System Design

System Design of SaX
SuSE advanced X-configuration

Project, Implementation and Maintenance
by Marcus Schaefer (ms@suse.de) *

Author: Marcus Schaefer
Datum: June 16, 2005

*SUSE LINUX AG, Maxfeldstrasse 5, D-90409 Nuremberg

Contents

1 The SaX Principle, based on X11 R6, Version 4.x

Contents

In contrast to the existing SaX version (2.8), a three-layered model was developed for configuring X11. Briefly outlined, a registration of hardware is performed in the first layer, in the second layer the actual configuration takes place, which can either be completely automated or be done manually, using the information gathered up till now. The third layer serves to optimize the position and size of the image, after a successful configuration and start of the X server.

A further important difference to the existing version is that the graphical interface provides, as a result, a strictly formatted file which is then imported, and which can be used to create the actual configuration file. This approach allows the graphical interface to be developed in a completely transparent manner. In this case it also doesn't matter with what means this was created. Whether this was done with ncurses, Tk, Qt or whatever it is based on, you must only ensure that the result of the configuration of a file matches the corresponding format. The format of this interface is described in chapter C.

The implementation of the new SaX version is based on the languages **C** and **Perl**. Many small tools, for adjusting the mouse, for example, or for parsing various formats, were implemented in C. For reasons of performance, the storage and the rereading of the hardware registration was also implemented in C. The principle and procedure of the configuration, writing the configuration file and the GUI which SaX itself suggests, were implemented in Perl or Perl-Tk.

1.1 Level 1: Init

Init is represented by **init.pl** and takes on the following tasks:

- Creating the principle file structure and determining default settings.
- Detection of hardware with reference to PCI/AGP graphics cards, pointer devices, keyboard and monitor. The actual hardware scan is started by a sysp (System-Profiler) call-up. Sysp is an independent program which is given its

functionality through insertable modules, which in turn can take on a specific task. In the case of SaX, only modules were developed for detecting the above-mentioned components. A description of the individual modules can be found in chapter 2.

- Entry of data into the hardware registry. By means of functions from the perl module *AutoDetect.pm*, the information provided by sysp is integrated into the file structure. The result from default settings and hardware data corresponds to the hardware registry.
- Saving the registry. The registration data is then saved in its current form as a binary stream in */usr/share/sax/files/config*.

init.pl is usually not called up manually but via a startup script, described at the end of this chapter. There, all available options are listed in detail. It is always necessary to run init.pl whenever hardware has been changed, whether by inserting another graphics card, or just changing the mouse. In normal cases the hardware registry is only created once, which considerably speeds up the starting of SaX.

1.2 Level 2: XC

XC (X-Configuration) is represented by xc.pl and takes over the following tasks:

- Reading the hardware registry. Reading in the registry is done very quickly, since the data has already been saved in the form of a hash. Data can thus be read in directly to a perl hash and processed further.
- By means of the functions from *CreateSections.pm*, a first automatic X11 file is created from the hardware registry. This configuration is then used to start the X server. If the user is satisfied with this configuration, no further configuration tool needs to be started. The user can then choose if he wants to change the position and size of the image, start SaX, abort the configuration or save the current configuration. The mode tuning is supported by a perl module with the name *XFineControl.pm*. This module is used both at this point as well as later on in SaX, since the actual program *xfine* which makes changes to the screen, merely protocols the changes to the modeline in */var/cache/xfine/*.... The exact structure of the files in */var/cache/xfine/* and how xfine functions is described in chapter 9.
- Starting the graphical interface. The interface itself in turn makes use of the data stored in the hardware registry. Depending on whether an already existing configuration file should be read in or not, either data is used from the registry, or information from the configuration file. The reading in of an already existing configuration file is done by a perl module called *ParseConfig.pm*. This module represents an extension of the Perl interpreter and is based on the **libxf86config.a**, which is included from version 4.x of X11 R6. The interface itself only has the task of creating a Variables-API file, from which an X11 configuration file is newly created, through a later import, using the *ImportAPI.pm* module in conjunction with the *CreateSections.pm* module. The newly created X11 configuration file is used to start a second instance of the X server. On this server the tuning tool xfine is started, to be able to make possible changes in

position and size. Saving the configuration file is done through the SaveAndExit() function, which is a part of the SaX interface. This function, however, is only set off by xfine, which should prevent a non-functioning configuration file from being stored in the system.

1.3 xw

In the SaX2 running procedure there are a number of points at which one or more X servers need to be started. Starting a new server is always done by the **xw** program. This program was originally a Perl script, later replaced by a C program. The name *xw* is derived from **xw**rapper. The program fulfills the following tasks:

- Starting the X server in its own process. Here the first option is interpreted as the logfile name, the remaining options are passed on to the X server. The server is started in a process created by xw through fork(). The output of the X server in the child process, which is written to the STDERR channel, is diverted to the logfile by a freeopen() call. xw itself (father) remains in the foreground and waits.
- Only when the SIGUSR1 signal has been received, which is sent by the X server to the calling process, is the server ready to work. If the signal arrives, then the father process is ended, whilst the child process (X server) continues to function. The process number of the child process is here output to STDOUT.

1.4 Startup Script

The coordination of the processes **init.pl**, **xc.pl** and **xfine.pl** is controlled by the startup script *sax.sh*, which is located in */sbin* and is usually called up by the wrapper script */sbin/SaX2*. The wrapper here takes over the following tasks:

- Creating a secure, temporary directory with **mktemp**.
- Testing for root permissions. If a normal user calls up the program, then authentication is required. This is done by the xauth mechanism, via the *sux* command, or via the xhost mechanism, if sux does not exist. In both cases the root password has to be given.
- Testing the X11 link environment. From version 4.x. of X11 onwards there are various xinit calls to start the X server. The original */usr/X11R6/bin/xinit* is now just a link to one of the two following programs:
  - /usr/X11R6/bin/xinit3x for X11 R6 v3.3.x
  - /usr/X11R6/bin/xinit4 for X11 R6 v4.x

  The script tests and sets this link, if it is needed. A corresponding message and query is output, if the link does not point to xinit4.
- Calling up the actual SaX2 startup script, **sax.sh**

The ability to pass on options to the startup script is provided by the sum of options from init.pl and xc.pl. The following options are available:

- `-f | --forceinit`
  With this option the hardware registry is deleted and then newly created. If you include this option it will force init.pl to be called up.

- `-b | --batchmode <interactive|profile>`
  With this option the so-called batch mode is activated. In this case SaX will not start immediately, but depending on the parameters, will open an interactive shell or allow the input of data from STDIN. Starting a shell is done via the optional parameter *interactive*, with *profile* SaX awaits data from STDIN. The batch mode allows it to access the configuration procedure directly. Changes in the batch mode are also stored in the registry. The batch mode requires data to be in a special format. This format is briefly described if you enter *help*. It is very important, however, to understand what changes were made, and where. Examples of using the interactive shell and the STDIN mode can be found in Appendix A.

- `-a | --auto`
  With this option the automatic configuration is started. This means that SaX works in the background and creates an automatic configuration from the current registry data. In this case, no X server or configuration interface is started.

- `-l | --lowres`
  With this option the DDC detection is switched off. This means that possible information provided by the monitor on its resolution options will not be used. SaX will then start in 640x480 VGA mode.

- `-m | --modules`
  With this option, a server module can be assigned for each graphics chip. An example should illustrate its use:

  ```
  SaX2 -m 0=vga,2=mga
  ```

  Chip 0 is assigned to the vga module and chip 2 to the mga module. Which chip number matches which card can be seen with the option -p.

- `-c | --chip`
  With this option you can determine which chipsets should be used for configuration. For graphics cards with just one graphics processor, this option is the same as the number of graphics cards to be used:

  ```
  SaX2 -c 0,2
  ```

  Use the cards with the chips 0 and 2. It should be especially noted that the option -c changes the order of module allocations. If, for example, cards 0 and 2 out of 3 graphics cards are to be used, and here the modules for these cards are also to be set by options, then this will be given as follows:

```
SaX2 -c 0,2 -m 0=vga,1=mga
```

The sequence of module allocation always increases by a step of one.

- `-p | --pci`
  With this information, SaX outputs the result of the PCI/AGP detection. The output is important in determining which chip number was assigned to which graphics card.
- `-d | --display < Display-Number >`
  With this option the number of the display to be used can be defined. It should be noted here that this does not denote the format of an X display, but just a number. If SaX is to be started on display 5, then the command will be as follows:

  ```
  SaX2 -d 5
  ```

- `-x | --xmode`
  This option instructs SaX not to calculate any modelines. In this case the modelines installed in the X server will be used to start SaX.
- `-u | --automode`
  This option instructs the server to search for the best mode itself. This means that SaX does not write any resolution to the start configuration, but leaves this to the server to choose the mode.
- `-n | --node`
  With this option the device node for the main mouse cursor can be set.
- `-t | --type`
  With this option the protocol to be used for the main mouse cursor is given.
- `-g | --gpm`
  This option activates the gpm as a repeater. Then SaX uses *MouseSystems* as the mouse protocol and as the device node, the fifo */dev/gpmdata* provided by GPM.
  NOTE: Currently this option does not work for X1 R6 v4.0-based X servers.
- `-o | --noscan`
  This option creates a new registry from the current data of the system profiler (sysp) without carrying out a hardware scan.
- `-w | --nowarn`
  This option suppresses the warning messages on the use of a possibly existing registry.

1.5 Diagram of procedures

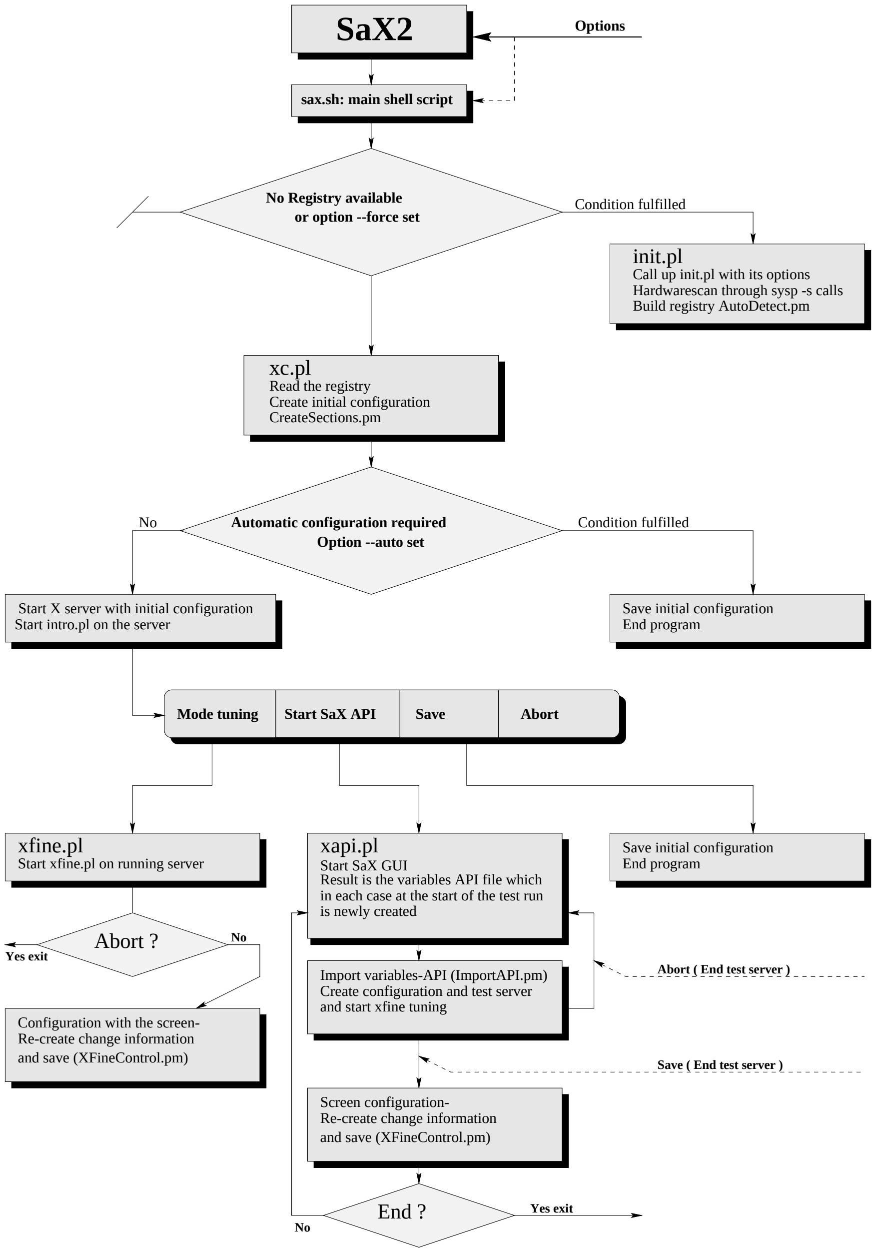


Figure 1.1: SaX2 Flow diagram

2 Sysp

Contents

Sysp stands for **system profiler** and is an independent program for detecting hardware data. Sysp is constructed modularly and saves once detected data in so-called perl DBM hash files. The data in these files can be read out and processed again through a sysp call. Detecting and saving information is a central part of *init.pl*. This action is carried out once by SaX2 during the initialization. Use is made of the sysp data, which as a whole makes up the SaX registry, at various points in the configuration process:

- In *xc.pl* to create the initial configuration.
- In *xapi.pl* to make data visible in configuration dialogs.

Information which was detected by individual sysp modules is stored in the directory

- /usr/share/sax/sysp/rdbms

and re-read from there as well.

2.1 Sysp Modules

In the course of development for SaX2 the following sysp modules were written:

- **Keyboard**
  This module determines, using the KEYTABLE variable in /etc/rc.config, which type of protocol can be used under X11. With systems such as Sparc, for example, a direct hardware scan is started to detect this data.
- **Mouse**
  This module determines the connection and protocol of all pointer devices connected to the system. A condition of this is that these are PnP-capable pointer devices, which also provide a checkback signal.
- **Server**
  This module determines all PCI/AGP graphics cards which are inserted in the PCI or AGP bus. Furthermore, it attempts to set an X11 R6 v4.x module allocation by means of the unique vendor and device ID‘s of these cards, as well as finding special options and extensions. If a single ISA card isa used, then it attempts to find out, by registry dump, which X11 R6 v4.x module can be responsible for this. If there is a mixture of AGP, PCI and ISA cards, the ISA cards will definitely not be automatically detected.

- **Xstuff**
  This module collects card-specific data, such as video memory, RamDAC speed, possible resolutions per DDC and the synchronization range of the monitor in accordance with the resolutions detected. To detect this data a minimal X11 configuration is created from the result of the preceding sysp module, and the X server is started for a brief test run. The output of the server is then processed and provides the above-mentioned information. The minimum configuration for this server start can be found under */tmp/config.sax*.

2.2 Calling up sysp

Calling up sysp does not have to be regulated in a script, but can also be run by hand. For this case, two modes should be differentiated:

- Starting a query
- Starting a hardware detection (scan)

Sysp Query

The following command can be used to query data:

- `sysp -q <Name of the sysp module>`

Sysp Scan

The following command can be used to start a hardware scan:

- `sysp -s <Name of the sysp module> [ -o options ]`

When scanning, it also possible, using the option `-o` , to pass on options for the scan. The option list always contains a colon as a separator. An example of such a command could look like this:

- `sysp -s server -o all:0=mga,1=ati`

3 ISaX

Contents

ISaX stands for **interfacing sax** and is a program to transport information from or to the engine. It is possible to query information from isax as well as give information to isax which is then able to create or modify the X11 configuration. If asking isax for data there are two possible data sources:

- The auto probed values from the sax registry
- The current configuration represented by the file */etc/X11/xorg.conf*

3.1 ISaX Modules

The option **-b** is used to obtain data from the sax registry. If no option is set the current configuration is used. The engine of SaX is based on seven sections which can be queried and manipulated:

- **Keyboard**
  defines all information about the core keyboard
- **Mouse**
  defines all information about mice
- **Card**
  defines all information about the graphics hardware
- **Desktop**
  defines all information about the desktop which includes settings like resolution and colordepth
- **Path**
  defines all information about search paths for fonts and special flags for the X-Server
- **Layout**
  defines all information about the server layout which includes information about multihead arrangements as well as priority lists for keyboard and pointers
- **Extensions**
  defines all information about new X-Server extensions

3.2 Calling up isax

Calling up isax does not have to be regulated in a script, but can also be run by hand. The following command can be used to query data about the graphics hardware from the current configuration:

```
/sbin/isax -l Card
```

When obtaining data from the SaX registry the call for graphics hardware will look like this:

```
/sbin/isax -l Card -b
```

3.3 Creating configurations with isax

As mentioned in the first section of this chapter isax can be used to create or modify X11 configurations as well. To do this it is necessary to specify a so called **apidata** file. Detailed information and an example of such a file can be found in appendix C (The Variables API File). The important point here is to mention that the information for reading and/or writing data with isax provides a common interface for all operations which can be done with sax. The following command can be used to create a new configuration from the information specified in the sample file */var/lib/sax/apidata*:

```
/sbin/isax -f /var/lib/sax/apidata -c /tmp/myconfig
```

When only modifying based on the currently installed configuration the command will look like the following. The apidata file in this case contains only information about the changes which should be migrated with the current configuration data.

```
/sbin/isax -f /var/lib/sax/apidata -c /tmp/myconfig -m
```

Both examples will create an output file specified with the option **-c**. In this case this results in a file named **/tmp/myconfig**.

4 libsax

Contents

Between the SaX GUI and the SaX engine an interface exists to transport the information from the GUI into the engine which is then able to create or modify the X11 configuration. This interface is called **ISaX**. A complete explanation about how SaX2 is structured can be found within the documentation at */usr/share/doc/packages/sax2*. The ISaX interface is the basis for the C++ library explained here. The library is based on the following major topics:

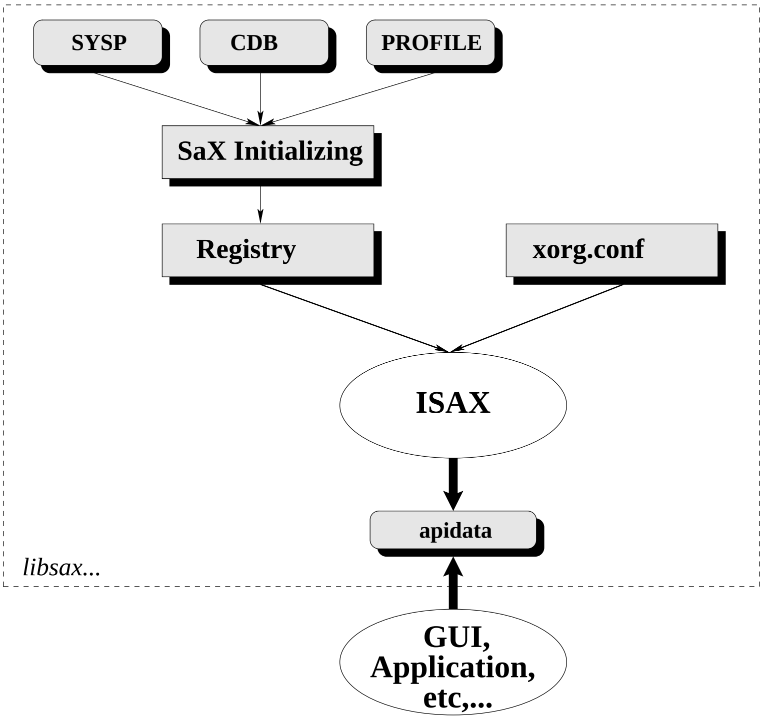
1. **Init:**
   Provide session cache
2. **Import:**
   Provide classes to obtain all necessary information
3. **Manipulate:**
   Provide classes to manipulate imported data
4. **Export:**
   Provide classes to create or modify the X11-Configuration

The programmer starts with an init() sequence to be able to access the automatically generated configuration suggestion which is based on the hardware detection. After this it is possible to import,manipulate and export information.

4.1 Rough diagram of SaX2 procedures

To give an introduction about how the ISaX interface is working as part of SaX2 the following rough overview will show the basics of the SaX2 engine.

Figure 4.1: SaX2 Flow diagram



4.2 SaX Import Classes

- **ISAX**
  The data of the currently used X11 configuration or the automatically generated configuration suggestion can be obtained by using the ISaX interface respectively by using the **isax** command. The information is stored into **SaX-Import** objects.

- **SYSP**
  Information near to the hardware like PCI IDs, BusID, etc... can be obtained by using the Sysp interface respectively by using the **sysp** command. The information is stored into **SaXImportSysp** objects

- **CDB**
  Manually maintained data refering stuff like Mice, Tablets, Graphics Cards, Monitors, etc... can be obtained from the exported files of the CDB (Component Data-Base). The information is stored into **SaXImportCDB** objects

- **PROFILE**
  Profile information for a specific card can be obtained using a special ISaX interface script called *createPRO.pl*. The information given here is stored into a **SaXImportProfile** object.

4.3 SaX Manipulation Classes

Once the needed data has been imported the programmer can start to manipulate it. The information from the SYSP, CDB and PROFILE methods are helpful but only the data concerning the ISAX import are used for the later export respectively the later configuration file. If the programmer is familiar with the ISaX interface there would be no need to provide further manipulation classes but to make it comfortable the library should provide SaXManipulation... classes to be able to do the common configuration tasks easily. At this point we need to define what the common configuration tasks are. Currently the following manipulation classes are specified:

- Baseclass: **SaXManipulateCard**
  Handle hardware related configuration settings including stuff like graphics drivers options etc...
- Baseclass: **SaXManipulateDesktop**
  Handle desktop related configuration settings including stuff like resolution colordepth etc...
- Baseclass: **SaXManipulateDevices**
  Handle device creation including stuff like creating or deleting a desktop adding input devices etc...
- Baseclass: **SaXManipulateExtensions**
  Subclass: *SaXManipulateVNC*
  Handle X-Server extensions for example VNC
- Baseclass: **SaXManipulateKeyboard**
  Handle keyboard configuration settings
- Baseclass: **SaXManipulateLayout**
  Handle layout configuration settings of multihead environments
- Baseclass: **SaXManipulatePath**
  Handle fontpath serverflags and server modules configuration settings
- Baseclass: **SaXManipulatePointers**
  Subclass: *SaXManipulateMice,SaXManipulateTablets,SaXManipulateTouchscreens*
  Handle pointer devices including stuff like mice tablets or touchscreen configuration settings

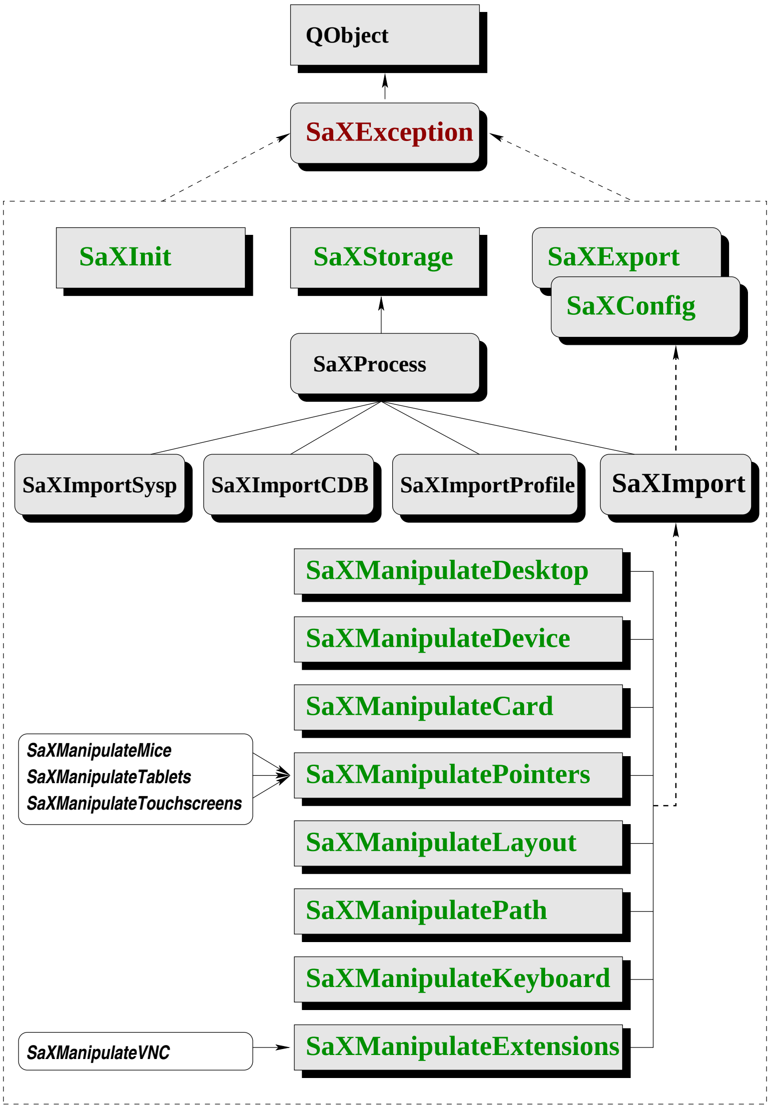
After all manipulations to all **SaXImport** objects have been done the programmer needs to add the affected SaXImport objects to a **SaXConfig** object which handles the export.

4.4 SaX Export Classes

the library provides a **SaXExport** and a **SaXConfig** class whereas the SaXConfig class is able to include multiple SaXImport objects. The SaXConfig object will create a corresponding SaXExport object for each SaXImport object bound to the SaXConfig object. With this list of SaXExport objects it is possible to create a new - or modify an existing X11 configuration.

4.5 libsax classes and inheritance

Figure 4.2: libsax: object reference



5 libsax - Error Handling

Contents

There are two possible methods to handle errors from the library:

- **Exception handling**
  asynchronous method using callback functions to handle the errors. The programmer needs to inherit from SaXException and bind an instance of this class to an instance of a SaX* class which itself inherits from SaXException as well.

- **Traditional *error* functions**
  synchronous method calling error methodes after each call to check if the return code is ok or not.

5.1 Exception handling

Every SaX class which is able to throw exceptions inherits from the SaXException class and therfore provides an interface to make use of the signal/slot concept provided by Qt. If an error occurs the library will emit a signal which can be catched. The following example will illustrate that:

exception.h

```
#include <sax.h>
class myException : public SaXException {
    Q_OBJECT

    public:
    myException ( SaXException* );

    private slots:
    void processFailed (void);
    void permissionDenied (void);
};
```

exception.cpp

```
#include <sax/sax.h>
#include <sax/exception.h>

myException::myException (SaXException* mException)  {
    connect (
        mException, SIGNAL
            ( saxProcessFailed (void) ),
        this, SLOT
            ( processFailed (void) )
    );
    connect (
        mException, SIGNAL
            ( saxPermissionDenied (void) ),
        this , SLOT
            ( permissionDenied (void) )
    );
}

void myException::processFailed (void) {
    printf ("Process call failed\n");
}

void myException::permissionDenied (void) {
    printf ("Permission denied\n");
}
```

main.cpp

```
#include <sax/sax.h>
#include <sax/exception.h>

int main (void) {
    SaXInit* init  = new SaXInit;
    myException* e = new myException (init);
    ...
}
```

5.2 Traditional *error* functions

If using signals is not appropriate for the current language environment the programmer can call one of the following public error methods:

- code = errorCode()
- info = errorString()
- value = errorValue()
- errorReset()

Every SaX class which can throw an exception will provide these error functions. The following example show how to make use of the error functions instead of the exception handling:

main.cpp

```
#include <sax.h>

int main (void) {
    SaXInit* init = new SaXInit;
    ...
    printf ("%d : %s\n",
        init->errorCode(),
        init->errorString().ascii()
    );
}
```

6 libsax - Examples

6.1 New configuration...

The following example will create a new configuration based on the suggestion made by SaX. We will add a new resolution 1600x1200 and want a modeline for the mode to be created.

```
#include <sax.h>

int main (void) {
    SaXInit init;
    if (init.needInit()) {
        init.doInit();
    }
    SaXException().setDebug (true);
    QDict<SaXImport> section;
    int importID[7] = {
        SAX_CARD, SAX_DESKTOP, SAX_POINTERS,
        SAX_KEYBOARD, SAX_LAYOUT, SAX_PATH, SAX_EXTENSIONS
    };
    SaXConfig* config = new SaXConfig;
    for (int id=0; id<7; id++) {
        SaXImport* import = new SaXImport ( importID[id] );
        import -> setSource ( SAX_AUTO_PROBE );
        import -> doImport();
        config -> addImport (import);
        section.insert (
            import->getSectionName(),import
        );
    }
    SaXManipulateDesktop mDesktop (
        section["Desktop"],section["Card"],section["Path"]
    );
    if (mDesktop.selectDesktop (0)) {
        mDesktop.addResolution (24,1600,1200);
        mDesktop.calculateModelines (true);
    }
    config -> setMode (SAX_NEW);
    config -> createConfiguration();
}
```

6.2 Change current configuration...

The next example will change the current configuration to use 24 bit as default color depth.

```
#include <sax.h>

int main (void) {
    SaXException().setDebug (true);
    QDict<SaXImport> section;
    int importID[] = {
        SAX_CARD,
        SAX_DESKTOP,
        SAX_PATH,
    };
    SaXConfig* config = new SaXConfig;
    for (int id=0; id<3; id++) {
        SaXImport* import = new SaXImport ( importID[id] );
        import -> setSource ( SAX_SYSTEM_CONFIG );
        import -> doImport();
        config -> addImport (import);
        section.insert (
            import->getSectionName(),import
        );
    }
    SaXManipulateDesktop mDesktop (
        section["Desktop"],section["Card"],section["Path"]
    );
    if (mDesktop.selectDesktop (0)) {
        mDesktop.setColorDepth (24);
    }
    config -> setMode (SAX_MERGE);
    config -> createConfiguration();
}
```

7 libsax - Thread safety

To be thread safe there are a view code points which needs a locking. The following list describes the serialized lock parts of the library:

- library included debug messages to STDERR are embedded into flockfile() / funlockfile() calls
- library initializing calls will lock each other using flock()
- library exporting code which creates the apidata files will apply an flock() during file creation

Refering to this the library can be used in a simultaneously way without crashing and without leaving the configuration in an inconsistent state. Of course it does not make much sense to simultaneously configure two different issues in such a case the last one will win. The following example will demonstrate a thread example including a simultaneously initialization.

```
#include <pthread.h>
#include <sax.h>

void* myFunction (void*);

int main (void) {
    pthread_t outThreadID1,outThreadID2;
    pthread_create (&outThreadID1, 0, myFunction, 0);
    pthread_create (&outThreadID2, 0, myFunction, 0);
    pthread_join (outThreadID1,NULL);
    pthread_join (outThreadID2,NULL);
    return 0;
}
void* myFunction (void*) {
    printf ("Checking cache...\n");
    SaXInit* init = new SaXInit;
    if (init->needInit()) {
        printf ("Initialize cache...\n");
        init->doInit();
    }
    printf ("%d : %s\n",
        init->errorCode(),init->errorString().ascii()
    );
    pthread_exit (0);
}
```

8 libsax - Language bindings

Contents

libsax has been developed as a C++ written library which means including this library into languages providing an object model is an easy task. To be able to use the library within languages providing only support for basic types an interface wrapper has been included which handles objects as ID's and provides the basic public constructors and members as simple functions.

8.1 SWIG

SWIG (Simple Wrapper Interface Generator) is a software development tool that simplifies the task of interfacing different languages to C and C++ programs. In a nutshell, SWIG is a compiler that takes C declarations and creates the wrappers needed to access those declarations from other languages including csharp, java, perl5, php4, python or tcl.

Concerning development requirements the following should be covered:

- In case of well written libraries the process of generating language bindings can be automated.
- typemaps for transforming the C++ types used within the library must be provided for each destination language.
- STL written libraries can make use of the typemaps already provided by swig

8.2 Interface template file for libsax (SaX.i)

The following template illustrate the steps which needs to be implemented to be able to map all libsax types into the destination language.

```
//==================================
// Interface definition for libsax
//----------------------------------
#define NO_OPERATOR_SUPPORT 1
%module SaX
%{
#include "../sax.h"
%}
//==================================
// SWIG includes
//----------------------------------
%include exception.i

//==================================
// Typemaps
//----------------------------------
//==================================
// Allow QString return types
//----------------------------------
// [ destination language dependant ]
// ...

//==================================
// Allow QString refs as parameters
//----------------------------------
// [ destination language dependant ]
// ...

//==================================
// Allow QDict<QString> return types
//----------------------------------
// [ destination language dependant ]
// ...

//==================================
// Allow QList<QString> return types
//----------------------------------
// [ destination language dependant ]
// ...
```

```
//==================================
// Exception class wrapper...
//----------------------------------
class SaXException {
    public:
    int   errorCode          ( void );
    bool  havePrivileges     ( void );
    void  errorReset         ( void );

    public:
    QString errorString  ( void );
    QString errorValue   ( void );

    public:
    void setDebug ( bool = true );
};
//==================================
// ANSI C/C++ declarations...
//----------------------------------
%include "../storage.h"
%include "../process.h"
%include "../export.h"
%include "../import.h"
%include "../init.h"
%include "../config.h"
%include "../card.h"
%include "../keyboard.h"
%include "../pointers.h"
%include "../desktop.h"
%include "../extensions.h"
%include "../layout.h"
%include "../path.h"
%include "../sax.h"
```

8.2.1 Interface explanations

The interface file is divided into four *sections* which handles the following interfacing problems:

1. Module name and include files to be able to create the wrapper code. The namespace used here is called **SaX**
2. type mappings from C++ into the destination language. Currently perl,python,java and csharp types are supported.
3. Exception class wrapper which doesn't include the Qt signal/slot definitions
4. Declarations used to create an interface for the destination language. Refering to perl this information is used to create the appropriate SaX.pm file

8.3 Example: libsax used with perl...

The following examples will do the same as the last example from the Examples chapter; Changing the default colordepth of the current configuration to 24 bit.

```
use SaX;
sub main {
    my %section;
    my @importID = (
        $SaX::SAX_CARD, $SaX::SAX_DESKTOP, $SaX::SAX_PATH
    );
    my $config = new SaX::SaXConfig;
    foreach my $id (@importID) {
        $import = new SaX::SaXImport ( $id );
        $import -> setSource ( $SaX::SAX_SYSTEM_CONFIG );
        $import -> doImport();
        $config -> addImport ( $import );
        $section{$import->getSectionName()} = $import;
    }
    my $mDesktop = new SaX::SaXManipulateDesktop (
        $section{Desktop},$section{Card},$section{Path}
    );
    if ($mDesktop->selectDesktop (0)) {
        $mDesktop->setColorDepth (24);
    }
    $config -> setMode (SaX::SAX_MERGE);
    $config -> createConfiguration();
}
main();
```

9 XFine tuning

XFine in SaX2 represents both a module and an independent X11 application. The module **XFineControl.pm** is used within SaX2 to save changes in the image geometry and to write these to the configuration file.

The **xfine.pl** main script writes this change information to the image geometry as a file in the directory:

- /var/cache/xfine

Per resolution a file is created with change information. The files are named according to the *SCREEN:XxY* convention. The format of the files has the following convention:

```
SCREEN:OLDMODE:NEWMODE:DACSPEED
```

When using XFineControl.pm, there are two different modes:

- **Using from within SaX2:**
  The main task in using XFineControl.pm is in the creation of the so-called *tune* hash. This hash serves in SaX2 as a reference for already changed modelines and is checked with each test run. It contains the original modeline, the last changed modeline and the current modeline. By means of the timing values and the number of original modelines, a check is made on whether the tune hash needs to be newly created, or if it can serve as a reference.

- **Using from within XFine in local mode:**
  If XFine is used as an independent tool, then the tune hash is created in advance from a configuration file given as a reference. The tool then works on this data in the same way as it would if used from within SaX2.

XFine has the following options:

- `-d | --display`
  With this option the display is set in which XFine should be started.

- `-l | --local`
  With this option XFine is started in local mode. This means that it will work on a reference to be specified. The reference configuration here is also changed.

- `-c | --config`
  With this option the name of the reference configuration is determined. If this is not specified, then /etc/X11/xorg.conf will be used.

- `-q | --quiet`
  This option includes all change information from /var/cache/xfine/ into the specified reference configuration. The option -q makes the option -l necessary. After this action, XFine is ended without access being made to the X display.

A Examples of Using the SaX Batch Mode

Contents

The batch mode in SaX2 allows you to make special settings directly after the hardware scan. This mode can be switched on in the *init.pl* stage. There are two different modes:

- Interactive mode
  A shell is provided to enter commands.
- Profile mode
  STDIN is read in and you can specify a profile file which contains shell commands.

Changes at this point directly influence the contents of the %var hash which is used to construct the registry and create the initial configuration. It is absolutely essential to understand the hash structure if you want to use this mode sensibly.

This structure is not a static form. It can be extended at will with the batch mode, but it is not clear that all data of the hash can also be used in the configuration, since simply everything in the hash can be included. The automatically created structure, when SaX2 is started, is built up as follows:

```
#------------------------------------------#
# Files specification...                   #
#------------------------------------------#
Files         ->    0  -> FontPath
Files         ->    0  -> RgbPath
Files         ->    0  -> ModulePath
Files         ->    0  -> LogFile

#-----------------------------------------#
# Module specification...                 #
#-----------------------------------------#
Module        ->    0  -> Load

#-----------------------------------------#
# ServerFlags specification...            #
#-----------------------------------------#
ServerFlags   ->    0  -> Option
```

```
ServerFlags   ->    0  -> blank time
ServerFlags   ->    0  -> standby time
ServerFlags   ->    0  -> suspend time
ServerFlags   ->    0  -> off time

#----------------------------------------#
# Keyboard specification...              #
#----------------------------------------#
InputDevice   ->    0  -> Identifier
InputDevice   ->    0  -> Driver
InputDevice   ->    0  -> Option            ->  Protocol
InputDevice   ->    0  -> Option            ->  XkbRules
InputDevice   ->    0  -> Option            ->  XkbModel
InputDevice   ->    0  -> Option            ->  XkbLayout
InputDevice   ->    0  -> Option            ->  XkbVariant
InputDevice   ->    0  -> Option            ->  AutoRepeat
InputDevice   ->    0  -> Option            ->  Xleds
InputDevice   ->    0  -> Option            ->  XkbOptions

#----------------------------------------#
# Mouse specification...                 #
#----------------------------------------#
InputDevice   ->    1  -> Identifier
InputDevice   ->    1  -> Driver
InputDevice   ->    1  -> Option            ->  Protocol
InputDevice   ->    1  -> Option            ->  Device
InputDevice   ->    1  -> Option            ->  SampleRate
InputDevice   ->    1  -> Option            ->  BaudRate
InputDevice   ->    1  -> Option            ->  Emulate3Buttons
InputDevice   ->    1  -> Option            ->  Emulate3Timeout
InputDevice   ->    1  -> Option            ->  ChordMiddle
InputDevice   ->    1  -> Option            ->  Buttons
InputDevice   ->    1  -> Option            ->  Resolution
InputDevice   ->    1  -> Option            ->  ClearDTR
InputDevice   ->    1  -> Option            ->  ClearRTS
InputDevice   ->    1  -> Option            ->  ZAxisMapping
InputDevice   ->    1  -> Option            ->  MinX
InputDevice   ->    1  -> Option            ->  MaxX
InputDevice   ->    1  -> Option            ->  MinY
InputDevice   ->    1  -> Option            ->  MaxY
InputDevice   ->    1  -> Option            ->  ScreenNumber
InputDevice   ->    1  -> Option            ->  ReportingMode
InputDevice   ->    1  -> Option            ->  ButtonThreshold
InputDevice   ->    1  -> Option            ->  ButtonNumber
InputDevice   ->    1  -> Option            ->  SendCoreEvents

#----------------------------------------#
# Monitor specification...               #
#----------------------------------------#
```

```
Monitor       ->    0  -> Identifier
Monitor       ->    0  -> VendorName
Monitor       ->    0  -> ModelName
Monitor       ->    0  -> HorizSync
Monitor       ->    0  -> VertRefresh
Monitor       ->    0  -> Modeline         ->  0                ->  640x480
Monitor       ->    0  -> Option

#-----------------------------------------#
# Device specification...                 #
#-----------------------------------------#
Device        ->    0  -> Identifier
Device        ->    0  -> VendorName
Device        ->    0  -> BoardName
Device        ->    0  -> Videoram
Device        ->    0  -> Driver
Device        ->    0  -> Chipset
Device        ->    0  -> Clocks
Device        ->    0  -> BusID
Device        ->    0  -> Option
Device        ->    0  -> Special          ->  hw_cursor

#-----------------------------------------#
# Screen specification...                 #
#-----------------------------------------#
Screen        ->    0  -> Identifier
Screen        ->    0  -> Device
Screen        ->    0  -> Monitor
Screen        ->    0  -> DefaultDepth
Screen        ->    0  -> Depth            ->  8                ->  Modes
Screen        ->    0  -> Depth            ->  8                ->  ViewPort
Screen        ->    0  -> Depth            ->  8                ->  Virtual
Screen        ->    0  -> Depth            ->  8                ->  Visual
Screen        ->    0  -> Depth            ->  8                ->  Weight
Screen        ->    0  -> Depth            ->  8                ->  Black
Screen        ->    0  -> Depth            ->  8                ->  White
Screen        ->    0  -> Depth            ->  8                ->  Option

#-----------------------------------------#
# ServerLayout specification...           #
#-----------------------------------------#
ServerLayout  ->  all  -> Identifier
ServerLayout  ->  all  -> InputDevice      ->  0                ->  id
ServerLayout  ->  all  -> InputDevice      ->  0                ->  usage
ServerLayout  ->  all  -> InputDevice      ->  1                ->  id
ServerLayout  ->  all  -> InputDevice      ->  1                ->  usage
ServerLayout  ->  all  -> Screen           ->  0                ->  id
ServerLayout  ->  all  -> Screen           ->  0                ->  top
ServerLayout  ->  all  -> Screen           ->  0                ->  bottom
```

```
ServerLayout  ->  all  -> Screen           ->  0                -> left
ServerLayout  ->  all  -> Screen           ->  0                -> right
```

A.1 Interactive Mode

The interactive mode provides the user with the following commands:

- **list**
  The *list* command lists all settings of the current registry.
- **see**
  The *see* command allows a certain setting to be displayed. For example, to see the modules used: `see Module->0->Load`.
- **calc**
  The *calc* command lets you calculate modeline timings. For example:
  `calc 1024x768->70` calculates a modeline for the mode 1024x768 at 70 Herz.
- **abort**
  Ends interactive mode and discards all changes.
- **exit**
  Ends interactive mode and saves all changes.
- **Setting variables**
  Setting variables is done by setting the full variable path, by including a value allocation. For example:
  `Module->0->Load = glx,dri`.

A.2 Profile Mode and Creating Profile Files

In some very specific cases it may be necessary for a profile for a card to be created. SaX2 provides a mechanism which allows you to include known profiles in the SaX2 package. These profile files are located under:

- /usr/share/sax/profile/

If there is an entry in */usr/share/sax/sysp/modules/maps/Identity.map* which starts with PROFILE=... then the profile for this card is integrated. The profile files essentially consist of variable values, as the following example illustrates:

```
Screen ->[X]->DefaultDepth         = 24
Monitor->[X]->Modeline->0->640x480 = 36.00 640 680 760 768 480 490 497 520
Monitor->[X]->Modeline->1->800x600 = 49.50 800 856 992 1000 600 612 619 651
```

Since it is never known in advance for which monitor, screen or device the new setting is to be made, it is possible to set a place holder in the form of an **[X]** mark, otherwise a number must be entered at this point.

B Examples of the Problem of Options

1. Four cards are inserted, of which the last 3 should be used. For cards 2 and 4, modules should be set. In this case the command would be:

   ```
   SaX2 -c 1,2,3 -m 0=mga,2=nv
   ```

   The numbering of the chips begins with 0, as does the order of modules. The device 0 is connected to chip 1, device 1 to chip 2, device 2 to chip 3.

2. Two cards with a total of 4 chipsets are inserted. Three of the 4 chips are on the first card, the other one on the 2nd card. A multihead setup should be created which in each case uses the first chip on both cards:

   ```
   SaX2 -c 0 3
   ```

   If modules need to be allocated for these chipsets, then it should be noted that these are detected as card 0 and card 1 and consequently the module option needs to be set to 0 and 1:

   ```
   SaX -c 0,3 -m 0=mga,1=glint
   ```

C The Variables API File

Contents

In this chapter all variables which may be found in a variables API file are explained. The variables API is normally created by SaX's own configuration interface, but this is not absolutely necessary. As soon as a variables API exists, this can be used to create a configuration file. In conjunction with the *ImportAPI* module and the *CreateSections* module, an X11 configuration can be created from the API file.

C.1 API File Keyword Explanations

The individual tables in their format description use various keywords, which are explained in the following list.

- **String:**
  Refers to any sequence of characters which are **not** embedded in quotation marks.
- **Subsection:**
  Refers to the name of a subsection in the X11 configuration. This word is followed by an entry in the subsection.
- **Flagname:**
  Refers to the name of a server flag. This is followed by the value for the server flag.
- **Integer:**
  Refers to a whole number. Is usually used in connection with variables for defining size.
- **ButtonX:**
  Refers to the number of the mouse button which should be adapted for the wheel movement in the X axis.

- **ButtonY:**
  Refers to the number of the mouse button which should be adapted for the wheel movement in the Y axis.

- **Clocks:**
  Refers to a list separated by spaces. with clock values. These values can be whole numbers as well as fractions.

- **Mode:**
  Refers to a resolution string in the form of `[Xpixel]x[Ypixel]`

- **Algorithm:**
  Refers to the two possible algorithms *CheckDesktopGeometry* or *IteratePrecisely*.

- **Modeline:**
  Refers to a modeline string, starting with a name in quotation marks which must match a *resolution string*, followed by the RamDAC speed and 8 further parameters.

- **Sync:**
  Refers to a frequency range. This is specified through a number range in the format: `[Minimum]-[Maximum]`

- **Left,Right,Up,Down**
  Refers to a screen position. The value matches an identifier string in accordance with the monitor. If there is no screen at this point then `<none>` should be entered.

C.2 API File Overview Tables of all Possible Variables

Below, all variables which may appear in an API file are listed in tabular form. The contents of each table refer to a section in the API file. It should be noted that one API section covers a number of xorg.conf sections.

C.2.1 Path Variables: Section Files, Modules and Server Flags

Card	Variable	Format
Integer	FontPath	String,String,String,...
Integer	RgbPath	String,String,String,...
Integer	ModulePath	String,String,String,...
Integer	ModuleLoad	String,String,String,...
Integer	Extmod	Subsection,String\nSubsectio,String,...
Integer	SpecialFlags	Flagname,String\nFlagname,String,...
Integer	ServerFlags	String,String,String,...

C.2.2 Graphics Card Variables: Device Section

Card	Variable	Format
Integer	Identifier	String
Integer	Driver	String
Integer	Memory	Integer
Integer	BusID	String
Integer	Vendor	String
Integer	Name	String
Integer	DacChip	String
Integer	GraphicsChip	String
Integer	ClockChip	String
Integer	DacSpeed	String
Integer	Clocks	Clocks,Clocks,...
Integer	Option	String,String,...
Integer	RawData	String,String,...
Integer	MaxDac	Integer

C.2.3 Mouse Variables: InputDevice Section

Card	Variable	Format
Integer	Identifier	String
Integer	Driver	String
Integer	Protocol	String
Integer	Device	String
Integer	Baudrate	Integer
Integer	Samplerate	Integer
Integer	Emulate3Buttons	Yes | No
Integer	Emulate3Timeout	Integer
Integer	ChordMiddle	Yes | No
Integer	MinX	Integer
Integer	MaxX	Integer
Integer	MinY	Integer
Integer	MaxY	Integer
Integer	ScreenNumber	Integer
Integer	ReportingMode	String
Integer	ButtonNumber	Integer
Integer	ButtonThreshold	Integer
Integer	SendCoreEvents	Yes | No
Integer	ClearDTR	Yes | No
Integer	ClearRTS	Yes | No
Integer	ZAxisMapping	Off | None | ButtonX ButtonY | X | Y
Integer	ZAxisNegMove	Off | ButtonX
Integer	ZAxisPosMove	Off | ButtonY
Integer	Vendor	String
Integer	Name	String
Integer	TabletMode	String
Integer	TabletType	String

C.2.4 Screen Variables: Section Monitor,Modes and Screen

Card	Variable	Format
Integer	Identifier	String
Integer	Device	String
Integer	Monitor	String
Integer	VendorName	String
Integer	ModelName	String
Integer	Virtual	Integer Integer
Integer	Visual	String
Integer	HorizSync	Sync
Integer	VertRefresh	Sync
Integer	MonitorOptions	String,String,...
Integer	ScreenOptions	String,String,...
Integer	Modelines	Modeline,Modeline,...
Integer	SpecialModeline	Modeline,Modeline,...
Integer	ColorDepth	Integer
Integer	CalcModelines	Yes | No
Integer	CalcAlgorithm	Algorithm
Integer	ViewPort	Integer Integer
Integer	ScreenRawLine	String,String,...
Integer	Modes:4	Mode,Mode,...
Integer	Modes:8	Mode,Mode,...
Integer	Modes:15	Mode,Mode,...
Integer	Modes:16	Mode,Mode,...
Integer	Modes:24	Mode,Mode,...
Integer	Modes:32	Mode,Mode,...

C.2.5 Layout Variables: Section ServerLayout

Card	Variable	Format
Integer	Identifier	String
Integer	Keyboard	String
Integer	InputDevice	String,String,..
Integer	Xinerama	On | Off
Integer	Screen:`<Identifier>`	Left Right Up Down

C.2.6 Keyboard Variables: InputDevice Section

Card	Variable	Format
Integer	Identifier	String
Integer	Driver	String
Integer	Protocol	String
Integer	XkbRules	String
Integer	XkbModel	String
Integer	XkbLayout	String
Integer	XkbVariant	String
Integer	XkbOptions	String,String,...
Integer	AutoRepeat	String
Integer	Xleds	String
Integer	XkbDisable	Yes | None
Integer	VTSysReq	Yes | None
Integer	VTInit	String
Integer	ServerNumLock	Yes | None
Integer	LeftAlt	String
Integer	RightAlt	String
Integer	ScrollLock	String
Integer	RightCtl	String
Integer	XkbKeyCodes	String

C.3 Example of API Variables

```
Keyboard {
 0 Protocol         =    Standard
 0 XkbLayout        =    de
 0 Identifier       =    Keyboard[0]
 0 XkbModel         =    pc104
 0 Driver           =    keyboard
}

Mouse {
 1 Name             =    AutoDetected
 1 Identifier       =    Mouse[1]
 1 Driver           =    mouse
 1 Vendor           =    AutoDetected
 1 Device           =    /dev/pointer0
 1 Protocol         =    PS/2
}

Card {
 0 Name             =    RivaTNT
 0 Identifier       =    Device[0]
 0 BusID            =    1:0:0
 0 Driver           =    nv
 0 Vendor           =    Nvidia
}

Desktop {
 0 VertRefresh      =    50-160
 0 Device           =    Device[0]
 0 ModelName        =    Vision Master Pro 450 (A901HT)
 0 CalcModelines    =    yes
 0 Identifier       =    Screen[0]
 0 ColorDepth       =    16
 0 Monitor          =    Monitor[0]
 0 Modes:16         =    1800x1350,640x480
 0 HorizSync        =    27-115
 0 VendorName       =    Iiyama
}

Path {
 0 RgbPath          =    /usr/X11R6/lib/X11/rgb
 0 ModulePath       =    /usr/X11R6/lib/modules
 0 ServerFlags      =    AllowMouseOpenFail
 0 FontPath         =    /usr/X11R6/lib/X11/fonts/misc:unscaled
 0 ModuleLoad       =    dbe,type1,speedo,extmod,freetype
}

Layout {
 0 Screen:Screen[0] =    <none> <none> <none> <none>
 0 InputDevice      =    Mouse[1]
 0 Keyboard         =    Keyboard[0]
 0 Identifier       =    Layout[all]
}
```

Glossary

SaX
Is an abbreviation for SuSE advanced X-configuration SaX is available for X11 R6 from version 3.3.3 onwards.

Device File
The interface between the functions of a driver and the access to these functions is formed by a device file. By means of the major and minor number of this file (also called node), allocation is made to a specific driver.

Device Node
Another term for *device file*.

rc.config
Contains configuration and start options for all services of the installed system.

batchmode
The batch mode stands for the concept of batch processing, and symbolizes a series of actions which are processed in the form of a stack. In SaX2 the batch mode is a kind of command interface into which you can enter commands or define variables for later use. The batch mode in SaX2 can be controlled automatically, or via a file.

Index