: public	AppBase
gxsm_app.h:	class	SpaLeedControl	: public	AppBase
gxsm_app.h:	class	DSPControl	: public	AppBase
gxsm_app.h:	class	GrOffsetControl	: public	AppBase
gxsm_app.h:	class	MoverControl	: public	AppBase
gxsm_app.h:	class	DriftControl	: public	AppBase
gxsm_app.h:	class	MonitorControl	: public	AppBase, Monitor
gxsm_app.h:	class	HistogrammControl	: public	AppBase
gxsm_app.h:	class	OptiControl	: public	AppBase
gxsm_app.h:	class	OsziControl	: public	AppBase
app_mkicons.h:	class	MkIconsControl	: public	DlgBase
app_print.h:	class	PrintControl	: public	DlgBase
app_profile.h:	class	ProfileElement		
app_profile.h:	class	ProfileControl	: public	AppBase, LineProfile1D
app_remote.h:	class	RemoteControl	: public	DlgBase, public remote_crtl
app_view.h:	class	ViewControl	: public	AppBase
app_vobj.h:	class	ViewInfo		
app_vobj.h:	class	VObject		
app_vobj.h:	class	VObPoint	: public	VObject
app_vobj.h:	class	VObLine	: public	VObject
app_vobj.h:	class	VObParabel	: public	VObject
app_vobj.h:	class	VObRectangle	: public	VObject
app_vobj.h:	class	VObCircle	: public	VObject

File IO				
dataio.h:	class	Dataio		
dataio.h:	class	DatFile	: public	Dataio
dataio.h:	class	GnuFile	: public	Dataio
dataio.h:	class	NetCDF	: public	Dataio
dataio.h:	class	PsiHDF	: public	Dataio
dataio.h:	class	DumpableNcFile	: public	NcFile
epsfutils.h:	class	EpsfTools		
epsfutils.h:	class	SPM_epsftools	: public	EpsfTools
epsfutils.h:	class	SPA_epsftools	: public	EpsfTools

Units and Input management				
unit.h:	class	UnitObj		
unit.h:	class	LinUnit	: public	UnitObj
unit.h:	class	BZUnit	: public	UnitObj
unit.h:	class	SUnit	: public	UnitObj
unit.h:	class	CPSCNTUnit	: public	UnitObj
pcs.h:	class	Param_Control		
pcs.h:	class	Gtk_EntryControl	: public	Param_Control

Math				
regress.h:	class	Zeile		
regress.h:	class	LGSys		
regress.h:	class	StatInp		
bench.h:	class	bench		
xsmmath.h:	plain C	all filters		
xsmmath.h:	class	Filter		
xsmmath.h:	class	F2D_Conv3x3	: public	Filter
xsmmath.h:	class	F2D_HistoHOP	: public	Filter
lineprofile.h:	class	LineProfile1D		
histogramm.h:	class	Histogramm		in progess

Event Logging				
monitor.h:	class	Monitor		

2D Memory Management				
mem2d.h:	class	LineInfo		
mem2d.h:	class	ZData		
mem2d.h:	template	<class ZTYP>	class	TZData : public ZData
mem2d.h:	class	Mem2d		
mem2d.h:	class	MemDigiFilter	: public	Mem2d
mem2d.h:	class	MemSmoothKrn	: public	MemDigiFilter
mem2d.h:	class	MemLclhtKrn	: public	MemDigiFilter
mem2d.h:	class	MemDeriveXKrn	: public	MemDigiFilter
mem2d.h:	class	MemCurvatureKrn	: public	MemDigiFilter
mem2d.h:	class	MemTderiveKrn	: public	MemDigiFilter

Remote Control				
remote.h:	class	remote		
remote.h:	class	remote_crtl	: public	remote

Data Representation/non GUI				
view.h:	class	View		
view.h:	class	Grey2D	: public	View
view.h:	class	Profiles	: public	View
view.h:	class	Surf3d	: public	View
xshmimg.h:	class	ShmImage2D		

Hardware Abstractation				
xsmhard.h:	class	XSM_Hardware		
xsmhard.h:	class	dspobj	: public	XSM_Hardware
xsmhard.h:	class	dspdev	: public	XSM_Hardware
xsmhard.h:	class	dspspm	: public	dspdev
xsmhard.h:	class	dspspa	: public	dspdev
xsmhard.h:	class	ccdobj	: public	XSM_Hardware

16.3 2D data management

16.3.1 General Information

With the extension of the data aquisition for SPA-LEED and the demand for more universal 2D data management the need for more datatypes than 16bit-short arised.

To meet this, a class-structure (see mem2d.h) was created which offers:

- dynamic 2D memory-handling of any scalar typ: use of BYTE, SHORT, ULONG, LONG, FLOAT, DOUBLE, (no COMPLEX type, it's not scalar) types is available,
- all covered in uniform base class "Mem2d",
- access through purely-virtual base class members.
- math operations and type-independent data manipulations are covered by special virtual member functions, all optimized for speed
- simple Get/PutDataPkt(int x,y) data access functions
- or interpolated data access via GetDataPktInterpol(double x,y)
- potential use of 'memory copy' (memcpy) within same type is possible to put and get data from and to the mem2d object (Get/PutDataLine)
- copying rectangular data windows between same type mem2d objects is supportet, for different type objects a slower routine with automatic type conversion is available (CopyFrom/ConvertFrom),
- getting data via a transferfunction for displaying in several representations: "direct" (scaled(contrast)+offset(bright)), "quick" with line regression (with caching of fit parameters), logarithmic, periodic (modulo mode), differential and horizontal (line average value is subtracted),
- finding High and Low value in a given area, possibly accellerated using subgrid,
- DataRead/Write to raw stream,
- Memory field resizeing,
- convolution with other data fields.

16.3.2 Details of several classes

The class LineInfo stores and managed the line regression data cache.
class ZData → class TZData[Typ] uses the template class TZdata to provides the type independet data handling used by the Mem2d class, which is used externally to hold the 2D data.

16.3.3 Description of classes

Class LineInfo

Storage of statistical information on every line. (Offset and gradient for line regression: $Y = ax + b$)

Class ZData

Base class, with only virtuel members.
Members:
constructor: ZData(int Nx=1, int Ny=1)
int GetNx() returns number of points in X.
int GetNy() returns number of points in Y.
size_t Zsize() returns the size of the elements: sizeof(element-TYPE)
double Z(int x, int y) returns the value at position (x,y).
double Z(double z, int x, int y) sets the value at position (x,y) and returns it.
void Zadd(double z, int x, int y) adds z to the value at position (x,y).
void Zmul(double z, int x, int y) multiplies with z.
void Zdiv(double z, int x, int y) divides by z.
int Resize(int Nx, int Ny) (???)
void ZPutDataLine(int y, void *src) copies an array of data from src to line y (Warning: care for the datatype!).
void ZPutDataLine(int y, void *src, int mode)
void ZGetDataLine(int y, void *dest) copies a line of data from y to src (Warning: care for the datatype!).
efficient Datamanipulation:
void SetPtr(int x, int y) Set internal pointer to position (x,y). Warning: *there is no range check!, increasing x++ may never cross the border of the set!*
double GetNext() Returns the value of the internal position, then the pointer jumps to the next datapoint (x++).
double GetThis(double x=0.) Returns the internal position.
long GetThis(long x), unsigned long GetThis(unsigned long x), SHT GetThis(SHT x), unsigned char GetThis(unsigned char x) for special (internal) use.
Hint: Neighbouring points can be addressed! See also mem2d.h.
void SetNext(double z) Sets the value at the current position to z, then moves on (x++).
int CopyFrom(ZData *src, int x, int y, int tox, int toy, int nx, int ny=1) Copies data from source *src to itself in a rectangular fashion. (x,y) is the upper left corner, (nx,ny) is the size, (tox,toy) is the upper left of the target position.
void* GetPtr(int x, int y) Warning: Not for external use!
Overloaded Operators: Warning: All following operations are related to the current position pointer (x,y).

void operator = (ZData &rhs)
void operator += (ZData &rhs)
void operator -= (ZData &rhs)
void operator *= (ZData &rhs)
void operator /= (ZData &rhs)
void operator ++ () increase internal position x++
void operator – () decrease internal position x–
double operator [] (int idx) returns value of (internal x-position + idx).
Warning: the resulting index must be valid!

Class TZData : public ZData

template class, derived from ZData. See ZData !

Class Mem2d

Control class, keeps an instance of class ZData[currentType].
Constructors:

```
Mem2d(int Nx, int Ny, ZD_TYPE type=ZD_SHORT)

Mem2d(Mem2d *m, MEM2D_CREATE_MODE Option=M2D_ZERO)

size_t GetEsz() Returns size of element (sizeof(\dots)).

ZD_TYPE     GetTyp() Returns type of element (see enum ZD_TYPE).

const char* GetEname() Returns name of element as string.

int GetNx(), int GetNy() Returns the Arraysize.

void Modus(int mod) See Ablagemode (*)???

int  Resize(int Nx, int Ny, ZD_TYPE type=ZD_IDENT)

void PutDataLine(int y, void *src)

int CopyFrom(Mem2d *src, int x, int y, int tox, int toy, int nx, int ny=1)

int ConvertFrom(Mem2d *src, int x, int y, int tox, int toy, int nx, int ny=1){

void PutDataLine(int y, void *src, int mode)
```

```
void   GetDataLine(int y, void *dest)

double GetDataPkt(int x, int y)

void   PutDataPkt(double value, int x, int y)

double GetDataPktInterpol(double x, double y)

double GetDataPktLineReg(int x, int y)

double GetDataPktHorizont(int x, int y)

ZVIEW_TYPE GetDataVMode(int x, int y) Value in displaymode.

double GetDataMode(int x, int y) Value in datamode.

ZVIEW_TYPE ZQuick(int &x, int &y), ZDirect(\dots), ZLog(\dots), ZPeriodic(\dots),
ZHorizontal(\dots) Functions for display.

void SetDataPktMode(int mode)
See Displaymode.

void SetDataRange(ZVIEW_TYPE Min, ZVIEW_TYPE Max)
Set area of valid display.

void SetDataSkl(double contrast, double bright)
Sets display parameters.

void AutoDataSkl(double *contrast, double *bright)
Calculates displayparameters from min and max.

void CalcLinRegress(int yfirst, int ylast)

int  GetDataLineFrom(Point2D *start, Point2D *end, Mem2d *Mob, SCAN_DATA *ScDat)

void HiLoMod(Point2D *p1=NULL, Point2D *p2=NULL, int Delta=4)

void HiLo(double *hi, double *lo, int LinReg=FALSE, Point2D *p1=0, Point2D *p2=0)

void DataRead(ifstream &f)
```

```
void DataD2DRead(ifstream &f, double gate)

void DataWrite(ofstream &f)

int  DataValid()

int  SetDataValid()
```

16.3.4 Objects: rectangles, circles, etc.

The Gxsm user can mark arbitrary areas within 2D views using objects such as rectangles, circles, and lines. These objects can be accessed from the Gxsm core or plug-ins using the interface classes/methods described below.
To get the number of objects attached to a Scan object use the memberfunction `unsigned int number_of_object ()` of the `Scan` class. `Scan` also provides the function `scan_object_data* get_object_data (int i)` to get the i-th object represented by the class `scan_object_data`.
The `scan_object_data` class is defined in `scan.h`. It offers several member functions to access the represented objects. Only the most important are described below:
`gchar* scan_object_data::*get_name ()` returns the type/name of the object. `int scan_object_data::get_num_points ()` returns the number of points the object is containing. For instance, rectangles are represented by two corner points. The coordinates x, y of the point i can be retrieved by `void scan_object_data::get_xy (int i, double &x, double &y)`.
The coordinates used here are in "real world" units, i.e. in Å and inclusive scan offsets.
For further details, see `scan.h` and `scan.C` of the Gxsm core source-code. An application example can be found in the `spectrocut` plug-in source-code.

16.4 Data Input and Output

In- and Output of aquired data is implemented in dataio.C. The interface is defined by the baseclass `dataio` (see `dataio.h`).

16.4.1 Files in dat-format

The ancient dat-format from PMSTM (or even older?) was replaced by the more flexible and easier extendable NetCDF-format (see 16.4.3). The input and output of the old format is defined in datio.C

The dat-format consists of a header which contains for example the scansize and the data. The body follows without further introduction after the header and contains the information on every datapoint as 16 value line by line. There exist three different versions of dat-files, which can be distinguished by their header-ids (see DatFILE::Read() in datio.C). Die different headers have the same length, so that the body of data always begins at the same place.

16.4.2 "GNU" Files

The in- and output of "GNU"-files is implemented in the GnuFile class in datio.C. The files of this group are listed in table (16.1)

ID	description
d2d	D2D format from SPA-LEED program
byt	byte format: raw 8 Bit data
sht	Short format: raw 16 Bit data
flt	Float format: floats, single precision
dbl	Double format: floats, double precision
pgm	Portable Greymap (P5) format
tga	TARGA bitmap format (8, 16 and 24 Bit colordepth)

Table 16.1: "GNU"-formats

As time of this writing only the d2d-format was importable (???), all other formats (except d2d which is read-only) can be exported.

16.4.3 NetCDF Files

To reach better portability and extensibility on 3.8.1998 the NetCDF-file format was introduced into the project. (See http://unidata.ucar.edu/packages/netcdf) All needed routines for handling NetCDF-files are stored in dataio.C. NetCDF is the default data-format for storage of data collected with Gxsm.
To pervent the loss of the primary filename of your data (through renaming) the original filename is stored within the NetCDF file since dataio.C v. 1.24 with the name 'basename'. The variable 'basename' is declared in the class SCAN_DATA in xsmtypes.h On start-up basename is set to 'unknown' (???) (this happens in function Surface::DoScan() in Surface.C). Upon your first save this variable is set to the given filename and no more changed. If Gxsm reads a file without basename information it is set to "No basename information available." (See NetCDF::read() in dataio.C)

16.4.4 PSI HDF Files

Our institue has access to a Park Scientific Instruments (PSI) Atomic Force Microscope. [1] This leads to the need for import-ability of Gxsm for the data format used by this device. The PSI/AFM data format is compatible to NCSA HDF 3.3 format for which development libraries are freely available. (See ???)
Implementation of the PSI HDF in Xxsm was difficult because of the following obstacles:

1. The NCSA HDF lib includes parts of the NetCDF (see 16.4.3) format. Compatibility is compromised.

2. PSI used a format which is basically HDF compatible but contains several hooks: Several "TAGs" were redefined, so that generic read functions from the HDF libraries do not work correctly.

For this reason, PSI HDF files are read directly without interference of the accompagnied librariy. This is accomplished through the "dump" of data into the PsiHEADER data structure, as defined in psihdf.h. The implementation can be found in dataio.C and dataio.h.
The current (???) implementation of the PsiHDF class of course depends on the version of the software used for writing the files and is tested only with version 1.1

16.4.5 Autosave

The Autosave-feature[2] used the internal counter `counter` for automatically naming the files in AUTOSAVE_MODE.
To allow the automatic naming of backup (intermediate) scans a second counter `subcounter` was defined (see xsm.h, xsm.C).
In xsmtype.h the gchar *AutosaveUnit and gchar *AutosaveOverwritemode and gint AutosaveValue were added.
The most interesting parts happen in surface.C.
At the beginning of a scan (Surface::DoScan) the `nextAutosaveEvent` is set to `AutosaveValue`. During the scan, the runtime conditions of seconds, percentage or lines are checked. If the appropriate variable overtook the nextAutosaveEvent, the file is saved with the special parameter '2' and with overwrite permissions set or unset, depending on AutosaveOverwritemode. In `Surface::save` we read, that 'autoon == 2' is the identifier for autoscans. Here the additional naming strings are attached and the counters are increased (if counter is increased, subcounter is set to zero).

[1] This instrument was bought after initiation of the "Innovationsoffensive Nanoelektronik" and is maintained by the institute for semiconductors and located in the institute for information technology, all in Hannover, Germany.
[2] See also users-guide

Note: Almost all changes done for this new autosave modes can be found when grepping for 'utosave'.

16.5 Defaultvalues for parameters

Several parameters have to be adjusted for correct performance of an instrument (e.g. scansize). To simplify software-setup default Xxsm can store values for many parameters in .xsmvalues in the users home-directory which will be read and set on program launch.
Gxsm saves all parameters using a "recource manager" in the "~/.gnome/gxsm" file. Have a look at it.

16.5.1 Defaultvalues at programm statup (Xxsm only)

When Xxsm starts, the function Xsm::loaddefaults() is executed (see xsm_main.C). Default parameters can be inserted and added there, preferably at the end.

16.5.2 Storing defaultvalues in .xsmvalues (Xxsm only)

Individual parameter-sets are stored in .xsmvalues in your home-directory. Those are determined *after* static default values are assigned to all parameters (see Xsm::loaddefaults()) to prevent an uncertain state if .xsmvalues is missing. Reading and writing of these parameters s implemented in Xsm::loadvalues() and Xsm::savevalues() in xsm_main.C.
The fileformat for .xsmvalues is straightforward. Key-Values pairs are dupmped commentless into the file. For correct rereading the succession is important. This implies, that new parameters have to be added at the end of the file.
Warning: There is no error detection. Wrong insertions in this file may leave Xxsm in an undefined state.

16.6 Inserting a new dialog (Xxsm only)

This section describes how a new dialog can be introduced to Xxsm after being designed with fdesign.
The arguments of the individual forms of the dialog must be inserted in actionid.h. The obligate use of the enum structure ensures the correct assignment of values to arguments.
The dialog must be inserted in xsm_main.h with an entry of the form FD_¡name of dialog in fdesign¿ and a second entry with FD_¡name of fialogs in fdesign¿Cpy.
The dialogs is created with a constructor in xsm_main.C. The memory allocated for the copy Cpy must be freed in the destructor.

The dialog is now inserted into the program but has no functionality. To add this, a callback function must be inserted in xsm_cb.C. The name of the function must be equal to that used in fdesign. To open and close the dialog at appropriate places the xforms functions fl_show_form() and fl_hide_form() are used. (Examples see xsm_menu_cb.C) Alternatively use the macros FL_SHOW_FORM() and FL_HIDE_FORM() if a definit placement is recommended.

16.7 Add X Resources (Xxsm only)

X resources (see XSm.ad and .Xdefaults) provide a convinient method for configuration of Xxsm. This section describes how to add a new resource.
First of all, the new resource has to be created in xsmtypes.h. Simply add it into the XSMRESOURCES struct.
Next, the resource must be prepared. Insert the resource in xsm_main.C in xsm_res_def. Every resource is defined with five parameters.

1. The name, as it is entered in X resource database. (See XSm.ad.)
2. The class name (unused?).
3. The datatype of the resource (int, float or string).
4. A pointer to the variable carrying the value.
5. A default value.

Just in front of xsm_res_def the preprocessing constant XRES_COUNT must be set to the number of resources.
The resources may be set using command line options. To make use of this functionality, an entry has to be added to xsm_cmd_opts. Like in xsm_res_def the maximum number of command line parameters must be set in CMD_COUNT.

16.8 DSP-Card PC31 and PCI32 – TMS320

16.8.1 Kernel Module for PC31 and PCI32

Hint: The device information is valid for versions using kernel 2.2, the newer versions use the new devfs!
[XG]xsm and the DSP tools use the device /dev/pcdsp (???) for communication with the signal processor. A kernel module was developed to handle this communication. Depending on the available card the appropriate modules pc31.o or pci32 are inserted into the running kernel using insmod. This can be done at system startup (place a simple script in /etc/re.boot for Debian systems). The root user may insert the

module at any time if you don't like this. check correct insertion with lsmod. Remove the module with rmmod pc31 or rmmod pci32 respectivly.
If the device /dev/pcdsp does not exist the insmod will fail. Root has to create the device with
"mknod -m 666 /dev/pcdsp c 127 0"

16.8.2 A very quick intro 'how it works' by PZ

The PC31/PCI32 DSP maschine is booted in some multistep way:

1. the kernelmodule should be loaded by "insmod pc31 or pci32" have a look at /var/log/messages ! ⟶ now /dev/pcdsp should be aktive
2. the program "loadpci" starts and openes /dev/pcdsp
3. a small loader is transfered to the dsp and started (some differencies PC31/PCI32 !!)
4. the xafm.out (COFF-File) is interpreted and uploaded section by section using the loader
5. the loaded program is launched via reset / jump
6. now DSP bootes up via int_00
   ⟶ look at ⟶ "boot.asm" (boot31/32.lib) for _c_int_00 [pci32/dspC32/periph31/rts/]
7. some initialisations, call _ii_init, call _main, call _exit (ii_bo31.c/ii_boot.c)
8. if main() returns _exit is called. – bye

16.8.3 xafm.c

After uploading and starting "xafm.out" we are here in _main() after exec of ii_init() !
1. Setup Hardware
2. install Interrupt
3. start main dsploop

- main loop: dsploop()
  - check for host cmd request (if CMD_MODE enabled) ggf. exec cmdmanager()
  - do moving, scanning, approching, etc...
  - do no blocking things here !!!

- in background (interrupt):
  - feedback_isr() is running
  - do no blocking things here !!!

. . . that's it - easy ? :=)

16.8.4 OLD: DSP-Kommunikation

stmdspos.c

Compile with the special DSP compiler. It initializes the hardware and sets interrupt 09, then enters the infininite loop 'dsploop'. The feedback is controled by an interrupt routine 'c_int09()'. The control frequency depends on the timer 0 and must leave enough processing time for the main 'dsploop'.
The gouverning variable is 'STMMode'. It contains a bitcode, which determines the current processing mode of the STM system.
All constants MD_XXX are bitmasks for these modes. If a bit is set the according mode is active.
Example: if(STMMode & MD_CMD) printf("Modus CMD ist aktiv");
This variable is instantanously mapped to port PIA_A and can be used for status control with a number of leds. That way a full control of the program flow can be obtained.

MD_CMD: Command evaluation mode, the SRQ-register is read and the given command evaluated.

MD_PID: Feed-Back-Control runs.

MD_MOVE: A MoveTo(Xnew, Ynew) is executed.

MD_SCAN: Linescan is exectued.

MD_BLK: Blinking of blinker, for administrative program control.

MD_TIPDN: Automatic/manual tip approach.

MD_ITU: Tunnel current detected.

MD_CRASH: Tunnelcurrent cannot be driven to desired value. The current is higher than the desired values: risk of tip crash!

The single modes interact, e.g. the start of a linescan disables MD_CMD, the end reenables it.
A quick calculation of the logarithm is vital for the tip control, because of the exponential dependency between distance and tunnel current. Math-libraries log function is to slow, so a replacement 'MyLog()' using a table and linear approximation is used.

dspcom.c (old non Objective Version - similar)

Provides service functions DSP ↔ PC. Additional dialog procedures (for DSP-menu) implemented.

struct DspParam Mirrors all important parameters, which are used for the DSP program.

PC31memput(Adr, Dlow, Dhigh) Transmit 32-bit data (2*16bit Dlow, Dhigh) to the dsp memory beginning with adresse Adr.

PC31Fmemput(Adr, Value) Transmit a float to the dsp memory. A special format is used (32 bit).

PC31memget(Adr, *Dlow, *Dhigh) Transmit 32-bit data (2*16bit Dlow, Dhigh) from the dsp memory beginning with adress Adr to the PC.

PC31reset() Reset for PC31.

int PC31acked() True, if DSP completed last command. (PC31 acknowledged)

int PC31sqred() True, if DSP sent a request to the PC. (PC31 Service Request to PC)

PC31srq(value)

value = 1 Commandrequest to DSP. (Post Service Request to PC31)

value = 2 Simulated DSP request at PC. (not used)

PC31ack() Answer to DSP request. (acknowledges a PC31 service request. i.e. says to PC31 that service has been completed)

WaitDSPacked() Wait until completion of DSP's last command.

If no is DSP available or the DSP program is not running the request is terminated after DSP_TIMEOUT seconds (20s).

Hint: During scan have some patience if response is not immediate. Long timeouts are necessary, because lengthy linescans may not be interrupted with DSP commands.

WaitDSPsrqed() Waits for DSP message to PC (not used).

DSP_SetWerte() Transmit all control parameters on DSP.

DSP_SetLogWerte() Transmit parameters for the logarithmic control characteristic to the DSP.

DSP_SetRotParam() Transmit parameter for image rotation.
(???) ToDo: Move-Offset PM-Programm und Rotationsoffset im DSP-Programm sinnvoll einsetzten. (jetzt: Drehung um Ursprung bezogen auf U_X/Y , nicht um Bildmitte !)

DSP_SetMoveParam() Transmit the dynamic parameters for Moveto-movements.

DSP_SetLineScanParam() Transmit the dynamik parameters for Linescan-movements and data collection.

DSP_SetLSP(nx, dx) Set Linescanlength (nx) and dotdistance (dx). Calls DSP_SetLineScanParar

DSP_SetAppWerte() Set parameters for tip approach mechanism.

LoadDSPParameter() Load DSP parameter set from file.

SaveDSPParameter() Save DSP parameter set to file.

DSP_Par_DlgProc(...) Dialogbox control function for generic parameters (Control, Log., Move, Scan)

DSP_Cmd_DlgProc(...) Dialogbox Control function for DSP commands.

DSP_APPParam_DlgProc(...) Dialogbox control function for tip approach parameters.
(Control, Log., Move, Scan)

Wirings – Quantum

```
*
*      Hardware Connections:
*
*      Digital IO      82C55
*      ================================================================
*   *) PIA_A: Status (Variable == STMMode)     (IOPA0..7)
*      Bit     0       1       2       3       4       5       6       7
*      MD_...  CMD     PID     MOVE    SCAN    Blink   TIPDN   ITU     CRASH
*      Description at #define MD_XXXX
*      For Display with LED's, using driver-circuit ULN2003 o...
*
*   *) PIA_B:  Controlbox, release of X,Y,Z +/-, speed (IOPB0..7)
*      Bit     0       1       2       3       4       5       6       7
*              X+      X-      Y+      Y-      Z-      Z+      R1      R2
*      release of R1,R2: fixed resistors for (???) Raupen-Schrittweite,
*      sonst Potiwert
```

```
*
*   *) PIA_C:  Diverses        (IOPC0..7)
*      Bit     0       1       2       3       4       5       6       7
*              Test    Test    Gate    Oszi-   -       -       -       -
*                              Trigger Trigger
*                                      for approach control
*      Analog IO
*      ========================================================================
*      DA0:    Z-Piezo (control variable)
*      DA1:    tunnel current          (bauer BNC plugin (Omicrom))
*      DA2:    X-Piezo
*      DA3:    Y-Piezo
*      AD0:    tunnel current (control variable)  (red BNC Stecker (Omicron))
*
```

16.9 Plug-in interface

16.9.1 How it works

PlugIns are "on the fly" loadable add on libs, simply using the shared object libary format. But there is a special design, which allows Gxsm to make automatically use of there libs. Gxsm doese not know of the lib itself, but it first searches at predefined locations for Gxsm-PlugIns libs trys to open it dynamically and looks for the PlugIns Descriptor Function

```
GxsmPlugin *get_gxsm_plugin_info ( void )
```

.
If it the Symbol "get_gxsm_plugin" is resolved, gxsm calls this function. The return value is a pointer to the static PlugIn-Descripto Table as defined below. Then Gxsm evaluates the Type of the PlugIn and installs it in the desired way and adds it's Descriptor to a List for future use – e.g. to remove it again.
First play a bit with the PlugIn "Plugin Details" found at Tools Submenu. Expand the box pressing "Details"! This will show the PlugIn Control Structure of eatch Plugin currently loaded, found and inspected by Gxsm.
There are three main types of Plugins: Run-Cb, Math-Plugin, and Query-Selfinstall. Additionally there is a Category mechanism, which allow to autoselect Plugins to load. E.g. we will no need a Mover-Control Plugin, if we are running without hardware.
Lets have a look at the PlugIns Descriptor Table (defined in Gxsm/src/plugin.h):

```
/*
```

```
 * Gxsm PlugIn Infostruct:
 *
 * PlugIn Function Call:
 *   (void*) get_plugin_info( void )
 * should return a valid Pointer to "struct GxsmPlugin" as defined here:
 *
 */

typedef struct
{
  void *handle;           /* Filled in by Gxsm */
  char *filename;         /* Filled in by Gxsm */
  int  Gxsm_session;      /* The session ID for attaching to the control socket */
  App *app;               /* Calling Application Object */
  char *name;             /* unique Plugin name */
  char *category;         /* Plugin's Category - used to autodecide on Pluginloadi
  char *description;      /* The description that is shown in the preferences box
  char *authors;          /* Plugins author */
  char *menupath;         /* The plugins menuposition to append to */
  char *menuentry;        /* The plugins menuentry */
  char *help;             /* The plugins help text */
  char *info;             /* Additional info about Plugin */
  char *errormsg;         /* Plugin Status Message */
  char *status;           /* Plugin Status, managed by Gxsm */
  void (*init) (void);    /* Called when the plugin is enabled */
  void (*query) (void);   /* Called after init and app is set. */
  void (*about) (void);   /* Show the about box */
  void (*configure) (void); /* User Configuration */
  void (*run) (GtkWidget *, void *); /* if run != NULL this handler is installed a
                                        default, else for special Gxsm-Plugins is se
  void (*cleanup) (void); /* Called when the plugin is disabled or when Gxsm exits
}
GxsmPlugin;


/*
 * Gxsm special PlugIns...
 */

/* Math Plugins */

/* returned by
```

```
 * (void*) get_gxsm_math_one_src_plugin_info( void )
 */
typedef struct
{
  BOOL (*run) (Scan *Src, Scan *Dest);
}
GxsmMathOneSrcPlugin;

/* returned by
 * (void*) get_gxsm_math_two_src_plugin_info( void )
 */
typedef struct
{
  BOOL (*run) (Scan *Src1, Scan *Src2, Scan *Dest);
}
GxsmMathTwoSrcPlugin;

/* returned by
 * (void*) get_gxsm_math_one_src_no_dest_plugin_info( void )
 */
typedef struct
{
  BOOL (*run) (Scan *Src);
}
GxsmMathOneSrcNoDestPlugin;
```

16.9.2 HowTo make a new PlugIn

A simple way to start a new math plug-in is to use the shell script `generate_math_plugin.sh` from the `plugins` directory in the Gxsm source code. This script will prompt you for some information and then generates a ready-to-compile source code template for you.
If you prefer doing the stuff by hand, follow the steps below:
first:

```
cd Gxsm/plug-ins/math/
```

the file HACKING is here:

```
Plugin Hacking Guide          20001124PZ
```

```
----------------------------------------

Step One:
Have a look at ./statistik/vorlage.C,
and copy it into the best matching subdir, one of

background  filter1d  filter2d  misc  statistik  transform

using a new plugin name. (here: "myplugin.C")
The file "myplugin.C" includes a quick GxsmPlugin Guide, please follow...


This is found at top of vorlage.C:
--- snip ---
 * Quick nine points Gxsm Plugin GUIDE by PZ:
 * ------------------------------------------------------------
 * 1.) Make a copy of this "vorlage.C" to "your_plugins_name.C"!
 * 2.) Replace all "vorlage" by "your_plugins_name"
 *     --> please do a search and replace starting here NOW!! (Emacs doese preserv
 * 3.) Decide: One or Two Source Math: see line 54
 * 4.) Fill in GxsmPlugin Structure, see below
 * 5.) Replace the "about_text" below a desired
 * 6.) * Optional: Start your Code/Vars definition below (if needed more than the
 *       Goto Line 155 ff. please, and see comment there!!
 * 7.) Fill in math code in vorlage_run(),
 *     have a look at the Data-Access methods infos at end
 * 8.) Add vorlage.C to the Makefile.am in analogy to others
 * 9.) Make a "make; make install"
 * A.) Call Gxsm->Tools->reload Plugins, be happy!
 * ... That's it!
--- snip ---

Change into the subdir!

Step Two:
Adding your myplugin.C to the Makefile.am, located in the same directory:

1.) The Makefile.am has a list of all libs to build, add "libmyplugin.la":

lib_LTLIBRARIES = \
        libhistogram.la \
        libhistoHOP.la \
```

```
        libspasim.la \          <=== the line before should end with "\" !!
        libmyplugin.la          <=== add "myplugin" as here

2.) add the compiler instructions like:

libmyplugin_la_SOURCES = myplugin.C [mypluginhelp.C, ...]  <=== here you cann add mor
libmyplugin_la_LDFLAGS = -export-dynamic -avoid-version

That's all to do for Makefile.am.


Step Three:
Build:

make
make install

Test:
Start gxsm   or call   Gxsm->Tools->Reload Plugins
Check Plugins Description, [About, Configure] using the "Gxsm->Tools->Plugin Details'
Test Plugins action


Step Four:
please publish!

If you are a SF-Gxsm project member:
sfcvs update
sfcvs add myplugin.C ...
sfcvf commit

or send Plugin-Srcs to zahl@users.sourceforge.net
```

16.9.3 Probe Modes (PlugIn)

Implementation of a generic "Probe-Method" with regard to the specialties of AFM, STM, SPA-LEED. No 2D data-ccollection, but "Probing" at one point/small area.

Probing: Force-Distance-Curves

For AFM...

Probing: Spectroskopie, STS

For AFM and STM...

16.9.4 Peak fit and find algortithms, "aka Beam-Find" (PlugIn)

For SPA-LEED and perhaps SARLS.

Parabel least Squares Fit

XY-scans, parable fit.

Gauss Fit

Log. transformation, parable fit.

Lorenz Fit

Inv. transformation, parable fit.