Manual

Lord of User Interface

For program version 3.0.9

Krzysztof Mroczek

February 1, 2015

Contents

1 Manual in a nutshell **2**

2 Overall information **3**

3 Program license **4**
3.1 LordUI online shop 5
3.2 Signing Lordui procedures 5

4 Manual **6**
4.1 Saving the project 6
4.1.1 LUI LordUI files 6
4.1.2 FUI files 6
4.1.3 XML LordUI files 6
4.2 Lordui view 7
4.2.1 Player Window 7
4.2.2 Global Variables 8
4.3 Lordui tools 9
4.3.1 Procedures 9
4.4 Players 10
4.5 Lordui as a Java library 11
4.5.1 Lordui library interface 11
4.5.2 Connecting Lordui library to different kinds of Java applications 13
4.5.3 Listeners 15
4.6 Expressions 16
4.6.1 Expression types 16
4.6.2 Editing expressions 17
4.6.3 Objects variables 17
4.6.4 Functions 18
4.6.5 Expressions examples 25
4.7 Procedure commands 25
4.7.1 Arrows 25
4.7.2 Tray icons 26

4.7.3 User mouseclick 27
4.7.4 GUI 28
4.7.5 Drawing 29
4.7.6 Image operations 30
4.7.7 Input device (Mouse or keyboard operations) 34
4.7.8 Database 37
4.7.9 Java objects support 38
4.7.10 OS commands 38
4.7.11 Socket 40
4.7.12 Integration 41
4.7.13 Sounds 42
4.7.14 Sound and Video 43
4.7.15 Lordui objects creation 44
4.7.16 Lordui meta elements 46
4.7.17 Standard syntax 47
4.7.18 Extensions 52
4.8 Creating macro procedures 52
4.9 Preparing image variables 52
4.10 Lordui native part 53
4.10.1 Text bluring 53
4.10.2 Native operations 53
4.10.3 Window transparency 54
4.11 Lordui Modules 54
4.11.1 Remote Java Module 54

Chapter 1

Manual in a nutshell

Lordui is the object programing tool. The main component is **procedure**. You are able to run procedures - the you create **process**. There may be many processes in one time, but only one process may be active at the moment. Procedures are grouped into **packages**. They consist of **commends** like: sleeping, input device action, loop, image operation, sound operation etc.

Chapter 2

Overall information

The author of the project is Krzysztof Mroczek. While createing the program, some third party librarys were used. These were i.e.:

- kSquaredHook - Handling the input device operations. The library is being developed by: Kristian Kraljic and Johannes Schüth.
- jLayer - mp3 sound files handling
- grammatica - Parsing regular expressions

Chapter 3

Program license

The license of 'Lordui' (named below also as 'program'), consists of the following points:

1. This program is created by Krzysztof Mroczek. Krzysztof Mroczek is it's author and all rights are reserved for him. The author reserves rights to make changes in the program, develop, edit, test, add and remove functionalities.

2. You may may copy and install it on any computer. However it is not allowed to copy with it the license codes - each operating system user has to have own license code.

3. The decompilation, making changes, including the program or it's parts into any project, distribution it's original or modified version is not allowed. The only exception are the third party libraries used in Lordui, that licences allows that.

4. The sources of the program and it's entire code is the property of programs author. It is forbiden to use it without the author's permition for any - private or commercial - target. The only exception are the third party libraries used in Lordui, that licences allows that.

5. It is no allowed to benefit any goods from selling, distributing, training or sharing the program.

6. The 5) point isn't in force, when having the special author's permision.

7. This program is published as is. There is no guaranty of correctness. The user takes whole the responsibilitie for any injuries, fails, missings or losses caused by the program. In particular the author is not responsible for data loosing, file system problems or inability of using the program.

8. The user takes all the responsibilities for using the program. The Author takes no responsibilities for using Lordui program in any action that breaks the law.

9. It is not allowed to change this license or separate it from the program. In particular every copy of this program should be covered by this license.

3.1 LordUI online shop

Base version of LordUI is the demo version only for private usage. It doesn't allow saving the project, that has more than one procedure, doesn't allow opening XML project files and pauses program every 10 minutes. You may extend your license for commerce usage and/or turn of limitations, buying the license code in our online shop. Visit www.lordui.com/en/shop for more information.

3.2 Signing Lordui procedures

You may sign procedures with the key. Signed procedure may be executed on any computer without demo limits - like pausing the player every 10 minutes. This is the way you may create tutorials to be published. To sign the procedure you have to select it and in "Procedure code" section choose "Regenerate" button. New code will appear. Please send it to the program author with an email (you may find contact adres on the lordui website). After getting the payment and generated code, we will send the reply to the email with "user code". It should be provided into procedure using "Enter code" button. Procedure signing should be repeated after every change done on prcoedure. The price is individual for every procedure. The program Author reserves the right to decline signing the procedure without giving any explanation.
Procedures signed with above instruction may be executed without any limits on any computer. You may use the Lordui player, to execute the procedures. Please find it on www.lordui.com/en/download. The player is smaller then Lordui editor but also can be configured to run procedure without showing any Lordui meta-elements. The syntax to run player is as following:

```
[-file <filepath *.lui>|-url <url to *.lui file>] <Procedure full name> <Procedure
```

.

Chapter 4

Manual

4.1 Saving the project

There are dwo different formats of files, to save LordUI project: *.XML and *.LUI. You may save in any of these formats using File->save option (select the file format from the combobox in the bottom part of the save dialog). You may also load a file in any of this formats using File->open option. Note, that by default the open dialog displays the *.lui files - provide other format (like .xml) to use the other one.

4.1.1 LUI LordUI files

The LordUI (LUI) file format is the default one. It stores the projects and global variables. Actually shown pallet or compiled procedures element may be stored optionally. Compiled objects may be added to final procedures to eg publish (in that case you don't have to attach some compilation libraries to Lordui)

4.1.2 FUI files

Not available anymore. Since Lordui 3.0.4 all the functionalities are available in LUI format.

4.1.3 XML LordUI files

XML files are plain text files (with the given structure). They are dedicated for all the users, that like to implement in notepad and like to type a lot. It is not recomended to use this format. XML files are heavy and need a lot of typing. You also can't view the images using the text editor. However, if you still want to edit your project in XML format, you are welcomed.

4.2 Lordui view

The first opened Lordui window is the main window, which provides most of the needed functionalities. In below section we will present basic lordui functionalitites available in Lordui windows. Lordui main window has few

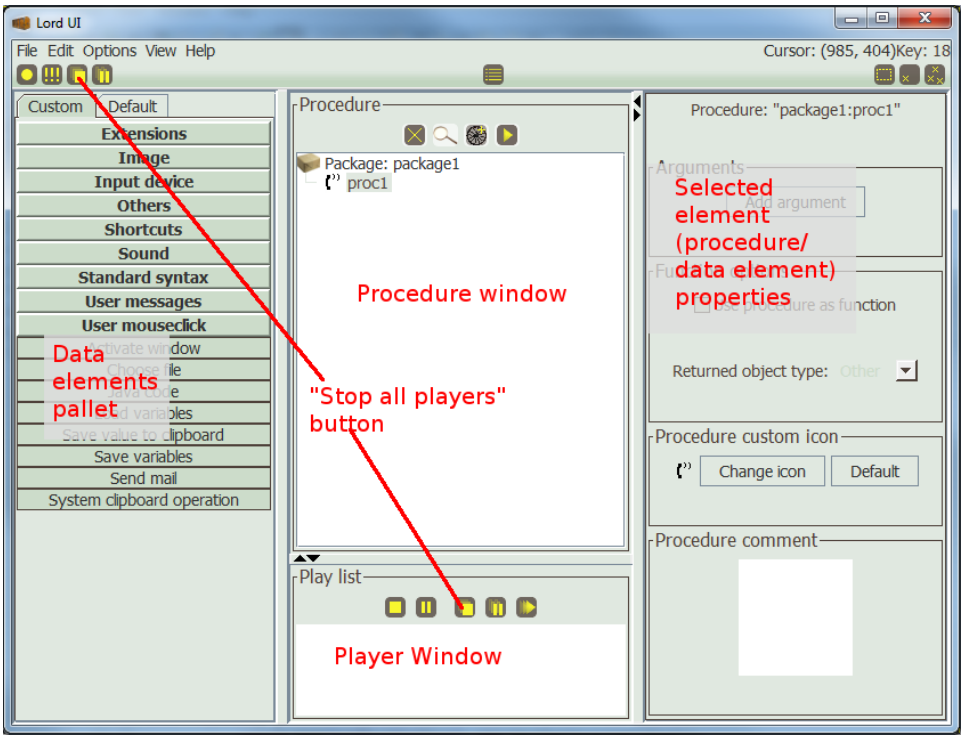

Figure 4.1: Basic elements of Lordui main window

panels. These are:

- Lordui Data Element Pallet on left hand side. Pallet contains commands, that may be inserted into Lordui Procedures. (You may edit Pallets by choosing: ”Edit’ -> ”Edit Pallet toolbar”),
- Procedure window, that presents the procedures in the form of tree view,
- On the bottom you may see Player Window, which shows all the active Processes,
- and selected object properties on the right hand side.

4.2.1 Player Window

The Player window contains the toolbar and the list of active processes. Processes may have attached some executed procedures. When the proce-

dures process is paused, after clicking it with mouse right button, you may see some information about that procedure, like: the stack trace or current Procedure call memory (all the objects that are available from the procedure). On the toolbar, you can find following buttons:

-  - stops selected process,
-  - pauses/resums selected process,
-  - stops all processes,
-  - pauses all processes,
-  - resumes all processes,
-  - Performs single step on selected process (available only if selected process is paused).

4.2.2 Global Variables

Global variables are the variables, that are available for all of the processes. To edit Global Variables, please select  icon from the top toolbar on the main window
The Global Variables window contains the tree of variables (the variables are organized into a tree using variable packages) on the left hand side, and the edit panel on the right side. Global Variable may be added, removed or renamed, using the toolbar icons over the Variables tree (See buttons:  ,  , ).
On the right hand side of Global Variables window you can see value of the selected variable. These could be numbers, booleans, string (Caution! We provide string using double quotes), but also images, sounds or arrays of variables, that could differ in type eg

```
[1, false, ,,text’’]
```

is a free elements array, that contains number (1), boolean value (false) and a string (”text”). Some object types (like image) provide also a dedicated edit view. However all of the objects may be edited in the basic edit view, which has a edit field and a toolbar over it. The toolbar contains following elements:

-  - insert the image (into the place of the cursor in edit field). Image can be loaded from file, past from system buffer (clipboard) or copied from the currently shown screen (like screenshot),

-  - insert the sound. Sound me be loaded (wav, mp3) or recorded (wav only),
-  - insert the point (point coordinates). Provide the coordinates by selecting proper point on your screen. The coordinates may be also provided manualy in Edit Field using format: (<X value>, <Y value>),
-  - insert area coordinates. Provide the area by selecting it on the screen. You may provide it also manually in Edit Field using format (<X value>, <Y value>, <width>, <height>),
-  - undo the changes - restores the variable value that from the time, editting was started,
-  - Switch the edit view between standard and dedicated mode,
- The type of the provided expression.

If the provided expression is not supported or it contains an error, the error message will appear on the bottom of the Global Variables Edit Window.

4.3 Lordui tools

4.3.1 Procedures

The first step while working with Lordui is to create a Procedure. You may create procedure by hitting the button  or by selecting "Add procedure" from the right mousebutton click-menu on procedure window. Procedure name should contain letters, digits and colons only. Every colon will be threaten as the package separator, where the last ingredient is the name of procedure, and all previous ingredients are package names e.g. name it "package1:package2:proc" to create a package1, that will contain package2, where the Procedure "proc" will be stored.
Procedures may have arguments. To be able to edit the procedure arguments you should select the procedure in procedures window. After adding the argument, you should fill in it's type and name. **It is recommended to have single type per single name for whole project.** The procedure may be deleted, renamed (by renaming the packages name, it will be moved to another package), duplicated or copied. There is also a tool to search for procedures.
While there is any procedure executed on any player, there are two very import keyboard shortcuts available:

1. **Scroll lock - Stops running all currently executed procedures (all the players)**

2. **Pause/break - Pauses or resumes all players**

Running the procedure

There are few ways of running the procedure.

- The first one is to select the procedure or one of the procedures sub-command and the procedure tree and hitting the  button on the toolbar. The new window should pop up - you will be able to enter the number of procedure calls, minimal and maximal time interspace between procedures call, the time before the first proceure call, all procedure arguments values and the player name as well.

- The second way of executing the procedure, is to define the keyboard shortcut: *'Edit' -> 'Keyboard shortcuts'*. The variable choice, the number of executions and other settings are available under the *'Set.'* button for every shortcut row. Please note the option *'shortcuts active while any of the procedure is executed'* option in keyboard shortcuts window and the option *'Shortcuts active'* in main menu *'Edit'*. If any of this option is disabled, keyboard shortcuts may not work.

- Another way to execute the procedure is to insert proper element into another procedure. Then the new procedure will be executed during that element execution. Procedure (player) execution may be stopped or paused at any moment. To stop all executed procedures, press keyboard **Scroll lock** key or toolbar button  . You may also stop one, selected process by selecting it with right mouse button and choosing 'Stop this process'.

- Procedures may also be executed automaticly by listeners (see section 4.5.3: "Listeners").

To pause all players select keyboard key **Pause/break** or toolbar button  . When the player is paused, there are some debug options available on it - Select the process with mouse right click from 'Play list'. Then

1. select 'Show memory', to see all values of variables from memory of the selected process,

2. select 'Show stack trace', to see the stack trase of selected process.

4.4 Players

The Procedure (thread) is being executed by Players. Every procedure run is being executed in separated thread. Every Player can execute only single

thread at the time - the rest of the threads scheduled for the given player are waiting until current thread will end the job or will pause execution (e.g. see 4.7.17: "Sleep"). There are many ways to execute (assign) thread to the player - see section 4.3.1: "Running the procedure" for more.
Every Thread scheduled to the Player has it's own memory cache. There is also a common memory for whole Lordui Project. The Player is being created while assigning first Thread to it. It's being closed, after last Thread removal. Some of the Players (i.e. the ones having any listener assigned) will not stop after last thread end and need to be stopped manualy (which will stop all it's assigned listeners).

4.5 Lordui as a Java library

Lordui may be integrated with any Java language application. This allows better control on User Interface elements using the references between object. This improves the implementation simplicity, projects quality and performance. You may also comunicate between Lordui and application. Some functionlities may support only Swing objects. You may contact Lordui Author to add the support of other User Interface objects.

4.5.1 Lordui library interface

Lordui library interface consists only single Java Class::

`ktm.lordui.Lordui`

. To initialize Lordui you have to call the static method *createInstance*. You may create only one instance of Lordui class - every next call will return the previously returned object. When ending the work, call *close* method. Please find below the list of functions available over Lordui class:

1. *public final void loadProject(File luiFile) throws IOException* - opens project from given *.lui file,
2. *public final void setValue(String name, Object value)* - stores in Lordui given object (like instance of Window, int, String, Point, etc.) under given global name,
3. *public final Object getValue(String name)* - gets from Lordui the object stored under given global name,
4. *public final void runProcedure(String procedureName)* - Executes the procedure,
5. *public final void runProcedureAndWait(String procedureName)* - Executes the procedure. Function *runProcedureAndWait* won't end execution until all procedures from the player won't stop running,

6. *public static final Lordui createInstance()* - returns the instance of Lordui object,

7. *public final void close()* - close Lordui library. Without calling it, Lordui may not dispose all the objects correctly. After calling it, you should no more use the Lordui class instance neither call *Lordui:createInstance* function,

8. *public final void setVisible(boolean visible)* - shows/hides Lordui editor window.

Here is the example of using Lordui library:

```
public class LorduiLibraryUsage {
private Lordui lui;

public void useLordui() {
lui = Lordui.createInstance();
try {
lui.loadProject(new File("myLorduiProcedureFile.lui"));
} catch (IOException e) {
e.printStackTrace();
return;
}

//Here implement the code start your application, show your user interface
try {
SwingUtilities.invokeAndWait(new Runnable() {
@Override
public void run() {
JFrame fr = new JFrame("Test window");
fr.setSize(800, 600);
fr.setVisible(true);
}
});
} catch (InvocationTargetException e) {
e.printStackTrace();
return;
} catch (InterruptedException e) {
e.printStackTrace();
return;
}

lui.runProcedureAndWait("LorduiProcedureName");
lui.close();
```

```
}
}
```

4.5.2 Connecting Lordui library to different kinds of Java applications

Before reading this chapter, please double check 'Remote Java' (see chapter: 4.11.1: "Remote Java Module".) module isn't something better for your needs.

There are various ways to connect Lordui into your project. To do it, you don't have to modify, or even have access to the code of the application. Before you will to it, please make sure you won't break your software license.

Desktop/Webstart applications

We name the classic *.jar the desktop application here. The information about main class should be in the *.jar file. Such a file we may call with *java -jar myFile.jar* command. This command will run application starting from the given class that contains *main(String[] args)* function. Ie.:

```
public class MainClass {
public static main(String[] args) {
//Run my app
}
}
```

In this case lets make the connection this way.:

```
public class MainLorduiClass {
public static main(String[] args) {
ktm.lordui.Lordui lui = ktm.lordui.Lordui.createInstance();
try {
lui.loadProject(new File("myLorduiProcedureFile.lui"));
} catch (IOException e) {
e.printStackTrace();
return;
}
lui.setVisible(true);
MainClass.main(args);
lui.runProcedureAndWait("LorduiProcedureName");
}
}
```

Please find we missed *close()* command on the end. If only there is possibility to run it somewhere on the end - please do it. If not, please close the Lordui window manualy to perform correct resources dispose(before it will be close eg with

```
System.exit(0)
```

command). Such a prepared Class you may pack into jar and run as a main one.

Applet

In case of applets it's very similar to applications. Imagine, we have an applet, that we want to connect with Lordui:

```
public class EntryApplet extends JApplet {
@Override
public void init() {
//Some init stuff
}

@Override
public void destroy() {
//Some destroy stuff
}
}
```

Now let's prepare our connection applet:

```
public class LorduiApplet extends EntryApplet {
private ktm.lordui.Lordui lordui;
@Override
public void init() {
lordui = ktm.lordui.Lordui.createInstance();
try {
lui.loadProject(new File("myLorduiProcedureFile.lui"));
} catch (IOException e) {
e.printStackTrace();
return;
}
super.init();
}

@Override
public void start() {
super.start();
lui.setVisible(true);
}

@Override
public void stop() {
```

```
lui.setVisible(false);
super.stop();
}

@Override
public void destroy() {
lui.close();
super.destroy();
}
}
```

This new class you may pack into jar file and run it as a main Applet Class.

tests

Running tests is the most comfortable example here. You may easily combine it with eg jUnit tests, but it's not the only way. Before running tests, please put something like:

```
lui = Lordui.createInstance();
try {
lui.loadProject(new File("myLorduiProcedureFile.lui"));
} catch (IOException e) {
e.printStackTrace();
return;
}
```

then, after creating your UI you may safely run Lordui procedure and dispose all allocated resources after all:

```
lui.runProcedureAndWait("LorduiProcedureName");
lui.close();
```

Such an example of Lordui usage you may find in chapter: 4.5.1: "Lordui library interface".

4.5.3 Listeners

Listeners are the special objects to observer e.g. user action. You may listen using the data element 4.7.7: "Start keyboard listener" or 4.7.7: "Start mouse listener" data element. The listener is being assigned to the Player. Every time, the event occurs the Procedure is executed (or more formal: to the selected Player the new Thread is being assigned to run the selected Procedure). To stop the Listener you have to stop the listener (attention: the player with any listener assigned won't stop when it's last thread will stop running) or run the command 4.7.16: "Stop listeners".

4.6 Expressions

The program is able to compute set of expressions. As an expresion we mean here the set of operations on objects. The object may be string, number, image, point etc.
The variables, similar to procedures, may be grouped into collection. To place the variable into collection, there should be added the collection name as the prefix of variable name. The names should be separated with colon(In Lordui colon is the synonim of package separator). I.e. giving a variable name *a:b:c*, this variable will be called *c* and it will be stored int collection *b*, that will be stored in collection *a*.
Lordui allows finding all uses of the variable in the procedures. To run the search, click *'Edit' -> 'Show variable calls'*. In shown window the full variable name should be provided.

4.6.1 Expression types

There are many object types in Lordui, like e.g.:

- List,
- Point ((0,0) is in upper left corner),
- String,
- Number,
- Image,
- Positioned image - Image with stored coordinates (Image + Point pair)
- boolean (true/false - the values are case sensitive),
- Area (x, y, width, height), where (0,0) coordinates are in upper left corner,
- Sound,
- Other object.

The objects of given types may be saved in global variables - they will be available from all the processes. To open the global variables edition window, choose  button from upper toolbar. The window contains the list of all declared global variables.
The objects may also be declared and calculated during runtime. Every process has it's own memory, so it's independent from other processes. Every assigment during runtime takes place in local process memory, so it has no influence on global variables or other processes variables. While reader the variable value, process checkes if the object of given name is stored in local memory. If not it checks in global variables.

4.6.2 Editing expressions

Exprission edit is available while creating procedures commands. You may fill in simple values but also fill in more complex expressions. You may use expressions while defining global variables as well, but they will be stored as simple values there. Please note, that while defining expressions, if you want to enter a string, you must use double quotes. Strings without quotes will be interpret as other objects (like eg variables).

Figure 4.2: Example of expression edit field

The expression edit field, depending on the context, may contain some of following buttons:

- , - respectively local/global variable (click to switch). Option is available only for expression defining the name of variable to store the procedure command result. You may save the variable localy (in memory of current procedure call) or as a global variable,
- - Image preview - preview the image (if only the expression evaluates to image in context of global variables),
- - Select area/image - click to select image or area on screen,
- - Choose point - click to choose the point on you screen,
- - Display available/insert build-in functions and attributes.

4.6.3 Objects variables

Some objects have defined some variables(fields) over them. To read such a field, after writing the object expression add dot and name of the field. There are following fields available:

- **x**
  Active window (focus owner) coordinates relative to top left position of main monitor and size. Returned value is area typed (x, y, width, height)
- **y**
  Mouse cursor position relative to top left position of main monitor

- **w**
  Gets the value of the given name from the memory (this is the dynamic alternative for static field variable calls)
- **h**
  Active window (focus owner) title
- **red**
  Draws the text on the image with transparent background
- **green**
  Creates the new image object
- **blue**
  Whole screen (including all screen devices) size

4.6.4 Functions

There are also some functions defined in Lordui. There are two types of functions. The first one are defined on objects. The second one, the system functions, should be accessed in static way.

Functions on objects

There are following functions on objects:

- **function get(index)**
  Gets the array object of the given index. Undefined behavior for out of range index (most likly procedure will stop)
  - **index** - index of the object in the array
- **function set(index, value)**
  Stores the value in array under the given index. Returns the new array (may be the same object as the oryginal one). Won't do any shift's.
  - **index** - index of the object in the array
  - **value** - object to store in array
- **function size()**
  Size of the array - maximum index of object stored in array + 1
- **function trim()**
  Returns trimmed (on the beggining and end) string from spaces, tabs, new lines
- **function replace(pattern, replace_with)**
  Returns string with all the patterns replaced with given value

  - **pattern** - pattern to find in string - see java's Pattern class (* is a wildchar)
  - **replace_with** - value to replace the found pattern occurance with.

- **function getPosition()**
  Position of the given positionImage

- **function split()**
  Splits given string into array of strings - see Java's String:split function for extreme cases

- **function pushEnd(object)**
  Pushes the object to the end of the array. Returns new array.
  - **object** - Object to push

- **function pushBegin(object)**
  Pushes the object to the begining to the object. Shifts all the elements one position right. Returns new array.
  - **object** - Object to push

- **function push(object, index)**
  Pushes the object to the given index of array. Shifts the elements starting from index position one step right. Returns new array.
  - **object** - Object to push
  - **index** - Index to push at

- **function toPoint()**
  Returns point based on the value

- **function toArea()**
  Area of the position image

- **function subImage(x, y, width, height)**
  Positioned subimage from the subimage. Result image contains summed coordinates of parent and subimage
  - **x** - X position of the subimage relative to left side of parent image
  - **y** - Y position of the subimage relative to top side of parent image
  - **width** - Width of the subimage
  - **height** - Height of the subimage

- **function subImage(x, y, width, height)**
  Subimage of the image

  - **x** - X position of the subimage relative to left side of parent image
  - **y** - Y position of the subimage relative to top side of parent image
  - **width** - Width of the subimage
  - **height** - Height of the subimage

- **function location()**
  Location (Point - left top corner coordinates) of the window on the screen

- **function middlePoint()**
  Location (Point) of the center pixel relative to the top left screen corner

- **function findUISubcomponent(functionName, value)**
  returns the UI object subcomponent of the given UI object and that returns the proper value of the chosen function

  - **functionName** - Name of the function to be called on the subcomponent
  - **value** - Expected value to be returned by searched subcomponent

- **function readyForReading()**
  Returns boolean - true if there is any data/message available on Socket to read

- **function mergeValues(separator)**
  Merges the array of objects casting them to string and separating with given string

  - **separator** - String to separate the objects

- **function getColor(x, y)**
  Returns the color on given image over given coordinates

  - **x** - horizontal coordinate
  - **y** - vertical coordinate

- **function subImage(area)**
  Calculates subimage of the image. The subimage will share the data with the oryginall image - any change will be reflected on both images

  - **area** - Area of the subimage on the oryginal image

- **function subImage(area)**
  Calculates subimage of the image. The subimage will share the data with the oryginall image - any change will be reflected on both images

  - **area** - Area of the subimage on the oryginal image

- **function copy()**
  Creates the copy of the given image
- **function copy()**
  Creates the copy of the given image

System functions

There are following system functions defined:

- **static function activeWindowCoordinates()**
  Active window (focus owner) coordinates relative to top left position of main monitor and size. Returned value is area typed (x, y, width, height)
- **static function cursorPosition()**
  Mouse cursor position relative to top left position of main monitor
- **static function getValue(index)**
  Gets the value of the given name from the memory (this is the dynamic alternative for static field variable calls)
  - **index** - array index of the object to return (starting from 0)
- **static function createImage(width, height)**
  Creates the new image object
  - **width** - width of the image
  - **height** - height of the image
- **static function captionImage(text, fontName, fontSize, fontStyle, color)**
  Draws the text on the image with transparent background
  - **text** - text to paint
  - **fontName** - name of the font to draw the string (family name, case sensitive)
  - **fontSize** - size of the text
  - **fontStyle** - Font style - 0 to use default style.
  - **color** - Color of the text - HTML (hexadecimal) format like FFFFFF
- **static function activeWindowTitle()**
  Active window (focus owner) title
- **static function screenSize()**
  Whole screen (including all screen devices) size

- **static function BOLD()**
  Bold font style constant
- **static function compareImages(image1, image2)**
  Calculate the area of the images that differs
  - **image1** - First image to compare
  - **image2** - Second image to compare
- **static function screenArea()**
  Returns the area of the screen - the size and the coordinates of top left corner
- **static function concat(area1, area2)**
  new object - intersection of two areas
  - **area1** - First area to concatenate
  - **area2** - Second area to concatenate
- **static function popFirst(arrayName)**
  Pops the first element from array shifting all remaining items one step left. Result (one element shorter array) is stored in memory under given variable name. Returns popped element.
  - **arrayName** - Variable name of the array
- **static function popLast(arrayName)**
  Pops the last element from array. Result (one element shorter array) is stored in memory under given variable name. Returns popped element.
  - **arrayName** - Variable name of the array
- **static function pop(arrayName, index)**
  Pops the given indexed element from array shifting all remaining items one step left. Result (one element shorter array) is stored in memory under given variable name. Returns popped element.
  - **arrayName** - Variable name of the array
  - **index** - Index of the element to pop
- **static function getClipboard()**
  Returns clipboard value - Image, integer, boolean or string
- **static function readImage(url)**
  Loads image from given file or url
  - **url** - Url or path (either relative or full) of the image file

- **static function color(htmlColor)**
  Creates color object
  - **htmlColor** - HTML format (like 0xFFFFFF) of color to create
- **static function color(red, green, blue)**
  Creates color object
  - **red** - Value of red color (0-255)
  - **green** - Value of green color (0-255)
  - **blue** - Value of blue color (0-255)
- **static function getColorArea(image, startPoint)**
  Calculates area of given color object - searches the pixels maximum to the left, top, right, bottom, that are connected to the given one
  - **image** - Image, where to check the area size (the coordinates of area are relative to image (0,0) position
  - **startPoint** - Starting pixel coordinates
- **static function getBackgroundColor(image, area)**
  Calculates the background color of the object (the color that most of the pixel have)
  - **image** - Image where to check the background color
  - **area** - Area on the image, where to check the coordinates
- **static function getBackgroundColor(image)**
  Calculates the background color of the object (the color that most of the pixel have)
  - **image** - Image where to check the background color
- **static function activeWindow()**
  Current active window object (pointer).
- **static function getActiveJavaWindow()**
  Current Java active window pointer. Null if none is active.
- **static function callJavaStaticFunction(ClassName, Function, Args)**
  Calls the static function over the given class. Returns returned value or null if fuction returns void
  - **ClassName** - Name of the class
  - **Function** - Name of the function to be executed
  - **Args** - Array of arguments

- **static function callJavaFunction(object, Function, Args)**
  Calls the function over given object. Returns returned value or null if function returns void
  - **object** - Object, over that the function will be called
  - **Function** - Name of the function to be executed
  - **Args** - Array of arguments
- **static function variableExisists(variableName, globalOnly)**
  Returns true if the variable is declared
  - **variableName** - Name of the variable
  - **globalOnly** - True for global variable only, false for global variable or current Procedure call variable
- **static function createScreenshot(area)**
  Create the screenshot
  - **area** - Screenshot area
- **static function getWindowWithTitlePrefix(titlePrefix)**
  Gets the pointer to the any application window, that starts with given text
  - **titlePrefix** - Text prefix of the searched window
- **static function getSystemProperty(propertyName)**
  System value. See Java call: System.getProperty(String)
  - **propertyName** - Name of the property to get
- **static function isImage(object)**
  Returns true if given object is the image
  - **object** - Object to be checked, if it's a image
- **static function abs(value)**
  Calculates area of given color object - searches the pixels maximum to the left, top, right, bottom, that are connected to the given one
  - **value** - Image, where to check the area size (the coordinates of area are relative to image (0,0) position

Defining own functions

To declare own functions, in procedure definition check the „Function" Checkbox. It is recommended also to choose the returned object type. User defined functions return the value with „Return" procedure element. They may be called just like a standard system function.

4.6.5 Expressions examples

Here are some expression examples:

1. *false || true* - evaluates to *true* - the boolean,
2. *1 <= 2 && 2 != 3* - evaluates to *true* - the boolean,
3. *"Hello " + "world!!!"* - evaluates to *"Hello world!!!"* - the string,
4. *7 + 8* - evaluates to *15* - the number,
5. *"a,b,c,d,e".split(",").get(1)* - evaluates to *"b"* - the string,
6. *screenSize().x + aVariableName* - assuming, that under *aVariableName* there is a number stored evaluates to number value,

4.7 Procedure commands

Procedures, the main component of the program, are build of commands. The command is the basic programming tool in Lordui. The following section describes commands available in the program. You may insert commands in few ways:

- with right mouse click on procedure or command and choosing the menu item 'Add step to/before/after',
- edit it in text editor in XML format and the paste it to procedure,
- with left commands menu.

Most of the commands have parameters, that you may edit clickng it on procedure tree or by selecting  to edit it in new window. Commands may also be cut, copied, moved with menu under right mouse click or with popular keyboard shortcuts *ctrl+x*, *ctrl+c*, *ctrl+v*. The commands are stored in memory in XML format.
To remove command from procedure, select it and click  toolbar button or simple *Delete* keyboard button.

4.7.1 Arrows

Hide arrow

Hiding the arrow of given variable.
Parameters:

- **Arrow variable name** - Variable name - Name of the variable holding the shown arrow.

Show/move arrow

Will show arrow to the given coordinates. If there is arrow declared under given variable, the arrow will only be moved.
Parameters:

- **Arrow variable name** - Variable name - Name of the arrow variable to store/modify,
- **Place pointed by arrow** - Point expression - Position of the screen to point out by arrow.

4.7.2 Tray icons

Tray icons are small icons behind the clock on start bar. LordUI supports creation of custom tray icons.

Add tray icon menuitem

Creates the try icon menu item.
Parameters:

- **Variable name** - Variable name - Name of the tray icon variable,
- **Menu item caption** - String expression - Caption to display,
- **Player name** - String expression - Name of the player, to run the procedure on menu element click.

Subprocedures:

- **itemAction** - Procedure executed on menu item mouse click.

Change tray icon

Changes the displayd icon of the tray icon.
Parameters:

- **Variable name** - Variable name - Name of the tray icon variable,
- **Icon image expression** - String expression - Image to show. The image will be scaled to fit the size of icon.

Hide tray icon

Hides the given tray icon.
Parameters:

- **Variable name** - Variable name - Name of the tray icon variable.

Show tray icon

Displays the tray icon.
Parameters:

- **Variable name** - Variable name - Name of the tray icon variable,
- **Icon image expression** - Image expression - Image to show. The image will be scaled to fit the size of icon.

Show tray icon message

Displays the tray icon tooltip message.
Parameters:

- **Variable name** - Variable name - Name of the tray icon variable,
- **Message caption** - String expression - The header of tray icon mesage,
- **Message text** - String expression - The text of tray icon mesage.

4.7.3 User mouseclick

The screen lock is implemented with the transparent window, that is displayd full screen in front. This lock may be stopped by pressing *Escape* keyboad button - information about that will be shown on screen.

Ask user to click

If possible, this command should be called before screen lock is turned on. This command adds area on screen to click. After clicking the area, the subprocedure corresponding to this area (this command) will be executed.
Parameters:

- **Area for the user to click** - Area expression - Area that will be highlighted on the screen. User will be allowed to click here,
- **Lock name** - String expression - Name of lock variable,
- **Player name** - String expression - Name of the player to run the procedure on user click.

Subprocedures:

- **OnClick** - Procedure executed on user click on the given area.

Remove user action

Removes every *'Ask user to click'* areas from lock. After calling this method screen lock is empty - should be populated with click areas once again.
Parameters:

- **Lock name** - String expression - Name of lock variable.

Screen lock - on/off

Turns of or on the lock.
Parameters:

- **Turn on of off** - Checkbox - Lock can be turned on or off,
- **Lock name** - String expression - Name of lock variable.

4.7.4 GUI

GUI - Graphical User Interface. All user-like operations on windows, pop-ups, tray icons etc.

Window activation

Activates the provided window. If there is no such window, nothing will happen.
Parameters:

- **Window expression** - Object expression - Window to be activated.

Choose button

Short text message with the set of buttons. Every button has it's subprocedure declared.
Parameters:

- **Shown text** - String expression - Text to display,
- **Variable name** - Variable name - Variable to store the users answer value.

Choose file

The file choosing window will be shown. The full path of selected file will be stored to the given variable.
Parameters:

- **Expression** - Variable name - Name of the variable (not an expression) under that the path of file should be stored.

Resize window

Move and resize window (any window of any application running on operating system)
Parameters:

- **Window** - Object expression - Pointer to the window to be moved,
- **Area** - Area expression - Target area of the window (position and size).

Text message

Short text message.
Parameters:

- **Message** - String expression - Text to display.

4.7.5 Drawing

With below commands you may draw some simple shapes on the image variables.

Draw image on image

Draws given image on other image.
Parameters:

- **Image to draw** - Image expression - Image object to be drawn,
- **Area** - Area expression - area on the target image, where the image will be painted,
- **Image name** - Image expression - Image object to draw on.

Draws or fills rectangle

Paints the rectangle on the given image.
Parameters:

- **Color** - Color expression - Color that should be used to fill the area (you may use color(r,g,b) function here),
- **Area coordinates** - Point expression - Coordinates on the image,
- **Image name** - Variable name - The source image and the result image at the same time,
- **Fill** - Boolean expression - if the Rectangle should be filled (else the rectangle frame only is drawn).

Set pixel color

Fills the pixel on the given image.
Parameters:

- **Pixel coordinates** - Point expression - Coordinates on the image,
- **Color** - Color expression - That should be used to fill the area (you may use color(r,g,b) function here),
- **Image name** - Variable name - The source image and the result image at the same time.

4.7.6 Image operations

Click image

This is a shortcut for searching an image on the screen. While inserting it into procedure the programer will be asked to select the pattern image on the screen and provide it's variable name. The image will be set as pattern to be searched on screen. If the option "Add error msg when image not found" is selected, then into the "notFound" subprocedure error msg popup data element will be added. By default, when the pattern will be found, it will be click on it's center point. If not - all players will be stopped.

Find image

Command will search the pattern on the image currently displayd on all screen devices. With this command you may check if the given pattern currently occurs on the device and make ny action (i.e. wait).
Parameters:

- **Expression of pattern image** - Image expression - The evaluated image will be searched,
- **Inversion** - Checkbox - If inverion is turned on, the command will wait until the pattern will disappear from screen. If unchecked, command waits until the pattern occurs on the screen,
- **Find all** - Checkbox - If turned on, as the result will be returned the array of points - the coordinates of all matches of the pattern on image. If tuned off, the returned value will be single Point object - the coordinate of single match of the pattern,
- **Name of the variable, to store the result(s) position** - Variable name - Name of the variable to store the point of the screen, where pattern was found on screen. The upper left point position will be

saved. The assigment takes place every time, just after the pattern was found. Leave empty, if no variable should be assigned. If command ends working and pattern was not found, variable will not be assigned. If "Find all" checkbox checked, the array of points will be stored.,

- **Level of comparison** - Scroll - How much the found image should be similar to the pattern (leave zero to 100
- **Timeout** - Integer expression - After how many miliseconds the searching should be stopped - no matter image was found or not,
- **Area/Point where the iamge should appear** - Point/area expression - Place where the pattern should be searched (area or the point of top left corner of the image). Leave empty if full screen/full image should be searched,
- **Source of image** - Image expression - Image where the patten will be searched. Leave empty to search on the image (screenshot will be created)..

Subprocedures:

- **FoundProcedure** - Executed after image was found, just before the end of command. It will be executed on image found (if inversion is off) or on timeout (if inversion is on),
- **NotFoundProcedure** - Executed every time if pattern was not found, just before leaving the command. It will be executed if image was not found (if inverion is on) or on timeout (if inversion is off),
- **OnSearching** - Will execute if pattern was not found (or pattern was found if inversion is turned on). If only it was possible, the coordinates of pattern are stored in variable. This subprocedure is executed only between searching - if image was being searched and not found, and it will be searched again in a moment.

Every time, after image was being searched one, and only one procedure is executed.

Save image to file

Command will save the image from Lordui memory to the file.
Parameters:

- **Image file path algorithm** - Options - there are three options:
    1. The full path - The file path is given exactly - the image should be written under this path even, if file exists.

2. Ask for file name - Ask user during runtime for the place of storing the file,
3. Add sufix to filename to make filename unique - Append the suffix to the filename to make the file name unique.

,

- **Default filename** - String expression - This is the default filename either to save the image or to propose the user to save it,
- **Image expression** - Image expression - Image to save to file.

Screenshot

The image screenschot will be created and stored to memory.
Parameters:

- **Image file path algorithm** - Options - there are three options:
  1. The full path - The file path is given exactly - the image should be written under this path even, if file exists.
  2. Ask for file name - Ask user during runtime for the place of storing the file,
  3. Add sufix to filename to make filename unique - Append the suffix to the filename to make the file name unique.

  ,
- **Variable name to store image** - String expression - Name of the variable that will be used to store image into memory,
- **Image expression** - Area expression - The area of the screen, that should be captured. Leave empty for full screen. note, that top left corner of the screen device isn't always the (0, 0) position.

Search string

The string will be searched. To search the string, the font, color, string and result variable name should be provided. The caption will be searched as image. If there is the area, of color of that string it will be threated as the searched caption. Also any string bluring (see Microsoft Cleartype technology) will make the command working wrong.
Parameters:

- **Searched text** - String expression - Text to be searched,
- **Font name** - String expression - Font family name of searched string,

- **Font size** - Integer expression - Font size of searched string,
- **Font style** - Integer expression - 0 for default style or BOLD() value for bold text,
- **Name of variable to store the result** - Variable name - Name of the variable, where the top left coordinates of the string will be stored,
- **Color in hexadecimal** - String expression - The HTML format string expression, with the color of the searched text.

Show image

Will show image in a popup dialog window.
Parameters:

- **Image expression** - Image expression - The image to be displayd on screen.

OCR - text recognision

The area of searching (empty if full screan), font, color and result variable name should be provided. In the given area there should be no distortion (i.e. pixel of given color, that is not the component of the caption). Current OCR algorithm contains no heuristic methods. That is why if only every condition is fullfilled, it is 100Parameters:

- **Search area** - Area expression - Area whith text to be searched. Only single line of the text should be on that area.,
- **Font name** - String expression - Font family name of searched string,
- **Font size** - Integer expression - Font size of searched string,
- **Font style** - Integer expression - 0 for default style or BOLD() value for bold text,
- **Name of variable to store the result** - Variable name - Name of the variable, where the text will be stored,
- **Color in hexadecimal** - String expression - The HTML format string expression, with the color of the searched text. On the checked image only the text should be displayd with given color. All the other pixels should have other color..

User marks image

User will have to mark the area on the screen. If the user will resign, the subprocedure "User cancelled" will be executed.
Parameters:

- **Variable name** - Variable name - Name of the variable, where to store the area if user provided an area.

Subprocedures:

- **OnCancel** - Executed when user cancelled marking area.

4.7.7 Input device (Mouse or keyboard operations)

Start keyboard listener

Keyboard listener will be executed. Every keypress that evaluates the filter into true value will cause execution of thread on the selected player. The executed listener will be active until it will be stopped - it may be stopped by command (like see efStop listeners: "Stop listeners" command)
Parameters:

- **Filter** - Boolean expression - Logic expression to determine whether the thread will be added to player or not,
- **Keycode variable name** - Variable name - Under this variable the keycode will be stored. This variable is available while evaluating the filter value,
- **Player name** - String expression - Player name to execute the threads,
- **Key action** - Options - You may start listening for only one of the two actions: key press and key release,
- **Timestamp** - Variable name - This value will contain the timestamp in miliseconds since 1st January 1970.

Subprocedures:

- **KeypressAction** - Action to run on keypress.

Keyboard operation

Keyboard key press, release or click (press + release). The keyboard key code should be provided. You may check the keyboard key on menubar - the right side under 'Key'. To see the last presset keyboard key value, you have to check the checkbox under *'Options' -> 'Show input devices values'*.
Parameters:

- **Key operation** - Options - there are three options:

  1. Keyboard key press - if press operation should be performed
  2. Keyboard key clicked - if press and release should be performed
  3. Keyboard key released - If release operation should be performed

  ,

- **Keycode** - Integer - Keycode of the key to perform the operation on.

Mouse event

Mouse mouse simple operation to be performed. Please note to distinct drag from move. Proper opereration should be choosed according to the current mouse state.
Parameters:

- **Cursor position** - Point expression - Coordinates of target pixel to perform the operation,
- **Mouse button** - Options - There are three buttons to choose:

  1. Left mouse button
  2. Middle mouse button
  3. Right mouse button

  ,

- **Button action** - Options - There are few operations you may perform:

  1. Press mouse button
  2. Click with mouse (press + release)
  3. Release mouse button
  4. Move the mouse cursor
  5. Drag the mouse cursor

  ,

- **Cursor move type** - Options - The type of cursor move. Following mouse moves are available:

  1. Jump (Cursor will appear on the target position immediately)
  2. Constant velocity move (You may provide the velocity of mouse cursor - it's in number of pixels per 1/100 second)

  .

Start mouse listener

Mouse listener will be executed. Every mouse click (button press action) that evaluates the filter into true value will cause execution of thread on the selected player. The executed listener will be active until it will be stopped - it may be stopped by command (like see efStop listeners: "Stop listeners" command)
Parameters:

- **Filter** - Boolean expression - Logic expression to determine whether the thread will be added to player or not,
- **Number of button variable name** - Variable name - Under this variable the number of button will be stored. This variable is available while evaluating the filter value. Values could be:
  1. 3001 - left mouse button
  2. 3002 - middle mouse button (or mouse wheel click)
  3. 3003 - right mouse button
  4. 3004 - no mouse button pressed

  ,
- **Positon variable name** - Variable name - Under this position on the screen will be stored. This variable is available while evaluating the filter value,
- **Player name** - String expression - Player name to execute the threads,
- **Mouse action** - Options - You may start listening for only one of the following three actions: mouse press, mouse release or mouse move/drag,
- **Timestamp** - Variable name - This value will contain the timestamp in miliseconds since 1st January 1970.

Subprocedures:

- **ClickAction** - Action to run on mouse click.

Point out with cursor

The given screen point will be pointed out with mouse cursor. The point, radius and the time (in miliseconds) should be provided.
Parameters:

- **Pointed out position** - Point expression - The cursor will move around the given position,

- **Circle radius** - String expression - The radius of the circle to move the cursor,
- **Length** - Integer expression - The duration of the pixel pointing out in miliseconds.

Text input

Typing the text (this is kind of shortcut for multiple keyboard events).
Parameters:

- **Text to type** - String expression - The text to be typed,
- **Sleep time after key press/release** - Integer expression - How many miliseconds to wait after any key press/release operation.

Wait for mouseclick

Command will wait until the mouse click will be done.

4.7.8 Database

With database data elements you will be able to connect and query database elements. You will need add Java drivers to Lordui classpath, to use this functionality

Data base connection

Connect to the database. To connect to the database you have to add *.jar file with database driver to the application filepath. All the custom settings of the connection may be applied using the connections settings. The given default values are provided for Postgres database. Examples of the keys for connection setings (like "user" or "password") may differ depending on used database.
Parameters:

- **Result varaible name** - String expression - Name of the variable to store the database connection for further operations,
- **Driver** - String expression - Name of the driver (full path to driver class),
- **Protocol** - String expression - Database comunication protocol,
- **Server url** - String expression - Database server URL,
- **Database name** - String expression - Name of the database.

Disconnect database

Close connection to the database.
Parameters:

- **Database connection** - Object expression - The database connection to be closed. It shouldn't be used after this call anymore.

Database operation

Perform database operation (like select, update, truncate or drop).
Parameters:

- **Database connection** - Object expression - Connection to the database,
- **Operation (Command)** - String expression - Full command to be performed on database,
- **Result variable name** - String expression - Name of the variable to store the result - 2d array (the 1d array of 1d arrays).

4.7.9 Java objects support

With Java support you will be able to better control Java applications. To use these functionality you have to attach Lordui to your application as a library (or attach your application as a library to Lordui).

Find Java UI object

On the given Java UI container (like eg Window) a Swing object will be searched. It will be searched after the given pattern - iterating layer after layer starting from the given container.
Parameters:

- **Result variable name** - String expression - Name of the variable, where the result Swing object will be stored,
- **Java UI container** - Object expression - The pointer to Java (Swing) object - the container over that the search should be performed.

4.7.10 OS commands

Operating system commands such as command execution of file copy.

Copy file(s)

Copy the file or the directory (recursive) under the given file path.
Parameters:

- **Source file path** - String expression - Name of source file/directory,
- **Destination file path** - String expression - Name of target file/directory.

Delete file(s)

Delete the file or the directory under the given file path.
Parameters:

- **File path** - String expression - Name of file/directory to delete.

Open document

Open document or any other file data. File will be opened by default web browser or with the program, that is is assigned by default to the choosen file extension
Parameters:

- **File URI** - String expression - File path (local or remote).

Run OS command

Executes the command from operating system command level (like from terminal). To run commands like from 'Run as' prompt on windows, you can type (on example of calculator) ["cmd.exe", "/C", "start", "calc"]
Parameters:

- **Command to execute** - Array expression - The array of command components. The first one should be command name. Next come the arguments. Ie. to run "Java –version" the array should evaluate to: ["java", "–version"],
- **Run in background** - Checkbox - If the player should wait until command termination.

Write to file

Write text tot file
Parameters:

- **Text to write** - String expression - Text to write to file.,
- **File path** - String expression - Path of the file.,
- **Clear before writing** - Boolean expression - If file should be cleared before writing the text (otherwise the text will be appended)..

4.7.11 Socket

Socket connection operations

Close Socket

Closes the given socket. The Socket variable is not to be used after this operation
Parameters:

- **Socket** - String expression - Socket to be closed.

Open Socket

Opens the socket and stores the connection under given variable.
Parameters:

- **Host** - String expression - Host to connect (IP or host address),
- **Port** - String expression - Port to connect to,
- **Variable to store connection** - Variable name - Name of the variable to store the connection to..

Print to socket

Sends the message to given Socket connection. Throws an exception if message not sent.
Parameters:

- **Socket** - String expression - The Socket connection, that the message is going to be sent to.,
- **Message** - String expression - Message to be sent.

Read from socket

Reads the message (string) from socket into given variable.
Parameters:

- **Socket** - String expression - Socket variable name,
- **Name of variable to store message** - Variable name - Name of the variable, where the reveceived message will be stored.

Subprocedures:

- **Timeout operation** - The procedure, that is being executed, when the Timeout exception occurs.

4.7.12 Integration

Different ways to comunicate with Operating System/external applications/databases/hosts/...

Java code

Java code is the user defined java code. It is the overwritten, whole Java class. Every Java code must contain following commands:

```
import ktm.lordui.nativeOperations.dynamicJavaCode.DynamicClassPrototype;

import ktm.lordui.data.Memory;

public class LorduiDynamicClass extends DynamicClassPrototype {
public void run(Memory state) {
}
}
```

To be able to compile the java code, Java JDK 1.6 or newer has to be properly installed. Also the system Path has to be set properly.
Parameters:

- **Java code** - String - The Java language Code.

Read clipboard value

System clipboard is the fragment of memory in operating system, that is shared for shared for all processes - it is also the place, where operating system stores data after hiting *ctrl+c*. The value stored in RAM will be passed to the local process variable - the name of the variable has to be provided.
Parameters:

- **Target variable name** - Variable name - Value will be stored under given variable name.

Save value to clipboard

Puts the value into the system clipboard (like after *ctrl+c* press).
Parameters:

- **Value to store** - Object expression - Object to save to system clipboard (String/image).

Email sending

The sent email can not be anonyous - the email account has to be provided. All the data (SMTP address, server address - see the connection properties. Keywords are same as in the default Java Mail Library) and other email data (from, to, title and body) has to be provided. Many of the internet providers are blocking the possibility of sending a email from java application - so it may sometimes not be possible even if the command is properly executed.
Parameters:

- **From** - String expression - Mail adress of sender,
- **To** - String expression - Mail adress of receiver,
- **Title** - String expression - Title of the mail,
- **Body** - String expression - Body text of the mail.

4.7.13 Sounds

Play sound

The sound will be playd. This sound may be loaded while developing (option *'Play sound from variable'*) or loaded from given url while executing the procedure. The command starts sound playing and ends it's execution immidiatly, without waiting for player to stop. Should not play multiple sounds in one time. Sound stopping will be performed with the other command. Sound may be in mp3, wav and (depends on Java implementation) au snd, aiff and aiffc format.
Parameters:

- **Sound player name** - String - Name of sound player, that will play the sound,
- **Source of sound** - String expression - Source of the file to load - it may be file stored on a disk or a sound variable aswell.

Stop playing sound

Stops currently playd sound.
Parameters:

- **Sound player name** - String - Name of sound player, that should stop playing.

Wait for sound to stop

Command will wait until currently playd sound will end playing.
Parameters:

- **Sound player name** - String - Name of sound player, on that listener should wait.

4.7.14 Sound and Video

Video and sound recording tools

Add frames to video

Adds frames(images) to the video. Sound (if the video supports sound) has to be handled separatly.
Parameters:

- **Video variable** - String expression - Name of the video variable to append the frames,
- **Image** - Image expression - Image to append. Should be same size as set in the video settings.

Close video recorder

Closes the video recorder - after calling this method the video variable is unusable any more and movie will be ready.
Parameters:

- **Recorder object name** - String expression - Name of the recorder object variable.

Init video recorder

Initializes the video recorder object.
Parameters:

- **Recorder object name** - String expression - Name of the recorder object variable,
- **Target file** - String expression - Target file (*.mov) of the video,
- **Video frame size** - Point expression - Size of the frames (width, height) - point objects/format is accepted,
- **Frame rate** - Integer expression - Frame rate,
- **Audio settings** - Audio settings - Settings of the audio (*.wav format).

Start recording video

Starts recording the desktop video. The recorder object has to be already initialized.
Parameters:

- **Recorder object name** - String expression - Name of the recorder object variable,
- **RecordArea** - Point/area expression - Left-top corner or area of the recorded fragment of screen to record,
- **Sound input source** - String expression - Name of the sound input source. Shoose "Select input" to see input sources available currently on your computer,
- **Run image process procedure** - Boolean expression - If the image process procedure is to be executed (see image processor subprocedure).

Subprocedures:

- **ImageProcessor** - If "Run image process procedure" is selected, the procedure will be executed for every single frame. Frames will be available in the procedure under "image" variable by default.

Stop recording video

Stop recording the desktop video. The video will not yet be ready after calling it - call "Close" object to finalize the movie.
Parameters:

- **Recorder object name** - String expression - Name of the recorder object variable.

4.7.15 Lordui objects creation

The set of commands for dynamic modification/creation of the Lordui procedures. Create objects using XML format (You may see XML definitions of the objects by exporting the project as XML or copying the fragment of the project into system clipboard and pasting it into notepad)

Creating command

Adds new command to procedure. Throw an exception, if the given procedure doesn't exist.
Parameters:

- **Procedure name** - String expression - Full name (including path) of the new procedure,
- **XML definition** - String expression - The new command's definition in XML. You may see XML definitions of the objects by exporting the project as XML or copying the fragment of the project into system clipboard and pasting it into notepad,
- **Position** - Integer expression - Position number (starting from zero), where to insert the new command. For -1 appends the command on the end of the procedure.

Create procedure

Creates a new, empty Lordui Procedure. Throws an exception if the procedure already exists.
Parameters:

- **Procedure name** - String expression - Full name (including path) of the new procedure,
- **Result type** - String expression - Leave empty, if procedure is not supposed to return anything. Possible options are:
  - Point
  - PointList
  - String
  - Integer
  - Image
  - PositionedImage
  - Boolean
  - Rectangle
  - MusicPlayer
  - Other
  - Array
  - Collection
  - Sound
  - Expression
  - Color
  - ScreenBlockade
  - Null

- JavaUIObject
- Long
- Map

,

- **Arguments** - Dynamic list of arguments - Arguments exposed by procedure. The list of possible types is same as in case of result types.

Remove command

Removes procedure's command. If procedure doesn't exist, an exception will be thrown. If procedure doesn't contain data element on given position, nothing happens.
Parameters:

- **Procedure name** - String expression - Full name (including path) of the new procedure,
- **Command number** - Integer expression - The number of command to remove - starting from 0 till number of commands - 1.

Remove procedure

Removes the procedure of the given path (including package names). If procedure doesn't exist, an exception will be thrown.
Parameters:

- **Procedure name** - String expression - Full name (including path) of the new procedure.

4.7.16 Lordui meta elements

With Lordui meta elements you may control players.

Close Lordui

All the resources will be released. The program will be closed.

Pause

Pauses the execution of procedures. For more information about running procedure see section 4.3.1: "Running the procedure"

Stop all

Stop all the executed procedures. The command does exactly the same, as if the *Scroll lock* button would be pressed.

Stop listeners

Stops all the listeners that are running threads on the given player. On example of Keyboard listener: efter running this command on player that runs the threads listener will be stoped, but the player will complete the execution of already started threads.
Parameters:

- **Player name** - String expression - Name of the player that listeners are to be stopped.

Stop Player

Will stop the player and all the threads running on it.
Parameters:

- **Player name** - String expression - Name of the player to stop.

4.7.17 Standard syntax

All the classic programming language operations like if condition, loop operation, sleeping of variable assigment.

Add Lordui Listener

Creates a new listener on Lordui Procedure state. The listener will execute given procedure, when procedures will be stopped.
Parameters:

- **Listener name** - String expression - Name of the player, to play the procedure, when event occurs,
- **Run once** - Boolean expression - If listener should be stopped on first event execution,
- **Traced Player** - String expression - The player, that will be traced for stop signal.

Subprocedures:

- **Procedure on Stop** - The procedure, that is being executed, when the Stop event occurs.

Add to array

Stores the value into array.
Parameters:

- **Variable name** - Variable name - Name of the variable of the array. If there is no such variable, it will be created.,
- **Insertion type** - Options - There are few ways to store value into array:

  1. set value (no shift will be performed). If there is any object stored under given index it will be substituted,
  2. push value (all elements starting from the given index will be pushed single step right),
  3. append - this is the Push to the end syntax shortcut.

  ,
- **Index** - Integer expression - Index where the value should be stored (not used for Append insertion type),
- **Inserted value** - Object expression - Value to insert.

Procedure call

New subprocedure will be called.
Parameters:

- **Called procedure name** - String - The full name of the procedure to call i.e.: *package1:pacakge2:procedure_Name*,
- **Number of procedure executions** - Integer expression - How many times the procedure will be executed,
- **Run in new Thread** - Checkbox - if procedure should be called in separate process (next step after procedure may, and probably will, be executed before this command ends it's execution. There will be separated process created,
- **Procedure arguments** - Dynamic expressions - Every argument expression has to be filled in. Procedure arguments number and types are defined in procedure definition header.

Procedure dynamic call

The subprocedure with the given name will be called. You must provide the procedure name and the list of procedure arguments. Both fields are dynamic calculated during runtime. You may run the procedure in new thread (note, that only one thread may be active). If procedure will be executed in new thread, it will run when current thread will end or sleep with 'enable thread change' option.
Parameters:

- **Called procedure name** - String expression - the full name of the procedure to call i.e.: *"package1:pacakge2:procedure_Name"*,
- **Procedure arguments** - Array expression - The array of the arguments to call the procedure. Number of arguments and types have to match,
- **Run in new Thread** - Checkbox - If procedure should be called in separate process (next step after procedure may, and probably will, be executed before this command ends it's execution. There will be separated process created.

Close Lordui

All the resources will be released. The program will be closed.

If condition

The if condition works just like in every programming language. If the condition evaluates to true, the subprocedure *"on true"* will be called, otherwise *"on false"* subprocedure executes. These subprocedurs are defined in procedure window under the if condition command. The condition has to be filled in. I.e.: *val1=2 // val2 = "aa" // true*
Parameters:

- **Expression** - Boolean expression - The if expression.

Subprocedures:

- **OnTrue** - Executed if the expression evaluates to true value,
- **OnFalse** - Executed if the expression evaluates to false value.

While loop

While loop works just like in every standard programming language. The condition is evaluated to boolean expression. The subprocedure *"Loop body"* is executed every time, the expression evaluated to *true* value. The while command will stop running after the expression will evaluate to false. These subprocedurs are defined in procedure window under the if condition command. The condition has to be filled in. I.e.: *val1=2 // val2 = "aa" // true*
Parameters:

- **While loop expression** - Boolean expression - the if expression.

Subprocedures:

- **LoopBody** - Executed while the expression evaluates to true value.

Remove Lordui stop listener

Removes Lorui stop listener. The listener won't listen to Lordui Stop events anymore.
Parameters:

- **Listener name** - String expression - Name of the listener to stop.

Return

The command breaks the current procedure and returns the provided value. The returned value should be of the type defined in the procedure header (procedure should be the function)
Parameters:

- **Returned value** - Object expression - Value to be returned by function.

Sleep

Sleep/wait. Player will hang the execution of procedure for given time.
Parameters:

- **Sleep length** - Integer expression - The length of sleeping in miliseconds,
- **Allow thread change** - Checkbox - The ability of thread change - if during the sleep time of procedure, other thread wants to execute, if it should be allowed.

Try ... catch

Exceptions handling
Parameters:

- **Exception pattern** - String expression - Names of the patterns to catch..

Subprocedures:

- **Body** - Code to run and handle the exceptions when occur,
- **Exception handling** - The code executed on exception.

Variable assigment

Assigning the variable (or deleting it, if expression is empty) will assign the evaluated value to the variable of the given name. **It is strongly recommended to always assign to single variable values of the same type.**
Parameters:

- **Variable name** - Variable name - Name of the variable to assign the object. If variable exists it will be overwritted. While deleting, will delete only global variable or only local variable. If you want to remove both, you have to use two ”set variable” data elements,
- **Expression** - Object expression - Value to be assigned. If field is empty, variable will be removed..

Load variables

You may load the variables from XML file during runtime.
Parameters:

- **File path** - String expression - Name of the file to load.

Log

Add panalty logging entry to the log.
Parameters:

- **Logged value** - String expression - Text to write to the log.

Save variables

You may save the variables from runtime level. The variables will be stored in XML format. The stored XML variables may be later opened also as a LordUI project (however it will contain constants only).
Parameters:

- **File path** - String expression - Name of the file to save to,
- **Variable name** - Dynamic Variable names - Names of the variables to store.

4.7.18 Extensions

With extensions you are able to implement your own commands. Extensions may be easly implmented in java and appended during runtime. With extensions lordui may also comunicate with external systems or used as the library in a project.

4.8 Creating macro procedures

LordUI also enables creating clasic screen macros. Click  . There are some options available for recording a macro. You may choose, if mouse moves will be saved (you may save none, threat all moves between clicks as single move, or save each mouse move signal as separated one) and if mouse drags will be saved. Between each recorded operation you may insert a constant sleeping operation. There are also some shortcuts to create screenshots (or save fragments of screenshots), or pick the points on screen. You may also switch to small view, where only stop, record and pause options are available. To record a macro simply select the record button. Note, that mouse actions performed on the LordUI recording window are not recorded. You may pause or stop the recording. After stopping it, new procedure will appear (it will be named Proc with the number of procedure suffix)

4.9 Preparing image variables

There is a tool, to compare image variables with the screen view. To check if the variable differs from the screen view subimage, enter constants list, choose the image constant on the left and the press  button. You will see the window with the screenshot, and the image pattern on the left top corner. You may drag this pattern with mouse or keyboard arrows. Using the icons in toolbar (see tool tips for more information about their functionality) you may see how the pattern differs from the choosen screen

shot's subimage, you may remove the pixels, that differ. You may also make the pattern partialy transparent with the scrollbar.

4.10 Lordui native part

Lordui is written in Java. Thanks to that, it's engine works on many platforms of operating system. However because of programs functionality, there are many commands specific for operating system. Because of that, while creating projects in Lordui, the developer has to remember about all possible problems, that may occur while switching to other computer/operating system. This section presents, what may go wrong.

4.10.1 Text bluring

The OCR mechanism and text pattern searching is implemented in pure java to work 100% efficient. However, because no heuristic is used, these mechanisms are not ready for any noices nor text bluring. Unfortunately some operating systems (like Microsoft Windows) do use text blooring in order (in theory) to improve quality of text presentation. In some operating systems (like Microsoft Windows Vista and newers) text blooring is by default turned on. Microsoft calls the bluring option *'ClearType'*. While creating the projects, you should take care abour blooring because of two reasons:

1. text blooring makes OCR or text pattern searching not working,
2. the image patterns defined on computer with *'cleartype'* turned on may be not found on the computer, where this option is turned off (and vice verse) or has other settings of this functionality.

That is why **it is strongly recommended to turn the** *'Cleartype'* **off, or take the special care, it turning off is not possible.**

4.10.2 Native operations

Some native operations - like system operation events or external (from Lordui point of view) application windows support are being implemented with native commands. Author of Lordui successfully executed this functionality under:

- Microsoft Windows XP service pack 3, 32 bit version
- Microsoft Windows 7, 64 bit version

The Author will be very pleased for any information, about successfull or not successfull execution of these commands on any other operating systems and about every occuring problem aswell. The important information are: name and version of operating system and version of Java.

4.10.3 Window transparency

Lordui uses window transparency in some places. All modern operating systems from Microsoft Windows family do support window transparency. However Author of Lordui had some problems to create transparent windows on some of the versions of Linux family operating systems. In such case working with Lordui could be very hard and it is not recommended.

4.11 Lordui Modules

There is a set of Modules, that can be installed/uninstalled from Lordui. Modules are being managed by the Programs Author and only the Author my expose new Module on official Lordui Module Server. However there is always possibility of creating own, custom module. The aim of separating out Modules from Lordui is to enable turning off the Lordui's functionality, that may be not needed for big group of users.
You can manage modules in Module Manager. Module Manager is available under Menu 'Help' -> 'Modules'. You can browse, install or uninstall modules from the server or local file. By default, the official Lordui Modules Server is used, however you can change that using settings under ⚙ button. To install the module select it on the modules list on the left side and hit Install button on the top side. You can update all the modules, that have the update available on the server, using 'Update modules' button on the bottom side.

4.11.1 Remote Java Module

With Remote Java plugin you will connect Lordui to separate Java Virtual Machines. Therefore you will be able to control the Java Applications not only by standard Lordui tools like screen or system buffer but also using object references.

Connecting Remote Java Client

After installing Remote Java module, Lordui will handle as Remote Java server. There can be only one server running on the single operation system. Any next Server will not start (the application will simply start without the Remote Java server). The server uses Java RMI mechanism on Port 7231.
Any Java application will be connected as a client. To connect Java application you have to perform two steps (where the first step is not always necessery - see next steps).

1. add Remote Java Client jar to the classpath,
2. start the client.

To have the client included in Remote Java application classpath, please download it from www.lordui.com site. There are two recommended ways of including the client jar in classpath:

- adding it into application classpath like an additional library,
- adding it to virtual machine classpath, so every Java Application will have the client Jar file loaded.

When having Remote Java client code loaded into Application, the last step is to start the client, so it will connect to the server. There are two ways of doing that. First one is just to include command:

```
LorduiRMIClient.execute();
```

directly in your code.
The second one is more tricky, but doesn't need first step to be performed. We will run the client, that will start the connection and execute the Application after that. The format is to run all the jars (including Remote Java client) in classpath, starting with class:
modules.ktm.autoclicker.remoteJava.client.JavaRemoteClient
The arguments will be: "main_class", "name.of.your.applications.class", "arguments..." As an example, let's assume we have got the application that we execute using command like: java -jar my_app.jar "arg1". From my_app.jar, looking into the manifest file, we know, that the application start with the com.application.Main class to run the main(String[]) method. Let's start the application. The command could be eg:

```
java -cp "my_app.jar;RemoteJavaClient.jar"
modules.ktm.autoclicker.remoteJava.client.JavaRemoteClient
main_class com.application.Main arg1
```

. This command will start Remote Java Client and execute the Java application just like user would do that normally. The tray icon will inform you, that the client has started.

Remote Java Console

In Lordui Application, after installing Remote Java Module, the new Icon will appear on main Window toolbar on the right side: . Click it to open Remote Java Console.
The Clients tab will show you the list of all connected Clients. With the option 'Run client locally' you can make Lordui application to have to Remote Java client turned on too. This option is turned off (unchecked) on every application launch by default. Under Inspect tab you may inspect the GUI objects from your remote application. Please select 'Select remote object' button. Move you mouse cursor over the object, you want to inspect. The

window should get dark. Click the object. In the console window you will see set of object properties. You can copy any of the values you want - like you can save the icons using the right click on it. With the Console you can calculate the formula, to get the GUI object from you application's window. To get it, select some properties, that make you object unique. Then click 'Calculate formula'. You will see the information how many object on your window have been found, when applying these filters. You will also notice, that the formula will appear over the clicked button. You can copy it using the 'copy to clipboard' button on the right side. On 'Stack trace' tab you can check what is the trace, starting from the Window up to your chosen GUI object.
The console is also available under 'Call remote Java Method' Data Element's properties, where you can fill in the parent objects using the Lordui's GUI.