# MaXXsettings Configuration Management

## Technical Specifications

Version RELEASE v1.0

## 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. |
| RC2 | 2021-05-17 | Eric Masson | Adding LIST and HASH Common CLI Commands. |
| RC3 | 2024-12-08 | Eric Masson | Clean up, tightening the overall document. |
| 1.0 | 2024-12-23 | Eric Masson | Release 1.0 |

# Table of Content

**Table of Content** 3
**Synopsys** 5
**Requirements Reminder** 5
**Architecture** 6
Development and Execution Platform 6
Data Persistence 6
Database 6
**Naming Conventions** 8
**Instrumentation and Structure** 9
The Structural Elements of MaXXsettings 9
Class 9
Group 9
Schema 9
Attributes 10
**Stereotype: The Foundation of Schema Consistency** 11
Schema Categorization 11
Simple Stereotype 11
Complex Stereotype 13
Enums: Predefined Sets for Validation 15
Choice Stereotype 16
**Unlocking the Power of Instruments** 17
What is an Instrument? 17
The Importance of Complete Structure 18
Why Instruments Matter 18
System-wide Instruments 19
Access and Permissions 19
System-wide Instrument Deconstruction 19
A Practical Example: Desktop.Mouse.Acceleration 20
Schema and User Preferences: An OOP Perspective 20
Editing Instruments: Best Practices 20
Refer to the CLI Section for detailed instructions on managing Instruments 20
User Preference Instruments 21
Shared Classification and Storage Logic 21
Why This Matters 21
User Preference Deconstruction 21
**Exploring User Experience Instruments: Scopes and Flexibility** 23
Example 1: Managing Desktop.Mouse.NaturalScrolling 23
Instrument Overview 23
Example 23
Example 2: Managing Desktop.FileManager.IconSortBy 24
Instrument Overview 24
Key Characteristics of Simple Choice Instruments 24
Example 24
Example 3: Managing Desktop.DtUtilities.WinEditor 25
Instrument Overview 25
Key Characteristics of Complex Command Choice Instruments 25
A Real-World Example 26

Dynamic Resolution in Action ... 26
Key Takeaway ... 26
**Catalog: Extending Choice Instrument Options ... 27**
Key Benefits of Catalog ... 27
Example ... 27
Resolver: Dynamic Value Resolution ... 27
Key Features of Resolvers ... 27
Usage ... 27
Popular Use Cases ... 27
Resolver Workflow with a Simple Choice ... 28
Resolver Workflow with a Complex Choice and Catalog ... 28
**Command Line Interface - CLI ... 29**
CLI Commands and Parameters ... 30
CLI Search Mechanism ... 30
CLI Interaction Modes ... 30
Common CLI Commands ... 31
LIST Command ... 31
HASH Command ... 32
Administrative CLI Commands ... 33
INIT Command ... 33
CREATE Command ... 34
UPDATE Command ... 35
Standard CLI Commands ... 36
SET Command ... 36
GET Command ... 37
RESET Command ... 38
**Index and Lookup Mechanism ... 39**
Classification ... 39
Why This Matters ... 39
Lookup By UUID ... 39
Lookup By Instrument Name ... 40

# Synopsys

**MaXXsettings** is a robust and dynamic configuration management subsystem, purpose-built for simplicity without compromising flexibility or extensibility. It includes a powerful **CLI interface**, enabling seamless management, scripting automation, inline queries, and straightforward integration with applications.

In addition, **MaXXsettings** offers **Java** and **C++ bindings**, making it highly adaptable and easy to integrate into modern applications. It supports the definition of system-wide configurations, known as **Instruments**, as well as user-specific overrides, referred to as **User Preferences**, providing a versatile framework for both centralized and personalized settings management.

This document explains all that there is to know about **MaXXsettings Technical Specifications** and how to get started.

For in-depth instrumentation and implementation details, refer to the [MaXXsettings Instrumentations Guide for MaXXdesktop] document.

# Requirements Reminder

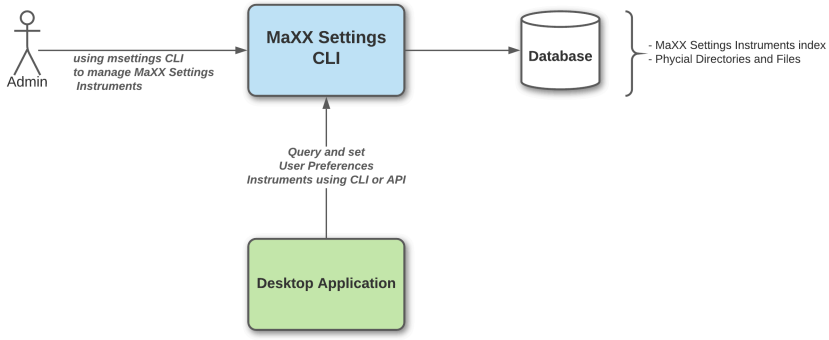
One of the benefits of starting fresh is the fact that we have a blank slate, put forward clear intentions, express technical requirements and build an architecture early on in the design process.

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

- Retrieve information as fast as possible (lookup speed is a flat-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 persist 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 MaXXsettings is built follows the classic Client/Server model, where the Client is represented by “users” using CLI interface or Applications to interact with the Server. The Server provides the necessary functionality to manage configurations, we call them instrumentations.




The diagram illustrates the overall architecture of MaXXsettings.

## 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 backend/service APIs and prebuilt components. The GraalVM was selected for its ability to compile Java byte-code into native code and offer optimizations with added benefits of reducing startup and boosting execution speed quite considerably.

## Data Persistence

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

### Database

The database format is a fixed-length field strategy to ensure simplicity and enable high-performance read and seek operations. Its design leverages computable hash codes derived from the **Instrument's** name or key, which are mapped to a hierarchical directory structure. The stored information is intentionally minimal, as the database primarily serves as a lookup mechanism, optimizing both storage efficiency and access speed.

#### Database Interface

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;

/**
* MaXXsettings 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);


   List<IndexEntry> fetchAll();

   Integer count();
}
```

# 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 MaXXsettings.

| | **Convention** | **Samples** |
|---|---|---|
| **Attribute** | One or multiple words in **camel case** 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 |

# Instrumentation and Structure

At the core of **MaXXsettings** is the concept of **Instrumentation**, which provides a structured approach to managing configurations and organizing complex hierarchical datasets. Through a set of well-defined abstractions, **Instrumentation** ensures that configurations adhere to **mandatory attributes** and **operational validation rules**, while maintaining data consistency across the system.

## The Structural Elements of MaXXsettings

In this section, we explore the foundational elements of **MaXXsettings**, their purpose, and how they work together to form a cohesive and hierarchical configuration management system.

### Class

The **Class** serves as the cornerstone of the information hierarchy, acting as the root node that organizes **Groups** and their associated **Schemas** into well-structured, hierarchical data sets.

Starting with the **MaXXdesktop Octane** release, the **MaXXsettings API** supports three distinct **Class types**:

1. **Desktop**: Manages user experience settings specific to the **MaXXdesktop** environment.
2. **Application**: Defines actions (e.g., `open`, `view`, and `edit`) linked to individual applications.
3. **FileType**: Identifies MIME file types and creates associations between file types and their respective applications.

### Group

A **Group** is always associated with a single **Class** and represents a logical, configurable entity or concept—such as "Mouse" or "Background"

While Groups do not hold data themselves, they act as **organizational placeholders** that group multiple **Schemas** under a single logical unit. In essence, Groups provide **context** to Schemas, much like Classes categorize Groups.

### Schema

A **Schema** is the final and most critical structural element in the **MaXXsettings Instrumentation hierarchy**. As the **leaf node**, it is always associated with a single **Group** and holds the actual configuration data that defines behaviors, characteristics, or attributes—collectively governed by a **Stereotype**.

To put it simply, a Schema is a text file containing **attributes** that describe settings or configurations. This abstraction enables:

- **Flexibility**: Accommodates diverse use cases with minimal overhead.
- **Structure**: Organizes settings into a predictable and maintainable format.
- **Scalability**: Supports configurations ranging from basic user preferences to complex, system-wide settings.

### Attributes

An **Attribute** represents the smallest unit of granularity within the **MaXXsettings** framework. Attributes are defined as **key-value pairs** and serve as the properties of a **Schema**.

To ensure **consistency** and **predictability**, every Schema must include the following **mandatory attributes**:

1. **Version Number**: Specifies the version of the Schema's data contract.
2. **Universally Unique Identifier (UUID v4)**: Ensures each Schema is uniquely identifiable across the system.
3. **Stereotype**: Indicates the Stereotype governing the Schema's structure and behavior.
4. **Unique Name**: Provides a clear and specific identifier for the Schema.

These attributes form the foundation of every Schema, ensuring it remains **distinct**, **consistent**, and adheres to a predictable structure.

By understanding these structural elements—**Class**, **Group**, **Schema**, and **Attributes**—you can fully appreciate how **MaXXsettings** achieves a clear, scalable, and robust configuration management system.

## 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 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 |

## Optional Attributes

| Attribute | Description | Example |
|---|---|---|
| **description** | Human readable text description of the Schema | description=SGI Color Scheme |

# Stereotype: The Foundation of Schema Consistency

A **Stereotype** is a powerful mechanism that defines the structure and behavior of a **Schema**. By using standardized attributes and optional validation rules, Stereotypes help ensure proper usage and deliver consistent, predictable outcomes. While Stereotypes also support advanced features like behavior modeling (a topic for the future), their primary role is to maintain a consistent framework for defining and managing information within Schema files.

## Schema Categorization

In **MaXXsettings**, **Schemas** are grouped into two categories: **Simple** and **Complex Stereotypes**—each with its own distinct personality and purpose.

**Simple Stereotypes** are the backbone of straightforward configurations, streamlining implementation and maintenance. They ensure reliability and predictability across the system with minimal effort, making them the go-to choice for basic, static settings.

Then there are the **Complex Stereotypes**—a whole different breed. These aren't just about storing values; they're designed for advanced use cases, offering specialized behaviors like managing static or dynamic value sets and enabling dynamic resolution. From the rich versatility of **Choice** Stereotype to the powerful extensibility of **Catalogs**, **Complex Stereotypes** elevate Schemas to a whole new level of capability.

Curious about their full potential? Stay tuned for the next section, where we dive deeper into what makes these Complex Stereotypes so dynamic and indispensable.

## Simple Stereotype

As the name suggests, **Simple Stereotypes** are used for straightforward, single-value settings. These represent basic data types such as numbers, strings, or booleans. With the exception of the **Chars Stereotype**, Simple Stereotypes do not support validation rules, making them lightweight and easy to use.

### 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.

#### Attributes

| Name | Sample |
|---|---|
| default | null |
| encoding | UTF-8* |
| maxLength | 256 |

* *mandatory attribute*

#### Example

```
version=1.0
uuid=d61fc339-25aa-4fba-96bb-98b2fe1e1435
stereotype=Chars
name=Desktop.Colors.SgiScheme
default=IndigoMagic
encoding=UTF-8
maxLength=128
```

## Decimal

Represents an unsigned numerical value with single precision decimal/floating-point.

### Attributes

| Name | Sample |
|---|---|
| default | 0.0 |

### Example

```
version=1.0
uuid=d61fc339-25aa-4fba-96bb-98b2fe1e1435
stereotype=Decimal
name=Desktop.Text.TextScaleFactor
default=1.0
```

## Logical

Represents a boolean value of either true or false.

### Attributes

| Name | Sample |
|---|---|
| default | false |

### Example

```
version=1.0
uuid=d61fc339-25aa-4fba-96bb-98b2fe1e1435
stereotype=Logical
name=Desktop.Window.MoveOpaqueWindow
default=true
```

## Number

Represents an unsigned integer numerical value.

### Attributes

| Name | Sample |
|---|---|
| default | 0 |

### Example

```
version=1.0
uuid=d61fc339-25aa-4fba-96ba-18b2fe1e1434
stereotype=Number
name=Desktop.WorkSpace.VirtualDesktopCount
default=1
```

## Complex Stereotype

**Complex Stereotypes** are where things get interesting. Designed to handle settings with multiple interdependent values, they are built to tackle more intricate and dynamic configurations. Unlike their simpler counterparts, **Complex Stereotypes** bring structure and precision to advanced use cases by introducing validation rules that:

- **Enforce Constraints**: Ensure values adhere to specific rules and ranges.
- **Define Defaults**: Establish fallback values for seamless operation.
- **Restrict Ranges**: Prevent invalid configurations by limiting acceptable inputs.

This added layer of validation guarantees that settings behave exactly as intended, no matter how dynamic or sophisticated the requirements become.

In short, **Complex Stereotypes** don't just store values—they manage relationships, enforce order, and ensure everything works together flawlessly. It's precision engineering for your configuration management.

| Name | Description | Attributes | Example |
|---|---|---|---|
| **Dimension** | Represents a two dimensional measurement composed of width and height as positive only single precision decimals. | default=1.0x1.0 ! | stereotype=Dimension<br>default=1.0x1.0<br>value=290.0x100.0 |
| **Location** | Represents a 2D location composed of X and Y as signed integers. | default=+0+0 ! | stereotype=Location<br>default=+0+0<br>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<br>default=1x1+0+0<br>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 user preference is using Gauge for example.<br>→ The *default* and *value* are specific to each Gauge but their values must be between the *minimum* and *maximum*. | default=1.0 !<br>minimum=1.0 *<br>maximum=10.0 *<br>scale=1.0 * ! | stereotype=Gauge<br>default=1.0<br>value=7.0<br>minimum=1.0<br>maximum=10.0<br>scale=1.0 |
| **Color** | Represents a Color commonly used in user preferences. BackgroundColor is such an example. Color is composed of a mandatory *colorSpace* and optional attribute *alpha*. The attribute *value* is populated with matching *colorSpace* color components separated with commas. → No Default value. | default !<br>colorSpace*<br>alpha | stereotype=Color<br>default=255,255,255<br>value=127,231,48<br>colorSpace=RGB255<br>alpha=1.0 |
| **Image** | Represents an Image user preference. BackgroundImage is such an example. Image is composed of a mandatory *filePath* with the optional attributes *crop, dimension, resizeTo* which can be used to apply a transformation on the original size. The attribute *value* is populated with the image filename.→ No Default value. | filePath*<br>dimension<br>crop<br>resizeTo | stereotype=Image<br>default=image.png<br>value=image.png<br>filePath=/temp<br>dimension=256x256 |
| **Typeface** | Represents a Typeface used in user preference. TerminalFont is such an example. Typeface is composed of mandatory *font* name and a *size* with the optional *style*, *weight* and *slant* attributes. The attribute *value* is generated from a concatenation of all present attributes and cannot be set directly.<br>→ Both default and value attributes are using fully qualified format and here's an example:<br>*Noto:size=10:slant=Italic:weight=Medium* | font*<br>size*<br>style<br>weight<br>slant | stereotype=Typeface<br>default=Mono:size=10<br>font=Noto Sans<br>size=12<br>?value=Noto Sans:size=12 |
| **Command** | Represents an executable Command used to launch an application. A Command is composed of the mandatory attributes *execName* and *execPath*, and optional attributes *execParams*, | execName*<br>execPath*<br>execParams | stereotype=Command<br>default=/usr/bin/nedit<br>execName=xnedit |

| | | | |
|---|---|---|---|
| | *envBinaryPath*, *envLibraryPath* and *geometry*. The attribute *value* is generated from the concatenation of *execPath* with *execName* and cannot be set directly.<br>→ Both default and value attributes are using a fully qualified command line format and here's an example:<br>*/opt/MaXX/bin64/xnedit file.txt -s param*<br>[execPath]execName [execPrams] | envBinaryPath<br>envLibraryPath<br>geometry<br>runInTerminal | execPath=/opt/MaXX/bin64<br>*value=/opt/MaXX/bin64/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. (Still WIP)<br>→ Default value is the "open" action | viewCommand<br>editCommand<br>openCommand | stereotype=Application<br>~~default=?~~<br>value=@Desktop.Editor.Nedit<br>editCommand=@Desktop.Editor.Nedit |
| **Enum** | (Still WIP) | | |
| **Catalog** | (Still WIP) | | |
| **Choice[TYPE]** | Represents a typed indexed container of values. In most programming languages, they are called arrays. The *type* 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 *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 !<br>type=Chars * !<br>option[ i ] ! | stereotype=Choice<br>value=1<br>type=Chars<br>option[0]=foo<br>option[1]=bar |

** mandatory attribute in User Preferences*
*! updatable default value via CLI*

## Enums: Predefined Sets for Validation

**Enums** (Enumerations) will be used, in the next major version of **MaXXsettings**, to associate a **predefined** set of static/read-only values for **Complex Choices**. **Scope** is limited to **system-wide**, meaning they are not extensible unless the **Enum** itself is changed. The **type** of **values** are limited to **Chars** and they do not support dynamic resolver.

For example, the **FontStyle Enum** could be used to define predefined values for a Font:

- Normal
- Bold
- Slant
- Italic

**Enums** play a critical role in **MaXXsettings** by:

- **Ensuring Consistency**: Providing a shared data contract for users and administrators to reference.
- **Enforcing Stronger Type Constraints**: Reducing the risk of invalid or unexpected configurations.
- **Validating Inputs**: Restricting attribute values to only the predefined sets.

To maintain robustness, **Enums** are exclusively defined at the **system-wide level**, ensuring they serve as a stable and reliable validation mechanism.

### Enums Location

**$Root:** `$MAXX_SETTING/Enums`
**$Filename:** `$Root/<Schema>.Enum`

## Choice Stereotype

The **Choice Stereotype** is a special type of **Complex Stereotype** used to define a list of predefined, system-wide options called **Choices**. These are commonly used with **Desktop User Experience Instruments** for settings that involve selecting from a predefined set of values. Chars, **Enums** and **Catalogs** are the Schemas used for the options (values) as they offer robust mechanisms for predefined value sets. More on them later, read on...

*Example:* A Schema for "UI theme selection" that offers predefined options such as Light, Dark, or System Default.
Refer to the MaXXsettings Instrumentations Guide for MaXXdesktop document for more details.

# Unlocking the Power of Instruments

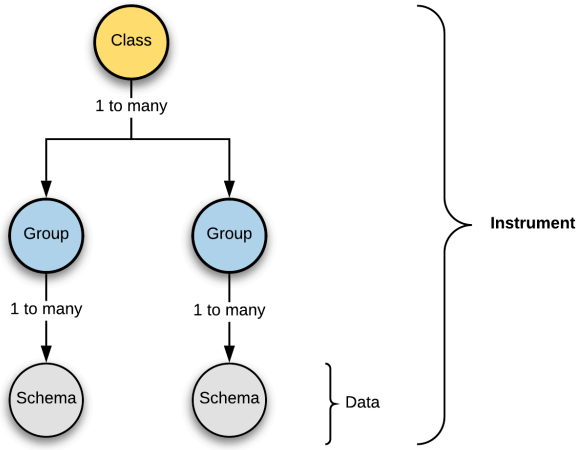
This section provides a deep dive into the inner workings of **Instruments**, exploring their categories, roles, and how they enable a dynamic and robust configuration management system. By understanding **Instruments**, you'll gain insights into how system-wide settings, and user-specific preferences by the same token, are defined, managed, and persisted.

## What is an Instrument?

An **Instrument** in **MaXXsettings** is composed of three hierarchical elements, each occupying a fixed position in the structure:

1. **Class**: The highest level, used to categorize and organize related **Groups**.
2. **Group**: A logical collection that groups related **Schemas** under a common concept.
3. **Schema**: The lowest level, where the actual configuration data and attributes are defined.

This structured approach allows for clear classification and an intuitive organization of information, making it accessible and easy to manage.




*The diagram illustrates the hierarchical structure that makes up MaXXsettings Instrumentation.*

As we already discussed, **Schema files** are highly versatile and operate within a defined context/scope determined by the **Class/Group** combination. Think of it as a modular building block for organizing and managing configuration settings in a systematic and scalable way.

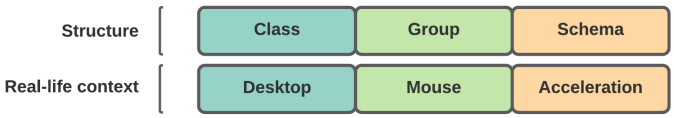
The **Schema** acts as an abstraction layer—a collection of values grouped and described by a conceptual framework called a **Stereotype**. This abstraction provides:

- **Flexibility**: Accommodating diverse use cases with minimal overhead.
- **Structure**: Ensuring all settings are organized, predictable, and easy to maintain.
- **Scalability**: Supporting a wide range of configurations, from basic user preferences to complex system-wide settings.

# The Importance of Complete Structure

An **Instrument** only becomes meaningful when all three elements—**Class**, **Group**, and **Schema**—are present and combined. Individually:

- **Classes** serve as containers for **Groups**
- **Groups** hold collections of **Schemas**




*Instrument Structure vs. real-life example.*

But without the full structure, the elements lack cohesion and context. A complete **Instrument** integrates these elements, ensuring the data is organized in a way that is both systematic and human-readable.

In essence, **Instruments** provide a clear and consistent framework for managing configuration data, enabling MaXXsettings to organize information efficiently while maintaining the integrity and usability of hierarchical datasets.

# Why Instruments Matter

**Instruments** form the backbone of **MaXXsettings**, bridging the gap between system-level configurations and runtime user preferences. By combining a **Schema**, a **Stereotype**, and the **Class/Group** hierarchy, Instruments provide:

1. **System-wide Configuration**: Standardized settings called **System-wide Instruments** are stored in `$MAXX_SETTINGS/Instruments`, ensuring global consistency.
2. **User-Specific Customization**: Runtime **User preferences Instruments** are stored in `$HOME/.maxxdesktop/msettings/Preferences`, offering personalized overrides.

This dual-purpose design makes **MaXXsettings** a powerful tool for both end-users and administrators, balancing simplicity with advanced configurability.

# System-wide Instruments

As previously explained, the root directory for **MaXXsettings** is defined by the environment variable `$MAXX_SETTINGS`. Within this directory, all **System-wide Instruments** are stored in the following location: `$MAXX_SETTINGS/Instruments`. These **system-wide Instruments** are crucial for maintaining consistent and global configurations across the system.

## Access and Permissions

To ensure stability and prevent accidental modifications:

- **System-wide Instruments** are **read-only** for standard users.
- Modifications can only be made using the **Administrative Command Line Interface (CLI)** with **superuser privileges**.

This safeguard ensures that Instruments remain secure and consistent, while still allowing administrators the flexibility to manage system configurations when necessary. For detailed instructions on managing Instruments, refer to the CLI documentation.

# System-wide Instrument Deconstruction

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 *(this information is managed by MaXXsettings)* |
| **Physical File Path** | $MAXX_SETTINGS/Instruments/3b/2a/d4/d6/Acceleration.Gauge |

## System-wide Instrument Detail

**$Root:** $MAXX_SETTING/**Instruments**
**$Filename:** $Root/$Classification/***<Schema>.<Stereotype>***

## Example

**$Filename:** /opt/MaXX/share/msettings/**Instruments**/14/ab/58/**Acceleration.Gauge**

## A Practical Example: Desktop.Mouse.Acceleration

Let's apply what we've learned so far with the **Desktop.Mouse.Acceleration** Instrument.

- **Definition**: This Instrument is defined as a **Gauge Stereotype**, and its **Schema** specifies attributes such as `minimum`, `maximum`, `scale`, and `default`.
- **Runtime Counterpart**: Its User Preference, of the same name, allows runtime overrides, storing only the chosen value and the Schema attributes relevant to the user. All User Preferences are stored in the user's `$HOME/.maxxdesktop/msettings` directory structure.

The **Gauge Stereotype** ensures compliance by enforcing behavior and optional validation rules. This separation of responsibility ensures clarity:

- **Schema**: Stores the data contract (definition).
- **User Preference**: Stores runtime values and overrides.

## Schema and User Preferences: An OOP Perspective

If you're familiar with Object-Oriented Programming, think of a **Schema** as a **Class** definition and a **User Preference** as an **instance** of that class containing live data. Just as a class defines the blueprint for its objects, a Schema encapsulates data and rules, while the **Stereotype** ensures compliance and validation.

## Editing Instruments: Best Practices

To safeguard system integrity, **Schema files** are only editable with superuser privileges. We recommend using the provided **Command Line Interface (CLI)** tool for any modifications, additions, or deletions. This ensures changes are consistent and properly validated.

Refer to the CLI Section for detailed instructions on managing Instruments.

# User Preference Instruments

While **system-wide Instruments** are read-only for standard users and focus on defining validation rules and default values, **User Preference Instruments**, or **User Preferences** for short, offer a way to create custom settings for individual users or multiple users on the same system. The approach is simple yet powerful: instead of modifying system-wide Instruments, **User Preferences** extend them by reusing the same **Class.Group.Schema** classification strategy to store only the user-defined values.

**User Preferences**, often referred to as **user-land Instruments**, are stored in a user-specific directory and are fully editable by normal users. By default, the location is `$HOME/.maxxdesktop/msettings/Preferences`. This directory mirrors the storage convention of system-wide Instruments, maintaining the same structure and classification logic.

## Shared Classification and Storage Logic

User Preferences and system-wide Instruments share:

1. **Classification Structure**: Both follow the **Class.Group.Schema** format for consistency.
2. **Hashed Storage Structure**: Both use the same hashed directory structure, ensuring calculated hashcodes are identical between the two. This enables seamless integration of system-wide defaults with user-defined overrides.

## Why This Matters

By leveraging the same classification and storage logic, **User Preferences** provide a flexible and user-friendly way to customize settings without interfering with global configurations. This design ensures that:

- **System-wide Instruments** remain stable and secure.
- **User Preferences** offer personalization while adhering to the same efficient and scalable architecture.

## User Preference Deconstruction

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 *(this information is managed by MaXXsettings)* |
| **Physical File Path** | $HOME/.maxxdesktop/msettings/Preferences/3b/2a/d4/d6/Acceleration.Gauge |

### User Preference Instrument Detail

**$URoot:** $HOME/.maxxdesktop/msettings/**Preferences**/
**$Filename:** $URoot/$Classification/***Schema.Stereotype***

## Example

**$Filename:** $HOME/.maxxdesktop/msettings/**Preferences**/14/ab/58/**Acceleration.Gauge**

# Exploring User Experience Instruments: Scopes and Flexibility

In this section, we take a deeper dive into the **User Experience Instruments** category, examining their different scopes and how they can be effectively utilized. We'll begin by exploring three (3) real-world examples to clarify the distinctions between various scopes and their levels of flexibility. Following this, we will delve into the **Desktop User Experience Instruments** to understand their unique role in shaping the user environment.

## Example 1: Managing Desktop.Mouse.NaturalScrolling

This example demonstrates how **MaXXsettings** handles the simple Instrument **Desktop.Mouse.NaturalScrolling**.

### Instrument Overview

- **Instrument Name**: `Desktop.Mouse.NaturalScrolling`
- **Stereotype**: Logical
- **Purpose**: Defines whether the mouse's scrolling direction follows a "natural" scrolling behavior (e.g., similar to touchscreen gestures).
- **Value Type**: Boolean (`true` or `false`)

<table>
<tr><th colspan="4">Instrument Name: Desktop.Mouse.NaturalScrolling</th></tr>
<tr><td>Type:</td><td>system-wide</td><td>Type:</td><td>User Preference</td></tr>
<tr><td>$Root:<br>$Location:<br>$Filename:</td><td>$MAXX_SETTING/Instruments<br>(calculated value)<br>$Root/$Location/NaturalScrolling.Logical</td><td>$URoot:<br>$ULocation:<br>$Filename:</td><td>$HOME/.maxxdesktop/msettings/Preferences<br>(calculated value)<br>$URoot/$ULocation/NaturalScrolling.Logical</td></tr>
<tr><td colspan="2">File content:<br><pre>version=1.0<br>uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c<br>stereotype=Logical<br>name=Desktop.Mouse.NaturalScrolling<br>default=false</pre></td><td colspan="2">File content:<br><pre>version=1.0<br>uuid=76c69f36-e0c1-4ec3-8d8b-8f0e1fb35c3c<br>stereotype=Logical<br>name=Desktop.Mouse.NaturalScrolling<br>value=true</pre></td></tr>
</table>

### Example

Let's experiment quickly with this Instrument. For more command line insight, refer to the CLI section.

**Command-line examples:**

```
$ msettings GET Desktop.Mouse.NaturalScrolling

false
```

# Example 2: Managing Desktop.FileManager.IconSortBy

This second example explores the **Instrument** `Desktop.FileManager.IconSortBy`, which defines the sorting algorithm for icons in **fm**, the **MaXXdesktop File Manager**.

## Instrument Overview

- **Instrument Name**: `Desktop.FileManager.IconSortBy`
- **Stereotype**: Choice Simple
- **Purpose**: Determines the sorting method for icons in the file manager.
- **Option Type**: Simple Chars Stereotype

## Key Characteristics of Simple Choice Instruments

1. **Choice Stereotype**:
   This Instrument uses the **Choice Stereotype** to manage a list of predefined sorting options.
2. **Option Type**:
   The options for **Simple Choice Instruments** always use the **Chars type**, representing basic textual values.
3. **Storage**:
   All options are stored within the same Schema file, keeping it simple and self-contained.

**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
```

## Example

Possible options for `Desktop.FileManager.IconSortBy` might include:

- **Name**: Sort icons alphabetically by file or folder name.
- **Type**: Sort icons by their file type.
- **Date**: Sort icons by creation or modification date.

**Command-line examples:**

```
$ msettings GET Desktop.FileManager.IconSortBy

Size
```

---

# Example 3: Managing Desktop.DtUtilities.WinEditor

This third example focuses on the **Instrument** `Desktop.DtUtilities.WinEditor`, which defines a list of **Default Graphical Text Editor Application** used throughout **MaXXdesktop**.

## Instrument Overview

- **Instrument Name**: `Desktop.DtUtilities.WinEditor`
- **Stereotype**: Choice Complex
- **Purpose**: Manages the list of default graphical text editor applications.
- **Option Type**: Application Stereotyped as Command

## Key Characteristics of Complex Command Choice Instruments

1. **Complex Choice Mechanism**:
   Unlike simple Choices, Complex Choice Instruments rely on additional Instruments and Schemas to function. These options are part of a **Choice Stereotype** but are enriched by:
   - **External Catalogs**: Used to organize and manage the options dynamically.
   - A **Resolver Mechanism**: Dynamically resolves and manages these options based on runtime requirements.
2. **Dynamic Options Management**:
   The combination of **Catalogs** and **Resolvers** enables **Complex Choice Instruments** to:
   - Separate system-wide options from the Schema file.
   - Expand dynamically with user-defined options or modifications.

| **Instrument Name:** Desktop.DtUtilities.WinEditor2 | | | |
|---|---|---|---|
| **Type:** | **Instrument** | **Type:** | **User Preference** |
| **$Root:**<br>**$Location:**<br>**$Filename:** | $MAXX_SETTING/**Instruments**<br>*(calculated value)*<br>$Root/$Location/**WinEditor2.Choice** | **$URoot:**<br>**$ULocation:**<br>**$UFilename:** | $HOME/.maxxdesktop/msettings/**Preferences**<br>*(calculated value)*<br>$URoot/$ULocation/**WinEditor2.Choice** |
| **File content:**<br>`version=1.0`<br>`uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6`<br>**`stereotype=Choice`**<br>**`type=WinEditor.Catalog`**<br>**`name=Desktop.DtUtilities.WinEditor`**<br>`default=0` | | **File content:**<br>`version=1.0`<br>`uuid=f353b007-0c3b-472f-8c6d-5e4a7e985ee6`<br>**`stereotype=Choice`**<br>**`type=WinEditor.Catalog`**<br>**`name=Desktop.DtUtilities.WinEditor`**<br>`value=1` | |

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

## A Real-World Example

For `Desktop.DtUtilities.WinEditor`, the Instrument works as follows:

- **System-Wide Configuration**: Stores predefined default graphical text editor applications, such as:
  - `gedit`
  - `vim`
  - `nano`
- **Catalog Extension**: Allows users to add personal preferences or custom applications to the list without modifying the system-wide Schema file.

**Command-line examples:**

```
$ msettings GET Application.WinEditor.XNEdit

/opt/MaXX/bin/xnedit
```

## Dynamic Resolution in Action

The **Resolver Mechanism** plays a key role by linking the options defined in the Catalog to their respective User Preferences or system-wide defaults. When the system or user invokes this Instrument:

- The Resolver checks if a **User Preference** exists for the option (e.g., a specific application selected by the user).
- If no User Preference is found, the system defaults are used.

## Key Takeaway

The **Desktop.DtUtilities.WinEditor** Instrument exemplifies the power and flexibility of **Complex Command Choice Instruments**. By leveraging Catalogs and Resolvers, it enables dynamic and scalable management of application options while maintaining a clear separation between system-wide configurations and user-defined preferences. This approach ensures adaptability without sacrificing consistency or control.

# Catalog: Extending Choice Instrument Options

**Catalog** is an advanced mechanism for managing alternative storage and lookup for **Choice Instrument**. When the number of options becomes too large or a use case requires dynamic behavior, **Catalogs** provide a way to separate these options from the schema file into a dedicated reference file.

## Key Benefits of Catalog

- **Simplified Schema Files**: Offloading numerous options to a Catalog reduces clutter and improves manageability.
- **User Extensions**: Catalogs can include user-defined options, enabling a more personalized configuration experience.
- **Dynamic Resolver Behavior**: Supports flexible options without modifying the original schema.

## Example

The **Desktop.DtUtilities.WinEditor** Instrument demonstrates Catalog usage, combining system-wide options with user-defined extensions for a tailored experience.

### Resolver Integration with Catalogs

Catalogs enhance the Resolver feature by allowing dynamic resolution of stored option values. For complex option types (excluding Chars), the Resolver looks up the enumerated Instrument and resolves it to either:

- A **User Preference value** if available.
- The **system-wide default** otherwise.

# Resolver: Dynamic Value Resolution

The **Resolver** feature in **MaXXsettings** introduces a powerful way to dynamically determine attribute values at runtime. Unlike static definitions, a **Resolver** infers an attribute's value through a recursive lookup mechanism, resolving the value from the most specific (leaf) Instrument available.

## Key Features of Resolvers

- **Dynamic Resolution**: Fetches attribute values dynamically during runtime.
- **Automatic Redirection**: Supports recursive lookups across Instruments.
- **Flexibility with Constraints**: While theoretically unlimited, redirections beyond two levels are often inefficient and indicate suboptimal strategies.

## Usage

To enable resolution, prefix an attribute value with an **"@"** followed by the Instrument name. Currently, only **Choice** and **FileType Instruments**, and **Catalog Schemas** support Resolvers.

## Popular Use Cases

1. **Default Value Inference**: Dynamically adapting values based on system configurations.

2. **Complex Preferences**: Allowing hierarchical settings to cascade through multiple Instruments.

## Resolver Workflow with a Simple Choice

In this example of an User querying the value of the User Preference *Desktop.DtUtilities.WinEditor* via CLI a GET Command. The User Preference is set up as a Simple Choice containing resolvable entries.

```
$ msettings GET -n Desktop.DtUtilities.WinEditor2
 /usr/bin/xnedit -s %f
```

**Here's the Simple ChoiceDetailed Workflow**

| 1. Load Instrument ⇩ | 2. Resolver Fetches option[0] Instrument ⇩ | 3. Resolver Get Value ⇩ |
|---|---|---|
| stereotype=**Choice**<br>name=**Desktop.DtUtilities.WinEditor**<br>type=**Command**<br>**option[0]=@Desktop.WinEditor.XNEdit**<br>option[1]=@Desktop.WinEditor.Gedit<br>default=0<br>**value=0** | stereotype=**Command**<br>name=**Desktop.WinEditor.XNEdit**<br>default=/usr/bin/nedit<br>execName=xnedit<br>execPath=/usr/bin<br>execParams=-s %f<br>envBinaryPath=/opt/MaXX/bin64<br>envLibraryPath=/opt/MaXX/lib64 | value=/usr/bin/xnedit -s %f |

## Resolver Workflow with a Complex Choice and Catalog

In this second example, the request is exactly the same but the underlying MaXXsettings Instrument is different. The User Preference in this case is set up as a Complex Choice that is extended via a Catalog that contains resolvable entries.

```
$ msettings GET -n Desktop.DtUtilities.WinEditor2
 /usr/bin/xnedit -s %f
```

**Here's the Complex Choice Detailed Workflow**

| 1. Load Instrument ⇩ | 2. Loads Catalog and Resolver Fetches Instrument at option[0] ⇩ | 3. Resolver Loads Instrument ⇩ |
|---|---|---|
| stereotype=**Choice**<br>name=**Desktop.DtUtilities.WinEditor**<br>type=**WinEditor.Catalog**<br>default=0<br>**value=0** | name=**WinEditor**<br>type=**Command**<br>**option[0]=@Desktop.WinEditor.XNEdit**<br>option[1]=@Desktop.WinEditor.Gedit<br>... | stereotype=**Command**<br>name=**Desktop.WinEditor.XNEdit**<br>default=/usr/bin/nedit<br>execName=xnedit<br>execPath=/usr/bin<br>execParams=-s %f<br>envBinaryPath=/opt/MaXX/bin64<br>envLibraryPath=/opt/MaXX/lib64 |

| 4. Resolver Gets Value ⇩ | |
|---|---|
| value=/usr/bin/xnedit -s %f | |

# Command Line Interface - CLI

MaXXsettings 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"*. MaXXsettings 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`
```

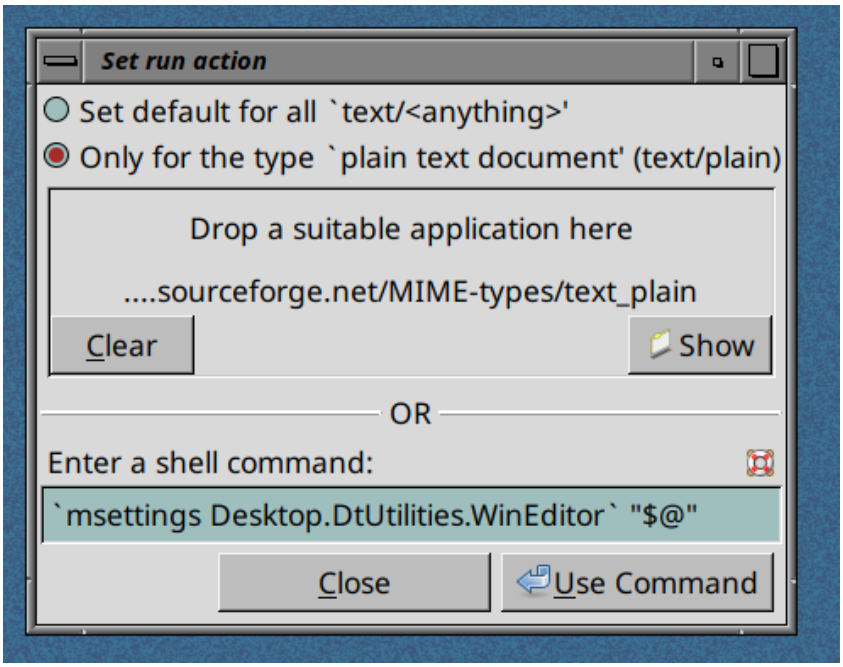
**From command-line - fetching from MaXXsettings the default Image Editor**

```
$ 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 MaXXsettings Command Line Interface or CLI with examples, commands and parameters documentation making your usage easier.

## CLI Search Mechanism

MaXXsettings 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*. MaXXsettings 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 MaXXsettings. No manually hacking.

## CLI Interaction Modes

For your convenience, MaXXsettings 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 MaXXsettings 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
```

# Common CLI Commands

This section describes the CLI commands that are common to both Administrative and Standard Modes.

## LIST Command

List command allows a user or system administrator to list the currently installed Instruments on a given system running MaXXsettings. The output is either a list of alphabetically sorted Instrument names with or without their default values.

### Parameters

| | | |
|---|---|---|
| **-K,** | **--key-value** | Output in key=value format for every requested Instrument. |

**LIST CLI Command-line example:**

```
$ msettings LIST [params]
$ ms LIST [params]

Examples:

# ms LIST

Total Installed Instruments : 6
Desktop.Background.DarkBackground
Desktop.Colors.SgiScheme
Desktop.Colors.UserInterfaceAccent
Desktop.DtSounds.DesktopSounds
Desktop.DtUtilities.EmailClient
Desktop.DtUtilities.FileBrowser

# msettings LIST -K

Total Installed Instruments : 6
Desktop.Background.DarkBackground=true
Desktop.Colors.SgiScheme=Sargent
Desktop.Colors.UserInterfaceAccent=blue
Desktop.DtSounds.DesktopSounds=true
Desktop.DtUtilities.EmailClient=thunderbird
Desktop.DtUtilities.FileBrowser=fm

#
```

## HASH Command

List command allows a user or system administrator to generate a Hashed Storage Location for the given Instrument name. This Hashed Storage Location is used to store both Instruments and User Preferences.

**HASH CLI Command-line example:**

```
$ msettings HASH [params]
$ ms HASH [params]

Examples:

# ms HASH Desktop.Mouse.Acceleration

Desktop.Mouse.Acceleration = /3b/2a/d4/d6

# msettings HASH Desktop.Mouse.Acceleration

Desktop.Mouse.Acceleration = /3b/2a/d4/d6

#
```

# Administrative CLI Commands

This section will focus on the administrative aspect of MaXXsettings CLI and the CLI commands that are only available in Admin mode. All commands require superuser permissions.

## INIT Command

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

**This command is only available in Admin mode and requires superuser permissions level in order to initialize MaXXsettings system-wide data structure.**

### Parameters

| | | |
|---|---|---|
| **-F,** | **--force** | Force the initialization over an existing MaXXsettings 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

MaXXsettings - system-wide Directory structure created.
MaXXsettings - system-wide Database was successfully initialized.
MaXXsettings - system-wide Indexes built and synched.
MaXXsettings - 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 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.**

## 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 UPDATE -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 MaXXsettings local database and directory structure are not present at the first invocation, MaXXsettings will install itself properly beforehand, then run the requested command.*

## SET Command

Set one or many User Preferences 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 Preferences 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

## Classification

One of the core design principles of **MaXXsettings** is to retrieve information as quickly as possible without adding unnecessary complexity. To meet this critical performance requirement, MaXXsettings employs a clever and efficient mechanism for classifying, partitioning, and retrieving data with remarkable speed. Less is more...

At its core, **Instruments** follow a logical **Class.Group.Schema** structure, which could theoretically map directly to physical directories and files on the file system. However, this traditional approach would introduce performance bottlenecks as the number of stored Instruments grows.

Instead, **MaXXsettings** leverages an **ultra-fast computable hashcode** based on the Instrument's name. This hashcode is then used to map into a **hashed directory structure**, which provides several advantages:

1. **Lightning-Fast Lookups**: Retrieval times remain consistent, regardless of the number of stored elements.
2. **Enhanced Stability**: The hashed structure minimizes the risk of manual errors, such as misplaced or corrupted files.
3. **Optimized Scalability**: The system handles large-scale configurations effortlessly without compromising speed or simplicity.

### Why This Matters

This hashing approach ensures **MaXXsettings** stays true to its mission: delivering high-speed access to configurations while keeping the process intuitive and user-friendly. By avoiding reliance on traditional file system hierarchies, MaXXsettings achieves a balance between **performance**, **reliability**, and **ease of management**.

**This is the way**—fast, efficient, and built to last.

*With this foundation, let's explore real-world examples and practical applications of Instruments in the following sections.*
Each Instrument under MaXXsettings can be looked up by its Instrument name, its unique ID (UUID) or full filename path. In order to provide fast and consistent performance, MaXXsettings 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
$
```