TS1 Graphical User Interface

Issued / changed	22.04.2013	Released / checked	22.04.2013	Signed	MC - Number	Page / Page (s)		
Document – Number						1	/	35

Table of Contents

1 Introduction

This document describes common concepts and UI paradigms used in TS1 software. It should not be consumed as specification but rather as guideline. The intention is not to fix the UI design to pre-defined pages and dialogs but to provide examples of the dialogs and pages in hope it helps to produce simliarly looking applications. Enforcing the designs will increase consistency between applications but reduces developers ability to make the best application for a specific machine at hand. **A word of warning though: the freedom to change the proposed designs comes with responsibilty to keep the applications consistent with reference designs** – if need arises it will easy to remove this chapter from the document and make the proposed designs mandatory for TS1. At present the document is written in good will and hope that it will spark the collaboration efforts between IPTE offices.

The Graphical User Interface (GUI) is only a part of the whole UI. The best dialog with operator is achieved with combination of the GUI and hardware HMI, a well placed physical button is much more convinient to use than a button in user interface. For instance, a physical "unload" button that is placed at the end of flatbelt conveyor, allowing to fetch all products on the conveyor, is much more accessible than corresponding button in GUI. The software button requires basic knowledge about computers in general and the machine software; the operator must go to computer (might be few meters away), activate the correct page in UI (requires mouse movement), find the right button (cognitive operation) and press it (more mouse movements). With well placed and labeled hardware button it's much more convinient.

Likewise it is much easier to move robot with a good teach pendant equipped with a handwheel than to jog using UI buttons. The tangible user interface of the handwheel enables much preciser control over robot axes than the UI button, especially because the UI buttons have a small lag.

The application graphical user interface (GUI) consists of *control bar*, *windows* and *pages* (Figure 1). In the context of this document a *window* is defined as group of *pages* that implement similar functionality as seen from operator perspective. The intention is to allow operator to perform an action on a machine from one window. **TS1 GUI shows how GCL (GUI Control Library) is used in different applications with one common philosophy of "*Observe*", "*Configure*", "*Manipulate*" and "*Analyze*" windows:**

- The **observe** window provides feedback to operator about what is going on in the machine, it allows to monitor the working process.
- The **configure** window enables operator to change settings, to teach new products or to calibrate the machine.
- The **manipulate** window is the service interface. It helps to diagnose and fix hardware problems by allowing direct hardware control.
- The **analyze** window is for statistics.

The *control bar* allows starting, stopping, pausing and resetting the machine. In addition there are indicators that reflect the status of service key and will notify if there is an active error or warning message on messages panel. The control bar resides at the bottom of the application window, its design borrows from UI paradigms used by windows image viewer and windows media player. Reusing the existing UI paradigms (buttons, dialogs, wizards, etc.) is important because it reduces the learning curve, especially for the operators who have had limited exposure to computing revolution.

Finally there are log-in, help and about buttons at the top right end of the tab bar. The log-in button will open a dialog that enables to log in as well as to manage users. The help button will open the machine manual and the about button will open a dialog that displays application version.

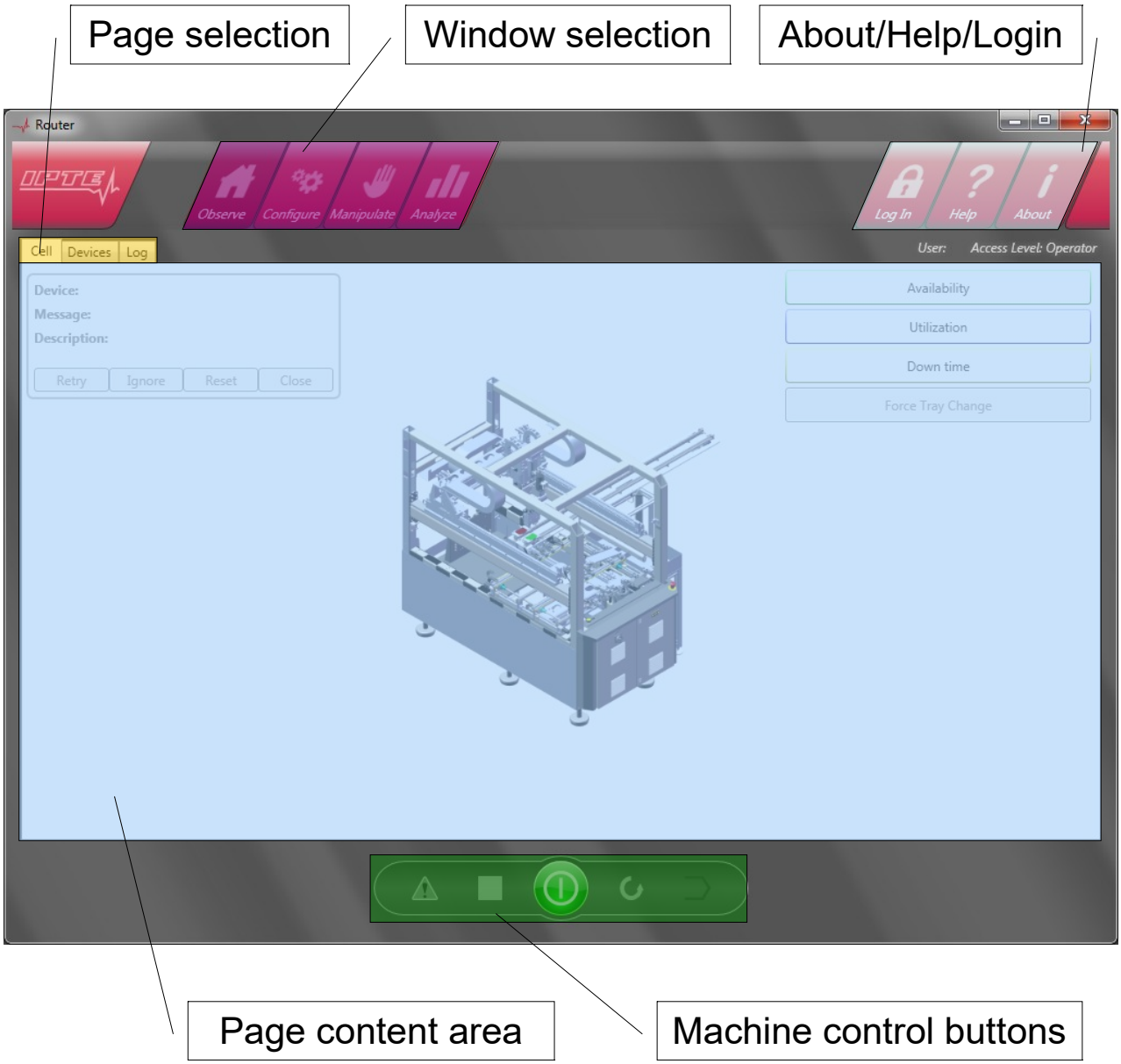
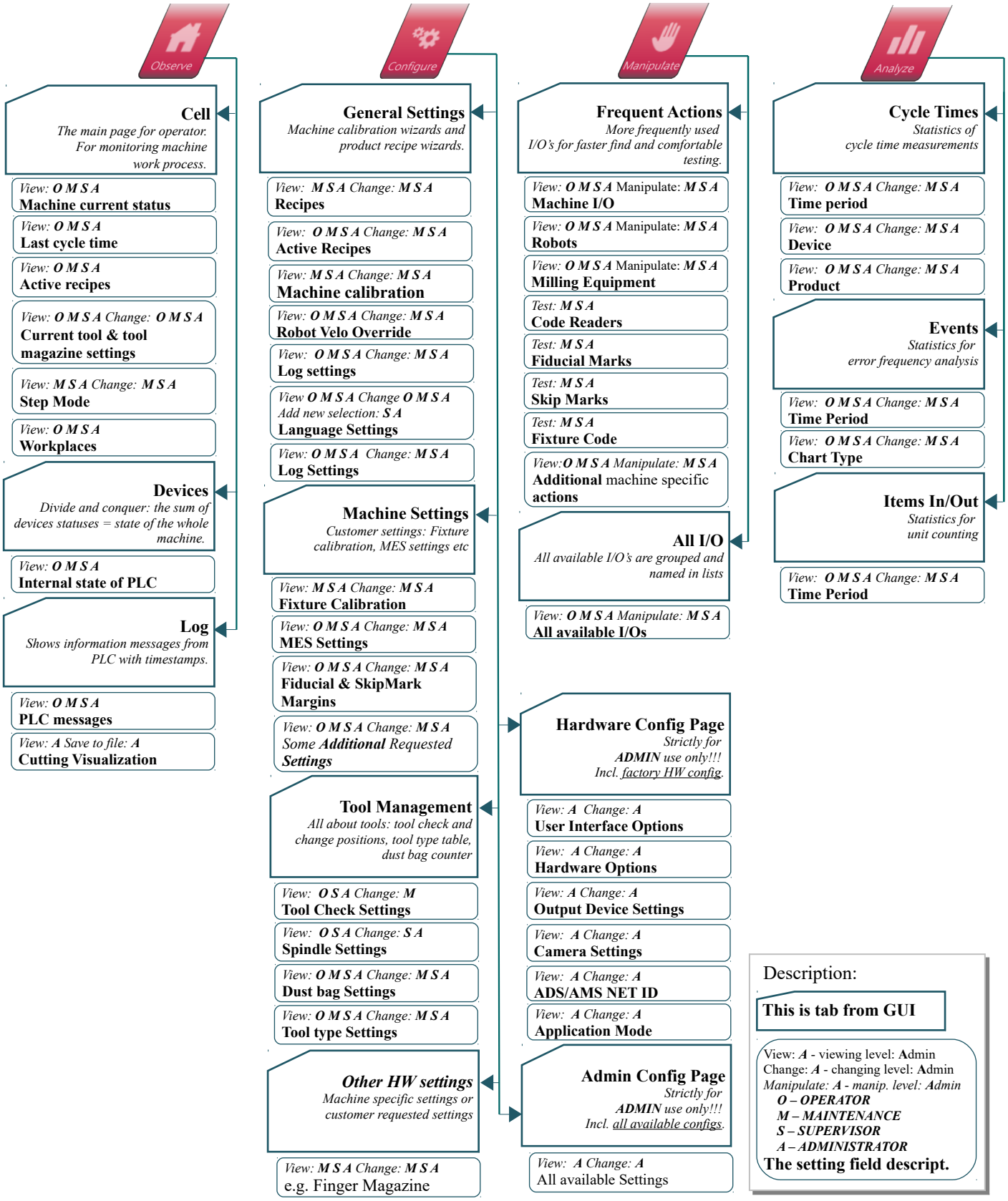


Figure 1: Application structure

TS1 GUI general look and feel are similar in different machines so that user can orientate faster, work more confidently and productively. Keeping similar structure in the user interface is a great solution for that. The following image brings out the principle of dividing settings and configuration options between the basic four main windows named Observe, Configure, Manipulate and Analyze. In addition, it is important to notice that each application setting view and changing possibility depends on the user level. Therefore, the following image also points out the required user levels respectively for each setting with a letter: O for operator, M for Maintenance, S for Supervisor and A for Administrator.



The user interface applications are typically tailored for a machine family (easy router, flex marker), it is uncommon to share the same application between different machine families. The reasoning is that by building the UI for a machine family we can provide the best user experience to end user and to reduce the development time. There are no ad hoc obstructions that would prohibit building an universal platform on top of controls provided by TS1, but doing it is strongly discouraged.

It is easy to see how the tailored UI improves the operator experience but the source of reduction in development time may be non-obvious. The single biggest factor that affects the development time is the reduced requirement on backward compatibility, since the application is designed with single purpose, it is much easier to change the pages and dialogs without worrying about breaking the program. It might not be a factor, if the application has to support only two different machine families, but it will be, when the application has to support ten machine families and there are custom requests for each of the machines.

All the pages and dialogs in TS1 applications are modular and self contained. The typically only communicate with a controller object for configuration and hardware access. This will make it easy to copy-paste them from one application to other with only minor changes needed. There are alternatives to the copy-paste approach, such as moving the common pages to libraries, but copy-paste is the simplest approach and in TS1 the simplicity is the paramount.

2 Observe Window

The *observe* group of pages reflect the status of the machine. These are the pages the operator will turn to when machine will stop delivering products. They provide some basic utilities for diagnostics (log page, devices page) and error recovery (cell page).

2.1 Cell Page

The cell page (Figure 2) is the main page of the application, it is the most used page over the life time of the machine. The page should reflect the state of the machine in generic terms: state of the machine, some basic statistics, etc. Most importantly the page should tell what is wrong in case the machine stops working.

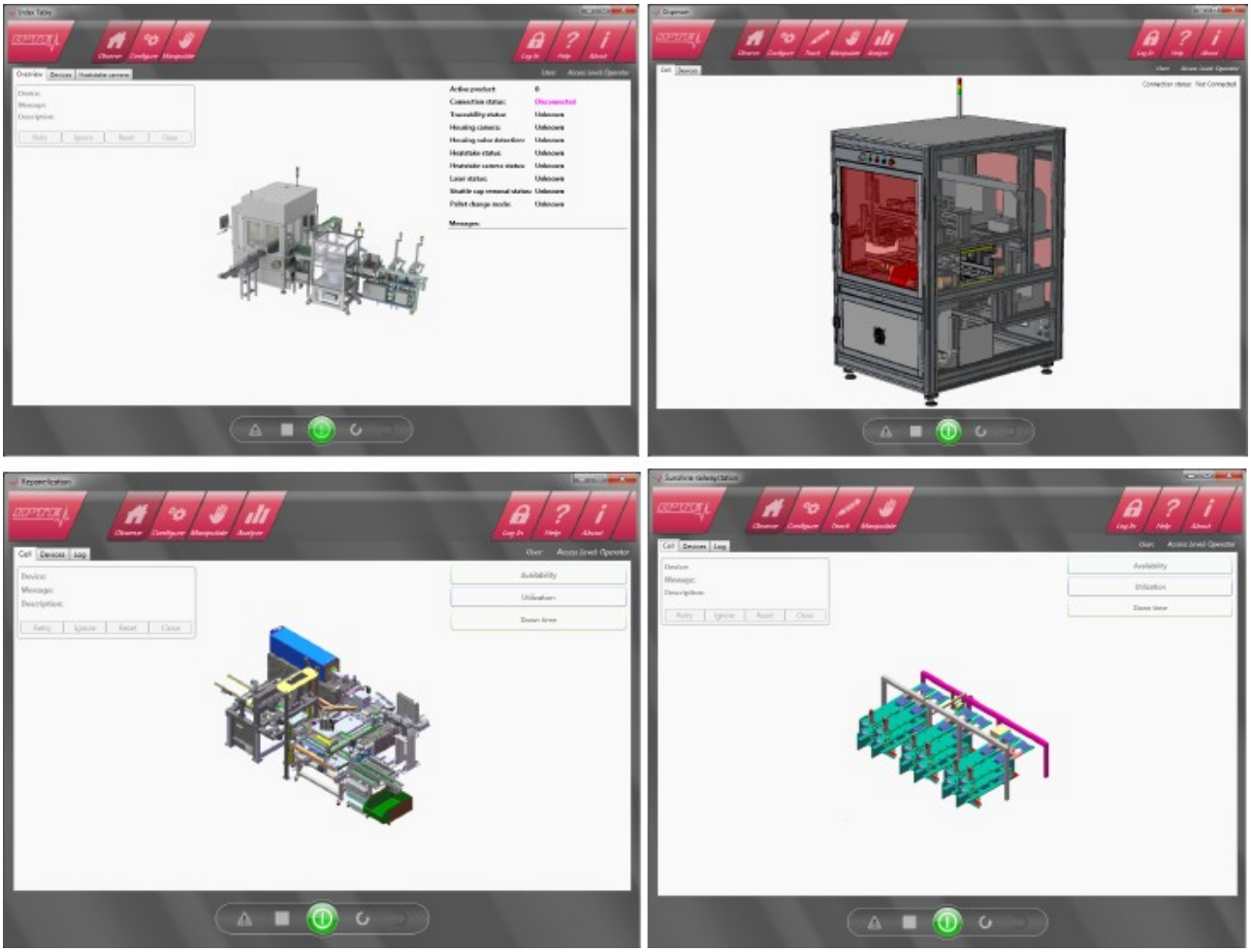


Figure 2: Cell page samples

The cell page also serves as the welcome page to the application, and thus is has the double duty of conveying the message of user friendliness and simplicity to the operator. The importance of the welcome page must not be under estimated, it is what makes the first impression and what puts the operator in right mood. **If user likes the application then he's willing to forgive even some rather significant bugs, but**

when he doesn't he will nitpick on every miscolored pixel on the screen. No user interface design can help out if there are critical show stopping bugs, but having a great user interface goes a long way towards creating the willingness to put up with glitches. The simplicity of application is, more often than not, an illusion. Having uncluttered and simple (with pictures) welcome page is a good place to start with it.

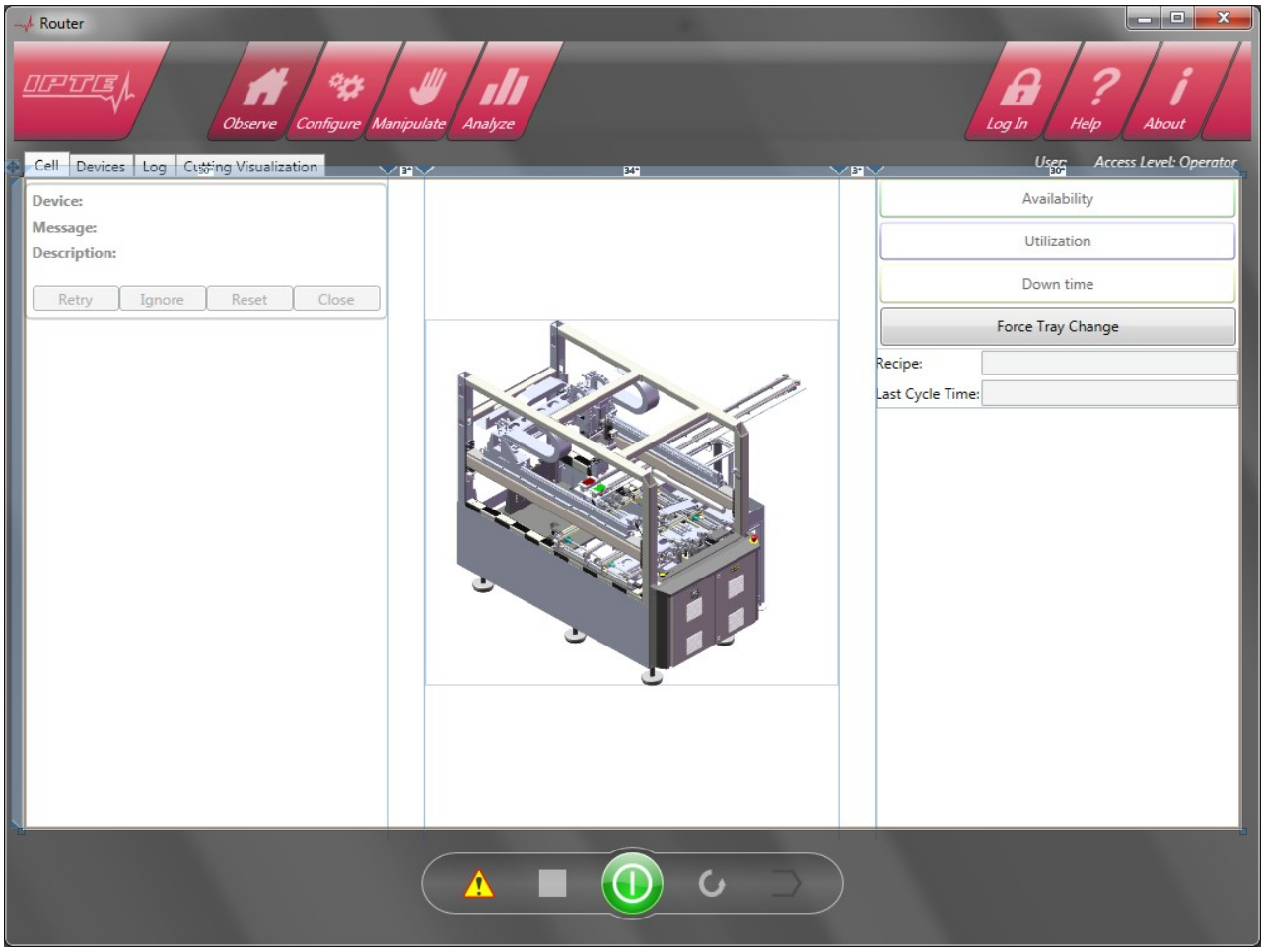


Figure 3: Cell page layout.

The layout of cell page consists of five columns (Figure 3): three columns for content and two empty columns for margins. The column widths are relative to window content area width; their proportions are as follows: 30%, 3%, 34%, 3% and 30%. Only the first, third and fift column will contain any content, the second and fourth columns are for adding some air between the elements -it will make the content more easily readable.

Since the real estate of the cell page is limited it takes a lot of consideration what should go there and what should not. It is common convention in TS1 applications that the left content column contains error panel, the middle content column contains cell picture (that can be animated to help error localization) and the right content column contains custom data. The primary indicator for machine state (started/stopped) is the physical beacon, but the control bar at the bottom of the page will give sufficient data about it so there's no need to add explicit state indicator to cell page.

Figure 4: Error panel

The error panel is essentially a column of stacked message panels (Figure 1). There can be more than one active message, but in case threre are no active messages a gray placeholder is displayed instead (Figure 3). The messages fall into one of three categories: warnings, errors and alarms. The warning messages are displayed with yellow background, errors and alarms with red background. For each message there are three information fields (*device*, *message* and *description*) and four buttons (*retry*, *ignore*, *reset* and *close*). The *device* field indicates the source of error or warning, *message* field is short description of the problem and the description field contains longer explanation about what went wrong and how can it be fixed. The buttons can be different for each message; for warnings it is sufficient to enable the *close* button, but the error messages are typically resolved with combination of *retry*, *ignore* and *reset* buttons.

The image in the center of the home screen is just a decorative item. Having a large image occupying a big chunk of home screen real estate is wasteful, to get more functionality out of the image we have added extra layers that indicate the source of warning and error messages (Figure 5).

Error panel in cooperation with animated central cell drawing will help operator to pinpoint error location and to respond to malfunction accordingly.

The final content column in the cell page is reserved for custom information about process. Typically it contains some basic statistics such as availability and cycle time, name of the active recipe, etc. Often the

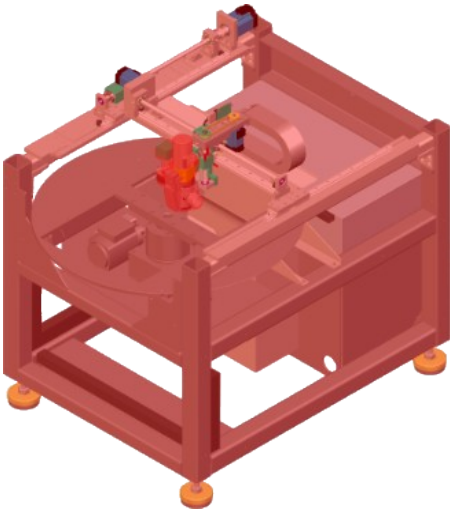


Figure 5: Animated cell drawing. In case of ESTOP the whole cell will pulse in red.

customers want to see extra information about some aspects of the production or factory communication; this is the place to put it.

Warning: implementation details! The error panel is typically connected to the TS1 "PLC" through events defined in IPC-2541 standard. When an IpcDevice has an error condition it will raise *IpcEquipmentError* event that is routed through controller (Figure 6). The cell page will hook itself to this event as handler, when the event is fired the page will show the message panel. The information to display and buttons available for each message panel are described in the event arguments. Similarly, warning messages are received through *IpcEquipmentWarning* and alarms through *IpcEquipmentAlarm* event.

When a button is pressed on a message panel the CellPage event handler will clear the sender device error/warning through custom interface (not defined in IPC standards), but does not clear the message panel. When the error is cleared on the device it will emit the *IpcEquipmentErrorCleared* event which is handled by the cell page, this event will remove the message panel from the screen. It is possible for a device to pre-emptively (without operator reaction) to remove the message panel by emitting the *IpcEquipmentErrorCleared* event. Same applies for warnings and alarms.

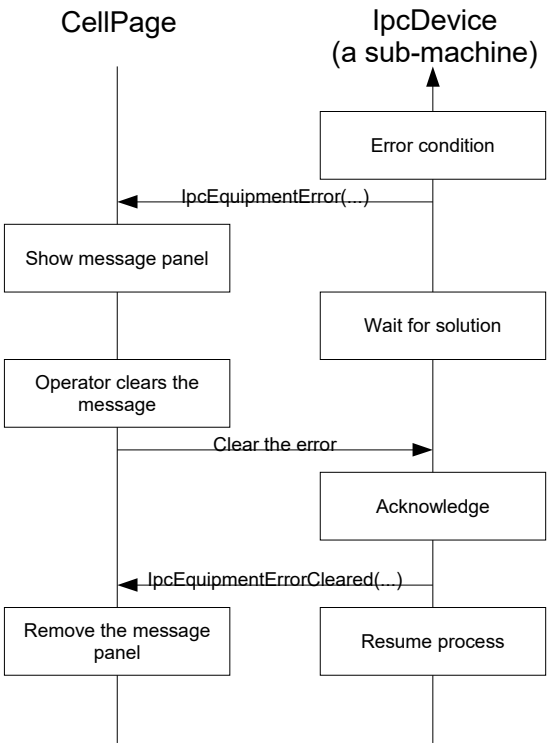


Figure 6: Error handling

The mechanism is widely used in TS1 platform, as such it has been improved for easier use.

- On the TS1 "PLC" side there are special functions, implemented in library, that take care of this procedure: Warning(...), Error(...) and Alarm(...).
- If the same device emits multiple warning messages with identical UUID, only one message panel is displayed on screen. This makes implementation of the "PLC" simpler.
- The warning messages will not be cleared in PLC. The PLC may emit warning and continue processing, it does not have to wait for clearance.
- The common log page implementation (see chapter 2.3) will automatically log error and alarm messages. No need for duplicate logging.

2.2 Devices Page

Warning: implementation details! The devices page (Figure 7, 8) will reflect internal state of the PLC in machines with "divide and conquer" style architecture. The architecture implies that a complex machine with multiple asynchronous processes are divided into simpler sub-machines (or *devices*) which may communicate with each other through SMEMA-like interface. In this case each of the machines has it's own internal state (off, down, setup, active, etc), each of them may be in error condition, etc. The state of the whole machine is then (to be compatible with IPC-2541 standard) the highest state of the sub-machines. The devices page will help to diagnose the status of individual sub-machines and to monitor the internals of the machine. It can be trivial (Figure 7) in machines containing only couple of sub-machines or complex (Figure 8) in machine that contains tens of sub-machines.

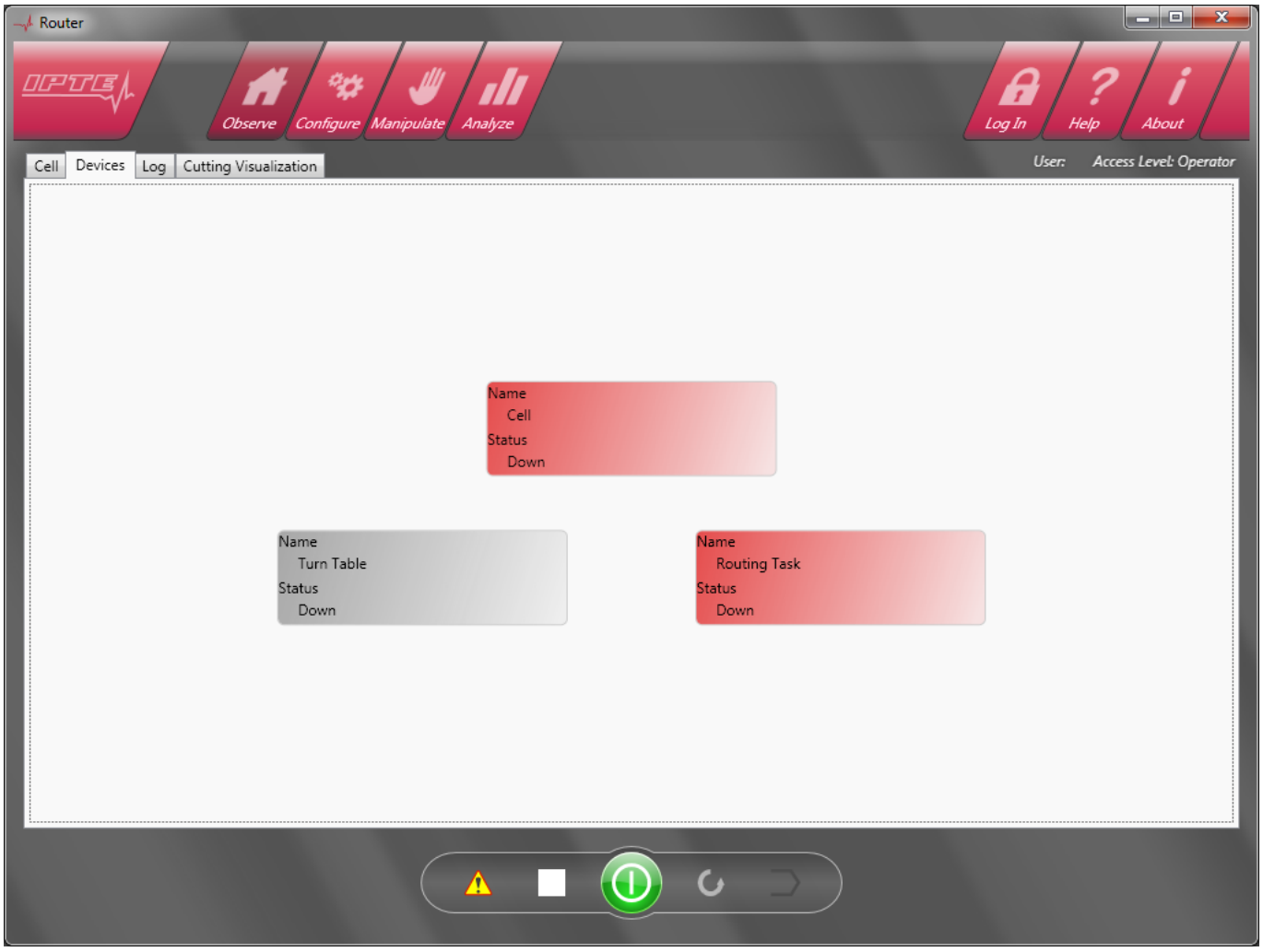


Figure 7: A trivial devices page for a machine with only two autonomous devices

In normal production the all the sub-machines are in one of the active state (*starved*, *blocked*, *active*, *executing*) or in *down* state if there's an error. The basic state monitoring will already give a lot of information about the cell and help to localize problems if production stops:

- If all the devices are in *off* state then the whole machine has been stopped.
- If only one of the devices is in *off* state it may indicate there's a software problem with this device. All unhandled exceptions in devies will shut down the device and show an error message on cell page.
- If all the devices are in *down* state it probably means that the machine is paused. NB! Switching to service mode will automatically pause the machine at first opportunity.
- If only one of the devices is in down state it will probably mean that there's an error in that device, check cell page to see details.
- If the input device (e.g. conveyor) is in *starved* state then the machine is ready to accept a new product, it indicates that there may be a problem with one of the previous machines in line.
- If the output device (e.g. tray loader) is in blocked state then the machine is ready to unload a product, it indicates that there may be a problem with one of the next machines in the line.
- If there are a series of devices, and one of the device is in *active/execution* state for a long time while previous device is in *blocked* state and next device is in *starved* state, it indicates that the one device in *active/executing* state is the bottleneck.
- If there is an empty device that does not import a product, even when previous device is offering one, it may indicate a software problem. Of course it can always be because the device is waiting for enabling signal from MES, waiting for operator action, etc.

The exact design and contents of this page is not well defined, it varies wildly from machine to machine. Often, when the machine that has been divided into sub-machines, the layout resembles of a production line that has been built from those sub-machines. In that case it is possible to quickly localize the sub-machine that has broken down or the sub-machine that is the bottleneck.

The contents of this page is geared towards developers and IPTE service personal, rather than the operators of the machine. Having a separate high-level diagnostics page will make the initial ramp-up and testing of the machine easier, especially when there are multiple sub-systems and factory communication involved.

Figure 8: A sample of complex devices page for 50+ autonomous devices. Each conveyor segment on this illustration has autonomous thread operating it.

2.3 Log Page

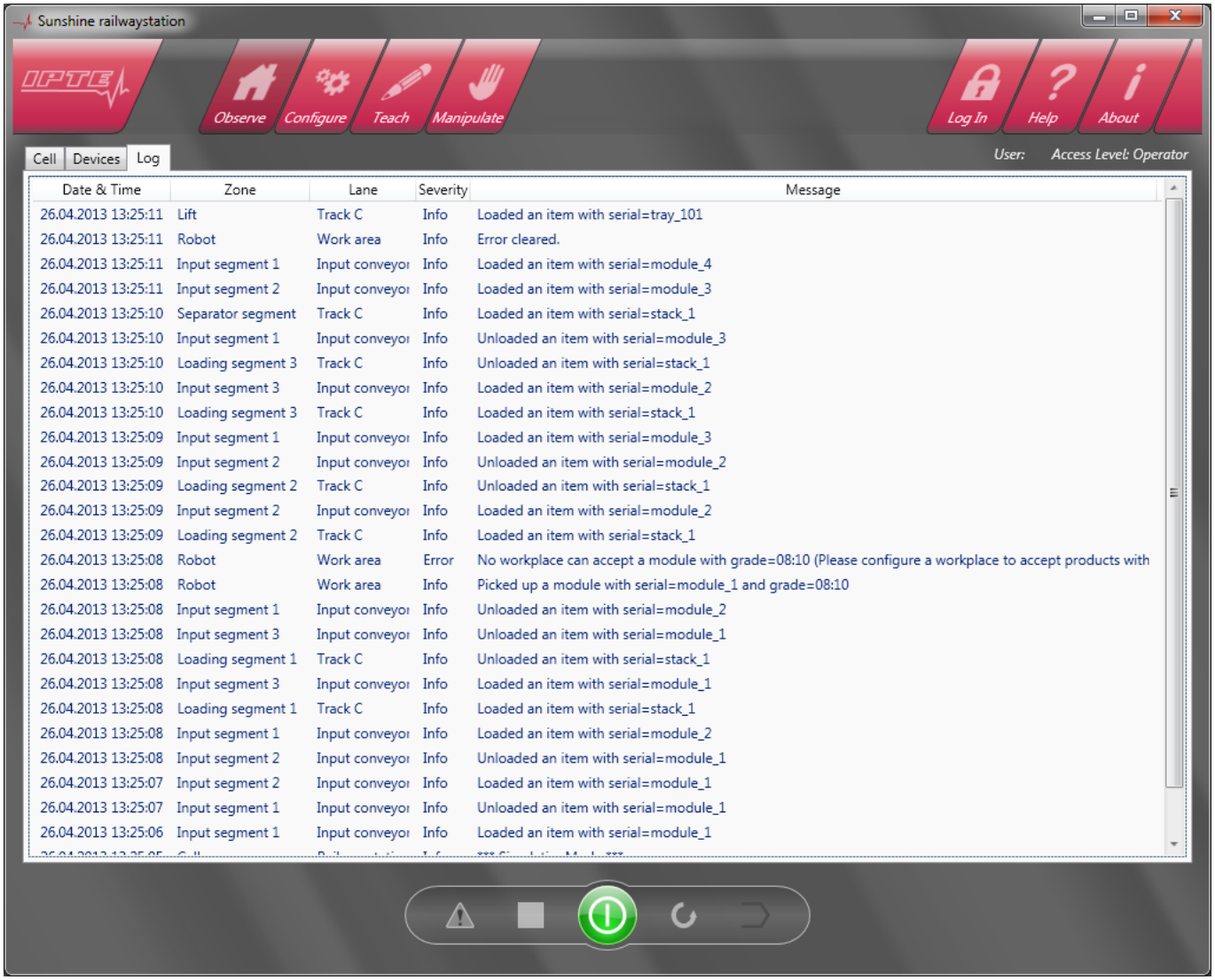


Figure 9: Log Page

The purpose of the log page (Figure 9) is self explanatory, it shows the information messages from PLC. The page only shows the laste 1000 messages to reduce computing overhead. The log files will keep the data for longer. The application should keep the logs for 100 days by default and automatically delete older log files to avoid filling up the hard drive.

Warning: implementation details! The log page hooks to Controller *IpcEquipmentInformation* event for log messages. In addition it will hook to Controller *IpcEquipmentError* and *IpcEquipmentAlarm* events for logging error and alarm messages.

3 Configure Window

The *configure* group of pages provides access to various settings. The configuration pages are customized for different machine types, and as such they vary wildly, but there are common aspects that are shared between most of the applications.

In general the settings are divided into two groups: *machine settings* and *recipe settings*. The recipe settings contain the information that is required for processing a recipe; if one machine can process multiple products, then all the settings that differ between the products are gathered into recipes. Recipe teaching is covered later in chapter Error: Reference source not found, this chapter focuses on machine settings.

* pages, dialogs and wizards

* wizards when multi step process – before usage of camera need to calibrate camera.

* similar settings into a group box

* most often used UI convention.

    * Recipe manager in general settings page

    * The recipe manager monitors recipes folder, allows copy/paste/rename recipes

    * Coordinate systems are calibrated with a wizard

3.1 General Settings Page

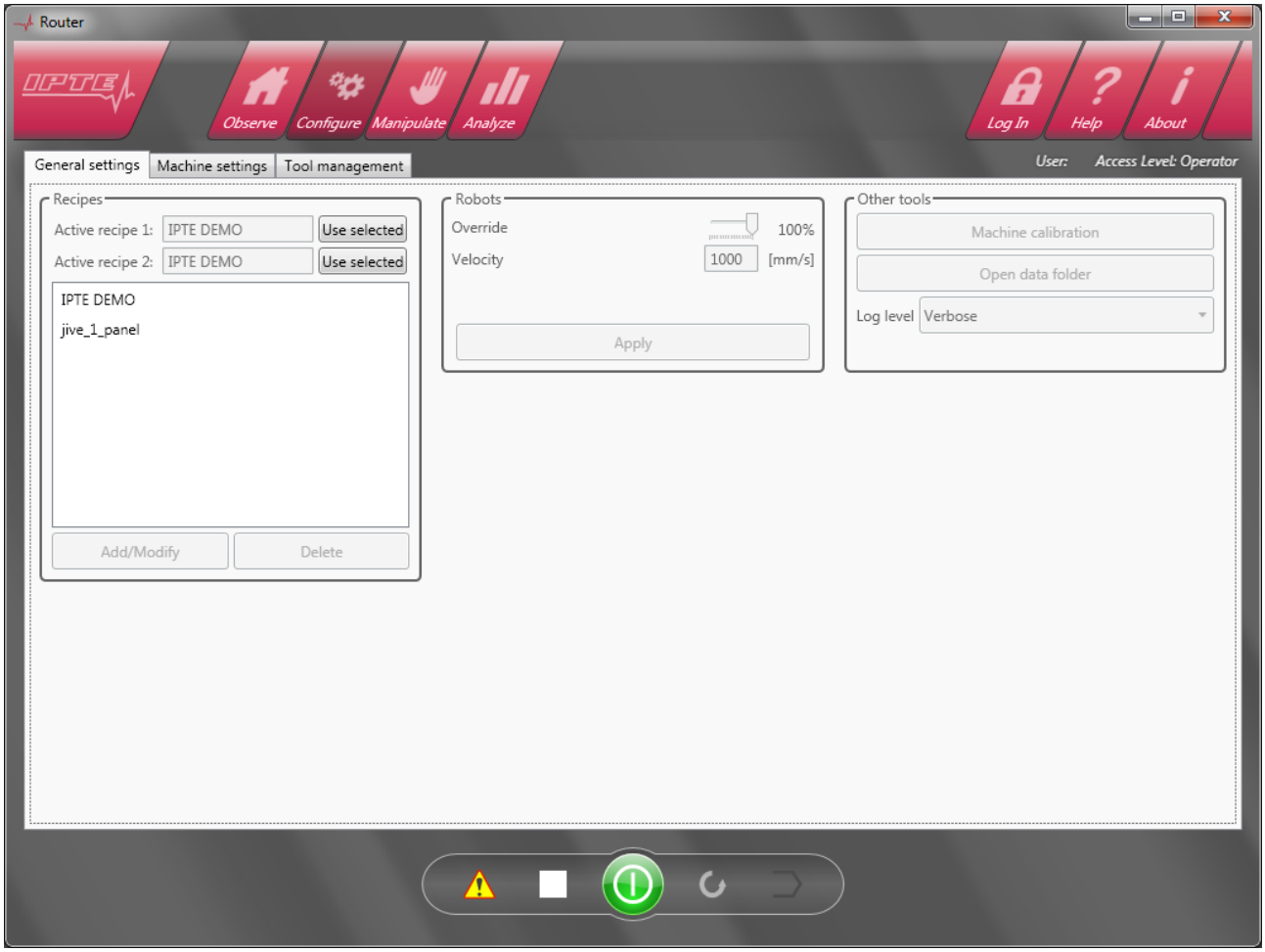


Figure 10: A typical general settings page, it contains recipe management and buttons to recipe and calibration wizards.

3.2 Admin Config Page

The admin config page (Figure 11) is, as name implies, only accessible to highest level user. The page allows direct access to machine settings, it is equivalent to modifying configuration file. The three primary drawbacks to directly modifying config file are:

- The config file is format xml, it is relatively easy to break the xml syntax rules making the config file unreadable.
- The xml does no validation on values whatsoever
- The program must be closed and reopened in order to reload settings from config file. (some settings still need restart of the program before they are activated)

Even tho the admin config page has limited access, it is still extemely useful, because there are settings that can only be changed from config file. The settings architecture, used by TS1, makes creation of new configuration options so simple that there isn't much incentive to keep hardcoded constants any more. Most of the settings have reasonable default values and don't need to be changed, it makes creation of extra configuration pages inefficient. Other settings only need to be changed once, such as camera type or barcode reader connection string. Those extra configuration pages would only clutter the user interface without adding much benefit.

Enter admin config page. It allows to conviniently change the configuration values that are otherwise hidden in config file. There is also some basic type checking in the page: for booleans it displays a checkbox, for enums it displays combo box with possible options and for numeric values it checks if inserted values are indeed numbers. It also enables quick filtering of the properties by name and displays the default value for each property.

It must be noted, that any changes to this page only change in-memory copy of settings object, in order to store the changes into config file the user must explicitly press the "Save" button or modify any of the settings elsewhere (other setting pages will automatically store config when changed).

Warning: implementation details! The page uses .net reflection to build a property tree, starting from the root object. The root object is typically the machine settings object.

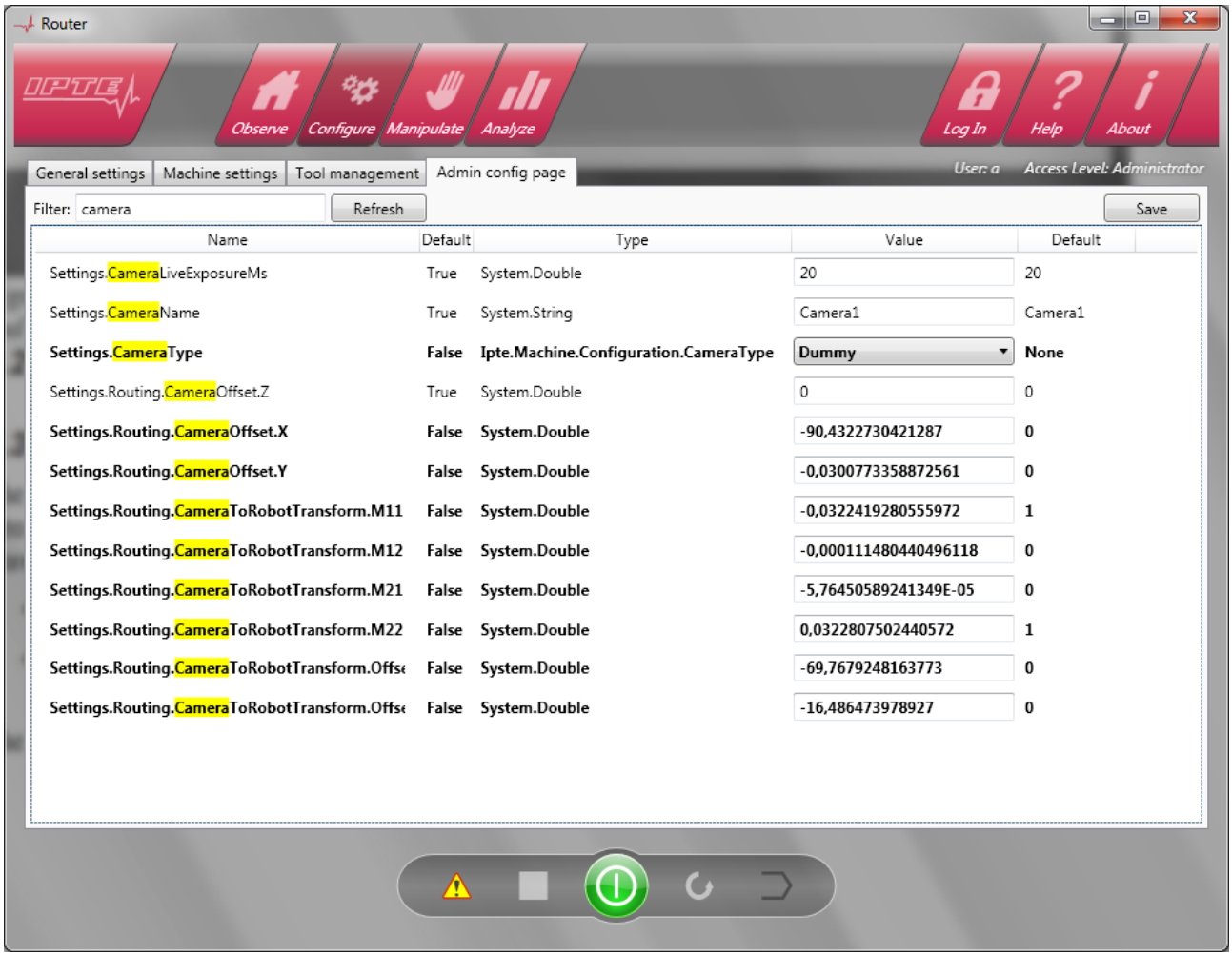


Figure 11: Admin config page

NB! Despite the fact, that we have used the page in numerous projects, it is still relatively experimental. There are ways to break the page, for example by creating infinite length property trees. There are plans to integrate the functionality of the page into a single control and, once it's finalized, to move it into GuiControlLibrary.

4 Manipulate Window

4.1 Frequent Actions Page

4.2 I/O Page

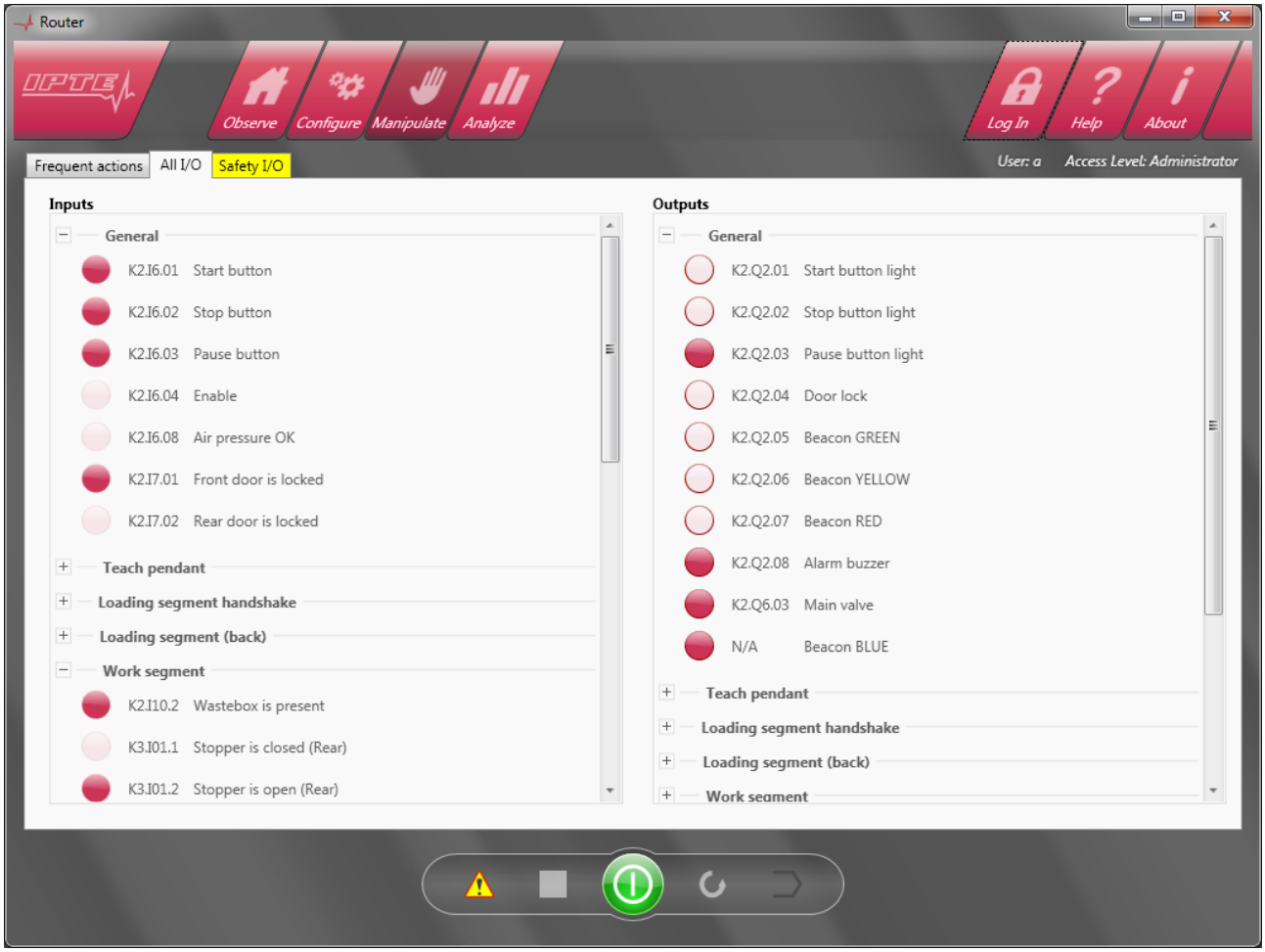


Figure 12: I/O page

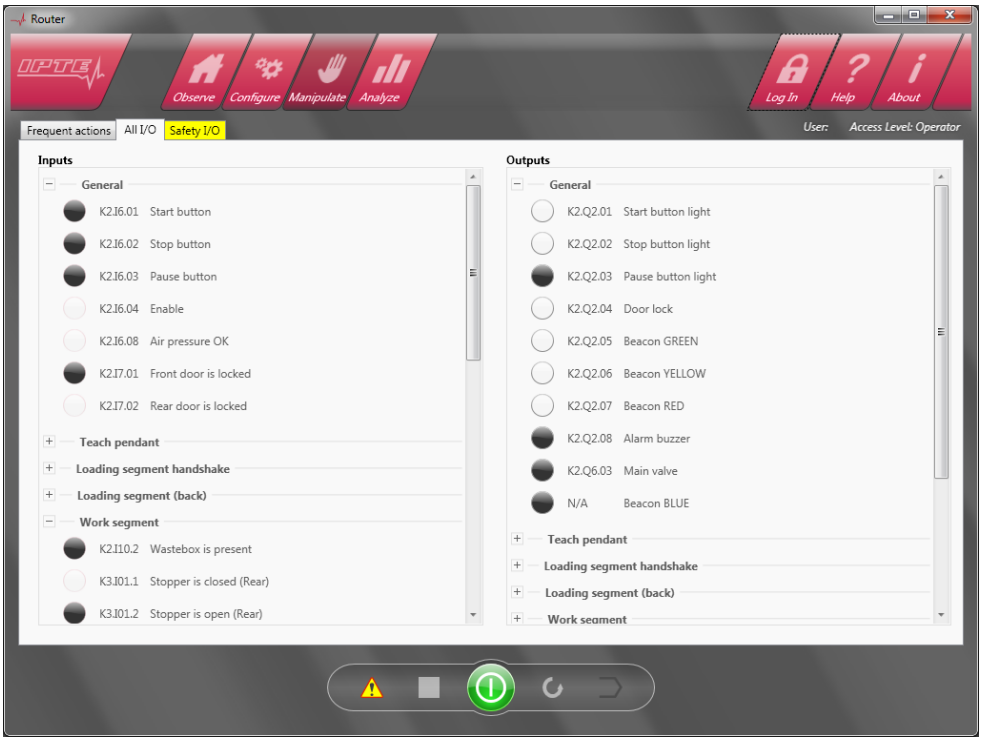


Figure 13: Disabled I/O page.

4.3 Safety I/O Page

5 Analyze Window

The *analyze* group of pages is built for displaying the process parameters over longer period of time. It offers some basic statistics functionality for cycle time measurement, unit counting and error frequency analysis.

Fetching of the data from database and processing the data might take a few seconds, especially if there's lots of data to display. During the time the application appears to be frozen. To mitigate this effect it is advisable to fetch and process the data in background thread and to display a notice on screen (Figure 14) during update.

Figure 14: Notice about updating

The current version of the statistics pages are somewhat limited in functionality and relatively complex to integrate into a project. At this point the integration of the statistics pages into a project consists of copy-pasting a bunch of files and slightly modifying them for the project. There are plans to build an advanced library for statistics that would make the integration easy and provide advanced functionality but it just hasn't been implemented yet.

The current implementation of statistics pages has two extra dependencies (besides .NET library): SQL compact edition (SQL CE 3.5 sp2) database for storage and WPF toolkit (NB! Feb 2010 version) for charting. The SQL CE is especially suitable for three reasons:

- it is a small install, that does not require any licenses and can be easily be included with visual studio one-click installer
- it supports LINQ to SQL object-relational mapping making programming easy. This is the area where the best alternative, sqlite, was lacking – it's object-relational wrapper was missing features.
- it does not install any database management services

The WPF toolkit provides WPF charting components that are missing from .NET. At the time, when the statistics pages were implemented, there wasn't much choice in royalty free WPF charting control department. The choice to use The WPF toolkit, available in codeplex, is essentially a beta version of various .NET controls, charting is just a part of it. Some of the controls have been included in .NET 4.0, such as DatePicker, but charting controls are yet to be included.

5.1 Cycle Times Page

Figure 15: Cycle times graph

The cycle times page is really a generic page that is built for displaying variation of a single process parameter over time. It can be cycle time, but it can also be process time or an SPC measurement value. The current implementation is built for fetching the cycle time from database, but it can be easily modified for displaying something else instead.

The charting control from WPF toolkit has problem displaying large data sets, the performance degrades to unusable if the count of points to be displayed in the chart reaches thousands. The workaround is to aggregate data; the data points are divided into 50 time slots and only the minimum, maximum and the average of each slot is displayed (Figure 16).

Figure 16: Aggregated cycle times chart

Warning: implementation details! Out of the box the WPF toolkit does not support a suitable chart series for displaying the aggregated data. The AggregateSeries, used for display, is defined within the context of a specific TS1 application; as opposed to defining it in separate library. This makes it's usage somewhat clumsy; the project must include two files that contain code for AggregateDataPoint and for the AggregateSeries. In addition there's XAML style defining how to display an AggregateDataPoint, it must be placed into globally visible resource e.g. App.xaml file.

5.2 Events Page

Figure 17: Event frequency graph

Figure 18: Event division graph.

Figure 19: Event density graph

5.3 Items In/Out Page

Figure 20: Product counter graph

6 Touch Screen Support

To support touch screens we need to ensure minimum size for all controls. The GUI Control Library is designed with touch screens in mind but windows controls are not. The default windows controls are too small to be used with fingers in gloves (Illustration 1).

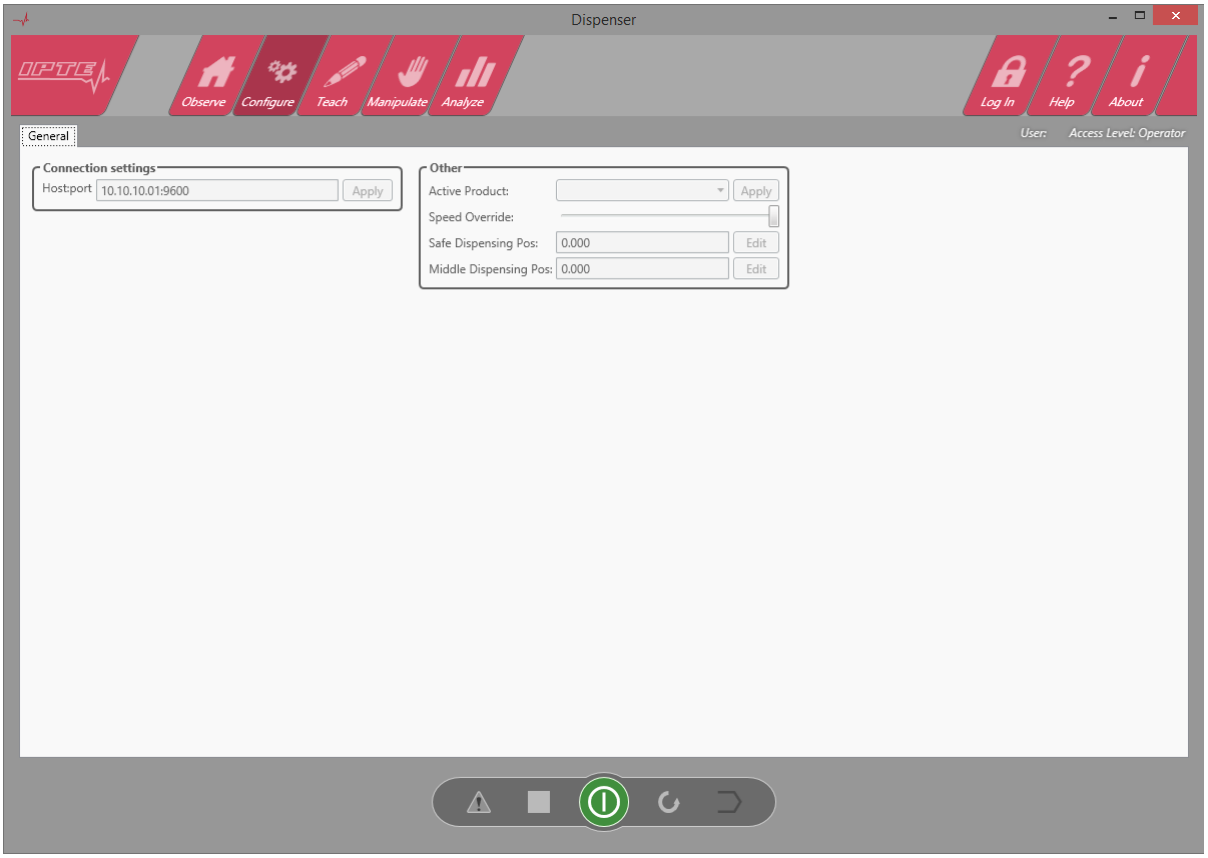


Illustration 1: UI for keyboard and mouse

Microsoft research gives an estimate for minimum button size; for phones and tablets it's about 7mm. In industrial application the operator may wear cloves and hence the buttons should be even bigger; about 9mm is sufficiently large.

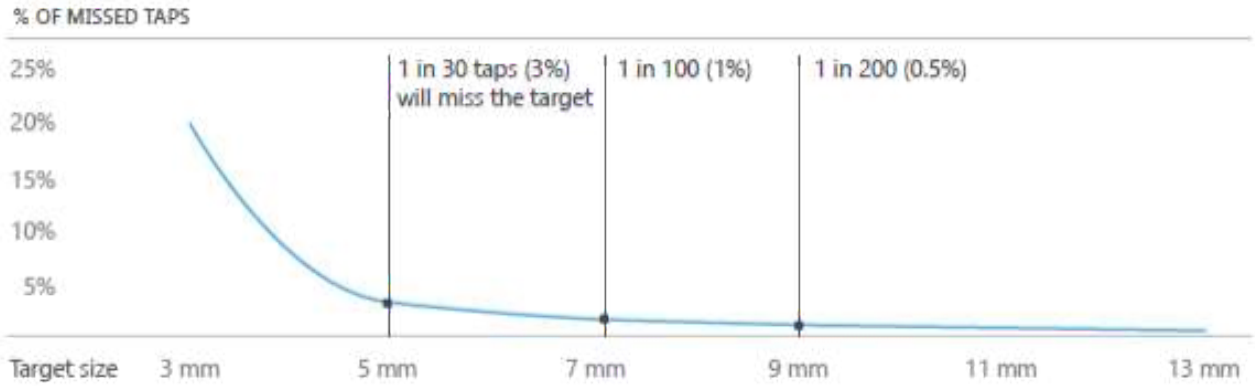


Illustration 2: "Size vs. efficiency: Target size influences error rate" - Windows 8 User experience guidelines

An easy way to increase the size of controls is to increase screen DPI setting from windows. The TS1 applications are based on WPF which respects the DPI setting. The problem with it is that it will increase the size of every item on screen which is not always desirable (Illustration 3). We would like to keep the images and texts normal to be able to fit reasonable amount of information onto screen and only scale the controls. In addition the GUI Control Library is designed with touch screens in mind and its controls are already large enough for touch screen; double scaling will make the controls unreasonably large.

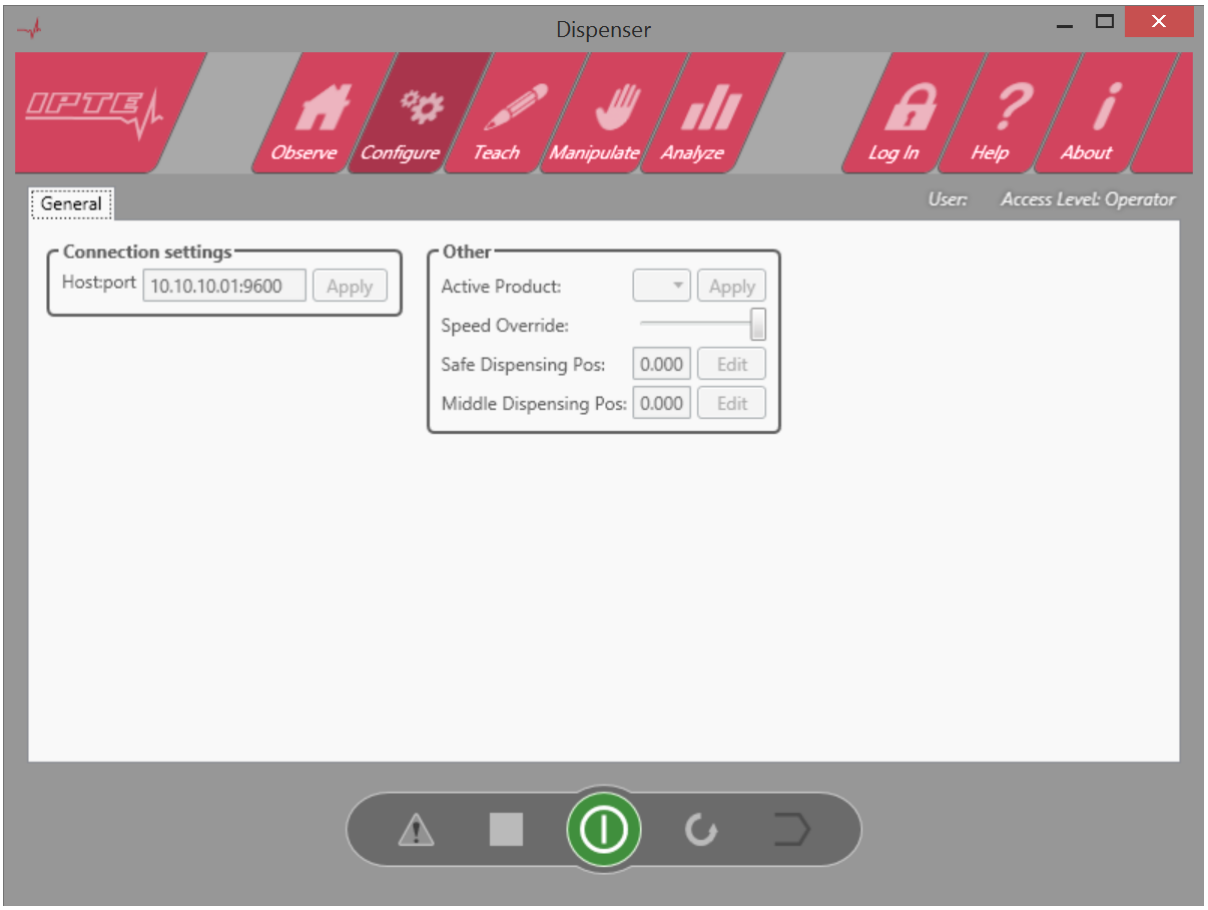


Illustration 3: DPI scaling

A better way to increase size for most controls in application is to override default style in app.xaml (Illustration 4). The following piece of code will increase minimum size of all buttons, toggle buttons, combo boxes (and combo box items), sliders, textboxes (and textbox fonts), tabitems and listbox items. It does not increase size of combo boxes, radio buttons and gridview items.

```
<!-- include this in namespace declaration -->
xmlns:gcl="clr-namespace:GuiControlLibrary;assembly=GuiControlLibrary"


<!-- style overrides -->
<Style TargetType="{x:Type Button}">
    <Setter Property="MinHeight" Value="30"/>
</Style>

<Style TargetType="{x:Type gcl:GuiButton}">
    <Setter Property="MinHeight" Value="30"/>
</Style>

<Style TargetType="{x:Type ToggleButton}">
    <Setter Property="MinHeight" Value="30"/>
</Style>
```

```
<Style TargetType="{x:Type gcl:GuiToggleButton}">
    <Setter Property="MinHeight" Value="30"/>
</Style>

<Style TargetType="{x:Type ComboBox}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type gcl:GuiComboBox}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type ComboBoxItem}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type Slider}">
    <Setter Property="MinHeight" Value="30"/>
</Style>

<Style TargetType="{x:Type gcl:GuiSlider}">
    <Setter Property="MinHeight" Value="30"/>
</Style>

<Style TargetType="{x:Type TextBox}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type gcl:GuiTextBox}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type PasswordBox}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type TabItem}">
    <Setter Property="MinHeight" Value="24"/>
    <Setter Property="MinWidth" Value="90"/>
    <Setter Property="FontSize" Value="16"/>
</Style>

<Style TargetType="{x:Type ListBoxItem}">
    <Setter Property="MinHeight" Value="30"/>
    <Setter Property="FontSize" Value="16"/>
</Style>
```

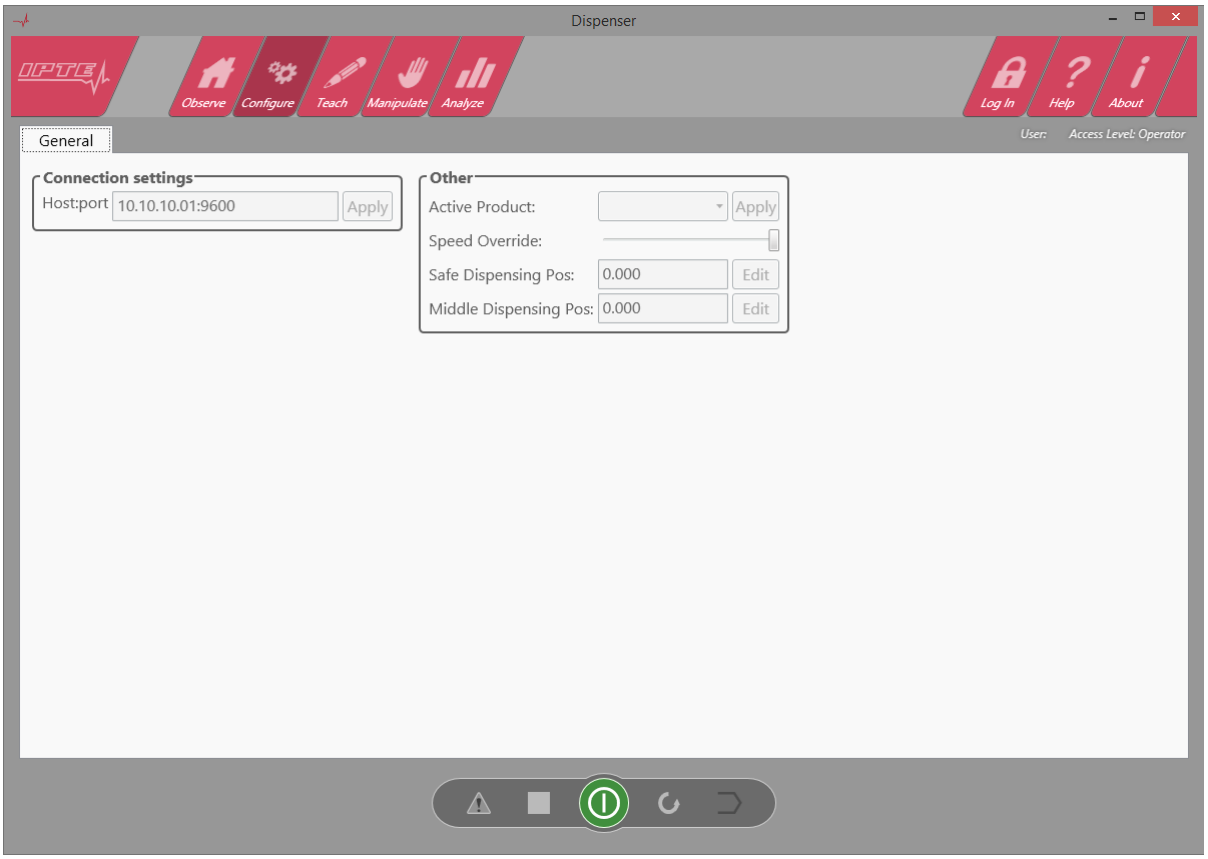


Illustration 4: Style based scaling

The objective in increasing the control sizes is to increase the chance of hit. The secondary, and more important, is to reduce the chance of mishit or chance of hitting wrong control. For some controls (buttons, and sliders) it is feasible to increase the size but others (checkbox, radiobutton) do not scale. For checkboxes and radiobuttons it is necessary to increase the margin around control to ensure large enough spacing.

7 Conclusion

This document is intended as a guidline, it provides samples from previous works. It does not impose limits on what can be done. For instance, to add luxurious feel to the software we could carve the window tabs out of red wood, but this is not what you should do. Keep your work aligend to IPTE common GUI layout.