

MaXX Settings Configuration Management

Instrumentation Guide

Version 0.24

Versions

Version	Date	Author(s)	Description
0.10	2020-12-30	Eric Masson	Initial and ongoing work
0.20	2021-01-06	Eric Masson	Complete documentation separation from Technical Specification.
0.21	2021-01-13	Eric Masson	Start adding Implementation details for Instruments, added SgiScheme Choice detail.
0.22	2021-04-10	Eric Masson	Sync with Technical specs v. 0.98
0.23	2021-04-15	Eric Masson	Adding Groups: KeyboardSettings and KeyboardShortcuts. Cleanup the format.
0.24	2021-05-07	Eric Masson	Adding Enum, Catalog, Resolver and implementation details. Syncing changes with MaXX Settings Architectures and Specifications RC1

Table of Content

Table of Content	3
Synopsys	5
Naming Conventions	5
MaXX Settings Convention	5
Instruments	6
Classification	6
Instrument Categories	6
System Wide Instruments	7
System Wide Nomenclature	7
System Wide Instruments Location	7
Example	7
User Preference	8
User Preference Nomenclature	8
User Preference Instruments Location	8
Example	8
Desktop Instrumentation	9
Enums	9
Desktop Enums	9
Enums Location	9
Example - FontStyle.Enum	9
Resolver	10
Resolver Workflow with a Simple Choice	10
Resolver Workflow with a Complex Choice and Catalog	10
Catalogs	11
Desktop Catalogs	11
Desktop Catalogs Location	12
Example - WinEditor.Catalog	12
Application Instruments	13
Application	13
FileType User Experience Instruments	14
Instruments Location	14
FileTypes	14
Implementation Details	16
MaXX Settings Integration	16
Desktop Class Instruments	16
Mouse Settings	17
Keyboard Settings	18
Keyboard Shortcuts	20
Default Application Settings	21
Window Settings	22
Desktop Settings	23
FileManager Settings	24
IconCatalog Settings	25
Background Settings	26
Colors Settings	27

Text Settings	28
Font Rendering Settings	29
Sounds Settings	30
Localisation Settings	31
External References	32
Code Snippets	33

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 dive into MaXX Settings implementation details and the MaXX Interactive Desktop Instrumentation.

The reader is expected to have read the [MaXX Settings Architecture & Technical Specification](#) Documentation prior reading this document.

Naming Conventions

Lowercase is a naming convention in which a name formed of a single word is written all letters in lowercase.

Example: name, version, uuid, etc.

Uppercase is a naming convention in which a name formed of a single word is written all letters in uppercase.

Example: HOME, SHELL, PATH, etc.

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

Example: Chars, Dimension, Geometry, etc.

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

Example: maximumSize, backgroundColor, darkColor, etc.

MaXX Settings Convention

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

Instruments

In this section, we go over the nuts and bolts regarding Instruments, their uses as System wide and User Preference settings, and the way they are persisted by MaXX Settings. An Instrument is in simple terms a Schema file used in a context (or use-case) defined by Class/Group Combo classification. Schema is the abstraction of a set of values grouped together described by a particular concept of information, a Stereotype.

Refer to the MaXX Settings Architecture & Technical Specifications for more details.

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

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		FileType	
Applications		Application	WinEditor ImageEditor ImageViewer ...

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

System Wide Instruments Location

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

Example

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

User Preference

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

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

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

User Preference Nomenclature

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

User Preference Instruments Location

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

Example

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

Desktop Instrumentation

This section is focusing on all Desktop Instrumentation related topics ranging from Enumerations used in Instrument validation, to dynamic resolver, to Catalogs for expandable System wide and User defined options for Choice Instruments to Applications and finally FileTypes matching.

Enums

Enums are used to associate a set of predefined values to a notion or concept. For example, we can list or enumerate all possible values for the notion `FontStyle` as: *Normal* and *Bold*. This allows both users and admins to agree on some form of data-contacts on expected values. Enums bring stronger type constraints to MaXX Settings in a simple way, and for robustness can only be defined in System wide space. Basically Enums are value sets used for validation.

Desktop Enums

Schema	Option Type	Options	Description / Comment
<code>ColorSpace.Enum</code>	Chars	RGB255, RGB100, YUV, HSL, CMYK,	Refer to Apple Developer Site
<code>FontStyle.Enum</code>	Chars	Normal, Bold	
<code>FontWeight.Enum</code>	Chars	Light, Medium, Demibold, Bold, Black	
<code>FontSlant.Enum</code>	Chars	Italic, Oblique, Roman	
<code>XftLcdFilter.Enum</code>	Chars	lcddefault, lcdlight, lcdnone, lcdlegacy	The lcddefault filter will work for most users. Other filters are available that can be used in special situations: lcdlight ; a lighter filter ideal for fonts that look too bold or fuzzy, lcdlegacy , the original Cairo filter; and lcdnone to disable it entirely.
<code>XftHintStyle.Enum</code>	Chars	hintnone, hintslight, hintmedium, hintfull	While hintfull will be a crisp font that aligns well to the pixel grid but will lose a greater amount of font shape. hintslight implicitly uses the autohinter in a vertical-only mode in favor of font-native information for non-CFF (.otf) fonts.
<code>XftRgba.Enum</code>	Chars	rgb, bgr, vrgb, vbrg	Monitors are either: RGB (most common), BGR , V-RGB (vertical), or V-BGR . A monitor test can be found here .

Enums Location

\$Root: \$MAXX_SETTING/Enums

\$Filename: \$Root/<Schema>.Enum

Example - FontStyle.Enum

\$Filename: /opt/MaXX/share/msettings/Enums/FontStyle.Enum

```
version=1.0
stereotype=Enum
name=FontStyle
type=Chars
option[0]=Normal
option[1]=Bold
```

Resolver

Resolver is a very powerful feature in MaXX Settings which allows the default and attributes to be dynamically “resolved” at runtime vs. the normal static behavior within the Instrument’s Schema file. A **Resolver** works by **inferring** an attribute’s runtime value to an automatic and dynamic recursive lookup mechanism that fetches the final value defined in the last leaf Instrument. There are no limits on the number of redirections, but after 2 your strategy is most likely not optimal.

To request MaXX Settings to dynamically resolve a value at runtime, use the **at-sign** “@” as prefix to an Instrument name in an attribute value field. Only Choice, and FileType Instruments and Catalog Schemas supports Resolver at the moment.

Let’s explore two popular use-cases with a detailed example for each.

Resolver Workflow with a Simple Choice

In this example of an User requisiting 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.WinEditor
/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 name= Desktop.DtUtilities.WinEditor type= Command option[0]=@Desktop.WinEditor.XNEdit option[1]=@Desktop.WinEditor.Gedit default=0 value=0	stereotype= Command name= Desktop.WinEditor.XNEdit default=/usr/bin/nedit execName=xnedit execPath=/usr/bin execParams=-s %f envBinaryPath=/opt/MaXX/bin64 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 MaXX Settings 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.WinEditor
/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 name= Desktop.DtUtilities.WinEditor type= WinEditor.Catalog default=0 value=0	name= WinEditor type= Command option[0]= @Desktop.WinEditor.XNedit option[1]= @Desktop.WinEditor.Gedit ...	stereotype= Command name= Desktop.WinEditor.XNedit default=/usr/bin/nedit execName=xnedit execPath=/usr/bin execParams=-s %f envBinaryPath=/opt/MaXX/bin64 envLibraryPath=/opt/MaXX/lib64
4. Resolver Gets Value ↕		
value=/usr/bin/xnedit -s %f		

Catalogs

Catalogs are predefined sets of value used to provide alternative storage and lookup mechanisms for **Choice Instrument** options. Catalog can be used to separate System wide Choice's options from its schema file into a reference file, a catalog. It makes sense to move options out of a schema file when they get too numerous or a specific use-case requires a more dynamic behavior with possible User extension. This way a Choice Instrument can be expanded without being modified for say. Catalogs are defined as System wide schemas and can be extended by the User with his/her personal options providing a more personalized experience. The *Desktop.DtUtilities.WinEditor* Instrument is a great example of System wide and User defined options.

Catalog adds an interesting feature that allows the **Resolver** to assist in the resolution of its stored option values by doing a lookup of the option's counterpart Instrument. The resolution mechanism only works for complex option types (not Chars) by resolving the enumerated Instrument into a User Preference value if present or the System wide default otherwise.

Desktop Catalogs

Schema	Option Type	Options	Description / Comment
Language.Catalog	Chars	...	List of supported Language
KeyboardInput.Catalog	Chars	...	List of supported KeyboardInput
DefaultSoundOutput.Catalog	Chars	...	List of supported Sound Output Devices ??
SGIScheme.Catalog	Chars	Arizona, Bayou,BlackAndWhite, DarkBliss, Gainsborough, Gotham, IndigoMagic, Inverness, Lascaux, Leonardo, Metropolis, Milan, Pacific, Potrero, RedGreenSafe, Rembrandt, RoseGarden, Sargent, VanGogh, Willis, Buckingham, GrayScale, KeyWest, Mendocino, Monet, Print, Rio, Titian, Turner, Vancouver, Whistler	List of supported SGI Schemes

MaXX Settings - Configuration Management Simplified

WinEditor.Catalog	Command	XNEdit, Gedit, Sublime, Gvim, Nano	
FileBrowser.Catalog	Command	Rox-filer, fm	
ImageViewer.Catalog	Command	Feh, Eog	
ImageEditor.Catalog	Command	gimp	
WebBrowser.Catalog	Command	firefox, chrome	
EmailClient.Catalog	Command	thunderbird, evolution	
MediaViewer.Catalog	Command	Vlc, ffplay	
VectorEditor.Catalog	Command	inkscape	
PDFViewer.Catalog	Command	Xpdf, evince	
BackgroundColors.Catalog	Color	...	List of predefined BackgroundColors
BackgroundImages.Catalog	Image	...	List of predefined Background Images

Desktop Catalogs Location

\$Root: \$MAXX_SETTING/**Catalogs**

\$Filename: \$Root/<**Schema**>.**Catalog**

Example - WinEditor.Catalog

\$Filename: /opt/MaXX/share/msettings/**Catalogs**/WinEditor.Catalog

```

stereotype=Catalog
name=WinEditor
type=Command
option[0]=@Desktop.WinEditor.XNEdit
option[1]=@Desktop.WinEditor.GEdit
option[2]=@Desktop.WinEditor.Sublime
option[3]=@Desktop.WinEditor.Gvim

```

Application Instruments

Application Instruments are using the same classification and lookup mechanism as Desktop Instruments and they are based on the Command Schema definition. Applications are strongly relying on Resolvers to do the resolution of their final value. This brings flexibility and modularity to Applications management.

Application

Class	Group	Schema	Name
Desktop	WinEditor	NEdit.Command	NEdit
		GEdit.Command	GEdit
		XNEdit.Command	XNEdit
	ImageEditor	Gimp.Command	Gimp
	ImageViewer	ImageViewer.Command	ImageViewer
		Feh.Command	Feh
		Eog.Command	Eog

WORK IN PROGRESS :)

FileType User Experience Instruments

MaXX Settings provides an extensible mechanism to associate file types and actions with corresponding applications in a truly limitless way. FileType Instruments offers a powerful file type to application matching engines based on MIME types, file extensions and content matching rules with programmable actions like view, edit, run, print, compile, etc.

FileTypes are using the same classification and lookup mechanism as Desktop User Experience Instruments, however their Class is FileType.

Classification : FileType

Group: (SUPERTYPE)

Schema : (Type)

WORK IN PROGRESS :)

Instruments Location

\$Root: \$MAXX_SETTING/**Instruments**

\$Classification: /<Class>/<Group>/<Schema>.<Stereotype> **which is a calculated Hashed Directory Structure**

\$Filename: \$Root/\$Classification/<Schema>.<Stereotype>

FileTypes

Class	Group	Schema	Description
FileType	SUPERTYPE	Type.Application	Command to execute to View, Edit or Run a specific Class/Group filetype.
	Executable	Generic.Application	
	Text	Plain.Application XML.Application JSON.Application	
	Audio	WAV.Application Edit.Application	
	Video	View.Application Edit.Application Run.Application	

SUPERTYPE The type-name is the TYPE name of any valid file type. Use SUPERTYPE to identify the file type as a “subset” of one or more other file types. This information can be accessed by other file types by calling isSuper(1) from within their CMD rules (OPEN, ALTOPEN, and so on). A file type can have multiple SUPERTYPES. (For example, the Script file type has both Ascii and SourceFile SUPERTYPES.)

Example: SUPERTYPE Executable

TYPE The type-name is a one-word ASCII string. You can use a legal C language variable as a type name. Choose a name that is in some way descriptive of the file type it represents. All rules that follow a **TYPE** declaration apply to that type, until the next **TYPE** declaration is encountered in the FTR file. Each Type declaration must have a unique Type name.

Example:

SUPERTYPE Executable	TYPE <i>Generic</i>
SUPERTYPE Text	TYPE Plain.Application

Implementation Details

This section focuses on the implementation details of MaXX Desktop Preference Panels, how they integrate with MaXX Settings and finally how they apply those settings.

MaXX Settings Integration

<need some description>

<how the integration is done>

<C++ class diagram>

<how settings are applied>

<C++ class diagram>

Desktop Class Instruments

<need some description>

Breakdown Summary

13 Groups

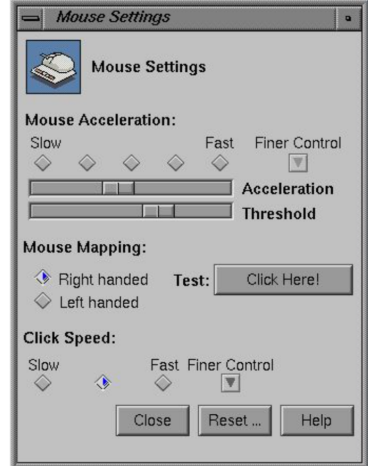
92 total User Experience Instruments

WIP

Mouse Settings

Classification: Desktop

Group: Mouse

Schema	Default	Preference Panel
Acceleration.Gauge	minimum=0.0 maximum=10.0 scale=1.0 default=2.0	
Threshold.Gauge	minimum=0.0 maximum=15.0 scale=1.0 default=4.0	
LeftHanded.Logical	false	
WheelMouseScroll.Logical	true	
NaturalScrolling.Logical	false	
DoubleClick.Gauge	minimum=0.0 maximum=10.0 scale=1.0 default=4.0	

Links:

https://wiki.archlinux.org/index.php/Mouse_acceleration
<https://askubuntu.com/questions/172972/configure-mouse-speed-not-pointer-acceleration>
<https://tronche.com/gui/x/xlib/input/keyboard-and-pointer-settings.html>
<https://tronche.com/gui/x/xlib/input/XChangePointerControl.html>
 /home/emasson/MaXX-Dev/cdesktopenv-code/cde/programs/dtstyle

XLib function calls:

- [XChangePointerControl](#)
- [XGetPointerControl](#)
- [XtGetMultiClickTime](#)
- [XtSetMultiClickTime](#)

Todo:

- Define Gauge values (table above) and feature+validate good default values. DoubleClick might be using decimals
- Write CLI commands (script)
- Integrate msettings CLI into **mouse** application
- Add command-line option **-apply** that just start, load, apply the settings and quit the **mouse** application (does not start the visual portion of the app)
- Add logic to apply settings with XLib calls above

```
$ xset q | grep -A 1 Pointer
acceleration: 2/1 threshold: 4
```

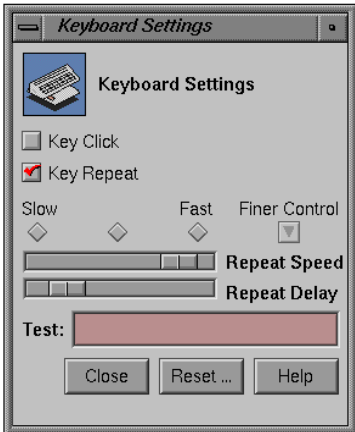
The **acceleration** is a *multiplier number* that defines how many times faster than the standard speed the cursor will move. Try numbers between 2 and 5, setting a high multiplier like 9 makes the mouse movements very jumpy. It does not need to be a whole number, you can use 1/2 to get half the standard speed or 5/2 (=2.5) if 2 is too slow and 3 is too fast.

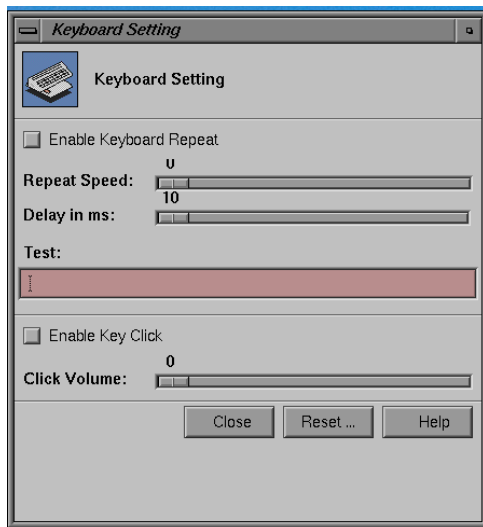
The **threshold** defines how many pixels the mouse must move in a short period of time before the acceleration setting is used. Using a threshold of 1, as in `xset m 5 1`, disables this and gives you the same mouse speed at all the time. Setting `xset m 5 10` requires the mouse to move 10 pixels before the pointer is accelerated.

Keyboard Settings

Classification: Desktop

Group: KeyboardSettings

Schema	Default	Preference Panel
KeyClick.Logical	true	
KeyRepeat.Logical	true	
RepeatSpeed.Gauge	-	
RepeatDelay.Gauge	-	
KeyClickVolume.Gauge	-	
Links: https://wiki.archlinux.org/index.php/Xorg/Keyboard_configuration https://tronche.com/gui/x/xlib/input/keyboard-and-pointer-settings.html		
XLib function calls: <ul style="list-style-type: none">• XSetInputFocus()• XGetInputFocus()• XChangeKeyboardControl()• XGetKeyboardControl()• XAutoRepeatOn• XAutoRepeatOff		
Todo: <ul style="list-style-type: none">• Define Gauge values (table above)• Write CLI commands (script)• Integrate msettings CLI into keyboard application• Add command-line option --apply that just start, load, apply the settings and quit the keyboard application (does not start the visual portion of the app)• Add logic to apply settings with XLib calls above		



Keyboard Shortcuts

Classification: Desktop

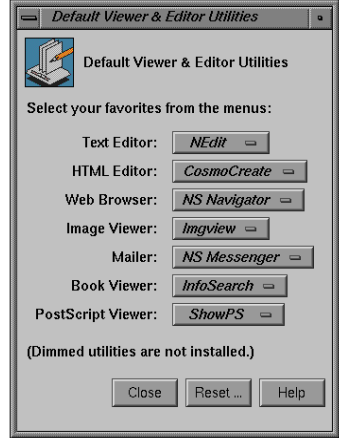
Group: KeyboardShortcuts

Schema	Default	Preference Panel
-	-	NEW IMAGE
-	-	
-	-	
-	-	
Links: https://wiki.archlinux.org/index.php/Xorg/Keyboard_configuration https://tronche.com/gui/x/xlib/input/keyboard-and-pointer-settings.html		
XLib function calls: <ul style="list-style-type: none">• XSetInputFocus()• XGetInputFocus()• XChangeKeyboardControl()• XGetKeyboardControl()• XAutoRepeatOn• XAutoRepeatOff		
Todo: <ul style="list-style-type: none">• Define Gauge values (table above)• Write CLI commands (script)• Integrate msettings CLI into keyboard application• Add command-line option --apply that just start, load, apply the settings and quit the keyboard application (does not start the visual portion of the app)• Add logic to apply settings with XLib calls above		

Default Application Settings

Classification: Desktop

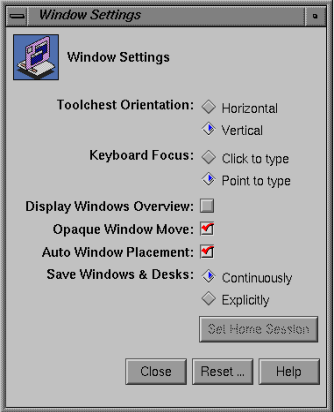
Group: DtUtilities

Schema	Default	Preference Panel
FileBrowser.Choice	-	
WinEditor.Choice	-	
TextEditor.Choice	-	
WebBrowser.Choice	-	
EmailClient.Choice	-	
ImageEditor.Choice	-	
ImageViewer.Choice	-	
MediaViewer.Choice	-	
VectorEditor.Choice	-	
PDFViewer.Choice	-	
Links:		
Todo:		
<ul style="list-style-type: none"> Define the Choices (table above) Write CLI commands (script) Integrate msettings CLI into dtutilities application Add command-line option --apply that just start, load, apply the settings and quit the dtutilities application (does not start the visual portion of the app) Edit \$HOME/.maxxdesktop/desktopenv script to load settings via CLI command (to set export values) Add logic to apply settings then run update-desktop afterwards 		

Window Settings

Classification: Desktop

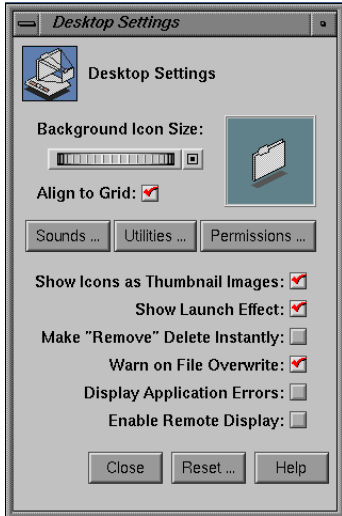
Group: Window

Instrument	Default	Preference Panel
ToolchestHorizontal.Logical	false	
KeyboardFocus.Logical	true	
DisplayOverview.Logical	false	
MoveOpaqueWindow.Logical	true	
OutlineThickness.Gauge	2 1-4:2	
WindowManagerAccent.Color	red	
AutoWindowPlacement.Logical	true	
SaveWindowsDesks.Logical	true	
Modern.WindowTitle.Typeface	-	
Modern.IconTitle.Typeface	-	
Classic.WindowTitle.Typeface	-	
Classic.IconTitle.Typeface	-	
MoveWithoutRaising.Logical	true	
Links:		
Todo: <ul style="list-style-type: none"> Define the Choices (table above) Write CLI commands (script) Integrate msettings CLI into window application Add command-line option --apply that just start, load, apply the settings and quit the window application (does not start the visual portion of the app) Edit \$HOME/.maxxdesktop/desktopenv script to load settings via CLI command (to set export values). Maybe extract some of the Xresources attributes (5Dwm and others): Add logic to apply settings then run update-desktop and tellwm afterwards 		

Desktop Settings

Classification: Desktop

Group: Settings

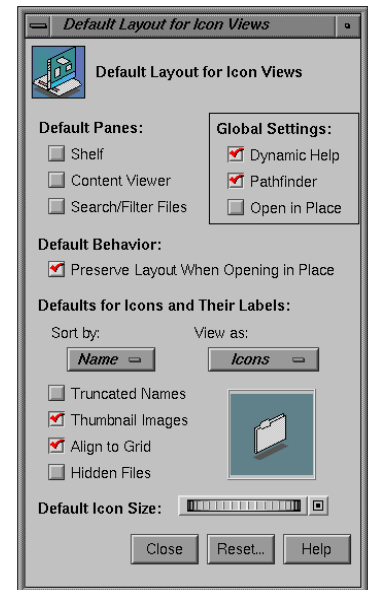
Schema	Default	Preference Panel
DesktopIconSize.Gauge	-	
DesktopIconAlignGrid.Logical	true	
DesktopAccemt.Color	yellow	
ToolchestSoundEffect.Logical	true	
ModernLookAndFeel.Logical	false	
IconAsThumbnaillImage.Logical	true	
ShowLaunchEffect.Logical	true	
MakeDeletelntantly.Logical	false	
WarnOnFileOverwrite.Logical	true	
DisplayApplicationErrors.Logical	false	
EnableRemoteDisplay.Logical	false	
Links:		
Todo: <ul style="list-style-type: none">Define the Choices (table above)Write CLI commands (script)Integrate msettings CLI into workspace applicationAdd command-line option --apply that just start, load, apply the settings and quit the workspace application (does not start the visual portion of the app)Add logic to apply settings then run update-desktop and tellworkspace afterwards		

FileManager Settings

Classification: Desktop

Group: FileManager

DisplayShelf.Logical	false
DisplayContent.Logical	false
DisplaySearchFilters.Logical	false
KeepLayoutOpenInPlace.Logical	true
IconSortBy.Choice	Name, Type, Size, CreationDate, ModifiedDate
IconViewAs.Choice	Icon, List, Detail
TruncateNames.Logical	false
ThumbnailImages.Logical	true
AlignToGrid.Logical	true
DisplayHiddenFiles.Logical	false
DefaultIconSize.Gauge	-
IconText.Typeface	-
DynamicHelp.Logical	true
PathFinder.Logical	true
OpenInPlace.Logical	false



Links:

Todo:

- Define the Choices (table above)
- Write CLI commands (script)
- Integrate msettings CLI into **fm** application
- Add logic to apply settings then run **update-desktop** and restart **fm** afterwards.

IconCatalog Settings

Classification: Desktop


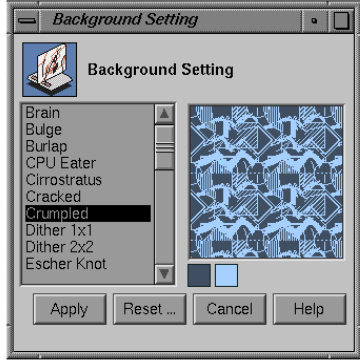
Group: IconCatalog

KeepLayoutOpenInPlace.Logical	false	NEW IMAGE
IconSortBy.Choice	-	
IconViewAs.Choice	-	
TruncateNames.Logical	false	
ThumbnailImages.Logical	true	
AlignToGrid.Logical	true	
DefaultIconSize.Gauge	-	
IconText.Typeface	-	
Links:		
Todo: <ul style="list-style-type: none">● Define the Choices (table above)● Write CLI commands (script)● Integrate msettings CLI into iconcatalog application● Add logic to apply settings then run update-desktop and restart iconcatalog afterwards		

Background Settings

Classification: Desktop

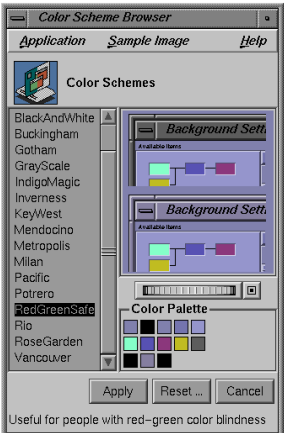
Group: Background

Schema	Default	Preference Panel
BackgroundColors.Choice		
BackgroundImages.Choice	-	
DarkBackground.Logical	true	
Pattern1.Color	-	
Pattern2.Color	-	
Pattern3.Color	-	
Links:		
Todo: <ul style="list-style-type: none">• Define the Choices (table above)• Write CLI commands (script)• Integrate msettings CLI into background application• Add command-line option -apply that just start, load, apply the settings and quit the background application (does not start the visual portion of the app)• Add logic to apply settings then run update-desktop afterwards		

Colors Settings

Classification: Desktop

Group: Scheme

Schema	Default	Preference Panel
SGIScheme.Choice	IndigoMagic	
SgiDarkScheme.Logical	false	
UserInterfaceAccent.Color	blue	
Links:		
Todo: <ul style="list-style-type: none">Define the Color (table above)Write CLI commands (script)Integrate msettings CLI into scheme applicationAdd command-line option -apply that just start, load, apply the settings and quit the scheme application (does not start the visual portion of the app)Maybe check current \$HOME/.maxxdesktop/desktopenv and Xresource scriptAdd logic to apply settings then run update-desktop and tellwm afterwards		


Text Settings

Schema	Default	Preference Panel
SmallText.Typeface	-	NEW IMAGE
NormalText.Typeface	-	
LargeText.Typeface	-	
Terminal.Typeface	-	
WindowTitle.Typeface	-	
WindowIconTitle.Typeface	-	
SmoothText.Logical	True	
Links:		
<p>Todo:</p> <ul style="list-style-type: none"> • Define the Typeface (table above) • Write CLI commands (script) • Integrate msettings CLI into ??? application • Add command-line option -apply that just start, load, apply the settings and quit the ??? application (does not start the visual portion of the app) • Add logic to apply settings then run update-desktop afterwards 		


Font Rendering Settings

Schema	Default	Preference Panel
XftAutoHint.Logical	0	NEW IMAGE
XftLcdFilter.Choice	lcddefault	
XftHintStyle.Choice	hintslight	
XftHinting.Logical	1	
XftAntialias.Logical	1	
XftRgba.Choice	rgb	
Links:		
<p>Todo:</p> <ul style="list-style-type: none">• Define the Choices (table above)• Write CLI commands (script)• Integrate msettings CLI into ??? application• Add command-line option -apply that just start, load, apply the settings and quit the ??? application (does not start the visual portion of the app)• Add logic to apply settings then run update-desktop and tellwm afterwards		

Sounds Settings

Schema	Default	Preference Panel
MuteSystem.Logical	false	
StartupShutdownTunes.Logical	true	
DesktopSounds.Logical	true	
SystemAlertsSounds.Logical	true	
KeyboardBell.Logical	true	
KeyClickVolume.Gauge	-	
DefaultSoundOutput.Choice	false	
SoundOutputVolume.Gauge	-	
Links: <ul style="list-style-type: none"> http://blog.chapagain.com.np/ubuntu-linux-increase-decrease-volume-from-command-line-keyboard-shortcut/ 		
Todo: <ul style="list-style-type: none"> Define the Choices (table above) Write CLI commands (script) Integrate msettings CLI into dtounds application Add command-line option -apply that just start, load, apply the settings and quit the dtounds application (does not start the visual portion of the app) Add logic to apply settings then run update-desktop afterwards 		

Localisation Settings

Schema	Default	Preference Panel
Language.Choice	-	
KeyboardInput.Choice	-	
Links:		
Todo: <ul style="list-style-type: none">• Define the Choices (table above)• Write CLI commands (script)• Integrate msettings CLI into ?? application• Add command-line option -apply that just start, load, apply the settings and quit the ?? application (does not start the visual portion of the app)• Add logic to apply settings then run update-desktop and tellwm afterwards		

External References

Colors Settings

https://en.wikipedia.org/wiki/Color_model

https://en.wikipedia.org/wiki/Color_space

<https://stackabuse.com/reading-and-writing-yaml-files-in-java-with-jackson/>

<https://overiq.com/c-programming-101/array-of-strings-in-c/>

https://wiki.archlinux.org/index.php/Font_configuration

<https://feh.finalrewind.org/>

<https://imagemagick.org/index.php>

Code Snippets

Read one time	Read text file line by line C++
<pre> std::string buffer; std::ifstream f("file.txt"); f.seekg(0, std::ios::end); buffer.resize(f.tellg()); f.seekg(0); f.read(buffer.data(), buffer.size()) #include <dirent.h> if (auto dir = opendir("some_dir/")) { while (auto f = readdir(dir)) { if (!f->d_name f->d_name[0] == '.') continue; // Skip everything that starts with a dot printf("File: %s\n", f->d_name); } closedir(dir); } </pre>	<pre> bool loadMenu(const char* filename) { ifstream ifs(filename, ios::in); if (ifs.fail()) { cout << "Toolchest: Menu file " << filename << " not found" << endl; return false; } while (!ifs.eof()) { string buffer; getline(ifs, buffer, '\n'); //ignore blank lines if (buffer.size() == 0) continue; string *data = new string(buffer); if ((data->find("#") == 0) (data->find("!") == 0)) { // comment line, do nothing } else { //add the line to the array! theFile[elements] = data; elements++; } } ifs.close(); return true; } </pre>

HashCode FilePath Generator in C++	HashCode FilePath Generator in Java
<pre>#include <iostream> #include <string> #include <ctime> #include <chrono> int hashCode(std::string); int main() { auto start = std::chrono::steady_clock::now(); std::string str = "Desktop.Settings.DisplayApplicationErrors"; std::size_t hash = hashCode(str); int mask = 255; int firstDir = hash & mask; int secondDir = (hash >> 8) & mask; int thirdDir = (hash >> 8 >> 8) & mask; int forthDir = (hash >> 8 >> 8 >> 8) & mask; auto end = std::chrono::steady_clock::now(); auto elapsed = std::chrono::duration_cast<std::chrono::nanoseconds>(end - start); std::cout << "It took me " << elapsed.count() << " nanoseconds." << std::endl; char filepath[50];std::sprintf(filepath, "%02x/%02x/%02x/%02x", firstDir, secondDir, thirdDir, forthDir); std::cout << str << " hash=" << hash << " path=" << filepath << std::endl; } g++ -std=c++11 hash.cpp -o hash -L/usr/lib64 -lstdc++ -lm</pre>	<pre>String str = "Desktop.Settings.DisplayApplicationErrors"; int hash = hashCode(); int mask = 255; int firstDir = hash & mask; int secondDir = (hash >> 8) & mask; int thirdDir = (hash >> 8 >> 8) & mask; int forthDir = (hash >> 8 >> 8 >> 8) & mask; String path = String.format("%02x/%02x/%02x/%02x", firstDir, secondDir, thirdDir, forthDir);</pre>
<pre>public int hashCode(std::string value) { int h = 0; int s = value.size(); char c_str[s + 1]; strcpy(c_str, value.c_str()); if (h == 0 && s > 0) { for (int i = 0; i < s; i++) { h = 31 * h + c_str[i]; } } return h; }</pre>	<pre>public static int hashCode(String strValue) { int h = 0; char[] value = strValue.toCharArray(); if (h == 0 && value.length > 0) { for (int i = 0; i < value.length; i++) { h = 31 * h + value[i]; } } return h; }</pre>
Average time 4500 nanoseconds on Xeon 5690 @4.13Ghz	Average time 143324 nanoseconds on Xeon 5690 @4.13Ghz Average time 6700 nanoseconds on Xeon 5690 @4.13Ghz (native)