MaXX Settings Configuration Management

Architecture & Technical Specifications

Version 0.98

Versions

Version	Date	Author(s)	Description
0.10	2020-06-06	Eric Masson	Initial and ongoing work.
0.90	2020-12-29	Eric Masson	Change version scheme.
0.91	2020-12-30	Eric Masson	Remove duplicated information, move out implementation details to another document.
0.92	2020-12-31	Eric Masson	Restructure and relocation of Choices, more cleanup, typo and text improvements.
0.93	2021-01-02	Eric Masson	Improve User Experience section.
0.94	2021-01-06	Eric Masson	Complete documentation separation with Instrumentations Guide
0.95	2021-01-09	Eric Masson	Improving Instrument definition, adding to Requirements and Architecture sections.
0.96	2021-03-03	Eric Masson	Adding new props: UserInterfaceAccent, WindowManagerAccent, ModernLookAndFeel, ThinWidgetMode & FlatMenuMode.Logical
0.97	2021-03-14	Eric Masson	Changing Gauge values to Float
0.98	2021-04-07	Eric Masson	Adding SmoothText, DesktopAccent & cleanup duplicated

Table of Content

Synopsys

MaXX Settings is a dynamic configuration management subsystem designed from the ground up with simplicity in mind while not sacrificing flexibility and extensibility. MaXX Settings comes with its own CLI interface allowing simple management, automation via scripting, inline-query and easy application integration. MaXX Settings also provides Java and C++ binding making it super easy to integrate within most modern applications. MaXX Settings allow the definition of System wide setting, we call them **Instruments**, and user's overridables called **User Preferences**.

This document will explain all there is to know about **MaXX Settings Architecture & Technical Specifications** and how to get Started.

For in-depth instrumentation and implementation details, refer to the **MaXX Settings Instrumentations Guide** document.

Requirements

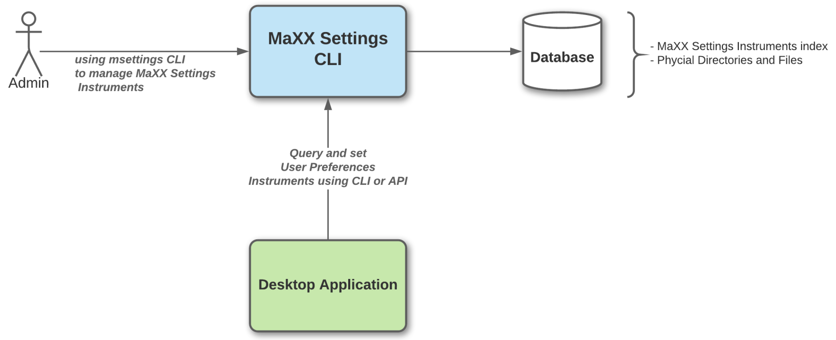
One of the benefits of starting fresh is the fact that we can start with a blank slate, put forward clear intents, express technical requirements and build an architecture early on in the design process.

Here are the requirements that MaXX Settings must strive to enforce or provide:

- Retrieve information as fast as possible (flat lookup speed curve).
- Provide different levels of verbosity (admin vs normal user).
- Software design based on current/modern technologies while future proofing the code with a component/modular approach.
- Use SOLID Principles (most): Single responsibility, open-close, interface segregation and dependency inversion.
- Favor simplicity over complexity.
- Support multiple OS.
- Provide a Command Line Interface (CLI) to administer, query and set data.
- Be human friendly with its interfaces.
- Provide an API for C++ and Java clients.
- Provide user based authentication.
- Support UTF-16 for its internal String encoding.
- Support hierarchical data structure suited for a dynamic typed configuration management system for Desktop, Application and FileType instrumentations.

Architecture

The Architecture on which MaXX Settings is built follows the Client/Server model, where the Client is represented by Users using a CLI type interface or Applications using an API both interacting with the Server. The Server provides the functionality to manage configuration instrumentations.


Admin
using msettings CLI to manage MaXX Settings Instruments
MaXX Settings CLI
Database
- MaXX Settings Instruments index
- Phycial Directories and Files
Query and set User Preferences Instruments using CLI or API
Desktop Application

The diagram illustrates the overall architecture of MaXX Settings.

Development and Execution Platform

Java was selected for the first implementation of the Server for its richness in features, robustness, maturity and special affinity with server/service type APIs and prebuilt components. The GraalVM was selected for its ability to compile Java bytecode into native code and run applications much faster. The optimizations offered by GraalVM have the added benefits to reduce startup and execution speed quite considerably.

Instrumentation Data Persistence

All information managed by the Server is persisted into regular files. No external dependency required for the database.

Database

The database is implemented using fixed length field strategy to ensure simplicity but high performance. Saved information into the database is kept to a minimum as it is mostly to provide lookup mechanism.

A generic interface is defined to ensure proper use and evolution. It also follows the OpenClose and Interface segregation SOLID Principles.

Here's the Java IDatabase Interface defining the core functionality offered.

```java
package com.maxxinteractive.msettings.database;

import org.jetbrains.annotations.NotNull;

/**
* MaXX Settings System wide Database specifications
*  @author Eric Masson
*  @version 1.0
*/
public interface IDatabase {

   boolean createDB(boolean forceCreation);

   void openDB();

   boolean isOpen();

   void closeDB();

   boolean put(IndexEntry entry);

   String findByUUID(@NotNull String uuid);

   String findByHashFilename(@NotNull String hashDirectory);

   boolean lookup(@NotNull String lookup);
}
```

Instrumentations

The actual instrumentation data is saved on the disk using an individual file for each instrument. The directory structure strategy for organizing the instrumentation is kept to a minimum and is designed to support hashed directory structure. The hashing is calculated from the unique instrument name and mapped onto a four (4) level hashing.

For example, the instrument **Desktop.Mouse.Acceleration** would have the calculated hashed directory structure **/3b/2a/d4/d6**. Then the instrument's file will be placed into that directory structure. The main goal here is efficiency by providing a consistently fast lookup mechanism. More on that throughout the document.

Naming Conventions

Lowercase is a naming convention in which a name formed of a single word is written all letters in lowercase.
Example: name, version, uuid, etc.

Uppercase is a naming convention in which a name formed of a single word is written all letters in uppercase.
Example: HOME, SHELL, PATH, etc.

Titlecase is a naming convention in which a name is written with all letters in lowercase except its first letter, which is uppercase. It follows a more natural style. No blank space allowed.
Example: Chars, Dimension, Geometry, etc.

Camelcase is a naming convention in which a **name** is formed of multiple words that are joined together as a single word with the first letter of each of the multiple words capitalized so that each word that makes up the **name** can easily be read. No blank space allowed.
Example: maximumSize, backgroundColor, darkColor, etc.

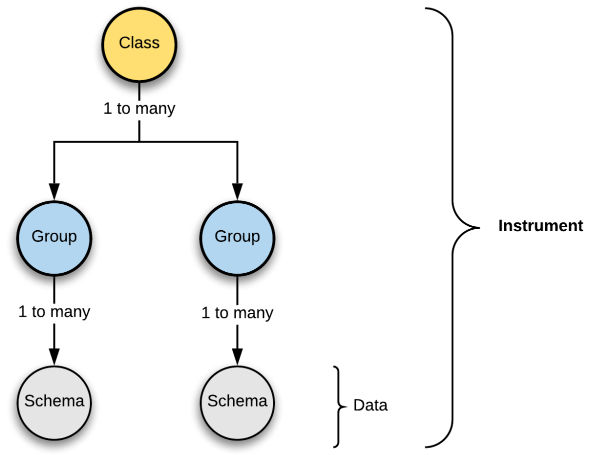
The table below lists the naming convention used in MaXX Settings.

	Convention	**Samples**
Attribute	One or multiple words in **camelcase** without blank space..	version, maxDuration, defaultAppName
Stereotype	One word in the **titlecase** without blank space.	Chars, Geometry, Image
Schema Name	Multiple words with no blank space where each word is in the **titlecase**. The last word usually defines the Schema's Stereotype name.	TextColor, DoubleClickGauge, AccelerationGauge
Schema Filename	Multiple words with no blank space where each word is in the **titlecase**. The last word usually defines the Schema's Stereotype name and is separated with a period.	Username.Chars, DoubleClick.Gauge, Acceleration.Gauge

Definitions

Structural Element

At the heart of MaXX Settings is the notion of **Instrument** representing the notion of a configuration setting containing mandatory attributes and operational validation rules that guarantee consistency in its data usages. An **Instrument** is composed of the three structural elements **Class, Group** and **Schema**. Each structural element belongs to a fixed hierarchical level facilitating clear classification and grouping of information in a natural and human readable way. Remember an **Instrument** has meaning only when it is composed of its three structural elements. Otherwise, they are just Classes containing Groups, Groups containing Schemas.



The diagram illustrates the hierarchical structure that makes up MaXX Settings Instrument.

Elementary...

Let's explore each structural element definition, their intent and how they all fit together.

Class

A **Class** sits at the top of the element food chain and is used to classify and organize Groups with their respective Schemas into meaningful collections of hierarchical information sets. As of MaXX Desktop v2.2, MaXX Settings supports three (3) types of classification: Desktop, Application and FileType. Desktop class of Instruments are used for the control of the User Experience aspect of the MaXX Desktop, whereas Application helps with the definitions of actions (open, view, edit, etc) per application and FileType defines MIME file-types and binds Application with their file-types.

Group

A **Group** is attached to a single **Class** while it defines a logical configurable grouping notion or thing such as 'Mouse' or 'Background'. Groups do not store information for themselves, but rather serve the purpose of a placeholder grouping any number of Schemas under one logical unit. **Group** is giving context to **Schema** similarly to the **Class** classifying Groups.

Schema

Schema is the last and the most important structural element of an Instrument. **Schema** is always attached to a single **Group** and it contains actual data defining as a whole a behavior characteristics, *or an* attitude. We call this behavior a **Stereotype**. But to keep things simple, a **Schema** is a text file that contains **attributes** governed by a **Stereotype**.

Attributes

An **Attribute** is the lowest level of granularity possible under everything MaXX Settings controls. Attributes are defined as key-value pairs for Schema properties. Below is the list of mandatory attributes for every Schema.

Mandatory Attributes

Attribute	Description	Example
version	Version identifier used when parsing and interpreting the Schema file.	version=1.0
uuid	Universally Unique IDentifier (UUID v4) is a 128-bit long value (36 chars length) used for reliably identifying information.	uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
stereotype	Stereotype name describing the Schema. It's like a data contract in a way.	stereotype=Dimension
name	The given name. to the Schema. Name must be unique and is case sensitive.	name=Desktop.Mouse.Acceleration
default	Define a default value when a user Preference is unset or resetted to its initial value.	default=0

Stereotype

Stereotype is a very powerful feature that helps define Schema characteristics with a set of standardized attributes, with optional validation rules, that help constrain and enforce proper usage with predictable outcomes. Stereotypes can even be used to behaviour modeling, but let's keep that for the future. We use Stereotypes in MaXX Settings to enforce a consistent definition and usage of important information within Schema files.

Every **Schema** under MaXX Settings is categorized as either a **Simple** or **Complex Stereotype**. To ensure consistency and predictability, a Schema must clearly define under which version its data-contract is based on and must contain an Universally Unique IDentifier (UUID v4) to uniquely identify itself, a Stereotype used by the Schema and finally an unique name. Those attributes are mandatory in all Schema files.

Refer to the two following tables below for more details on both **Simple** and **Complex Stereotypes**.

Simple Stereotype

As the name suggests, Simple Stereotypes are used to represent simple basic value type notions. At the exception of Chars Stereotype, none of the Simple Stereotypes supports validation rules.

Simple Stereotypes supported in MaXX Settings (inherits all Mandatory Attributes)

Schema	Description	Attributes	Example

Chars	Represents a sequence of characters. The *encoding* attribute is mandatory and is set to UTF-8 by default. The *maxLength* is optional and when present helps constraining the size of both default and value attributes. The size is calculated using (octets/bytes) with the character encoding.	default=null
		encoding=UTF-8 *
		maxLength
			stereotype=Chars
			value=Space1999
			encoding=UTF-8
			maxLength=256

Number	Represents an unsigned integer numerical value.	default=0	stereotype=Number
			value=12344

Decimal	Represents an unsigned numerical value with decimal single precision.	default=0.0	stereotype=Decimal
			value=13.467

Logical	Represents a boolean value of either true or false.	default=false	stereotype=Logical
			value=true

** mandatory attribute*

Complex Stereotype

A Complex Stereotype is used to represent complex value notions that requires several input parameters and generally used validation rules for its default and value attributes.

Complex Stereotypes supported in MaXX Settings (inherits all Mandatory Attributes)

Name	Description	Attributes	Example

Choice[TYPE]	Represents a typed indexed container of values. In most programming languages, it's called an array. The *type* defines the option's subtype. Each *option[]* entry is a possible choice. The *default* and *value* attributes are indexes pointing back to the Choice's *option[]* array. Index starts at 0.	default=0 !
		type=Chars * !
		option[i] * !
			stereotype=Choice
			value=1
			type=Chars
			option[0]=foo
			option[1]=bar

Dimension	Represents a two dimensional measurement composed of width and height as positive only single precision decimals.	default=1.0x1.0 !	stereotype=Dimension
			value=290.0x100.0

Location	Represents a 2D location composed of X and Y as signed integers.	default=+0+0 !	stereotype=Location
			value=+1090-300

Geometry	Represents a two dimensional measurement composed of width and height as integers and 2D location composed of X and Y as integers for a pixel drawable.	default=1x1+0+0 !	stereotype=Geometry
			value=290x100+1090+300

Gauge	Represents a single value measurement (as of linear scalar) according to predefined *minimum*, *maximum* and an incremental value as *scale*. Mouse Sensitivity preference is using Gauge for example.
	→ The *default* and *value* are specific to each Gauge but their values must be between the *minimum* and *maximum*.
		default=1.0 !
		minimum=1.0 *
		maximum=10.0 *
		scale=1.0 * !
			stereotype=Gauge
			value=7.0
			minimum=1.0
			maximum=10.0
			scale=1.0

Color	Represents a Color commonly used in user's preferences. BackgroundColor is such an example. Color is composed of a mandatory colorSpace and optional attribute alpha. Value is populated with matching colorSpace color components separated with comma. → No Default value.	default !
		colorSpace*
		alpha
			stereotype=Color
			default=255,255,255
			value=127,231,48
			colorSpace=RGB255
			alpha=1.0

Image	Represents an Image user's preferences. BackgroundImage is such an example. Image is composed of a mandatory filePath and the optional attribute resizeTo which can be used to apply a transformation on top of the original size. → No Default value.	filePath*
		dimension
		crop
		resizeTo
			stereotype=Image
			default=image.png
			value=image.png
			filePath=/temp
			dimension=256x256

Typeface	Represents a Font user's preferences. TerminalFont is such an example. Font is composed of mandatory values for the Font name (value attribute) and a size and optional attributes style, weight and slant. → No Default value.	size*
		style
		weight
		slant
			stereotype=Font
			default=Sans
			value=Noto Sans UI
			size=12

Command	Represents an executable command application or script. to launch. A Command is composed of the following optional attributes: execPath, execParams, envBinaryPath, envLibraryPath and geometry. The attribute value is the executable to run.	execPath
		execParams
		envBinaryPath
		envLibraryPath
		geometry
			stereotype=Command
			~~default=?~~
			value=nedit
			execPath=/usr/bin

Application	Represent a list of Command schema names for specific application actions such as: view, edit, etc. The value is used for the default action when no other actions are provided.
	→ No Default value.
		viewCommand
		editCommand
			stereotype=Application
			~~default=?~~
			value=Desktop.Editor.Nedit
			openCommand=Desktop.Editor.Nedit

** mandatory attribute*
! updatable default value via CLI

Choice Stereotype

Choice Stereotype is a special type of Stereotype that fulfills the single purpose of providing predefined System wide Choices. Many of those Choices are used throughout the Desktop User Experience Instruments.

Schema	Option Type	Options
ColorSpace.Choice	Chars	RGB255, RGB100, YUV, HSL, CMYK,
FontStyle.Choice	Chars	Normal
FontWeight.Choice	Chars	Light, Medium, Demibold, Bold, Black
FontSlant.Choice	Chars	Italic, Oblique, Roman
Language.Choice	Chars	...
KeyboardInput.Choice	Chars	...
DefaultSoundOutput.Choice	Chars	...
SGIScheme.Choice	Chars	...
IconSortBy.Choice	Chars	Name, Type, Size, CreationDate, ModifiedDate
IconViewAs.Choice	Chars	Icon, List, Detail
XftLcdFilter.Choice	Chars	lcddefault, lcdlight, lcdnone, lcdlegacy
XftHintStyle.Choice	Chars	hintnone, hintslight, hintmedium, hintfull
XftRgba.Choice	Chars	rgb, bgr, vrgb, vbrg
WinEditor.Choice	Command	Nedit, XNedit, Gedit
FileBrowser.Choice	Command	Rox-filer, fm
ImageViewer.Choice	Command	Feh, Eog
ImageEditor.Choice	Command	Gimp
WebBrowser.Choice	Command	Firefox, Chrome
EmailClient.Choice	Command	Thunderbird, Evolution
MediaViewer.Choice	Command	Vlc
VectorEditor.Choice	Command	Inkscape
PDFViewer.Choice	Command	Xpdf, Evince
BackgroundColors.Choice	Image	...
BackgroundImages.Choice	Color	...

Note: 24 totals

Refer to the MaXX Settings Instrumentations Guide document for more details.

1+1+1 > 3

When you combine those three structural elements together, you get an **Instrument** that can fulfil two purposes. First it defines at a **System wide** level what each instrumentation is (with the help of **Schema** and **Stereotype**), and second how it helps manage live/runtime user data as **User Preferences**. All System wide Instruments are stored within the system's **$MAXX_SETTINGS** directory. In practice, end-users are only exposed to a tiny portion of what Instruments can do, and this is intentional. KISS approach… However MaXX Settings power-users or administrators are able to tap into the full power of different types of User Preference, Instrument, Choices, Schema and Stereotypes.

Let's put all of what we learned so far in practice with the real life Instrument, **Desktop.Mouse.Acceleration**. This Instrument is defined as a **Gauge Stereotype** and its **Schema** defines the *minimum*, *maximum*, *scale and default* attributes. Its runtime counterpart is an **User Preference** of the same name to keep things simple. All User Preferences are stored within the user's **$HOME/.maxxdesktop** directory.

Our **Desktop.Mouse.Acceleration Instrument** stores its data-contract through a **Schema** file and its **User Preference** counterpart only stores the runtime overridable Schema's attributes and a chosen *value* by the end-user. The **Gauge Stereotype** enforces compliance based on the Stereotype defined behavior and optional validation rules. As you can see, there is a clear separation of responsibility between the definition, validation and utilization.

If you are familiar with Object-Oriented Programming, then a **Schema** is like a Class definition and an **User Preference** is the instance of that Class which contains live data. Schema provides similar data encapsulation mechanisms found in C++ and other OO programming languages. This responsibility of validation and compliance are handled by the **Stereotype**.

As a safeguard, Schema files are only editable by superuser privilege level. Therefore, we recommend using the provided Command Line Interface (CLI) tool to make any modification, addition or deletion. Refer to the CLI section for more detail.

How does it work?

Let's dive into how MaXX Settings's Configuration Management works under the hood.

Testing 1,2,3...

First we assume that MaXX Desktop v.2.2 or above is installed and running on your system. To confirm, login into a MaXX Interactive Desktop session from the login manager and follow the instructions.

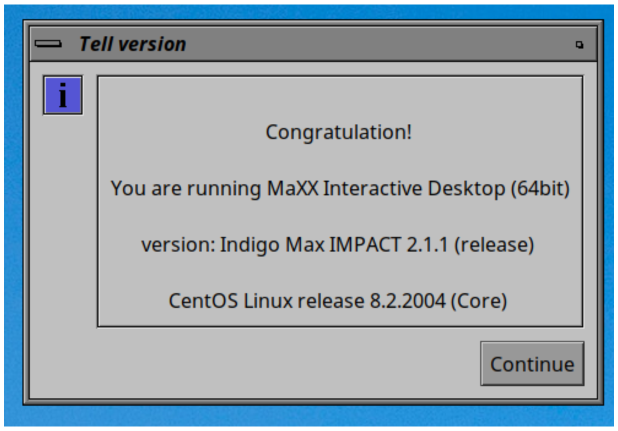
MaXX Desktop HOME

You may launch the Winterm application (Terminal app) from Toolchest or from the Winterm icon on the Desktop. Then, from the new Winterm window, type the command **echo $MAXX_HOME**. If you get something similar to the example below, congratulations MaXX Desktop is properly configured and running. If not, you may consult our online documentation at https://maxxinteractive.com

The **$MAXX_HOME** Environment Variable defined the location of your MaXX Desktop installation Root directory.

Test your MaXX Desktop Installation

```
</home/userbob1> $ echo $MAXX_HOME
/opt/MaXX
</home/userbob1> $ tellversion
```



Congratulation!

You are running MaXX Interactive Desktop (64bit)

version: Indigo Max IMPACT 2.1.1 (release)

CentOS Linux release 8.2.2004 (Core)

<need a new screenshot for version 2.2>

MaXX Settings System wide Root Directory

Following the same logic, MaXX Settings defines its own Environment Variable, **$MAXX_SETTINGS** which points to the location where MaXX Settings stores all its System wide files. Normally, MaXX Settings is installed inside the MaXX Desktop HOME directory and its Environment Variable is defaulted to: **$MAXX_HOME/share/msettings**.

Test System wide MaXX Settings Installation

```
</home/userbob1> $ echo $MAXX_SETTINGS
/opt/MaXX/share/msettings
</home/userbob1> $
```

We are all set, let's dive into it.

Instruments

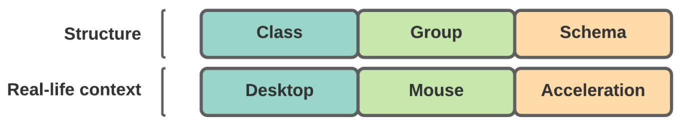
In this section, we go over the nuts and bolts regarding **different categories of Instruments** under MaXX Settings and how they can be used to create a dynamic configuration management system.

Instrument Categories

Category	Scope	Class Name	Group Name
User Experience	System Wide Instruments User Preference Instruments	Desktop	Mouse Keyboard Background DtUtilities Window Settings Colors DtSounds Localization Text FontRendering FileManager IconCatalog
FileTypes		FileTypes	
Applications		Application	WinEditor ImageEditor ImageViewer ...

Classification

One of the main design goals of MaXX Settings is to retrieve information as fast as possible and without introducing too much complexity in the process. So for this important performance requirement alone, MaXX Settings must provide an efficient mechanism for classifying and retrieving information. It is known that Instruments are made of a *Class.Group.Schema* structure could be mapped directly onto the file system with physical directories and files. Instead, MaXX Settings use an ultra fast computable hashcode of the Instrument's name, then mapped into a hashed directory structure. This allows lightning fast lookup regardless of the number of stored elements and is less prone to manual human intervention (messing things up). This is the way...



Instrument Structure vs. real-life

System Wide Instruments

As we saw previously, MaXX Settings Root directory is defined by the Environmental Variable **$MAXX_SETTINGS**. Therefore all MaXX Settings Instruments are stored in the **$MAXX_SETTINGS/Instruments** directory. Those **Instruments** are called **System wide Instruments**, they are read-only for normal users and only modifiable via the Administrative Command Line Interface with superuser privilege.

System Wide Nomenclature

Let's explore the System wide Instrument Desktop.Mouse.Acceleration and its various properties.	
Name	Desktop.Mouse.Acceleration
Class	Desktop
Group	Mouse
Schema	Acceleration
Stereotype	Gauge
Schema File	Acceleration.Gauge
Fully Qualified Name (FQ Name)	/Desktop/Mouse/Acceleration.Gauge
Hashed Storage Location	/3b/2a/d4/d6
Physical File Path	$MAXX_SETTINGS/Instruments/3b/2a/d4/d6/Acceleration.Gauge

User Preference

We know already System wide Instruments are read-only from a normal user point of view since they only define validation rules and default values. So how do we handle custom preferences for one or multiple users on the same system? The solution is rather simple, we just don't use them for say, but rather extend them and reusing the same classification strategy **<Class>.<Group>.<Schema>** for storing only the user defined values, but in a user specific location. Basically, they are user-land **Instruments** that can be editable by normal users, a.k.a. **User Preferences**.

By default User Preferences are located inside the **$HOME/.maxxdesktop/msettings/Preferences** directory and follow the same storage convention as System wide Instruments.

User Preferences are sharing the same classification and hashed storage location structure as System wide Instruments. This also means that the calculated hashcodes are the same.

User Preference Nomenclature

Let's explore an User Preference Instrument Desktop.Mouse.Acceleration and its various properties.	
Name	Desktop.Mouse.Acceleration
Class	Desktop
Group	Mouse
Schema	Acceleration
Stereotype	Gauge
Schema File	Acceleration.Gauge
Fully Qualified Name (FQ Name)	/Desktop/Mouse/Acceleration.Gauge
Hashed Storage Location	/3b/2a/d4/d6
Physical File Path	$HOME/.maxxdesktop/msettings/Preferences/3b/2a/d4/d6/Acceleration.Gauge

User Experience Instruments

This section will briefly explore in much more depth the User Experience Instruments category, their different scopes and how they can be used. We first look at three (3) real-life examples to better understand the differences between scopes and their level of flexibility. Then we explore the Desktop User Experience Instruments.

Simple *Logical* Instrument

This first example describes how MaXX Settings manages a *Simple* **System wide** Instrument **Desktop.Colors.SgiDarkScheme** and how it looks like from the **User Preference** (scope) perspective as well. This Instrument is of **Logical Stereotype**, and it is used to represent if the selected **SGI Scheme** is using dark colors.

Instrument Name:	Desktop.Colors.SgiDarkScheme		
Type:	**System wide**	**Type:**	**User Preference**
$Root:	$MAXX_SETTING/**Instruments**	**$URoot:**	$HOME/.maxxdesktop/msettings/**Preferences**
$Location:	*(calculated value)*	**$ULocation:**	*(calculated value)*
$Filename:	$Root/$Location/**SgiDarkScheme.Logical**	**$Filename:**	$URoot/$ULocation/**SgiDarkScheme.Logical**

File content: (System wide)

```
version=1.0
uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
stereotype=Logical
name=Desktop.Colors.SgiDarkScheme
default=false
```

File content: (User Preference)

```
version=1.0
uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
stereotype=Logical
name=Desktop.Colors.SgiDarkScheme
value=true
```

Important things to remember

1. Both the System wide and User Preference are using the same UUID and name in order to provide a lookup by UUID or name.
2. The calculated storage location is identical for both. This makes the switch between System wide and User Preference seamless.
3. The **Schema filename** is made of the Instrument name with its Stereotype as file extension.

Simple *Choice* Instrument

This second example explores the System wide Instrument **Desktop.FileManager.IconSortBy** which defines the sorting algorithm used for Icon display in **fm**, the MaXX Desktop File Manager. This Instrument is of **Choice Stereotype**, which manages a list of Simple **Chars** Stereotyped options. Simple Choice Instruments always used **Chars** as their options type.

Instrument Name: Desktop.FileManager.IconSortBy

Type:	**Instrument**	**Type:**	**User Preference**
$Root:	$MAXX_SETTING/**Instruments**	**$URoot:**	$HOME/.maxxdesktop/msettings/**Preferences**
$Location:	*(calculated value)*	**$ULocation:**	*(calculated value)*
$Filename:	$Root/$Location/**IconSortBy.Choice**	**$UFilename:**	$URoot/$ULocation/**IconSortBy.Choice**

File content:

```
version=1.0
uuid=e828aeec-de4e-4899-9ebf-14e418570a71
stereotype=Choice
name=Desktop.FileManager.IconSortBy
default=0
type=Chars
option[0]=Name
option[1]=Size
option[2]=Type
option[3]=Date
```

File content:

```
version=1.0
uuid=e828aeec-de4e-4899-9ebf-14e418570a71
stereotype=Choice
name=Desktop.FileManager.IconSortBy
type=Chars
value=2
```

Note:

- In a Simple Choice Instrument, the options are defined as Simple Stereotype.

Complex *Command* Choice Instrument

This third example explores the System wide Instrument **Desktop.DtUtilities.WinEditor** which defines a list of Default Graphical Text Editor **Applications** used throughout the MaXX Desktop. This Instrument is a **Choice** Stereotype managing a list of **Command** Stereotyped options. We call them Complex Choice Instrument because they rely on a second set of Instruments to populate their list of options. It's like a Complex Choice using a list of Complex Instruments as options. See below for more details, it will make more sense as you read through a real example.

Instrument Name:	Desktop.DtUtilities.WinEditor		
Type:	**Instrument**	**Type:**	**User Preference**
$Root:	$MAXX_SETTING/**Instruments**	**$URoot:**	$HOME/.maxxdesktop/msettings/**Preferences**
$Location:	*(calculated value)*	**$ULocation:**	*(calculated value)*
$Filename:	$Root/$Location/**WinEditor.Choice**	**$UFilename:**	$URoot/$ULocation/**WinEditor.Choice**

File content:

```
version=1.0
uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6
stereotype=Choice
type=Application
name=Desktop.DtUtilities.WinEditor
default=0
option[0]=Application.WinEditor.Gedit
option[1]=Application.WinEditor.Nedit
```

File content:

```
version=1.0
uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6
stereotype=Choice
type=Application
name=Desktop.DtUtilities.WinEditor
value=1
```

In a Complex Choice Instrument, the options are defined as Complex Instruments. This allows for limitless customizations and extensibility in the future.

Instrument Name:	Application.WinEditor.Nedit	**Instrument Name:**	Application.WinEditor.Gedit
Type:	**Instrument**	**Type:**	**Instrument**
$Root:	$MAXX_SETTING/**Instruments**	**$Root:**	$MAXX_SETTING/**Instruments**
$Classification:	*(calculated value)*	**$Classification:**	*(calculated value)*
$Filename:	$Root/$Location/**Nedit.Command**	**$Filename:**	$Root/$Location/**Gedit.Command**

File content:

```
version=1.0
uuid=034d3104-fba0-4e1d-9530-d2e948de000b
stereotype=Command
name=Nedit
execPath=$MAXX_BIN/xnedit
execParams=
```

File content:

```
version=1.0
uuid=fc3bbe1a-da71-47c7-ba81-f759579990dc
stereotype=Command
name=Gedit
execPath=gedit
execParams=
```

Desktop User Experience Instruments

Class	Group	Schema
Desktop	**Mouse**	Acceleration.Gauge Threshold.Gauge LeftHanded.Logical WheelMouseScroll.Logical DoubleClick.Gauge NaturalScrolling.Logical
	Keyboard	KeyClick.Logical KeyRepeat.Logical RepeatSpeed.Gauge RepeatDelay.Gauge
	Background	BackgroundColors.Choice BackgroundImages.Choice DarkBackground.Logical Pattern1.Color Pattern2.Color Pattern3.Color
	DtUtilities	FileBrowser.Choice WinEditor.Choice TextEditor.Choice WebBrowser.Choice EmailClient.Choice ImageEditor.Choice ImageViewer.Choice MediaViewer.Choice VectorEditor.Choice PDFViewer.Choice
	Window	ToolchestHorizontal.Logical KeyboardFocus.Logical DisplayOverview.Logical MoveOpaqueWindow.Logical OutlineThickness.Gauge WindowManagerAccent.Color AutoWindowPlacement.Logical SaveWindowsDesks.Logical
	Settings	DesktopIconSize.Gauge DesktopIconAlignGrid.Logical DesktopAccent.Color ToolchestSoundEffect.Logical IconAsThumbnailImage.Logical ShowLaunchEffect.Logical MakeDeleteInstantly.Logical WarnOnFileOverwrite.Logical DisplayApplicationErrors.Logical EnableRemoteDisplay.Logical
	Colors	SGIScheme.Choice SgiDarkScheme.Logical UserInterfaceAccent.Color
	DtSounds	MuteSystem.Logical StartupShutdownTunes.Logical DesktopSounds.Logical SystemAlertsSounds.Logical KeyboardBell.Logical KeyClickVolume.Gauge DefaultSoundOutput.Choice

	Localization	Language.Choice KeyboardInput.Choice
	FileManager	DisplayShelf.Logical DisplayContent.Logical DisplaySearchFilters.Logical KeepLayoutOpenInPlace.Logical IconSortBy.Choice IconViewAs.Choice TruncateNames.Logical ThumbnailImages.Logical AlignToGrid.Logical DisplayHiddenFiles.Logical DefaultIconSize.Gauge
	IconCatalog	KeepLayoutOpenInPlace.Logical IconSortBy.Choice IconViewAs.Choice TruncateNames.Logical ThumbnailImages.Logical AlignToGrid.Logical DefaultIconSize.Gauge
	Text	SmallText.Font NormalText.Font LargeText.Font Terminal.Font WindowTitle.Font WindowIconTitle.Font IconText.Font SmoothText.Logical
	FontRendering	XftAutoHint.Logical XftLcdFilter.Choice XftHintStyle.Choice XftHinting.Logical XftAntialias.Logical XftRgba.Choice
	UserInterface	ModernLookAndFeel.Logical ThinWidgetMode.Logical FlatMenuMode.LogicalBonjour

Refer to the MaXX Settings Instrumentations Guide Document for the complete and up to date list of User Experience Instrumentation

Command Line Interface - CLI

MaXX Settings has its own Command Line Interface or commonly called CLI that supports interrogation and edition of settings in a human friendly way. The Standard CLI executable is called ***msettings*** and the Administrative CLI called ***ms*** and both can be found in the **$MAXX_BIN** directory. For example ***msettings*** could be used in a shell script to expand the current setting for the Desktop.DtUtilities.ImageEditor Instrument, or directly from the command-line, in **Toolchest** menus or even in Rox-Filer *"Set Run Action"*. MaXX Settings is also integrated with all aspects of the MaXX Interactive Desktop configuration and User's Preference Panels.

From a shell script - launching the default Graphical Text Editor

```
#!/bin/bash
Exec `msettings --value-only Desktop.DtUtilities.WinEditor`
```

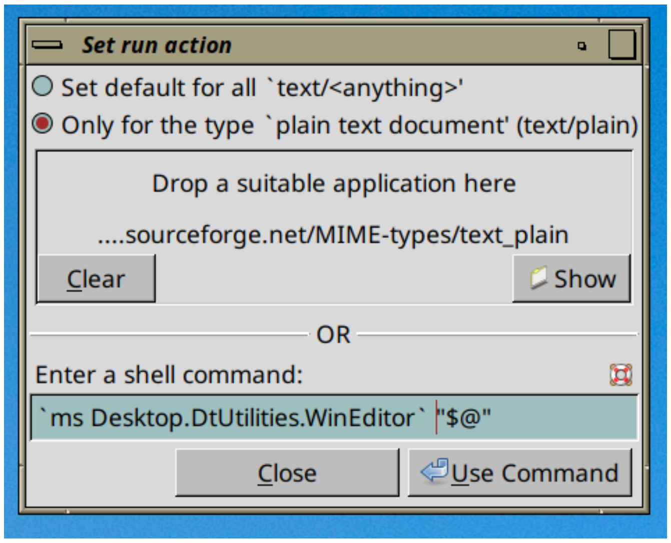
From command-line - fetching from MaXX Settings the default Image Editor

```
$ msettings --value-only Desktop.DtUtilities.ImgEditor
/usr/bin/gimp
```

From Toolchest Menu - launching the default Graphical Text Editor

```
"Text Editor"    f.checkexec.sh.le    "`msettings --value-only Desktop.DtUtilities.WinEditor`"
```

From ROX-Filer set 'Run Action' - setting the default Graphical Text Editor to launch for text/plain MIME type


Set run action
Drop a suitable application here
....sourceforge.net/MIME-types/text_plain
Clear
Show
OR
Close
Use Command

Set default for all `text/<anything>' [✓ Only for the type `plain text document' (text/plain)]

Enter a shell command:

```
`ms Desktop.DtUtilities.WinEditor` "$@"
```

CLI Commands and Parameters

This section will focus on the usage of MaXX Settings Command Line Interface or CLI with examples, commands and parameters documentation making your usage easier.

CLI Search Mechanism

MaXX Settings CLI provides two ways of searching for Instruments. First, it supports search by Instrument's Universal Unique IDentifier (UUID v4) and the second by Instrument's Name. MaXX Settings relies on internal indexes to make the search lightning fast regardless of the size of your data-set. This also means that manual addition/removal of Instrument might create unexpected results. Therefore, we always recommend to either use the CLI interface or one of the Java/C++ APIs.

CLI Interaction Modes

For your convenience, MaXX Settings CLI supports two interaction modes, comfort and admin. Comfort mode is aimed at providing a more padded user experience whereas the admin mode is a minimalist experience for die-hard console users.

Comfort Mode

The Confort CLI Mode provides help and an extra level of comfort with more human friendly output messages. A typical MaXX Settings CLI interaction could have one or many options, a mandatory command to execute, any number or parameters depending on the command itself and either one or a list of comma-separated lookup values.

Comfort Mode CLI command format

```
$ msettings [option] command [params] search-criteria
```

Comfort Mode Options

-h,	**--help**	Print the help information..
-v,	**--version**	Print the version information.
-D,	**--debug-mode-on**	Enable debug mode. Extra DEBUG information will be printed out in the console
-s,	**--silent-mode-off**	Turn OFF silent mode. This allows normal verbose outputs in the console.
-i,	**--no-case**	Disable case sensitivity when searching by name.

Admin Mode

Admin CLI Mode has no option, only the command to execute with its supported parameters, depending on the command itself. We dropped the dog and pony show in favor of simplicity. We encourage you to learn it and also use the short parameter form as well.

Admin Mode CLI command format

```
$ ms command [params] search-criteria
```

In Export mode, the GET command is defaulted, meaning it is the default command and can be omitted.

Administrative CLI Commands

This section will focus on administrative CLI commands, that are only available in export mode. All commands require superuser permissions level.

INIT Command

Init command will initialize the directory structure required by MaXX Settings in order to store Instruments and indexes. If the command was successful, a detailed report of the new MaXX Settings environment will be outputted.

This command is only available in Admin mode and requires superuser permissions level in order to initialize MaXX Settings System wide data structure. System wide initialization is defaulted.

INIT Command in Admin Mode CLI

```
$ ms INIT [params]

Examples:

# ms INIT

MaXX Settings - System wide Directory structure created.
MaXX Settings - System wide Database was successfully initialized.
MaXX Settings - System wide Indexes built and synched.
MaXX Settings - System wide Initialization completed. We are open for business.

Remember to set your MAXX_SETTINGS environment variable to : /opt/MaXX/share/msettings

# export MAXX_SETTINGS=/opt/MaXX/share/msettings

# ls -l $MAXX_SETTINGS

drwxrwxr-x. 2 root root 6 Jul  7 19:44 Applications
drwxrwxr-x. 2 root root 6 Jul  7 19:44 Choices
drwxrwxr-x. 2 root root 6 Jul  7 19:44 FileTypes
drwxrwxr-x. 2 root root 6 Jul  7 19:45 Indexes
drwxrwxr-x. 2 root root 6 Jul  7 19:44 Instruments

#
```

CREATE Command

Create a new global Instrument with a text file as input source using the *key=value* pairs Stereotype convention. If the command is successful, then a detailed report of the new Instrument creation will be outputted. To maintain data integrity and unicity, both the Instrument's name and UUID will be compared against the existing Instruments.

This command is only available in Admin mode and requires superuser permissions level.

Parameters

-f FILENAME,	**--filename=FILENAME**	Filename to use as input. The filename ideally describes the Instrument name, but must contain the Schema type name as extension.

Instrument Input File Format

Filename: *DesktopMouse_Acceleration.**Gauge***

```
name=Desktop.Mouse.Acceleration
minimum=1
maximum=20
scale=1
default=5
```

Create Input File attributes

- The ***uuid*** attribute is omitted from the input file as it is automatically generated while processing the create-transaction.
- The ***stereotype*** attribute can be omitted since the information is already available from the input filename's last portion.
- The ***version*** attribute can be omitted, if not present, the version ***1.0*** will be assigned at creation.
- The ***default*** attribute and all other Schema specific attributes are mandatory for Instrument creation operation.
- The ***value*** attribute is never required for any Instrument related operations.

CREATE Command in Admin Mode CLI

```
$ ms C [params] filename

Examples:

$ ms C -f ./DesktopMouse_Acceleration.Gauge

version=1.0
uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
stereotype=Gauge
name=Desktop.Mouse.Acceleration
minimum=1
maximum=20
scale=1
default=1
```

UPDATE Command

Update an existing global Instrument with a text file as input source using the *key=value* pairs Stereotype convention. If the command is successful, then a detailed report of the operation will be outputted. To maintain data integrity, only the editable attributes can be modified with this operation.

This command is only available in Admin mode and for superuser level users and ONLY a few attributes can be updated. Refer to the Instruments section for detail.

Parameters

-f FILENAME,	**--filename=FILENAME**	Filename to use as input. The filename ideally describes the Instrument name, but must contain the Schema type name as extension.

Instrument Input File Format

Filename: *`Desktop.Mouse.Acceleration.Gauge`*

```
uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
name=Desktop.Mouse.Acceleration
minimum=1
maximum=20
scale=2    <- - -  VALUE WE WANT TO UPDATE
default=10 <- - -  VALUE WE WANT TO UPDATE
```

Update Input File attributes

- The ***uuid*** and ***name*** attributes are mandatory and must both match the existing Instrument in question.
- The ***stereotype*** attribute can be omitted since the information is already available within the system.
- The attribute that requires an update. Not all attributes are editable. Consult the Schema's specification for detail.
- The ***value*** attribute is never required for any Instrument related operations.

UPDATE Command in Admin Mode CLI

```
$ ms U [params] filename

Examples:

$ ms U -f ./DesktopMouse_Acceleration.Gauge
version=1.0
uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
stereotype=Gauge
name=Desktop.Mouse.Acceleration
minimum=1
maximum=20
scale=2
default=10
```

Standard CLI Commands

From this point on, all CLI commands are intended for normal users and they all work in both comfort and export mode. Only the output might get a little more comfortable.

No initialization required when running Standard CLI Commands. If the user's MaXX Settings local database and directory structure are not present at the first invocation, MaXX Settings will install itself properly beforehand, then run the requested command.

SET Command

Set one or many User Preference by providing either the Instrument's UUID or Name as identifier. The Instrument must be present in the System wide database. See below for details.

Parameters

-u UUID=value	Single Instrument UUID.
-u UUID=value,UUID=value,UUID=value	Comma-separated list of Instrument UUIDs and their values.
-n name=value	Single Instrument name.
-n name=value,name=value,name=value	Comma-separated list of Instrument names and their values.

Comfort Mode

SET Command in Comfort Mode CLI

```
$ msettings [option] SET [params] identifier=value

Examples:

$ msettings SET -u 8f6e1638-91fe-4eae-9876-45a4e6686d74=True
8f6e1638-91fe-4eae-9876-45a4e6686d74=True

$ msettings SET -n Desktop.Mouse.Acceleration=True,Desktop.Mouse.Threshold=7
Desktop.Mouse.Acceleration=True
Desktop.Mouse.Threshold=7
```

Admin Mode

In Export mode, the SET command default its lookup mechanism to, by name. This reduces the required inputs. See the example below.

SET Command in Admin Mode CLI

```
$ ms S [params] indentifier=value

Examples:

$ ms S -u 8f6e1638-91fe-4eae-9876-45a4e6686d74=False
8f6e1638-91fe-4eae-9876-45a4e6686d74=False

$ ms S Desktop.Mouse.Acceleration=False,Desktop.Mouse.Threshold=10
Desktop.Mouse.Acceleration=False
Desktop.Mouse.Threshold=10
```

GET Command

Return one or many User Preference by providing either the Instrument's UUID or Name as identifier. If no User Preference is found, the command can be instructed to return its default value instead via the -d or --*default-value* parameter. The output format is customizable as well with the -f or --*output-format* parameter. See below for details.

The compact output format is defaulted for both modes and can be omitted from the CLI command. See the example below.

Parameters

-u UUID,	**--uuid=UUID**	Single Instrument UUID.
-u UUID,UUID,UUID		Comma-separated list of Instrument UUIDs. No space char allowed
-n name	**--name=NAME**	Single Instrument name.
-n name,name,name		Comma-separated list of Instrument names.
-x,	**--expand-detail**	Long output format where key=value pair is returned for every match.
-d,	**--default-value**	Returns and sets to the default value when the User Preference is not found.

Comfort Mode

GET Command in Comfort Mode CLI

```
$ msettings [option] GET [params] identifier(s)

Examples:

$ msettings -i GET -n desktop.mouse.lefthanded
False

$ msettings GET -x -n Desktop.Mouse.Acceleration
version=1.0
uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
stereotype=Gauge
name=Desktop.Mouse.Acceleration
minimum=1
maximum=20
scale=1
default=5
value=11

$ msettings -i GET -u 553e9f88-32c9-4477-910a-66fbeb104e3c
11
```

Admin Mode

In Export mode, the GET command is defaulted, meaning it can be omitted from the command line and the default lookup mechanism is by name. This reduces the required inputs. See the example below.

GET Command in Admin Mode CLI

```
$ ms G [params] search-criteria

Examples:

$ ms G Desktop.Mouse.Acceleration
11

$ ms Desktop.Mouse.LeftHanded,Desktop.Mouse.Acceleration
Desktop.Mouse.LeftHanded=False
Desktop.Mouse.Acceleration=11

$ ms -x -u 8f6e1638-91fe-4eae-9876-45a4e6686d74
version=1.0
uuid=8f6e1638-91fe-4eae-9876-45a4e6686d74
stereotype=Logical
name=Desktop.Mouse.LeftHanded
default=False
value=False
```

RESET Command

Reset to factory default value one or many User Preference by providing either the Instrument's UUID or Name as identifier. If no User Preference is found, the command can be instructed to create the missing User Preference and set it to its default value. See below for details.

Parameters

-u UUID,	**--uuid=UUID**	Single Instrument UUID and its value
-u UUID,UUID,UUID		Comma-separated list of Instrument UUIDs.
-n name	**--name=NAME**	Single Instrument name and its value.
-n name,name,name		Comma-separated list of Instrument names.

Comfort Mode

RESET Command in Comfort Mode CLI

```
$ msettings [option] RESET [params] search-criteria

Examples:

$ msettings -i RESET -u 8f6e1638-91fe-4eae-9876-45a4e6686d74
8f6e1638-91fe-4eae-9876-45a4e6686d74=False

$ msettings RESET -n Desktop.Mouse.Acceleration,Desktop.Mouse.Threshold
Desktop.Mouse.Acceleration=False
Desktop.Mouse.Threshold=5
```

Admin Mode

In Export mode, the RESET command default lookup mechanism is by name. See the example below.

RESET Command in Admin Mode CLI

```
$ ms R [params] search-criteria

Examples:

$ ms R -n Desktop.Mouse.LeftHanded
Desktop.Mouse.Acceleration=False

$ ms -u 8f6e1638-91fe-4eae-9876-45a4e6686d74
uuid=8f6e1638-91fe-4eae-9876-45a4e6686d74=False

$ ms R Desktop.Mouse.Acceleration,Desktop.Mouse.Threshold
Desktop.Mouse.Acceleration=False
Desktop.Mouse.Threshold=5
```

Index and Lookup Mechanism

Each Instrument under MaXX Settings can be looked up by its Instrument name, its unique ID (UUID) or full filename path. In order to provide fast and consistent performance, MaXX Settings relies on an internal database to reduce lookup time. This means that adding manually an Instrument without updating the indexes could result in lookup failures.

We always recommend to use the CLI interface when performing administrative tasks on Instruments. This way the database is kept in sync with the data and ensures optimal performance.

Lookup By UUID

From the CLI, a lookup to a User Preference by its UUID can be done this way.

```
$ msettings --uuid 76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
version=1.0
uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
stereotype=Logical
name=Desktop.Colors.SgiDarkScheme
value=True
$
$ msettings --value-only --uuid 76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
True
$
```

Lookup By Instrument Name

From the CLI, a lookup to a User Preference by its Instrument Name can be done this way.

```
$ msettings --name Desktop.Colors.SgiDarkScheme
version=1.0
uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c
stereotype=Logical
name=Desktop.Colors.SgiDarkScheme
value=True
$
$ msettings --value-only --name Desktop.Colors.SgiDarkScheme
True
$
```