Instrumentation Group

HCx

Software Manual

Author	Revision	Date
Alexandre Gobbo	0.1.0	05.08.2008
Alexandre Gobbo	0.1.1	31.08.2008
Alexandre Gobbo	0.2.0	11.03.2010
Alexandre Gobbo	0.2.1	18.05.2010

EMBL

Table of Contents

Requirements

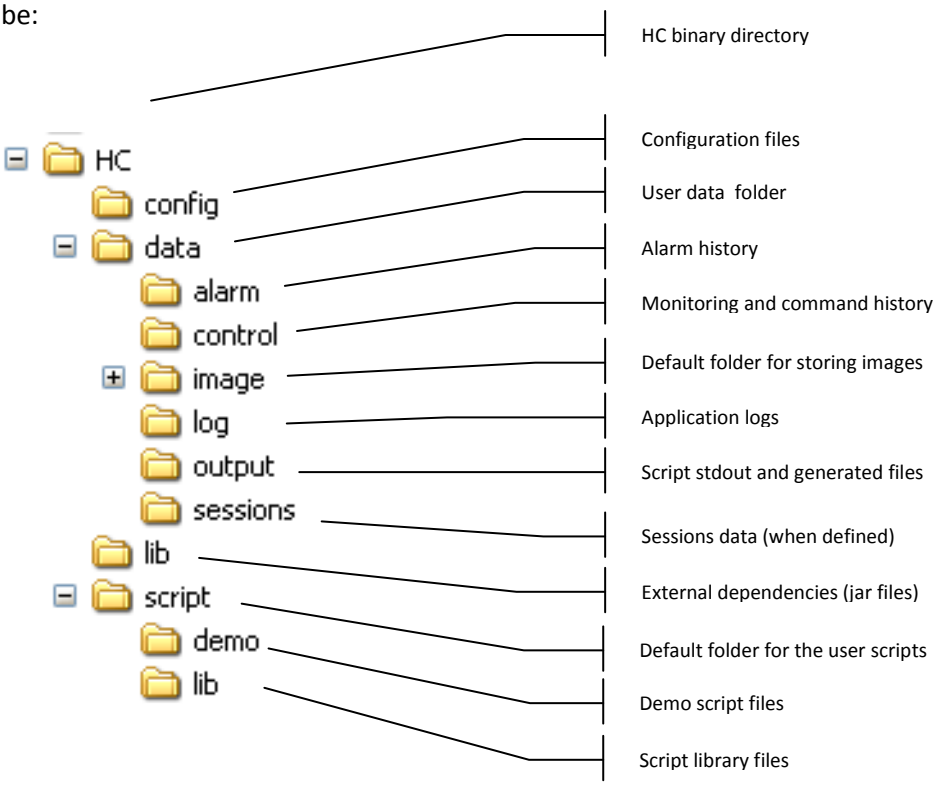
1) Hardware Requirements
    - PC (x86-compatible), recommended minimum 1GHz, 512 Mb RAM and 10 Gb free disk space.
    - One free RS232 serial port (not required for HC2).
    - Minimal screen resolution: 1024 x 768.
    - Ethernet (10BASE-T) connection (optional on HC1, mandatory on HC2):
        - Specific link through direct cross cable or
        - Through the LAN.
    - Additional hardware may be needed according to the imaging type.
        - E.g. if using a Prosilica camera, an additional Gigabit Ethernet port (in case of direct connection) or external hub/switch may be needed.

2) Software Requirements
    - Java 6 (JRE 6 Update 15 or higher).
    - Windows or Linux.
        - Java Communications API must be installed (only HC1).
        - Tested on Windows SP2, SP3, Ubuntu 8.03, 9.03.

Installation

Copy the distribution package to the destination folder.

By default the application the binaries (the jar file, the lib folder and eventually native imaging libraries) are located in the same directory as the configuration, data and script folders. In this case the directory tree will be:



Each one of the paths above may be changed to a user-defined folder. To do so the file ./config/config.xml must be edited, adding <path> entries and, below it, as needed:

- <script> (defines the application script folder).
- <scriptlib>(defines the location of script libraries).
- <data> (changes the base data path, that, by default, contains all the following ones) .
- <output> (changes the default folder for script stdout and output files) .
- <controldata> (changes the folder that stores the monitoring and command history).
- <log> (changes the folder that contains application logs).
- <alarm> (changes the folder that contains the application alarm history) .
- <image> (changes the default folder for storing images).
- <sessions> (changes the base folder to store session data (default is <data>/sessions). Data is then stored in <sessions>/sessionname.

E.g. the script and data paths can be customized in the config.xml like this:

```
</org.embl.hc.hcConfig>
…
 <path>
        <data>d:\data</data>
        <script>Z:\hc\script</script>
        <scriptlib>Z:\hc\lib</scriptlib>
 </path>
 …
</org.embl.hc.hcConfig>
```

Windows

When the distribution folder is copied, the application can be launched by double-clicking the jar file. In this way the data folder will be the same as the binary folder. Optionally different users can have separated data files. To do so a .bat file can be created, setting the current folder to the user data folder and then launching the jar file as:

java –jar < HC binary directory >\hc.jar

Linux

A application launcher batch file should be created setting the desired data folder path (eventually the same HC binary directory) and then launching the application as:

java –jar < HC binary directory >/hc.jar

The Java serial extension (Javax.comm) must be installed on the machine prior to the execution of the software. Optionally this can be done by typing, in the application binary folder:

- sudo java –jar serialsetup.jar (Linux)
- java –jar serialsetup.jar (Windows)

If the serial extension libraries are already installed this command prints a list of the available IO devices.

- On Linux machines make sure the user has access rights to the serial port (such as "/dev/tty0").

Operating Modes

1) Manual.
    - All commands are issued through the HC software GUI.

2) Automated.
    - Experiment logic implemented in scripts, executed by the HC software.
    - The functionalities available to scripts are:
        a. Get/set relative humidity and temperature values.
        b. Read drop size.
        c. Save snapshot and animations of drop images.
        d. Add logs to HC software and pop-up message boxes.
        e. Use most Python standard libraries and therefore execute local specific tasks such as start and control data collection.
        f. Access low level machine variables (advanced).

3) Remote.
    - The logic is implemented by external client software.
    - The functionalities exported by the HC software are:
        a. Get/set relative humidity.
        b. Read drop size.
        c. Save snapshot and animations of drop images.
        d. Start and control execution of script.
    - Existing protocols for remote clients:
        a. Exporter (JLib default protocol for distributed objects).
            - Client libraries in Java, Python and C++.
        b. Auto-generated Tango Server.
        c. Auto-generated Tine Server.
        d. Auto-generated Epics Server.
        e. Web service.

Imaging Support

The HC software supports several image sources to display drop images:

1) Prosilica.
    - Prosilica GigE cameras are natively supported by the application.

2) RTP.
    - Real-time Transport Protocol servers.

3) External application (Only on Windows).
    - An external application can be configured for rendering images within the HC.
    - MIL is currently supported in this mode and therefore the Matrox frame grabbers.

4) Capture Devices.
    - Capture devices (as USB cameras) are supported if recognized by the system.

5) Tango.
    - Tango video servers are supported if they export the frame as an image file.

6) Tine.
    - Tine image properties (CF_IMAGE) are supported.

7) Animation.
    - An animation can be configured for debug or demonstration purposes.

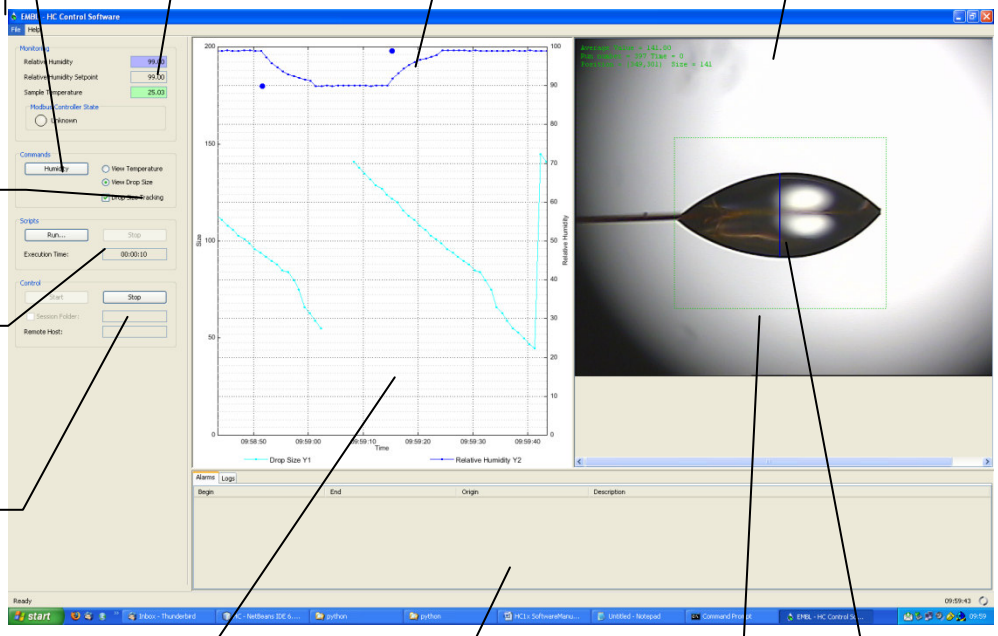
8) File.
    - Since the frame rate can be low (1fps or less), the HC software can poll an externally generated image file, reloading it when it changes.

9) Script.
    - For low frame rates a script can be used to retrieve every frame as an image file.

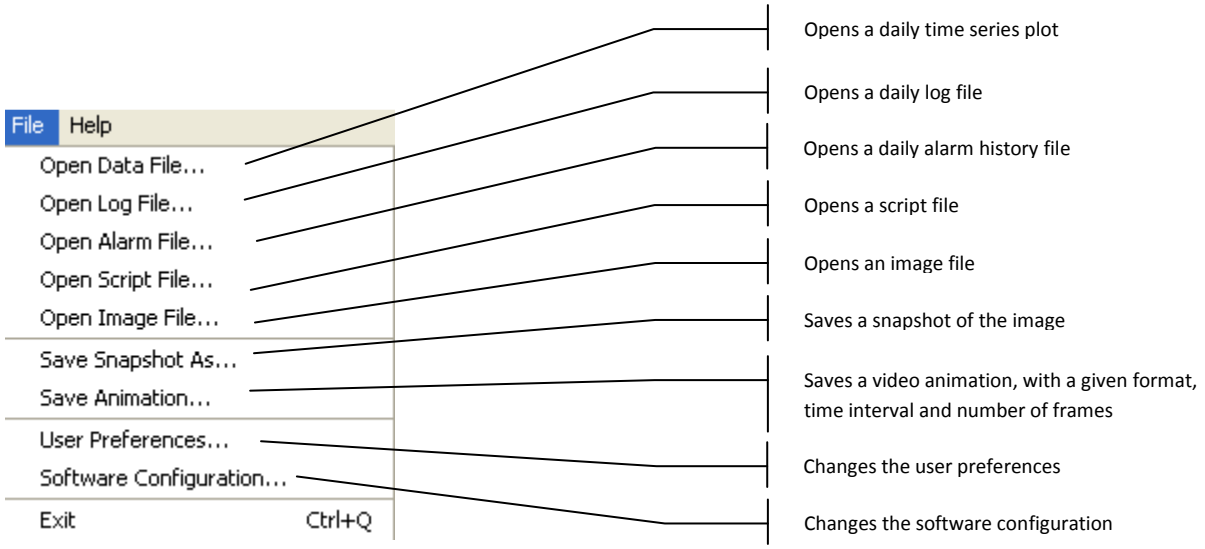
10) Streamer.
    - The built-in server for remote acquired image frames.

The HC Software GUI

The Main Window



The File Menu



The Configuration Window

The Software Configuration window contains the following tabs:

General	General application configuration of the HC model, logging level, look and feel, script communication port and simulation mode.
Serial	Configuration of the serial port (only for HC1).
Modbus	IP address of the modbus controller and its start-up parameters.
Image	Definition of the imaging source type and the parameters.
Img Proc	Definition of the image processing type and parameters.
Server	Configuration of an embedded server.

The User Preferences window presents the parameters that may be changed by users:

- Chart options.
- The ROI for drop size tracking (that may be equally changed by right-clicking the image).

Imaging Configuration Examples

The Configuration Window includes an “Image” tab for configure the image source. In there we must:

- Select the source type in the combo box.
- Edit the “Source” name field (using the “Browse” button or using the example table below).
- Edit the source’s “Pars”field, if any (meaning depends on the source type, see the table).
- In the image’s “Type” field enter the default format for saving images (as “jpg” or “png”).
- Use the “Rotate“ checkbox only if the drop image is vertical – for the drop size estimation to work since it only measures the vertical lenght.
- The “Interval” is only used in sources that are based on polling (File, Tango, Tine, Script) and in this case it means the refresh interval.

Animation	Source Name = File name (from wizard) Source Pars = Delay in seconds of the first frame (leave empty if no delay)
Application	Source Name = File name (from wizard) - as VideoServerStarter.exe Source Pars = 768 576 MORPHIS 0 False D: \image\overlays.png (parameters to the application)
Capture Device	Source Name = Device name (from wizard) Source Pars = width=X height=Y (if stretching is needed, otherwise leave it empty)
File	Source Name = File name (from wizard)
Prosilica	Source Name = Unique id (from wizard) or it can be left empty if only one camera is present Source Pars = PixelFormat=Mono8 ExposureValue=33000 (Prosilica internal parameters)
RTP	Source Name = ServerIP : ServerPort
Script	Source Name = Script name (from wizard) (argv[1] is the file to be written) Source Pars = lib folder as parameter
Streamer	Source Name = ServerIP : ServerPort
Tango	Source Name = pc251.embl.fr:20000/d14/microdiff/general (with DB) Source Name = tango://localhost:9999/embl/microdiff/general#dbase=no (Without DB) Source Pars = GrabImageEx 0,0,jpeg (“GrabImageEx” method and its atributes) Source Pars = ATT_JpegImage (JpegImage attribute – for attributes use the prefix “ATT_”)
Tine	Source Name = /DEFAULT/SGP_FAKEIMAGE/Output Source Pars = Frame

Scripts

Scripts are a convenient way to perform long custom experiments without requiring user intervention. The HC software scripts are Jython scripts. They should be stored in the script folder, and they can be started and stopped by the user interface.

Most of standard Python libraries can be used in the scripts. In fact, these scripts are Python compatible and can even be executed in Python, on the local or on a remote machine. In the script/lib folder, additional script libraries are stored. Any python IDE can be used to debug the scripts, providing:

- The script/lib folder is added to the Python path.
- The scripts call requestRemoteControl() at the beginning and releaseRemoteControl() at the end (see demo_simple_gradient.py). This is necessary because the HC software only allows one remote script at a time.

The communication with HC software is done through the library file hc.py. It exposes the functionalities available to scripts as Python functions such as:

a. Get/set relative humidity and temperature values.
b. Read drop size.
c. Save snapshot and animations of drop images.
d. Add logs and pop-up message boxes.
e. Access low-level machine parameters.

The script/demo folder contains several demonstration scripts that can be used as a reference for creating new scripts.

Since they are generic Python scripts, any functionality can be added, such as triggering data collection or retrieving or processing generated data. Scripts to perform such tasks can be created and extend the script library folder with the local specific needs.

- By default the HC is distributed with Jython 2.2.1, due to its small size, but the “jython.jar” file inside <HC binary directory>\lib can be replaced with Jython 2.5’s “jython.jar”. Jython 2.5 offers support to more Python standard libraries and solves some issues.

Script Interface

The script interface to the HC software is defined in the hc.py module. The available functions are:

log(log,level=INFO)	Registers a log in the HC software log=String to be logged level = FINE, INFO,W ARNING, or ERROR
showMessageBox(msg)	Displays a message box on the HC software
showOptionBox(msg, option)	Displays a message box with multiple choice buttons (OPTION_TYPE_YES_NO, OPTION_TYPE_YES_NO_CANCEL, OPTION_TYPE_OK_CANCEL) Returns the user choice (RESULT_YES, RESULT_NO, RESULT_CANCEL, RESULT_OK, RESULT_CLOSED)
showInputBox(msg)	Displays an input box on the HC software. Returns the string entered by the user (or None)
getWorkingPath()	Returns the script output path (the session path, if defined)
requestRemoteControl()	First statement on remote scripts to get control on the HC
releaseRemoteControl()	Last statement on remote scripts to release control
getRH()	Returns the current sample relative humidity
setRH(val)	Changes the relative humidity setpoint
getSetpointRH()	Returns the relative humidity setpoint
getSampleTemp()	Returns the current sample temperature
setSampleTemp(val)	Changes the sample temperature setpoint (only on HC2)
getSetpointSampleTemp()	Returns the sample temperature setpoint (only on HC2)
getDewPointTemp()	Returns the dew point temperature
setDewPointTemp(val)	Changes the dew point temperature setpoint
getSetpointDewPointTemp()	Returns the current dew point temperature
getWaterHeaterTemp()	Returns the current water heater temperature
setWaterHeaterTemp(val)	Changes the water heater temperature setpoint
getSetpointWaterHeaterTemp()	Returns the current sample temperature
getModbusControllerState()	Gets the state of the modbus controller (“Initializing”, “"Heating", "Running", "Force On", "Force Off", "Error" or "Changing Temperature")
setModbusControllerState(val)	Sets the state of Modbus controller
setWaterHeaterPIDParameters(pars)	Changes the water heater PID parameters
saveSnapshot(filename)	Saves an image snapshot in a given filename
saveAnimation(no_images,interval,prefix,format)	Saves an animation with the given parameters
getDropSize()	Returns the size of the drop (or None if not detected)

Remote Interface

The methods available to remote control are:

State getState()	Return the state of the HC (Initializing, Communication Error, Ready, Busy, Running, Stopped, Starting or Remote)
String getStatus()	Returns the status string of the application
double getRH()	Returns the current sample relative humidity
void setRH(double value)	Changes the relative humidity setpoint
double getSetpointRH()	Returns the relative humidity setpoint
void setSetpointRH(double value)	Changes the relative humidity setpoint (same as setRH)
double getSampleTemp ()	Returns the current sample temperature
void setSampleTemp (double value)	Changes the sample temperature setpoint (only on HC2)
double getSetpointSampleTemp ()	Returns the sample temperature setpoint (only on HC2)
void setSetpointSampleTemp (double value)	Changes the temperature setpoint (same as setSampleTemp)
State getModbusControllerState()	Gets the state of the modbus controller (“Initializing”, “"Heating", "Running", "Force On", "Force Off", "Error" or "Changing Temperature")
Double getDropSize()	Returns the size of the drop (or null if not detected)
resetDropReferenceSize()	If using relative sizes for drop estimation, this commend sets next drop size measure to 100
String getSession()	Return name of the current session (or null if not defined)
void setSession (String session_name)	Changes the current session.
void setROI(int[] location)	Changes the drop size detection ROI
String execScript(String filename, String args)	Starts the execution of a script: Filename= name of the script in the script path. Args=comma separated string with the script parameters The script will be executing while the application remains in state “Running”
void stopScript()	Stops the execution of the current script
String getScriptOutputPath()	Returns the script output path (the session path, if defined)
String getTaskOutputStr()	If script succeeds contains its return if any
String getTaskExceptionStr()	If script fails contains its exception description
void saveSnapshot (String filename)	Saves an image snapshot in a given filename
void saveAnimation (int no_images, int interval, String folder, String prefix,String format)	Saves an animation with the given parameters

Notes:

- On Tango, Tine and Epics servers the get/set methods are translated into an attribute (Tango), property (Tine) or process variable (Epics). It has the same name of the method excluding the get/set prefix.
- If only the set method is present the attribute/property/PV is “write-only”. If only the get method is present it is “read-only”. If both exist then it is “read-write”.
- The non get/set methods (generic methods) are translated into a method receiving an array of string as parameters (Tango), a string with write/read access (Tine) or a PV of type array of strings (Epics).