

MaXX Settings Configuration Management

Architecture & Technical Specifications

Version RC1

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
0.99	2021-04-15	Eric Masson	Improve and simplify CLI section
RC1	2021-05-06	Eric Masson	Refactoring of Choice and Introduction of Catalog. Add diagrams and small corrections here and there to make this document as sharp as possible.

Table of Content

Table of Content	3
Synopsys	5
Requirements	5
Architecture	6
Development and Execution Platform	6
Instrumentation Data Persistence	6
Database	6
Instrumentations	7
Instrumentation Class Diagram	8
Naming Conventions	8
Definitions	9
Structural Element	9
Elementary...	9
Class	9
Group	9
Schema	9
Attributes	10
Mandatory Attributes	10
Stereotype	11
Simple Stereotype	11
Complex Stereotype	11
Choice Stereotype	13
1+1+1 > 3	14
How does it work?	15
Testing 1,2,3...	15
MaXX Desktop HOME	15
MaXX Settings System wide Root Directory	15
Instruments	16
Classification	16
System Wide Instruments	16
System Wide Nomenclature	17
User Preference	17
User Preference Nomenclature	17
User Experience Instruments	18
Simple Instrument	18
Simple Choice Instrument	19
Complex Command Choice Instrument	20
Desktop User Experience Instruments	21

Command Line Interface - CLI	23
CLI Commands and Parameters	24
CLI Search Mechanism	24
CLI Interaction Modes	24
CLI Options	24
Standard Mode	24
Admin Mode	24
Administrative CLI Commands	25
INIT Command	25
Parameters	25
CREATE Command	26
Parameters	26
Instrument Input File Format	26
Create Input File attributes	26
UPDATE Command	27
Parameters	27
Instrument Input File Format	27
Update Input File attributes	27
Standard CLI Commands	28
SET Command	28
Parameters	28
SET CLI Command	28
GET Command	29
Parameters	29
GET CLI Command	29
RESET Command	30
Parameters	30
RESET CLI Command	30
Index and Lookup Mechanism	31
Lookup By UUID	31
Lookup By Instrument Name	31

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

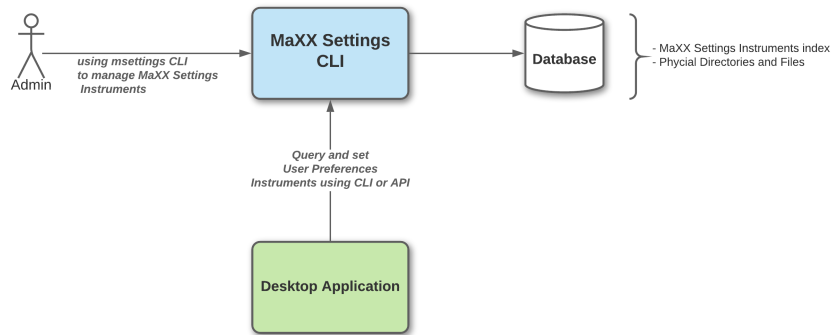
One of the benefits of starting fresh is the fact that we can start with a blank slate, put forward clear intents, express technical requirements and build an architecture early on in the design process.

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.



The diagram illustrates the overall architecture of MaXX Settings.

Development and Execution Platform

Java 8 was selected for the first implementation of the CLI Engine for its richness in features, robustness, maturity and special affinity with server/service type APIs and prebuilt components. The [GaalVM](#) 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 CLI Engine is persisted into regular files of different natures. No external dependency required for the database.

Database

The database file format is implemented using fixed length field strategy to ensure simplicity with high performance seek speed. The design of the database file allows an all in memory lookup operations. 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.

```
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);

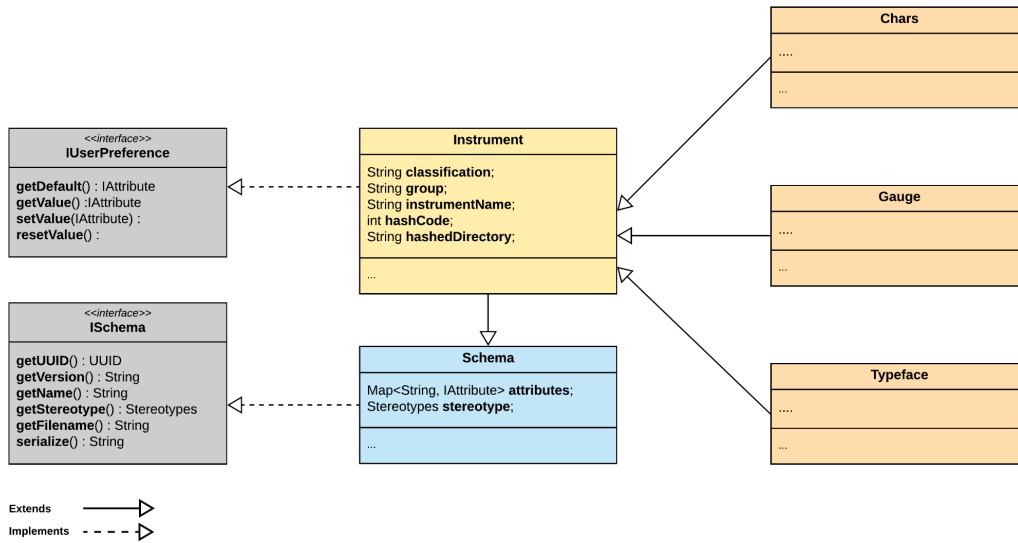
    boolean lookup(@NotNull String lookup);
}
```

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.

Instrumentation Class Diagram



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.

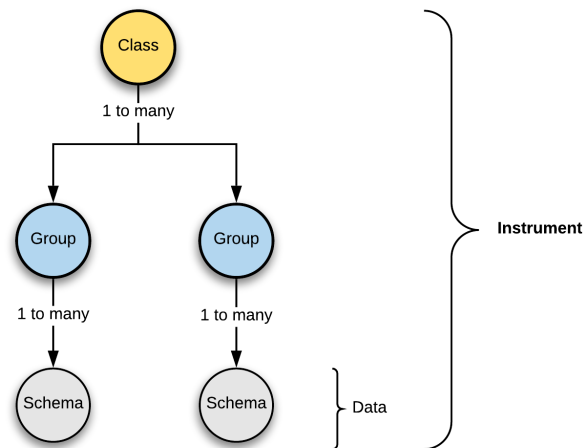
The table below lists the naming convention used in MaXX Settings.

	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 <i>encoding</i> attribute is mandatory and is set to UTF-8 by default. The <i>maxLength</i> 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, they are called arrays. The <i>type</i> attribute defines the option's subtype. Type can either be Chars for simple value or Catalog where the options are stored into another Schema. Each <i>option[]</i> entry is a possible choice. The <i>default</i> and <i>value</i> attributes are indexes pointing back to the Choice's <i>option[]</i> array. Index starts at 0.	default=0 ! type=Chars *! option[i] *!	stereotype=Choice value=1 type=Chars option[0]=foo option[1]=bar

MaXX Settings - Configuration Management Simplified

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 <i>minimum</i> , <i>maximum</i> and an incremental value as <i>scale</i> . Mouse Sensitivity user preference is using Gauge for example. → The <i>default</i> and <i>value</i> are specific to each Gauge but their values must be between the <i>minimum</i> and <i>maximum</i> .	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 preferences. BackgroundColor is such an example. Color is composed of a mandatory <i>colorSpace</i> and optional attribute <i>alpha</i> . The attribute <i>value</i> is populated with matching <i>colorSpace</i> 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 preference. BackgroundImage is such an example. Image is composed of a mandatory <i>filePath</i> with the optional attributes <i>crop</i> , <i>dimension</i> , <i>resizeTo</i> which can be used to apply a transformation on the original size. The attribute <i>value</i> is populated with the image filename.→ No Default value.	filePath* dimension crop resizeTo	stereotype=Image default=image.png value=image.png filePath=/temp dimension=256x256
Typeface	Represents a Typeface used in user preference. TerminalFont is such an example. Typeface is composed of mandatory <i>font</i> name and a <i>size</i> with the optional <i>style</i> , <i>weight</i> and <i>slant</i> attributes. The attribute <i>value</i> is generated from a concatenation of all present attributes and cannot be set directly. → Both default and value attributes are using fully qualified format and here's an example: <i>Noto:size=10:slant=Italic:weight=Medium</i>	font* size* style weight slant	stereotype=Typeface default=Mono:size=10 font=Noto Sans size=12 ?value=Noto Sans:size=12
Command	Represents an executable Command used to launch an application. A Command is composed of the mandatory attributes <i>execName</i> and <i>execPath</i> , and optional attributes <i>execParams</i> , <i>envBinaryPath</i> , <i>envLibraryPath</i> and <i>geometry</i> . The attribute <i>value</i> is generated from the concatenation of <i>execPath</i> with <i>execName</i> and cannot be set directly. → Both default and value attributes are using a fully qualified command line format and here's an example: <i>/usr/bin/nedit file.txt -s param</i> [execPath]execName [execPrams]	execName* execPath* execParams envBinaryPath envLibraryPath geometry	stereotype=Command default=/usr/bin/nedit execName=xnedit execPath=/opt/MaXX/bin *value=/opt/MaXX/bin/xnedit
Application	Represent a list of Command names for specific application's actions such as: open, view, edit, etc. The default attribute is pointing to the default action (open) action when no other actions are provided. → Default value is the "open" action	viewCommand editCommand	stereotype=Application default=? value=Desktop.Editor.Nedit openCommand=Desktop.Editor.Nedit

* mandatory attribute in User Preferences

! 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
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
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, ffplay
VectorEditor.Choice	Command	Inkscape
PDFViewer.Choice	Command	Xpdf, Evince
BackgroundColors.Choice	Color	...
BackgroundImages.Choice	Image	...

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.

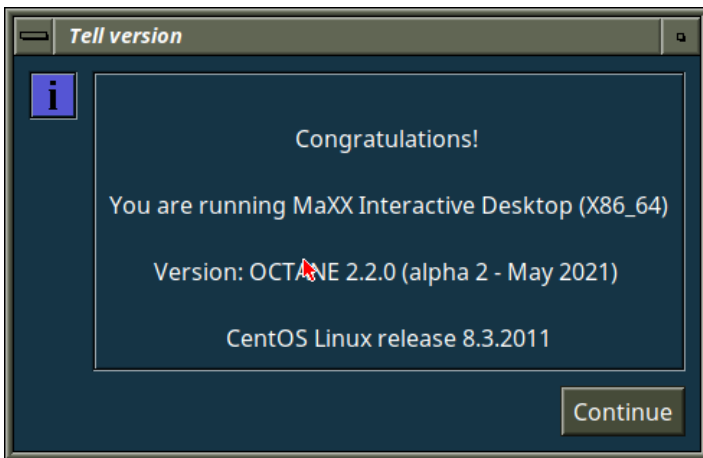
MaXX Desktop HOME

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
```



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 KeyboardSettings KeyboardShortcuts Background DtUtilities Window Settings Colors DtSounds Localization Text FontRendering UserInterface 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.	
Nme	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 will look at Catalog as a mechanism to 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 Instrument

This first example describes how MaXX Settings manages a *Simple System wide* Instrument like as **Desktop.Mouse.NaturalScrolling** and how it looks like from the **User Preference** (scope) perspective as well. This Instrument is a **Logical Stereotype**, and it is used to inform users mouse scrolling direction.

Instrument Name: Desktop.Mouse.NaturalScrolling			
Type:	System wide	Type:	User Preference
\$Root:	\$MAXX_SETTING/Instruments	\$URoot:	\$HOME/.maxxdesktop/msettings/Preferences
\$Location:	(calculated value)	\$ULocation:	(calculated value)
\$Filename:	\$Root/\$Location/NaturalScrolling.Logical	\$Filename:	\$URoot/\$ULocation/NaturalScrolling.Logical
File content:		File content:	
<pre>version=1.0 uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c stereotype=Logical name=Desktop.Mouse.NaturalScrolling default=false</pre>		<pre>version=1.0 uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c stereotype=Logical name=Desktop.Mouse.NaturalScrolling value=true</pre>	

Important things to remember

- Both the System wide and User Preference are using the same UUID and name in order to provide a lookup by UUID or name.
- The calculated storage location is identical for both. This makes the switch between System wide and User Preference seamless.
- 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. The Instrument is a **Choice Stereotype**, which manages a list of Simple **Chars** Stereotype options. Simple **Choice** Instruments always used **Chars** as its option type and stored them within the same Schema file as described below.

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:		File content:	
<pre>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</pre>		<pre>version=1.0 uuid=e828aeec-de4e-4899-9ebf-14e418570a71 stereotype=Choice name=Desktop.FileManager.IconSortBy type=Chars value=2</pre>	

Note:

- In a Simple Choice Instrument, the options are defined as Simple Stereotype Chars.
- Options are stored within the Choice Schema file

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 Application Stereotyped as **Command** options. We call them Complex Choice Instruments because they rely on an additional set of Instruments and Schemas to fulfil their purposes. They are the same Choices we learned about, but they are also using external **Catalogs** and a resolution mechanism that allows dynamic options management. 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: <pre>version=1.0 uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6 stereotype=Choice type=WinEditor.Catalog name=Desktop.DtUtilities.WinEditor default=0</pre>		File content: <pre>version=1.0 uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6 stereotype=Choice type=WinEditor.Catalog name=Desktop.DtUtilities.WinEditor value=1</pre>	

In a **Complex Choice** Instrument, the options are defined in a **vim** . Schema that is referenced in the Choice's Schema. This allows for limitless customizations and extensibility in the future.

Instrument Name: Application.WinEditor.XNEdit		Catalog Name: WinEditor.Catalog	
Type:	Instrument	Type:	Catalog
\$Root:	\$MAXX_SETTING/Instruments	\$Root:	\$MAXX_SETTING/Catalogs
\$Classification:	(calculated value)	\$Filename:	\$Root/\$WinEditor.Catalog
\$Filename:	\$Root/\$Location/XNEdit.Command		
File content: <pre>version=1.0 uuid=034d3104-fba0-4e1d-9530-d2e948de000b stereotype=Command name=XNEdit execPath=\$MAXX_BIN/xnedit execParams=</pre>		File content: <pre>version=1.0 uuid=fc3bbe1a-da71-47c7-ba81-f759579990dc stereotype=Catalog name=Command option[0]=@Application.WinEditor.XNEdit option[1]=@Application.WinEditor.Gedit</pre>	

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
	Sounds	MuteSystem.Logical StartupShutdownTunes.Logical DesktopSounds.Logical SystemAlertsSounds.Logical KeyboardBell.Logical KeyClickVolume.Gauge DefaultSoundOutput.Choice

MaXX Settings - Configuration Management Simplified

	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.Logical Bonjour

Refer to the **MaXX Settings Instrumentation Guide** for the complete and up to date list of all 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 GET -n Desktop.DtUtilities.WinEditor`
```

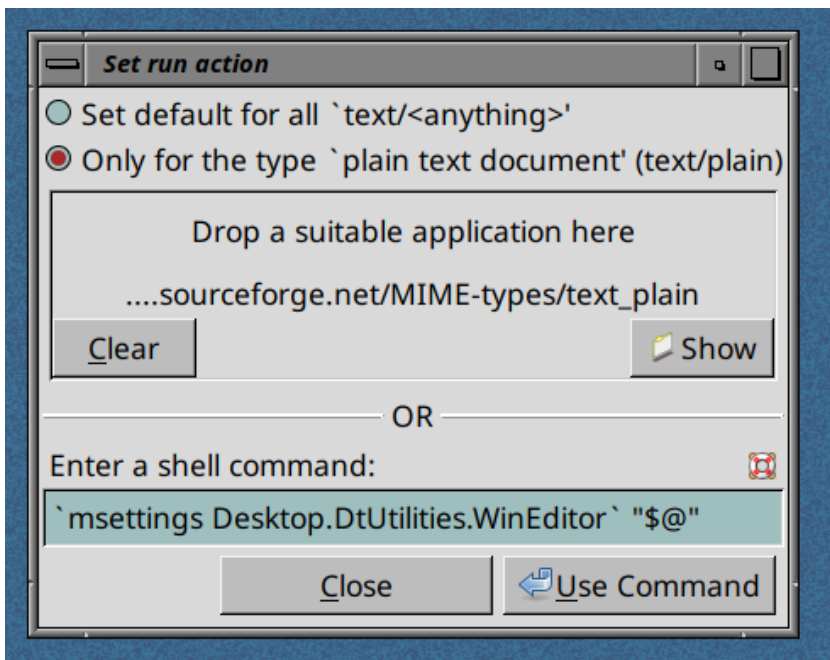
From command-line - fetching from MaXX Settings the default Image Editor

```
$ msettings GET Desktop.DtUtilities.ImgEditor
/usr/bin/gimp
```

From Toolchest Menu - launching the default Graphical Text Editor

```
"Text Editor"    f.checkexec.sh.le    "`msettings G Desktop.DtUtilities.WinEditor`"
```

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



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 such as *Desktop.Mouse.Acceleration*. MaXX Settings relies on internal indexes to make the search lightning fast regardless of the size of your data-set. It is recommended to always use the CLI interface or one of the Java/C++ APIs when interacting with MaXX Settings. No manually hacking.

CLI Interaction Modes

For your convenience, MaXX Settings CLI supports two interaction modes, standard and admin. Standard mode is aimed at providing support for User Preferences whereas the Admin mode is a minimalist interface for Instruments management, with superuser privilege.

CLI 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.

Standard Mode

The Standard CLI Mode provides a simple read and write access to User Preferences without complexity, but with optional powerful features. A typical MaXX Settings Standard CLI command is named **msettings** which is composed of a mandatory **command** to execute, **options** (if needed), a number of **parameters** depending on the command itself and a **value**. The value can either be a single name, an uuid or a comma-separated list of key=value pairs.

Standard Mode CLI command format

```
$ msettings command [options] [params] value
$ msettings command [options] [params] value1,value2,value3
$ msettings command [options] [params] key1=value1,key2=value2
```

Admin Mode

Admin CLI Mode provides support for Instrument management with superuser privilege. The Admin CLI command is named **ms** and follows the same command line scheme as the Standard mode.

Admin Mode CLI command format

```
$ ms command [options] [params] value
$ ms command [options] [params] value1,value2,value3
$ ms command [options] [params] key1=value1,key2=value2
```


Administrative CLI Commands

This section will focus on the administrative aspect of MaXX Settings CLI and the CLI commands that are only available in Admin 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.

Parameters

-F, --force Force the initialization over an existing MaXX Settings environment. This will erase everything stored in the Database and wipe out all Instruments.
Use this with caution.

INIT CLI Command-line example:

```
$ ms INIT [params]

Examples:

# ms INIT --force

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 19:44 Instruments

#
```

CREATE Command

Create a new global Instrument with a text file as input source and using *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.
-A key=value,key=value,key=value,...		Attributes in Key/Value pair format separated by comma.

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 CLI Command-line examples:

```
$ ms CREATE [options] [param] filename
```

Examples:

```
$ ms CREATE -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=5
```

```
$ ms CREATE -A \
stereotype=Choice,name=Desktop.FileManager.IconSortBy,type=Chars,default=1,option[0]=Name,option[1]=Date
```

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 CLI Command-line example:

```
$ ms UPDATE [options] [param] 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 Standard mode. Normal user access privileges are required.

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. Should always be the last param.
-u UUID=value,UUID=value,UUID=value	Comma-separated list of Instrument UUIDs and their values.
-n name=value	Single Instrument name. Should always be the last param.
-n name=value,name=value,name=value	Comma-separated list of Instrument names and their values.

SET CLI Command

Command-line examples:

```
$ msettings SET [options] [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=10  
Desktop.Mouse.Acceler!nation=10  
  
$ msettings 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 will output nothing unless you turn off silent-mode. The output is customizable as well with a full-detail, key-value or value only format to cover all integration needs. See below for details.

Parameters

-u UUID,	--uuid=UUID	Single Instrument UUID. Should always be the last param.
-u UUID,UUID,UUID		Comma-separated list of Instrument UUIDs. No space char allowed
-n name	--name=NAME	Single Instrument name. Should always be the last param.
-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.
-X,	--expand-detail	Detailed output format, print all attributes in Key=Value pair for every requested Instrument.
-x,	--value-only	Output only the Value for every requested Instrument. This is the DEFAULT settings.
-K,	--key-value	Output in key=value format for every requested Instrument.

GET CLI Command

Command-line example:

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

Examples:

$ msettings GET -n Desktop.Mouse.Acceleration
5.0

$ msettings GET -K Desktop.Mouse.Acceleration
value=5.0

$ msettings GET -u 553e9f88-32c9-4477-910a-66fbeb104e3c
5

$ msettings G -X Desktop.Mouse.Acceleration
version=1.0
uuid=553e9f88-32c9-4477-910a-66fbeb104e3c
stereotype=Gauge
name=Desktop.Mouse.Acceleration
minimum=1.0
maximum=20.0
scale=1.0
default=5.0
value=5.0

$ msettings G Desktop.Mouse.LeftHanded,Desktop.Mouse.Acceleration
False
5.0

$ msettings G -K Desktop.Mouse.LeftHanded,Desktop.Mouse.Acceleration
Desktop.Mouse.LeftHanded=False
Desktop.Mouse.Acceleration=5.0
```

RESET Command

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

The RESET CLI command is a great way to do a first-time initialization of User Preferences.

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.

RESET CLI Command

Command-line examples:

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

Examples:

$ msettings 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

$ msettings R Desktop.Mouse.LeftHanded
Desktop.Mouse.Acceleration=False

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

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