DiWaCS Documentation

Release 0.9.3.0

Nick Eriksson

July 15, 2013

DiWaCS is an application developed for DiWa smart space and should be used **only** inside **Diwaamo**. DiWaCS connects to address **239.128.128.1:5555** using Pragmatic General Multicast (PGM). DiWaCS is built on Python and WxPython is used for UI programming. Currently, only supported platform is **Windows 7**.

Required python modules for DiWaCS:

- Configobj
- lxml
- PIL
- PyAudio
- Python Pubsub
- SQLAlchemy
- Watchdog
- WMI
- WxPython
- ZeroMQ with openpgm support

Contents:

CHAPTER

ONE

AUTOMATED CODE DOCUMENTATION

Documentation generated on 2013-07-15 at 14:51.

1.1 Add file module

Created on 5.6.2012

platform Windows

synopsis Used to add a file in the current project.

warning Requires ZeroMQ.

author neriksso

`add_file.main()`

Main function of the sub program.

Sub program is meant to be bound to windows explorer context menu. Context menu allows the user to quickly add files to project without interacting with DiWaCS directly.

Transmits the add_file command to DiWaCS via interprocess socket.

Parameters **filepath** (*String*) – Path of the file to be added.

Returns windows success code (0 on success).

Return type Integer

1.2 Send file module

Created on 5.6.2012

author neriksso

requires Requires ZeroMQ

synopsis Used to send a file to another node.

`send_file_to.main()`

Main function of the sub program.

Sub program is meant to be bound to windows explorer context menu. Context menu allows the user to quickly send files without interacting with DiWaCS directly.

Transmits the send_to command to DiWaCS via interprocess connection.

Parameters

- **node_id** (*Integer*) – ID of the node to send the file to.
- **filepath** (*String*) – Path of the file to be sent.

Returns windows success code (0 on success).

Return type Integer

1.3 Controller package

Used to control the database.

1.3.1 controller.activity module

Created on 28.6.2013

author neriksso

`controller.activity.`**`add_activity`**(*project_id*, *pgm_group*, *session_id=None*, *activity_id=None*)

Add activity to database.

Parameters

- **project_id** (*Integer*) – ID of the project Activity is associated with.
- **pgm_group** (*Integer*) – The PGM Group number.
- **session_id** (*Integer*) – ID of the session Activity is associated with.
- **activity_id** (*Integer*) – ID of the activity.

Returns Activity ID of the added activity.

Return type Integer

`controller.activity.`**`get_active_activity`**(*pgm_group*)

Get the latest active activity id.

Parameters **pgm_group** (*Integer*) – The PGM Group number.

Returns Latest active activity ID.

Return type Integer

`controller.activity.`**`logger`**()

Return controller logger.

`controller.activity.`**`unset_activity`**(*pgm_group*)

Unsets activity for PGM Group.

Parameters **pgm_group** (*Integer*) – The PGM Group number.

1.3.2 controller.common module

Created on 28.6.2013

author neriksso

`controller.common.`**`connect_to_database`**(*expire=False*)

Connect to the database and return a Session object.

Parameters **expire** (*Boolean*) – Parameter passed to session maker as expire_on_commit.

Returns Session.

Return type `sqlalchemy.orm.session.Session`

`controller.common.`**`create_all`**`()`
Create tables to the database.

`controller.common.`**`delete_record`**`(`*record_model*, *id_number*`)`
Delete a record from database

Parameters

- **record_model** (`sqlalchemy.ext.declarative.declarative_base()`) – The model for which to delete a record.
- **id_number** (*Integer*) – Recond id.

Returns Success.

Return type Boolean

`controller.common.`**`get_action_id_by_name`**`(`*action_name*`)`
Get the static ID of action name.

`controller.common.`**`get_or_create`**`(`*database*, *model*, ***kwargs*`)`
Fetches or creates a instance.

Parameters

- **database** (`sqlalchemy.orm.session.Session`) – a related database.
- **model** (`sqlalchemy.ext.declarative.declarative_base`) – The model of which an instance is wanted.

Returns An object of the desired model.

`controller.common.`**`set_node_name`**`(`*name*`)`
Set the stored node name for own swnp node as global.

Warning This should be removed in the future as globals are bad.

`controller.common.`**`set_node_screens`**`(`*screens*`)`
Set the stored node screens settings for own swnp node as global.

Warning This should be removed in the future as globals are bad.

`controller.common.`**`test_connection`**`()`
Test the connection to database.

Returns Does the software have access to the database at this time.

Return type Boolean

`controller.common.`**`update_database`**`()`
Update the database connection engine.

Note: This only works when DB_STRING is completely defined by the log reader.

1.3.3 controller.computer module

Created on 28.6.2013

author neriksso

`controller.computer.`**`add_computer`**(*name*, *pc_ip*, *wos_id*)

Add a new computer to the database.

Parameters

- **name** (*String*) – Name of the computer.
- **pc_ip** (*String*) – IP address of the computer.
- **wos_id** (*Integer*) – Node ID of the computer (usually the last part of IP).

Returns The added computer

Return type `models.Computer`

`controller.computer.`**`add_computer_to_session`**(*session*, *name*, *pc_ip*, *wos_id*)

Adds a computer to a session.

Parameters

- **session** (`models.Session`) – A current session.
- **name** (*String*) – A name of the computer.
- **pc_ip** (*Integer*) – Computers IP address.
- **wos_id** (*Integer*) – Wos id of the computer.

`controller.computer.`**`get_active_computers`**(*timeout*)

Get all the active computers from database.

Parameters **timeout** (*Integer*) – The number of seconds an "active" computer may have been idle while still being considered active.

Returns A list of active computers.

Return type List of `models.Computer`

`controller.computer.`**`get_active_responsive_nodes`**(*pgm_group*)

Return the wos_id fields of all active responsive nodes.

Parameters **pgm_group** (*Integer*) – The responsive group we want.

Returns A list of node IDs that are both active and responsive.

Return type A list of Integer

`controller.computer.`**`last_active_computer`**()

Is the current node last active computer.

Note: This uses 10 seconds as timeout for definition "not active".

Return type Boolean

`controller.computer.`**`logger`**()

Return controller logger.

`controller.computer.`**`refresh_computer`**(*computer*)

Refresh the computer in database.

Parameters **computer** (`models.Computer`) – The computer to refresh.

Returns The Refreshed computer.

Return type `models.Computer`

`controller.computer.`**`refresh_computer_by_wos_id`**(*wos_id*, *new_name=None*, *new_screens=None*, *new_responsive=None*)

Refresh the computer by node id and give it optionally new configurations.

Parameters

- **wos_id** (*Integer*) – The ID of the node to refresh.
- **new_name** (*String*) – Optional new name for the node.
- **new_screens** (*Integer*) – Optional new screens configuration for the node.
- **new_responsive** (*Integer*) – Optional new responsive setting for the node.

Returns Success

Return type Boolean

1.3.4 controller.handlers module

Created on 28.6.2013

author neriksso

class `controller.handlers.`**`PROJECT_FILE_EVENT_HANDLER`**(*project_id*)

Handler for FileSystem events on project folder.

Parameters **project_id** (*Integer*) – Project id from database.

`on_created`(*event*)

On_created event handler. Logs to database.

Parameters **event** (an instance of `watchdog.events.FileSystemEvent`) – The event.

`on_deleted`(*event*)

On_deleted event handler. Logs to database.

Parameters **event** (an instance of `watchdog.events.FileSystemEvent`) – The event.

`on_modified`(*event*)

On_modified event handler. Logs to database.

Parameters **event** (an instance of `watchdog.events.FileSystemEvent`) – The event.

class `controller.handlers.`**`SCAN_HANDLER`**(*project_id*)

Handler for FileSystem events on SCANNING folder.

Parameters **project_id** (*Integer*) – Project id from database.

`on_created`(*event*)

On_created event handler. Logs to database.

Parameters **event** (an instance of `watchdog.events.FileSystemEvent`) – The event.

`controller.handlers.`**`logger`**()

Return controller logger.

1.3.5 controller.project module

Created on 28.6.2013

author neriksso

`controller.project.`**`add_file_to_project`**(*filepath*, *project_id*)

Add a file to project. Copies it to the folder and adds a record to database.

Parameters

- **filepath** (*String*) – A filepath.
- **project_id** – Project id from database.

Returns New filepath.

Return type String

`controller.project.`**`add_project`**(*data*)

Adds a project to database and returns a project instance

Parameters **data** (*A dictionary*) – Project information

Return type an instance of `models.Project`

`controller.project.`**`check_password`**(*project_id*, *password*)

Docstring here.

`controller.project.`**`create_file_action`**(*path*, *action_id*, *session_id*, *project_id*)

Logs a file action to the database.

Parameters

- **path** (*String*) – Filepath.
- **action_id** (*Integer*) – File action id.
- **session_id** (*Integer*) – Current session id.
- **project_id** (*Integer*) – Project id from database.

`controller.project.`**`edit_project`**(*id_number*, *row*)

Update the project info.

Parameters

- **id_number** (*Integer*) – Database id number of the project.
- **row** (*A dictionary*) – The new project information.

`controller.project.`**`get_active_project`**(*pgm_group*)

Get the active project.

Parameters **pgm_group** (*Integer*) – The PGM Group number.

Returns Active project ID.

Return type Integer

`controller.project.`**`get_file_path`**(*project_id*, *filename*)

Returns the filepath for filename.

Returns Filepath.

Return type String

`controller.project.`**`get_project`**`(`*project_id*`)`

Fetches projects by a company.

Parameters **company_id** (*Integer*) – A company id from database.

`controller.project.`**`get_project_id_by_activity`**`(`*activity_id*`)`

Docstring here.

`controller.project.`**`get_project_password`**`(`*project_id*`)`

Returns the project password.

Parameters **project_id** (*Integer*) – ID of the project.

Return type String

`controller.project.`**`get_project_path`**`(`*project_id*`)`

Fetches the project path from database and return it.

Parameters **project_id** (*Integer*) – Project id for database.

Return type String

`controller.project.`**`get_projects_by_company`**`(`*company_id*`)`

Fetches projects by a company.

Parameters **company_id** (*Integer*) – A company id from database.

`controller.project.`**`get_recent_files`**`(`*project_id*, *max_files_count=None*`)`

Fetches files accessed recently in the project sessions from the database.

New in version 0.9.3.0: Added a limit parameter, limits the number of returned results.

Note: Duplicate check has been added at some point in time.

Parameters **project_id** (*Integer*) – The project id

Returns The list of filepaths that have recently been used in this project.

Return type List of String

`controller.project.`**`init_sync_project_directory`**`(`*project_id*`)`

Initial sync of project dir and database.

Parameters **project_id** (*Integer*) – Project id from database.

`controller.project.`**`is_project_file`**`(`*filename*, *project_id*`)`

Checks, if a file belongs to a project. Checks both project folder and database.

Parameters

- **filename** (*String*) – a filepath.
- **project_id** (*Integer*) – Project id from database.

Return type Boolean

`controller.project.`**`logger`**`()`

Return controller logger.

1.3.6 controller.session module

Created on 28.6.2013

author neriksso

`controller.session.`**`add_event`**`(`*session_id*, *title*, *description*`)`

Adds an event to the database.

Parameters

- **session** (`models.Session`) – The current session.
- **description** (*String*) – Description of the event.

`controller.session.`**`end_session`**`(`*session_id*`)`

Ends a session, sets its endtime to database. Ends file scanner.

Parameters **session** (`models.Session`) – Current session.

`controller.session.`**`get_active_session`**`(`*pgm_group*`)`

Get the active session.

Parameters **pgm_group** (*Integer*) – The PGM Group number.

Returns The active session ID.

Return type Integer

`controller.session.`**`get_latest_event`**`()`

Get the latest event id.

Returns The ID of latest event.

Return type Integer

`controller.session.`**`get_session_id_by_activity`**`(`*activity_id*`)`

Docstring here.

`controller.session.`**`get_sessions_by_project`**`(`*project_id*`)`

Fetches sessions for a project.

Parameters **project_id** (*Integer*) – Project id from database.

`controller.session.`**`logger`**`()`

Return controller logger.

`controller.session.`**`start_new_session`**`(`*project_id*, *session_id=None*, *old_session_id=None*`)`

Creates a session to the database and return a session object.

Parameters

- **project_id** (*Integer*) – Project id from database.
- **session_id** (*Integer*) – an existing session id from database.
- **old_session_id** (*Integer*) – A session id of a session which will be continued.

1.4 Dialogs module

Created on 4.6.2013

author neriksso

class `dialogs.`**`AddProjectDialog`**`(`*parent*, *title*, *project_id=None*`)`

A dialog for adding a new project

Parameters

- **parent** (`wx.Frame`) – Parent frame.
- **title** (*String*) – A title for the dialog.

`OnAdd`(*event*)
Handles the addition of a project to database, when "Add" button is pressed.

Parameters **event** (*Event*) – GUI Event.

`OnClose`(*event*)
Handles "Close" button presses.

Parameters **event** (*Event*) – GUI Event.

`OnText`(*event*)
Event handler for text changed.

exception `dialogs.`**`CloseError`**(**args*, ***kwds*)
Class describing an error while closing application.

class `dialogs.`**`ConnectionErrorDialog`**(*parent*)
Create a connection error dialog that informs the user about reconnection attempts made by the software.

class `dialogs.`**`DeleteProjectDialog`**(*parent*, *title*, *project_id*)
A dialog for deleting project.

`OnCancel`(*event*)
Event handler for pressing Cancel button.

`OnOk`(*event*)
Event handler for pressing OK button.

class `dialogs.`**`ErrorDialog`**(*parent*, *message*)
Error dialog.

class `dialogs.`**`PreferencesDialog`**(*parent*, *config_object*)
Creates and displays a preferences dialog that allows the user to change some settings.

Parameters **config_object** (`configobj.ConfigObj`) – a Config object

`LoadPreferences`()
Load the current preferences and fills the text controls.

`OnCancel`(*event*)
Closes the dialog without modifications.

Parameters **event** (*Event*) – GUI event.

`OpenConfig`(*event*)
Opens config file.

Parameters **event** (*Event*) – GUI event.

`SavePreferences`(*event*)
Save the preferences.

Parameters **event** (*Event*) – GUI Event.

class `dialogs.`**`ProjectAuthenticationDialog`**(*parent*, *title*, *project_id*)
A dialog for project authentication.

`OnOk`(*event*)
Called on OK button press.

class `dialogs.`**`ProjectSelectDialog`**(*parent*)
A dialog for selecting a project.

Parameters **parent** (`wx.Frame`) – Parent frame.

`OnCancel`(*event*)
Handles "Cancel" button presses.

Parameters **event** (*Event*) – GUI Event.

`OnProjectAdd`(*event*)
Shows a modal dialog for adding a new project.

Parameters **event** (*Event*) – GUI Event.

`OnProjectDelete`(*event*)
Handles the selection of a project. Starts a `wos.CURRENT_PROJECT`, if necessary. Shows a dialog of the selected project.

Parameters **evt** (*Event*) – GUI Event.

`OnProjectEdit`(*event*)
Shows a modal dialog for adding a new project.

Parameters **event** (*Event*) – GUI Event.

`OnProjectSelect`(*event*)
Handles the selection of a project.

Starts a `wos.CURRENT_PROJECT`, if necessary. Shows a dialog of the selected project.

Parameters **event** (*Event*) – GUI Event.

`OnSelectionChange`(*event*)
Event handler for selection change of the listbox.

`UpdateProjects`(*company_id=1*)
Fetches all projects from the database, based on the company.

Parameters **company_id** (*Integer*) – A company id, the owner of the projects.

Returns The total number of projects.

Type Integer

class `dialogs.`**`ProjectSelectedDialog`**(*parent*, *title*, *project_id*)
A dialog for project selection confirmation.

class `dialogs.`**`SendProgressBar`**(*parent*, *title*, *ypos*)
Implements file send progress bar...

class `dialogs.`**`UpdateDialog`**(*title*, *url*, **args*, ***kwargs*)
A Dialog which notifies about a software update. Contains the URL which the user can click on.

Parameters

- **title** (*String*) – Title of the dialog.
- **url** (*String*) – URL of the update.

`dialogs.`**`set_logger_level`**(*level*)
Sets the logger level for dialogs logger.

Parameters **level** (*Integer*) – Level of logging.

`dialogs.`**`show_modal_and_destroy`**(*class_*, *parent*, *params=None*)
Used to show modal and destroy afterwards.

Note: The implementation is kind of ugly, but guarantees a safe execution of the dialog without memory leaks and with all exceptions logged.

Parameters

- **class** (*type*) – The type of dialog to show.
- **parent** (`wx.Window`) – The parent wx.Window of this object.
- **params** (*Dictionary.*) – The params to give for __init__ call.

Returns The modal result value.

Return type Integer

1.5 DiWaCS module

Created on 8.5.2012

author neriksso

class `diwacs.`**`EventList`**(*parent*, **args*, ***kwargs*)

A Frame which displays the possible event titles and handles the event creation.

`CheckVisibility`(*selection*)

Checks the visibility.

`HideNow`()

Method to hide the event list.

`OnEnter`(*event*)

Event handler for pressing ENTER button.

Parameters **event** (`wx.Event`) – The EVT_ON_TEXT_EVENT event.

`OnFocusLost`(*event*)

On focus lost event handler.

`OnSelection`(*event*)

On selection event handler.

`OnText`(*event*)

On text event handler.

`ShowNow`()

Method to show the event list.

class `diwacs.`**`GraphicalUserInterface`**

WOS Application Frame.

`DisableDirectoryButton`()

Used to disable the project directory button when project has been unselected.

Note: There should be no need for this as the software should always start a new project after the old one ends. But for the mid state to be legimate this is still usable.

`DisableSessionButton`()

Used to disable the needed buttons after session has been stopped.

Note: Does not actually disable to session button, only the session state of the button.

EnableDirectoryButton()
Used to enable the project directory button when project has been selected.

EnableSessionButton()
Used to enable the needed buttons after session has been started.

GetNodeByName(*name*)
From current session nodes, select a node with this name or return None.

Parameters **name** (*String*) – Name of the desired node.

Returns The desired node if one exists.

Return type `swnp.Node`

InitUICore()
Inits the Core UI (`guitemplates.GUItemplate.InitUI()`) and binds the functionality.

OnAboutBox(*event*)
About dialog.

Parameters **e** (*Event*) – GraphicalUserInterface Event.

OnEvtBtn(*event*)
Event Button handler.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

OnExit(*event*)
Exits program.

Parameters **event** (*Event*) – GraphicalUserInterface Event

OnIconify(*event*)
Window minimize event handler. Should toggle the minimized state of the application.

Parameters **evt** (*Event*) – GraphicalUserInterface Event.

OnInfoBtn(*event*)
Handles the pressing of Web-information button.

Directs the user to web-storage website/help.

OnMBBtn(*event*)
Handles the pressing of meetings browser button.

Directs the user to web-storage website/mb.

OnPreferences(*event*)
Preferences dialog event handler.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

OnProject()
Project selected event handler.

OnSession(*event*)
Session button pressed.

The user either desires to start a new session or end an existing one.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

OnTaskBarActivate(*event*)
Taskbar activate event handler.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

`OnTaskBarClose`(*unused_event*)
Taskbar close event handler.

Parameters **evt** (*Event*) – GraphicalUserInterface Event.

`OnWABtn`(*event*)
Handles the pressing of Web-application button.

Directs the user to web-storage website.

`OpenProjectDir`(*event*)
Opens project directory in windows explorer.

Parameters **event** (*Event*) – The GraphicalUserInterface event.

`PaintSelect`(*evt*)
Paints the selection of a node.

Note: For future use.

Parameters **evt** (*Event*) – GraphicalUserInterface Event

`SelectNode`(*event*)
Handles the selection of a node, start remote control.

Note: For future use.

Parameters **event** (*Event*) – GraphicalUserInterface Event

`SelectProjectDialog`(*event*)
Select project event handler.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

`SetProjectName`(*name*)
Set the project text. For example "No Project OnSelection".

Note: Requires None explicitly when the purpose is to set default label because writing SetProjectName(None) is more informative than SetProjectName()

Parameters **name** (*String*) – The name of the project to set as label.

`Shift`(*event*)
Caroussel Shift function.

Parameters **event** (*Event*) – GraphicalUserInterface Event.

`UpdateScreens`(*update*)
Called when screens need to be updated and redrawn.

Parameters **update** (*Boolean*) – Pubsub needs one param, therefore it is called update.

`diwacs.`**`main`**(*profile*)
Main function.

Warning The profiler has been pre-calibrated using the development machine so this should be changed for other development environments that wish to profile the execution of the diwacs system.

THIS ONLY WORKS WHEN diwavars.DEBUG HAS BEEN ENABLED.

Remember to disable it from release binaries.

Parameters **profile** (*Boolean*) – should the call be profiled?

`diwacs.`**`set_logger_level`**`(`*level*`)`

Used to set logger level.

Parameters **level** (*Integer*) – The level desired.

1.6 DiWaVars module

DiWaCS Variables

`diwavars.`**`add_logger_initializer`**`(`*logger_initializer*`)`

For initializing the loggers from main.

Parameters **logger_initializer** (*function*) – The logger initializer to add to initialize chain.

`diwavars.`**`add_logger_level_setter`**`(`*logger_level_setter*`)`

For setting application logger level globally.

Parameters **logger_level_setter** (*function*) – The logger level setter to add to level set chain.

`diwavars.`**`set_blank_cursor`**`(`*value*`)`

Set the blank cursor variable.

`diwavars.`**`set_config`**`(`*config*`)`

Set the CONFIG global...

`diwavars.`**`set_default_cursor`**`(`*value*`)`

Set the default cursor variable.

`diwavars.`**`set_run_cmd`**`(`*value*`)`

Update the RUN_CMD setting.

Parameters **value** (*Boolean*) – Desired value.

`diwavars.`**`set_running`**`()`

Set the currently running flag as true.

Causes other modules to redirect their stdout and stderr streams to files.

`diwavars.`**`update_PGM_group`**`(`*new_group*`)`

Update the PGM group for this node.

`diwavars.`**`update_audio`**`(`*audio*`)`

Docstring here.

`diwavars.`**`update_camera_vars`**`(`*url*, *user*, *passwd*`)`

Docstring here.

`diwavars.`**`update_database_vars`**`(`*address=None*, *name=None*, *type_=None*, *user=None*, *password=None*`)`

Docstring here.

`diwavars.`**`update_keys`**`(`*modifier=91*, *key=27*`)`

Update the key combination to stop remote controlling.

Parameters

- **modifier** (*Integer*) – The key to hold.

- **key** (*Integer*) – The key to press while holding modifier key.

`diwavars.`**`update_padfile`**(*padurl*)
Set the padfile address.

`diwavars.`**`update_responsive`**(*resp*)
Docstring here.

`diwavars.`**`update_storage`**(*storage*)
Update the address of storage.

Parameters **storage** (*String*) – The new address of storage.

`diwavars.`**`update_windows_version`**()

Updates the current version information to variables:

- WINDOWS_MAJOR
- WINDOWS_MINOR

1.7 Filesystem module

Created on 17.5.2013

author neriksso

`filesystem.`**`copy_file_to_project`**(*filepath*, *project_id*)
Copy file to project dir and return new filepath in project directory.

Parameters

- **filepath** (*String*) – The file path.
- **project_id** (*Integer*) – Project id from database.

Returns The path for this file in project directory or empty string.

Return type String

`filesystem.`**`copy_to_temporary_directory`**(*filepath*)
Copy a file to temporary folder.

Parameters **filepath** (*String*) – The file path.

`filesystem.`**`create_project_directory`**(*dir_name*)
Creates a project directory, if one does not exist in the file system

Parameters **dir_name** (*String*) – Name of the directory

`filesystem.`**`delete_directory`**(*path*)
Deletes a directory.

Returns Weather the function was successful or not.

Return type String

`filesystem.`**`file_to_base64`**(*filepath*)
Transform a file to a binary object.

Parameters **filepath** (*String*) – The file path.

`filesystem.`**`get_file_extension`**(*path*)
Returns the file extension of a file

Parameters **path** (*String*) – The file path.

Return type String

`filesystem.`**`get_node_image`**(*node*)

Searches for a node's image in STORAGE.

Parameters **node** (*Integer*) – The node id.

`filesystem.`**`is_subtree`**(*filename*, *parent*, *case_sensitive=True*)

Determines, if filename is inside the parent folder.

Parameters

- **filename** (*String*) – The file path.
- **parent** (*String*) – The parent file path.

`filesystem.`**`open_file`**(*filepath*)

Opens a file path.

Parameters **filepath** (*String*) – The file path.

`filesystem.`**`save_screen`**(*filepath*)

Saves the background image of the desktop.

Parameters **filepath** (*String*) – The filepath for the saved image.

`filesystem.`**`screen_capture`**(*path*, *node_id*)

Take a screenshot and store it in project folder.

Parameters

- **path** (*String*) – Path to the project folder.
- **node_id** (*Integer*) – NodeID

`filesystem.`**`search_file`**(*filename*, *search_path*, *case_sensitive=True*)

Search file in a given path.

Parameters

- **filename** (*String*) – The file name.
- **search_path** (*String*) – The search path.

Returns The path to the file.

Return type String

`filesystem.`**`set_logger_level`**(*level*)

Sets the logger level for filesystem logger.

Parameters **level** (*Integer*) – Level of logging.

`filesystem.`**`snapshot`**(*path*)

Start the worker thread for snapshot.

Parameters **path** (*String*) – File path where to store the snapshot.

`filesystem.`**`snapshot_procedure`**(*path*)

Worker for storing the snapshot.

Warning: This object has a timeout of 1 minute. So consider terminating the thread on shutdown if it's hanging.

Parameters **path** (*String*) – File path where to store the snapshot.

`filesystem.`**`test_storage_connection`**`()`

Try to access \StorageProjects

Returns Does the path exist.

Return type Boolean

1.8 Graphical Design module

Created on 6.6.2013

author neriksso

synopsis This file represents graphical designs of some GUI elements in DiWaCS.

class `graphicaldesign.`**`BlackOverlay`**`(`*pos*, *size*, *parent*, *text*`)`

Represents all black frame without a mouse.

`OnFocusLost``(`*evt*`)`

Event handler for focus losing of the window.

Parameters **evt** (`wx.Event`) – The focus lost event.

class `graphicaldesign.`**`DropTarget`**`(`*window*, *parent*, *i*`)`

Implements drop target functionality to receive files, bitmaps and text.

`OnData``(`*x*, *y*, *d*`)`

Handles drag/dropping files/text or a bitmap.

Parameters

- **x** (*Integer*) – The x coordinate of the drop-location.
- **y** (*Integer*) – The y coordinate of the drop-location.
- **d** – The data of drop.

class `graphicaldesign.`**`EventListTemplate`**`(`*parent*, **args*, ***kwargs*`)`

Represents an event list menu.

`GetProgramIcon``(`*icon*`)`

Fetches gui icons.

Parameters **icon** (*String*) – The icon file name.

Return type `wx.Image`

class `graphicaldesign.`**`GUItemplate`**`(`**args*, ***kwargs*`)`

Represents the main GUI window graphical template.

Parameters

- **parent** (`wx.Window`) – Parent frame.
- **id** (*Integer*) – ID of the new Frame.
- **title** (*String*) – Title for the frame, default = EmptyString.
- **pos** (*wx.Point*) – Position of the new frame.
- **size** (*wx.Size*) – Size of the new frame.
- **style** (*long*) – Style flags for the new frame.
- **name** (*String*) – Name of the new frame.

AlignCenterTop()
Aligns frame to Horizontal center and vertical top.

ClearStatusText()
Sets the status text to EmptyScreen string.

ConnectionErrorHandler(*error*)
Show connection error handler dialog.

GetProgramIcon(*icon*)
Fetches a GUI icon.

Parameters **icon** (*String*) – The icon file name.

Return type `wx.Image`

HideScreens()
Hides all screens.

InitScreens()
Inits Screens.

InitUI(*node_id*)
UI initializing.

Parameters **node_id** (*Integer*) – The id of current swnp node (self).

OnExit(*event*)
Exits program.

Parameters **event** (*Event*) – GUI Event

SelectNode(*evt*)
Handles the selection of a node, prototype.

Parameters **evt** (*Event*) – GUI Event

class `graphicaldesign.`**ImageViewer**(*parent*, *image*, **args*, ***kwargs*)
Used to show an image.

class `graphicaldesign.`**MySplashScreen**(*parent=None*)
Create a splash screen widget.

class `graphicaldesign.`**NodeScreen**(*node*, *parent*)
Represents a bitmap with node id.

EmptyScreen()
Make this screen EmptyScreen.

ReloadAs(*node*)
Reload the content of this bitmap.

class `graphicaldesign.`**SysTray**(*parent*)
Taskbar Icon class.

Parameters **parent** (`wx.Frame`) – Parent frame

CreateMenu()
Create systray menu.

ShowMenu(*event*)
Show popup menu.

Parameters **event** (*Event*) – GUI event.

`ShowNotification(title, message)`
Start a thread to show the notification.

Parameters

- **title** (*String*) – Title to diplay in the balloon.
- **message** (*String*) – Message to display in the balloong (max 255 chars).

1.9 Macro module

macro.py defines a few user input functions.

`macro.GetKeydown(code)`
Docstring here.

class `macro.HardwareInput`
Docstring here.

class `macro.Input`
Docstring here.

class `macro.Input_I`
Docstring here.

class `macro.KeyBdInput`
Docstring here.

class `macro.MacroPoint`
Stores the x and y components of coordinates.

Attribute x c_ulong

Attribute y c_ulong

class `macro.MouseInput`
Docstring here.

`macro.click()`
Send a mouse click_type: LeftButton down, LeftButton up.

`macro.get_mouse_position()`
Return the current position of the mouse.

Returns The position of the mouse.

Return type `MacroPoint`

`macro.get_sendkeys(code)`
Returns a character for a key code.

Parameters **code** (*Integer*) – The character code.

`macro.hold()`
Send a mouse hold: LeftButton down.

`macro.key_press(event, kcode)`
Used to send a single virtual keycode to the system.

Parameters

- **event** (`wx.Event`) – Captured key event.
- **kcode** (*Integer*) – Keycode.

`macro.`**`middle_click`**`()`
Send a mouse middle click_type: MiddleButton down, MiddleButton up.

`macro.`**`middle_hold`**`()`
Send a mouse middle click_type: MiddleButton down.

`macro.`**`middle_release`**`()`
Send a mouse middle click_type: MiddleButton up.

`macro.`**`move`**`(`*pos_x*, *pos_y*`)`
move the cursor for pos_x amount in horizontal direction and pos_y amount in vertical direction.

Parameters

- **pos_x** (*Integer*) – Amount to move in horizontal direction.
- **pos_y** (*Integer*) – Amount to move in vertical direction.

`macro.`**`move_to`**`(`*pos_x*, *pos_y*`)`
move the mouse cursor to point (pos_x, pos_y) on screen.

Parameters

- **pos_x** (*Integer*) – X coordinate of the desired position.
- **pos_y** (*Integer*) – Y coordinate of the desired position.

`macro.`**`release`**`()`
Send a mouse release_type: LeftButton up.

`macro.`**`release_all_keys`**`()`
Reset every keycode state to UP state.

`macro.`**`right_click`**`()`
Send a mouse right click_type: RightButton down, RightButton up.

`macro.`**`right_hold`**`()`
Send a mouse right hold: RightButton down.

`macro.`**`right_release`**`()`
Send a mouse right release_type: RightButton up.

`macro.`**`send_input`**`(`*intype*, *data*, *flags*, *scan=0*, *mouse_data=0*`)`
send_input sends virtual user input.

Parameters

- **intype** (*String*) – Input type, either ‘mouse_input’ for mouse input or ‘key_input’ for keyboard input.
- **data** (*Integer or (Integer, Integer)*) – Input data, keycode to input or a tuple of (x, y) for mouse.
- **flags** (*Integer*) – Input flags, used to separate keyup and keydown events.
- **scan** (*Integer*) – Input scancode. More info in: http://en.wikipedia.org/wiki/Scancode
- **mouse_data** (*Integer*) – Represents additional information about mouse events for example wheel amount.

`macro.`**`slide`**`(`*difference_x*, *difference_y*`)`
slide the mouse for difference_x amount in horizontal direction and difference_y amount in vertical direction.

Parameters

- **difference_x** (*Integer*) – The amount to slide in horizontal direction.

- **difference_y** (*Integer*) – The amount to slide in vertical direction.

`macro.`**`slide_to`**(*target_x*, *target_y*, *speed='normal'*)

Slides the mouse to point (target_x, target_y)

Parameters

- **target_x** (*Integer*) – The target X coordinate.
- **target_y** (*Integer*) – The target Y coordinate.
- **speed** (*String*) – The speed of motion 'slow', 'normal' or 'fast'.

1.10 Models module

Created on 23.5.2012

author neriksso

warning Requires `sqlalchemy` and `pywin32`

synopsis Used to represent the different database structures on DiWa.

class `models.`**`Action`**(*name*)

A class representation of a action. A file action uses this to describe the action.

Field:

- `id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the action, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of the action (Max 50 characters).

Parameters **name** (`String`) – Name of the action.

class `models.`**`Activity`**(*project*, *session=None*)

A class representation of an activity.

Fields:

- `id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of activity, used as primary key in database table.
- `session_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the session activity belongs to.
- `session (sqlalchemy.orm.relationship)` - Session relationship.
- `project_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the project activity belongs to.
- `project (sqlalchemy.orm.relationship)` - Project relationship.
- `active (sqlalchemy.schema.Column(sqlalchemy.types.BOOLEAN))` - Boolean flag indicating that the project is active.

Parameters

- **project** (`models.Project`) – Project activity belongs to.
- **session** (`models.Session`) – Optional session activity belongs to.

class `models.`**`Company`**`(`*name*`)`

A class representation of a company.

Fields:

- `id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the company, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of the company (Max 50 characters).

Parameters **name** (`String`) – The name of the company.

class `models.`**`Computer`**`(`***kwargs*`)`

A class representation of a computer.

Fields:

- `id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of computer, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of the computer.
- `ip(sqlalchemy.schema.Column(sqlalchemy.dialects.INTEGER))` - Internet Protocol address of the computer (Defined as unsigned).
- `mac(sqlalchemy.schema.Column(sqlalchemy.types.String)` - Media Access Control address of the computer.
- `time (sqlalchemy.schema.Column(sqlalchemy.types.DATETIME))` - Time of the last network activity from the computer.
- `screens (sqlalchemy.schema.Column(sqlalchemy.types.SMALLINT))` - Number of screens on the computer.
- `responsive (sqlalchemy.schema.Column(sqlalchemy.types.SMALLINT))` - The responsive value of the computer.
- `user_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the user currently using the computer.
- `user(sqlalchemy.orm.relationship)` - The current user.
- `wos_id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - **WOS** ID.

class `models.`**`Event`**`(`***kwargs*`)`

A class representation of Event. A simple note with timestamp during a session.

Fields:

- `id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the event, used as primary key in database table.
- `title (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Title of the event (Max 40 characters).
- `desc (sqlalchemy.schema.Column(sqlalchemy.types.String))` - More in-depth description of the event (Max 500 characters).
- `time(sqlalchemy.schema.Column(sqlalchemy.types.DATETIME))` - Time the event took place.

- `session_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the session this event belongs to.
- `session` (`sqlalchemy.orm.relationship`) - Session this event belongs to.

class `models.`**`File`**(***kwargs*)

A class representation of a file.

Fields:

- `id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the file, used as primary key in database table.
- `path` (`sqlalchemy.schema.Column(sqlalchemy.types.String)`) - Path of the file on DiWa (max 255 chars).
- `project_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the project this file belongs to.
- `project` (`sqlalchemy.orm.relationship`) - Project this file belongs to.

class `models.`**`FileAction`**(*fileobject*, *action*, *session=None*, *computer=None*, *user=None*)

A class representation of a file action.

Fields:

- `id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the FileAction, used as primary key in the database table.
- `file_id` (`sqlalchemy.schema.Column(sqlaclhemy.types.INTEGER)`) - ID of the file this FileAction affects.
- `file` (`sqlalchemy.orm.relationship)`) - The file this FileAction affects.
- `action_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the action affecting the file.
- `action` (`sqlalchemy.orm.relationship)`) - Action affecting the file.
- `action_time` (`sqlalchemy.schema.Column(sqlalchemy.types.DATETIME)`) - Time the action took place on.
- `user_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the user performing the action.
- `user` (`sqlalchemy.orm.relationship`) - User peforming the action.
- `computer_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the computer user performed the action on.
- `computer` (`sqlalchemy.orm.relationship`) - Computer user performed the action on.
- `session_id` (`sqlalchemy.schema.Column(sqlalchemy.types.INTEGER)`) - ID of the session user performed the action in.
- `session` (`sqlalchemy.orm.relationship`) - Session user performed the action in.

Parameters

- **fileobject** (`models.File`) – The file which is subjected to the action.
- **action** (`models.Action`) – The action which is applied to the file.
- **session** (`models.Session`) – The session in which the FileAction took place on.

- **computer** (`models.Computer`) – The computer from which the user performed the action.
- **user** (`models.User`) – The user performing the action.

class `models.`**`Project`**(*name*, *company*, *password*)

A class representation of a project.

Fields:

- `id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of project, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of the project (Max 50 characters).
- `company_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the company that owns the project.
- `company (sqlalchemy.orm.relationship)` - The company that owns the project.
- `dir (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Directory path for the project files (Max 255 characters).
- `password (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Password for the project (Max 40 characters).
- `members (sqlalchemy.orm.relationship)` - The users that work on the project.

Parameters

- **name** (`String`) – Name of the project.
- **company** (`models.Company`) – The owner of the project.

class `models.`**`Session`**(*project*)

A class representation of a session.

Fields:

- `id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of session, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of session (Max 50 characters).
- `project_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the project the session belongs to.
- `project (sqlalchemy.orm.relationship)` - The project the session belongs to.
- `starttime (sqlalchemy.schema.Column(sqlalchemy.types.DATETIME))` - Time the session began, defaults to *now()*.
- `endtime (sqlalchemy.schema.Column(sqlalchemy.types.DATETIME))` - The time session ended.
- `previous_session_id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the previous session.
- `previous_session (sqlalchemy.orm.relationship)` - The previous session.
- `participants (sqlalchemy.orm.relationship)` - Users that belong to this session.
- `computers (sqlalchemy.orm.relationship)` - Computers that belong to this session.

Parameters **project** (`models.Project`) – The project for the session.

`AddUser`(*user*)
Add users to a session.

Parameters **user** (`models.User`) – User to be added into the session.

`FileRoutine`()
File checking routine for logging.

Throws IOError When log.txt is not available for write access.

`GetLastChecked`()
Fetch `last_checked` field.

Returns `last_checked` field (None before `models.Session.start()` is called).

Return type `datetime.datetime` or `None`

`Start`()
Start a session. Set the `last_checked` field to current DateTime.

class `models.`**`User`**(*name*, *company*)
A class representation of a user.

Fields:

- `id(sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - ID of the user, used as primary key in database table.
- `name (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Name of the user (Max 50 characters).
- `email (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Email address of the user (Max 100 characters).
- `title (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Title of the user in the company (Max 50 characters).
- `department (sqlalchemy.schema.Column(sqlalchemy.types.String))` - Department of the user in the company (Max 100 characters).
- `company_id (sqlalchemy.schema.Column(sqlalchemy.types.INTEGER))` - Company id of the employing company.
- `company (sqlalchemy.orm.relationship)` - Company relationship.

Parameters

- **name** (`String`) – Name of the user.
- **company** (`models.Company`) – The employer.

1.11 Setup module

Created on 8.5.2012

author nick26

synopsis This file is used to compile a DiWaCS.exe file out of the python project using py2exe and setuptools packages available at: pypi.python.org/pypi/setuptools

1.12 Models state

Created on 4.7.2013

@author: neriksso

class `state.`**`State`**(*parent*)

classdocs

`end_current_project`()

End the current project.

`end_current_session`()

End the current session.

`get_random_responsive`()

Get a random node amongst all the responsive nodes.

`handle_file_send`(*filenames*, *progressdialog=None*)

Sends a file link to another node.

First parses all the files and folder structure, then confirms weather the users wishes to add the items to project before beginning the copy routine.

The copy routine first creates all the needed subfolders and then sums up all the file sizes to be copied. Then it will update the dialog in the beginning/end of every file transaction and whenever there's been more than 1 second from the last update dialog update. Assuming the progressdialog parameter has been given.

The progress dialog, if supplied, is updated as follows:

- If there's less than DEF_FILES (**40**) files the dialog will not be shown or updated.
- If the data size sum is less than DEF_SIZE (**2 MB**) the dialog will not be shown or updated.
- Title will contain the total percentage of data transfer.
- Message will contain the percentage of current file transfer.
- Progress bar is set to percent [0, 100] of the total data transfer.

Parameters

- **filenames** (*List of String*) – All the files/folders to be copied.
- **progressdialog** (`wx.ProgressDialog`) – The progress dialog to update (optional).

`initialize`()

Docstring.

`message_handler`(*message*)

Message handler for received messages.

Parameters **message** (an instance of `swnp.Message`) – Received message.

`on_project_selected`()

Docstring.

`remove_observers`()

Docstring.

`set_current_project`(*project_id*)

Start current project loop.

Parameters **project_id** (*Integer*) – The project id from database.

`set_current_session(session_id)`
Set current session.

Parameters **session_id** (*Integer*) – a session id from database.

`set_observers()`
Docstring.

`set_project_observer()`
Observer for file changes in project directory.

`set_responsive()`
Docstring.

`set_scan_observer()`
Observer for created files in scanned or taken with camera.

`start_audio_recorder()`
Starts the audio recorder thread.

`start_current_project()`
Start current project loop.

`start_current_session()`
Start current project loop.

`start_new_session()`
Start a new session.

`stop_responsive()`
Docstring.

`swnp_send(node, message)`
Sends a message to the node.

Parameters

- **node** (*String*) – The node for which to send a message.
- **message** (*String*) – The message.

`state.create_config()`
Creates a config file.

`state.initialization_test()`
Docstring.

`state.load_config()`
Loads a config file or creates one.

1.13 SWNP module

Created on 30.4.2012

author neriksso

class `swnp.Message(tag, prefix, payload)`
A class representation of a Message.

Messages are divided into three parts: tag, prefix, payload. Messages are encoded to json for transmission.

Parameters

- **tag** (*String*) – tag of the message.
- **prefix** (*String*) – prefix of the message.
- **payload** (*String*) – payload of the message.

static `from_json` (*json_dict*)

Return a message from json.

Parameters **json_dict** (*json*) – The json.

Returns Initializes a message from JSON object.

Return type `swnp.Message`.

static `to_dict` (*msg*)

Return a message in a dict.

Parameters **msg** (`swnp.Message`) – The message.

Returns Dictionary representation of the message.

Return type Dict

class `swnp.`**`Node`** (*node_id*, *screens*, *name=None*, *data=None*)

A class representation of a node in the network.

Parameters

- **node_id** (*Integer*) – Node id.
- **screens** (*Integer*) – Amount of visible screens.
- **name** (*String*) – The name of the node.

`get_age`()

Return the elapsed time since last refresh.

`refresh`()

Updates the timestamp.

class `swnp.`**`SWNP`** (*pgm_group*, *screens=0*, *name=None*, *node_id=None*, *context=None*, *error_handler=None*)

The main class of swnp.

This class has the required ZeroMQ bindings and is responsible for communicating with other instances.

Warning: Only one instance per computer

Parameters

- **pgm_group** (*Integer*) – The Multicast Group this node wants to be a part of.
- **screens** (*Integer*) – The number of visible screens. Defaults to 0.
- **name** (*String*) – The name of the instance.
- **node_id** (*Integer*) – ID of the current instance.
- **context** (`zmq.Context`) – ZeroMQ context to use.
- **error_handler** (`wos.CONN_ERR_TH`) – Error handler for the init constructor.

`close`()

Closes all connections and exits.

`do_ping`()
Send a PING message to the network.

`find_node`(*node_id*)
Search the node list for a specific node.

Parameters **node_id** (*Integer*) – The id of the searched node.

Return type `swnp.Node`

`get_buffer`()
Gets the buffered messages and returns them

Returns JSON formated string.

Return type String

`get_list`()
Returns a list of all nodes

Return type list

`get_screen_list`()
Returns a list of screens nodes.

Return type list.

`ping_handler`(*payload*)
A handler for PING messages. Sends update_screens, if necessary.

Parameters **payload** (*String*) – The payload of a PING message.

`ping_routine`(*error_handler*)
A routine for sending PING messages at regular intervals.

`send`(*tag*, *prefix*, *message*)
Send a message to the network.

Parameters

- **tag** (*String*) – The tag of the message; recipient.
- **prefix** (*String*) – The prefix of the message.
- **message** (*String*) – The payload of the message.

`set_name`(*name*)
Sets the name for the instance.

Parameters **name** (*String*) – New name of the instance.

`set_responsive`(*responsive*)
Sets the responsive flag for the instance.

Parameters **responsive** (*Integer*) – New number of screens.

`set_screens`(*screens*)
Sets the number of screens for the instance.

Parameters **screens** (*Integer*) – New number of screens.

`shutdown`()
Shuts down all connections, no exit.

static `start_sub_routine`(*target*, *routine*, *name*, *args*)
A wrapper for starting up subroutine threads.

Parameters

- **target** (`threading.Thread`) – Variable that contains the current thread for routine.
- **routine** – The routine to run.
- **name** (*String*) – Name of the routine.
- **args** (*List*) – Arguments for the routine.

Returns The thread of subroutine.

Return type `threading.Thread`

`sub_routine`(*sub_urls*)
Subscriber routine for the node ID.

Parameters **sub_urls** (*List of Strings*) – Subscribing URLs.

`sub_routine_sys`(*sub_urls*)
Subscriber routine for the node ID.

Parameters **sub_urls** (*List of Strings*) – Subscribing URLs.

`sys_handler`(*msg*)
Handler for "SYS" messages.

Parameters **msg** (`swnp.Message`) – The received message.

`timeout_routine`()
Routine for checking node list and removing nodes with timeout.

`swnp.`**`set_logger_level`**(*level*)
Sets the logger level for swnp logger.

Parameters **level** (*Integer*) – Level of logging.

1.14 Testing module

Created on 20.5.2013

author Kristian

1.15 Threads package

Set of threading functionality.

1.15.1 threads.audiorecorder module

Created on 5.6.2013

author neriksso

class `threads.audiorecorder.`**`AudioRecorder`**(*parent*)
A thread for capturing audio continuously. It keeps a buffer that can be saved to a file. By convention AudioRecorder is usually written in mixed case even as we prefer upper case for threading types.

Parameters **parent** (`threading.Thread`) – Parent of the thread.

find_input_device()
Find the microphone device.

open_mic_stream()
Opens the stream object for microphone.

run()
Continuously record from the microphone to the buffer.

If the buffer is full, the first frame will be removed and the new block appended.

save(*ide*, *path*)
Save the buffer to a file.

stop()
Stop audio recorder.

threads.audiorecorder.**logger**()
Get the common logger.

1.15.2 threads.checkupdate module

Created on 5.6.2013

author neriksso

class threads.checkupdate.**CHECK_UPDATE**
Thread for checking version updates.

static get_pad()
Returns the padfile object using PAD_URL setting.

Returns A Filelike object with additional methods geturl(), info() and getcode().

run()
Returns weather the update checking was successful.

Return type Boolean

show_dialog(*url*)
Shows the dialog that promps the user to download newer version of the software.

Parameters **url** (*String*) – URL address of the new version.

threads.checkupdate.**logger**()
Get the common logger.

1.15.3 threads.common module

Created on 5.6.2013

author neriksso

1.15.4 threads.connectionerror module

Created on 5.6.2013

author neriksso

class `threads.connectionerror.`**`CONNECTION_ERROR_THREAD`**(*parent*)

Thread for checking connection errors.

Parameters **parent** (*wx.Frame*) – Parent object.

`run`()

Starts the thread.

`threads.connectionerror.`**`logger`**()

Get the common logger.

1.15.5 threads.contextmenu module

Created on 27.6.2013

author neriksso

class `threads.contextmenu.`**`ContextMenuFailure`**(*self*, *EventType type=wxEVT_NULL*, *int winid=0*)

Represents a failure of CMFH initialization.

class `threads.contextmenu.`**`SEND_FILE_CONTEX_MENU_HANDLER`**(*parent*, *context*, *send_file*, *handle_file*)

Thread for OS context menu actions like file sending to other node.

Parameters

- **context** (`zmq.Context`) – ZeroMQ Context for creating sockets.
- **send_file** (*Function*) – Sends files.
- **handle_file** (*Function*) – Handles files.

`run`()

Starts the thread.

`stop`()

Stops the thread.

`threads.contextmenu.`**`logger`**()

Get the common logger.

1.15.6 threads.current module

Created on 27.6.2013

author neriksso

class `threads.current.`**`CURRENT_PROJECT`**(*swnp*)

Thread for transmitting current project selection. When user selects a project, an instance is started. When a new selection is made, by any DiWaCS instance, the old instance is terminated.

Parameters

- **project_id** (*Integer*) – Project id from the database.
- **swnp** (`swnp.SWNP`) – SWNP instance for sending data to the network.

`run`()

Starts the thread.

class `threads.current.`**`CURRENT_SESSION`**`(`*swnp*`)`
Thread for transmitting current session id, when one is started by the user. When the session is ended, by any DiWaCS instance, the instance is terminated.

Parameters

- **session_id** (*Integer*) – Session id from the database.
- **swnp** (`swnp.SWNP`) – SWNP instance for sending data to the network.

`run``()`
Starts the thread.

`threads.current.`**`logger`**`()`
Get the common logger.

1.15.7 threads.diwathread module

Created on 5.6.2013

author neriksso

class `threads.diwathread.`**`DIWA_THREAD`**`(`*target=None*, *name=None*, *args=()*, *kwargs=None*`)`
Doc string here.

`stop``()`
Stop the thread.

static **`stop_all`**`()`
Stop all program threads except the calling one.

`stop_is_set``()`
Is the thread supposed to stop.

exception `threads.diwathread.`**`TimeoutException`**`(`*message*`)`
Represents a thread timeout event.

`threads.diwathread.`**`logger`**`()`
Get the common logger.

1.15.8 threads.inputcapture module

Created on 5.6.2013

author neriksso

class `threads.inputcapture.`**`INPUT_CAPTURE`**`(`*parent*, *swnp*`)`
Thread for capturing input from mouse/keyboard.

Parameters

- **parent** (`GUI`) – Parent instance.
- **swnp** (`swnp.SWNP`) – SWNP instance for sending data to the network.

`hook``()`
Docstring here.

`on_keyboard_event``(`*event*`)`
Called when keyboard events are received.

on_mouse_event (*event*)
Called when mouse events are received.

- WM_MOUSEFIRST = 0x200
- WM_MOUSEMOVE = 0x200
- WM_LBUTTONDOWN = 0x201
- WM_LBUTTONUP = 0x202
- WM_LBUTTONDBLCLK = 0x203
- WM_RBUTTONDOWN = 0x204
- WM_RBUTTONUP = 0x205
- WM_RBUTTONDBLCLK = 0x206
- WM_MBUTTONDOWN = 0x207
- WM_MBUTTONUP = 0x208
- WM_MBUTTONDBLCLK = 0x209
- WM_MOUSEWHEEL = 0x20A
- WM_MOUSEHWHEEL = 0x20E

reset_mouse_events ()
Docstring here.

run ()
Starts the thread.

stop ()
Stops the thread.

unhook ()
Docstring here.

class threads.inputcapture.**MOUSE_CAPTURE** (*parent*, *swnp*)
Docstring.

parse_mouse_events ()
Docstring here.

threads.inputcapture.**logger** ()
Get the common logger.

threads.inputcapture.**set_capture** (*value*)
Set's the capture value for threads.

Parameters **value** (*Boolean*) – Is the capture on.

1.15.9 threads.worker module

Created on 27.6.2013

author neriksso

class threads.worker.**WORKER_THREAD** (*parent*)
Worker thread for non-UI jobs.

static **`add_project_registry_entry`**(*reg_type*)
Adds "Add to project" context menu item to registry. The item will be added to Software-Classes<reg_type>, where <reg_type> can be e.g. '*' for all files or 'Folder' for folders.

Parameters **reg_type** (*String*) – Registry type.

static **`add_registry_entry`**(*name*, *node_id*)
Adds a node to registry.

Parameters

- **name** (*String*) – Node name.
- **id** (*Integer*) – Node id.

`check_responsive`()
Docstring here.

`create_event`(*title*)
Docstring here.

`parse_config`(*config_object*)
Handles config file settings.

static **`remove_all_registry_entries`**()
Removes all related registry entries.

`run`()
Run the worker thread.

`threads.worker.`**`logger`**()
Get the common logger.

1.16 Utils module

Recreated on 17.5.2013

author neriksso

`utils.`**`DottedIPToInt`**(*dotted_ip*)
Transforms a dotted IP address to Integer.

Parameters **dotted_ip** (*String*) – The IP address.

Returns The IP address.

Return type Integer

`utils.`**`GetLANMachines`**(*lan_ip*)

Parameters **lan_ip** (*string*) – Local Area Network IP.

Returns lan machines

Return type string[]

`utils.`**`GetLocalIPAddress`**(*target*)
Used to get local Internet Protocol address.

Returns The current IP address.

Return type string

`utils.`**`GetMacForIp`**(*ip*)
Returns the mac address for an local IP address.

Parameters **ip** (*String*) – IP address

`utils.`**`IntToDottedIP`**(*intip*)
Transforms an Integer IP address to dotted representation.

Parameters **intip** (*Integer*) – The IP

Returns The IP

Return type string

`utils.`**`IterIsLast`**(*iterable*) → generates (item, islast) pairs.
Generates pairs where the first element is an item from the iterable source and the second element is a boolean flag indicating if it is the last item in the sequence.

Parameters **iterable** (*iterable*) – The iterable element.

`utils.`**`MapNetworkShare`**(*letter*, *share=None*)
Maps the network share to a letter.

Parameters

- **letter** (*String*) – The letter for which to map.
- **share** (*String*) – The network share, defaults to None which unmaps the letter.

`utils.`**`check_project_password`**(*project_id*, *password*)
Compares the the provided password with the project password.

`utils.`**`get_encrypted_directory_name`**(*name*, *hashed_password*)
Returns the encrypted name for project directory.

`utils.`**`hash_password`**(*password*)
Hashes the provided password.

`utils.`**`set_logger_level`**(*level*)
Docstring here.

CHAPTER

TWO

BUGS

Bug	Description	Status
Sample bug	Description for sample	Open / Closed / Will not be fixed

CHAPTER THREE

FEATURES

Feature	Description
Project	User can add, edit and select a project
Session	User can start, end and continue sessions
Event	User can tag an interesting event during a session
File Monitoring	Users' file actions are monitored during a session. It includes opening files.

CHAPTER FOUR

LICENSE

European Union Public Licence

22. 1.1

EUPL © the European Community 2007

This European Union Public Licence (the "EUPL") applies to the Work or Software (as defined below) which is provided under the terms of this Licence. Any use of the Work, other than as authorised under this Licence is prohibited (to the extent such use is covered by a right of the copyright holder of the Work).

The Original Work is provided under the terms of this Licence when the Licensor (as defined below) has placed the following notice immediately following the copyright notice for the Original Work:

> Licensed under the EUPL V.1.1

or has expressed by any other mean his willingness to license under the EUPL.

4.1 1. Definitions

In this Licence, the following terms have the following meaning:

- **The Licence:** This Licence.
- **The Original Work or the Software:** The software distributed and/or communicated by the Licensor under this Licence, available as Source Code and also as Executable Code as the case may be.
- **Derivative Works:** The works or software that could be created by the Licensee, based upon the Original Work or modifications thereof. This Licence does not define the extent of modification or dependence on the Original Work required in order to classify a work as a Derivative Work; this extent is determined by copyright law applicable in the country mentioned in Article 15.
- **The Work:** The Original Work and/or its Derivative Works.
- **The Source Code:** The human-readable form of the Work which is the most convenient for people to study and modify.
- **The Executable Code:** Any code which has generally been compiled and which is meant to be interpreted by a computer as a program.
- **The Licensor:** The natural or legal person that distributes and/or communicates the Work under the Licence.
- **Contributor(s):** Any natural or legal person who modifies the Work under the Licence, or otherwise contributes to the creation of a Derivative Work.
- **The Licensee or "You":** Any natural or legal person who makes any usage of the Software under the terms of the Licence.

- **Distribution and/or Communication:** Any act of selling, giving, lending, renting, distributing, communicating, transmitting, or otherwise making available, on-line or off-line, copies of the Work or providing access to its essential functionalities at the disposal of any other natural or legal person.

4.2 2. Scope of the rights granted by the Licence

The Licensor hereby grants You a world-wide, royalty-free, non-exclusive, sublicensable licence to do the following, for the duration of copyright vested in the Original Work:

- use the Work in any circumstance and for all usage,
- reproduce the Work,
- modify the Original Work, and make Derivative Works based upon the Work,
- communicate to the public, including the right to make available or display the Work or copies thereof to the public and perform publicly, as the case may be, the Work,
- distribute the Work or copies thereof,
- lend and rent the Work or copies thereof,
- sub-license rights in the Work or copies thereof.

Those rights can be exercised on any media, supports and formats, whether now known or later invented, as far as the applicable law permits so.

In the countries where moral rights apply, the Licensor waives his right to exercise his moral right to the extent allowed by law in order to make effective the licence of the economic rights here above listed.

The Licensor grants to the Licensee royalty-free, non exclusive usage rights to any patents held by the Licensor, to the extent necessary to make use of the rights granted on the Work under this Licence.

4.3 3. Communication of the Source Code

The Licensor may provide the Work either in its Source Code form, or as Executable Code. If the Work is provided as Executable Code, the Licensor provides in addition a machine-readable copy of the Source Code of the Work along with each copy of the Work that the Licensor distributes or indicates, in a notice following the copyright notice attached to the Work, a repository where the Source Code is easily and freely accessible for as long as the Licensor continues to distribute and/or communicate the Work.

4.4 4. Limitations on copyright

Nothing in this Licence is intended to deprive the Licensee of the benefits from any exception or limitation to the exclusive rights of the rights owners in the Original Work or Software, of the exhaustion of those rights or of other applicable limitations thereto.

4.5 5. Obligations of the Licensee

The grant of the rights mentioned above is subject to some restrictions and obligations imposed on the Licensee. Those obligations are the following:

Attribution right: the Licensee shall keep intact all copyright, patent or trademarks notices and all notices that refer to the Licence and to the disclaimer of warranties. The Licensee must include a copy of such notices and a copy of the Licence with every copy of the Work he/she distributes and/or communicates. The Licensee must cause any Derivative Work to carry prominent notices stating that the Work has been modified and the date of modification.

Copyleft clause: If the Licensee distributes and/or communicates copies of the Original Works or Derivative Works based upon the Original Work, this Distribution and/or Communication will be done under the terms of this Licence or of a later version of this Licence unless the Original Work is expressly distributed only under this version of the Licence. The Licensee (becoming Licensor) cannot offer or impose any additional terms or conditions on the Work or Derivative Work that alter or restrict the terms of the Licence.

Compatibility clause: If the Licensee Distributes and/or Communicates Derivative Works or copies thereof based upon both the Original Work and another work licensed under a Compatible Licence, this Distribution and/or Communication can be done under the terms of this Compatible Licence. For the sake of this clause, "Compatible Licence" refers to the licences listed in the appendix attached to this Licence. Should the Licensee's obligations under the Compatible Licence conflict with his/her obligations under this Licence, the obligations of the Compatible Licence shall prevail.

Provision of Source Code: When distributing and/or communicating copies of the Work, the Licensee will provide a machine-readable copy of the Source Code or indicate a repository where this Source will be easily and freely available for as long as the Licensee continues to distribute and/or communicate the Work.

Legal Protection: This Licence does not grant permission to use the trade names, trademarks, service marks, or names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the copyright notice.

4.6 6. Chain of Authorship

The original Licensor warrants that the copyright in the Original Work granted hereunder is owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each Contributor warrants that the copyright in the modifications he/she brings to the Work are owned by him/her or licensed to him/her and that he/she has the power and authority to grant the Licence.

Each time You accept the Licence, the original Licensor and subsequent Contributors grant You a licence to their contributions to the Work, under the terms of this Licence.

4.7 7. Disclaimer of Warranty

The Work is a work in progress, which is continuously improved by numerous contributors. It is not a finished work and may therefore contain defects or "bugs" inherent to this type of software development.

For the above reason, the Work is provided under the Licence on an "as is" basis and without warranties of any kind concerning the Work, including without limitation merchantability, fitness for a particular purpose, absence of defects or errors, accuracy, non-infringement of intellectual property rights other than copyright as stated in Article 6 of this Licence.

This disclaimer of warranty is an essential part of the Licence and a condition for the grant of any rights to the Work.

4.8 8. Disclaimer of Liability

Except in the cases of wilful misconduct or damages directly caused to natural persons, the Licensor will in no event be liable for any direct or indirect, material or moral, damages of any kind, arising out of the Licence or of the use of the Work, including without limitation, damages for loss of goodwill, work stoppage, computer failure or malfunction, loss of data or any commercial damage, even if the Licensor has been advised of the possibility of such damage. However, the Licensor will be liable under statutory product liability laws as far such laws apply to the Work.

4.9 9. Additional agreements

While distributing the Original Work or Derivative Works, You may choose to conclude an additional agreement to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or services consistent with this Licence. However, in accepting such obligations, You may act only on your own behalf and on your sole responsibility, not on behalf of the original Licensor or any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against such Contributor by the fact You have accepted any such warranty or additional liability.

4.10 10. Acceptance of the Licence

The provisions of this Licence can be accepted by clicking on an icon “I agree” placed under the bottom of a window displaying the text of this Licence or by affirming consent in any other similar way, in accordance with the rules of applicable law. Clicking on that icon indicates your clear and irrevocable acceptance of this Licence and all of its terms and conditions.

Similarly, you irrevocably accept this Licence and all of its terms and conditions by exercising any rights granted to You by Article 2 of this Licence, such as the use of the Work, the creation by You of a Derivative Work or the Distribution and/or Communication by You of the Work or copies thereof.

4.11 11. Information to the public

In case of any Distribution and/or Communication of the Work by means of electronic communication by You (for example, by offering to download the Work from a remote location) the distribution channel or media (for example, a website) must at least provide to the public the information requested by the applicable law regarding the Licensor, the Licence and the way it may be accessible, concluded, stored and reproduced by the Licensee.

4.12 12. Termination of the Licence

The Licence and the rights granted hereunder will terminate automatically upon any breach by the Licensee of the terms of the Licence.

Such a termination will not terminate the licences of any person who has received the Work from the Licensee under the Licence, provided such persons remain in full compliance with the Licence.

4.13 13. Miscellaneous

Without prejudice of Article 9 above, the Licence represents the complete agreement between the Parties as to the Work licensed hereunder.

If any provision of the Licence is invalid or unenforceable under applicable law, this will not affect the validity or enforceability of the Licence as a whole. Such provision will be construed and/or reformed so as necessary to make it valid and enforceable.

The European Commission may publish other linguistic versions and/or new versions of this Licence, so far this is required and reasonable, without reducing the scope of the rights granted by the Licence. New versions of the Licence will be published with a unique version number.

All linguistic versions of this Licence, approved by the European Commission, have identical value. Parties can take advantage of the linguistic version of their choice.

4.14 14. Jurisdiction

Any litigation resulting from the interpretation of this License, arising between the European Commission, as a Licensor, and any Licensee, will be subject to the jurisdiction of the Court of Justice of the European Communities, as laid down in article 238 of the Treaty establishing the European Community.

Any litigation arising between Parties, other than the European Commission, and resulting from the interpretation of this License, will be subject to the exclusive jurisdiction of the competent court where the Licensor resides or conducts its primary business.

4.15 15. Applicable Law

This Licence shall be governed by the law of the European Union country where the Licensor resides or has his registered office. This licence shall be governed by the Belgian law if: - a litigation arises between the European Commission, as a Licensor, and any Licensee; - the Licensor, other than the European Commission, has no residence or registered office inside a European Union country.

4.16 Appendix

"Compatible Licences" according to article 5 EUPL are:

- GNU General Public License (GNU GPL) v. 2
- Open Software License (OSL) v. 2.1, v. 3.0
- Common Public License v. 1.0
- Eclipse Public License v. 1.0
- Cecill v. 2.0

CHAPTER

FIVE

USER INTERFACE

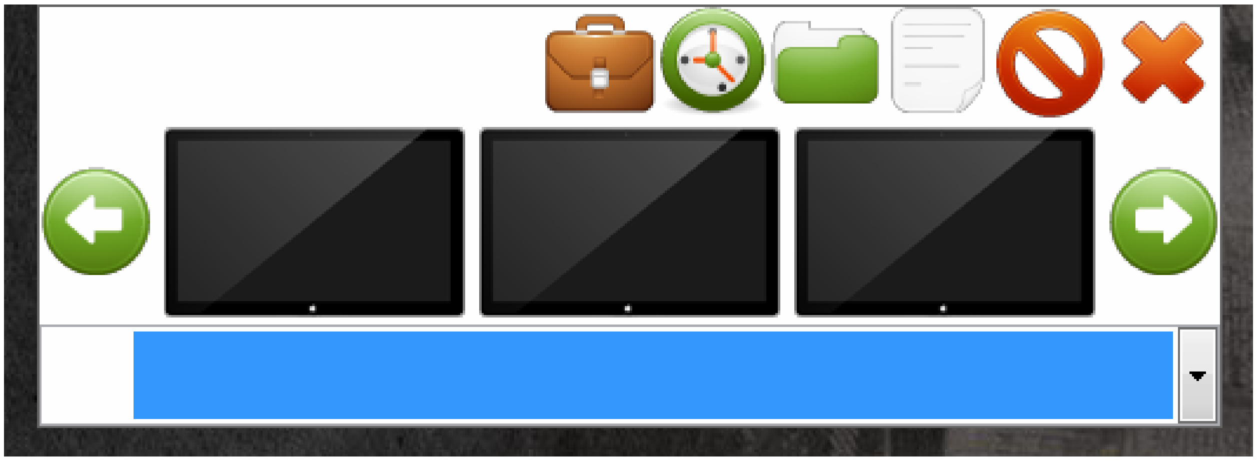

Figure 5.1: The UI of DiWaCS; Several icons for different functions.

The screen icons identify different screens nodes. The user can drop files on to these icons, causing the dropped files to be opened in the specific node. The arrows control the carousel of nodes, and are visble only if more than three nodes are connected. The drop-down list in holds recently viewed files in the selected project.

Table 5.1: Icons explained

Icon	Description
Briefcase	Select a project
Clock	Start / End a session
Folder	Open project directory
Note	Create an Event note
Circle	Hide the application
Cross	Exit the application

CHAPTER SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*