



Python Scripting guide

rev. 4

Feasa Python scripts · reference guide

by R&D Engineer Carles Martínez Rius, rev 4, © Feasa Enterprises Ltd.



0. Introduction

Feasa is proud to introduce a new feature added to the Feasa Terminal program which will allow to create Python scripts to extend the capabilities of the Terminal program.

Current script files are a handy tool to create macro-like scripts, in order to automate some repetitive tasks that could be easily executed and accessed from a script file (*.fcs). However, despite these scripts files are really useful they lack some basic instructions that any other programming language can offer like flow control (loops, conditionals, etc), file access, external libraries, etc.

In order to improve the user experience, a Python scripting system has been embedded in the Terminal program, while still keeping the old scripting engine to allow backwards compatibility.

All the Commands from the previous Scripting engine are still available in the Python scripts, through a class called “Feasa”. Please, read the following sections to find out more about the syntax and programming examples.

Why Python?

Python has become one of the most popular interpreted languages, highly adopted by the community and has been ported to many platforms and operating systems. Nowadays is integrated by default in many of those platforms and has become a common language to program software plug-ins or applications extensions.

1. Usage

Python is an object oriented language, so the native static class *Feasa* has been created to access to all the functions available in the Terminal program (functions available in the old *fcs* scripts). There is no need to import any module or library, since the class *Feasa* has been embedded in the source code as a native object.

Almost any native Python keyword or function can be accessed from the Feasa Terminal Python scripts, but external modules like “ctypes” need to be imported, and consequently the corresponding libraries need to be installed. So in order to have full Python support it is highly recommended to install the Python system/libraries from the official website.

The process of creating a Python script for the Terminal is very simple: any text editor can be used to write the source code of the script and then the file should be saved in “*.py” extension.

2. Functions and syntax

2.1 Logging

Print

This function prints a number or string to the default output stream (console or graphical mode)

Syntax: `Print(data [, color])`

Arguments:

- `data`: the number or string to be printed. The output stream will be the Messages box in GUI mode and the console in command-line/console mode.
- `color`: (optional parameter) used to set the color of the output text. Different colors are available: [blue, red, green, yellow, orange, purple, pink, gray and black] for the Windows/GUI mode and [blue, red, green, yellow, cyan, magenta, black, gray, and white] for the console/DOS mode.

Returned value: none

Examples:

```
Feasa.Print('Hello world!')
Feasa.Print('This is a submarine', 'yellow')
my_number = 1980.0310
value = 36
Feasa.Print(my_number, 'green')
Feasa.Print(value)
Feasa.Print('Total:' + str(my_number + 36), 'red')
```

Message dialog

This function shows a dialog box in graphical mode and a plain message in console mode.

Syntax: `Message(string message)`

Arguments:

- `message`: string to be displayed

Returned value: none

Examples:

```
Feasa.Message('The process has finished sucessfully!')
```

Confirmation box

Displays dialog box with a message and the buttons OK and Cancel.

Syntax: `AbortBox(string message)`

Arguments:

- `message`: string to be displayed

Returned value: boolean; True when OK is pressed and False otherwise.

Examples:

```
continue = Feasa.AbortBox('Do you want to continue')
if continue:
    Feasa.Print('Go on!')
else:
    Feasa.Print('See you later...')
```

Enable Logging

Sets the destination path of the Logging file and enables the logging. The messages sent or received to the Feasa device will be automatically logged if this function is called. Other data can also be logged to the file using the command `Log()`.

Syntax: `bool LogOn(string DestinationFilePath, bool LogTx, bool LogRx)`

Arguments:

- `DestinationFilePath`: string that contains the path of the file where the data will be logged to. Some wild-cards, that refer to system paths are accepted for the file path: “[MYDOCUMENTS]” and “[DESKTOP]”.
- `LogTx`: boolean; the commands transmitted to the Feasa Device will be logged when this value is True
- `LogRx`: boolean; the commands received from the Feasa Device will be logged when this value is True

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
Feasa.LogOn('f:\\test.txt', False, True)
Feasa.Log('Ground control to Major Tom...')
Feasa.LogOn('[MYDOCUMENTS]\\serial_number.dat', False, True)
Feasa.Run('GETSERIAL')
Feasa.LogOff()
```

Disable Logging

Disables the logging, which was previously enabled by the function LogOn().

Syntax: `LogOff()`

Arguments: none

Returned value: none

Examples:

```
Feasa.LogOn('[MYDOCUMENTS]\\serial_number.dat', False, True)
Feasa.Run('GETSERIAL')
Feasa.LogOff()
```

Log data to file

Stores data in the logging file, previously defined in the function LogOn()

Syntax: `bool Log(data)`

Arguments:

- `data`: string or numeric value to be stored in the file

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
Feasa.LogOn('[MYDOCUMENTS]\\my_data.dat', False, True)
Feasa.Log('Data:')
Feasa.Log(1980)
Feasa.Log(10)
Feasa.Log(3)
Feasa.Log(36)
Feasa.LogOff(0.3479)
```

Empty Log file

Clears the content of the logging file, previously defined in the function LogOn().

Syntax: `bool EmptyLog()`

Arguments: none

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.



Examples:

```
Feasa.LogOn('f:\\test.txt', False, True)
Feasa.Log('There was a guy, an underwater guy...')
Feasa.LogOff()
```

2.2 Communication with Feasa Devices

Run Feasa Command

The command Run give access to the communications library of the Feasa Terminal, so commands can be sent to the Feasa products and retrieve their responses.

Syntax: `string Run(string command [, int Timeout])`

Arguments:

- `command`: string containing the command to be sent to the Feasa Device.
- `Timeout`: optional parameter to define the response Timeout. This value is used in order to avoid to wait infinitely if something goes wrong (maximum wait time). If this parameter is not given, then the command takes the default general Timeout, which has a default value of 3 seconds. This default time can be adjusted using the function “SetResponseTimeout()”, that will be explained in further sections.

Returned value: string; the data returned by the Feasa Device. Empty string in case of Timeout or Syntax Error; in this situation the class Feasa has two flags: “TimeoutError” and “SyntaxError” which will be set in case that any of those situations occur, this way there is a better control of the errors.

Examples:

```
# Retrieve HSI values
response = Feasa.Run('GETHSI01')
if response == '' :
    if Feasa.SyntaxError:
        print 'Syntax Error. The command to the Analyser does
not seem to be valid.';
    elif Feasa.TimeoutError:
        print 'Timeout. The Analyser is taking too much to
respond';
    else
        print Feasa.Error_Description
```

Timeouts

The command Run has a default maximum wait time of 3 seconds (Timeout), in order to avoid to wait indefinitely if something goes wrong. But some commands can take some time to be executed, so this time can be modified using this function:

Syntax: `SetResponseTimeout(int Timeout)`

Arguments:

-
- `Timeout`: numeric value of the timeout in milliseconds.

Returned value: none

Examples:

```
Feasa.SetResponseTimeout(5000)    # Set timeout to 5 seconds
response = Feasa.Run('CAPTURE1')
```

End of transmission control

All responses from Feasa Devices end with a predefined set of character, like CR + LF or CR + LF + EOT. This format is used to detect the end of a response (the reception will end when those characters are received) and despite it is set by default to the Device's format, it can be set to any value using the following function:

Syntax: `bool SetStringToWait(string value)`

Arguments:

- `value`: string representing the characters that define the end-of-transmission for a received response from the Feasa Device. This parameter accepts the wild-cards “[CR]”, “[LF]”, “[EOT]”, “[CRLF]” and “[CRLF EOT]”;

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
# Test with CR + LF
Feasa.Run('DISABLEEOT')
Feasa.SetStringToWait('[CRLF]')
serial_number = Feasa.Run('GETSERIAL')
print serial_number

# Test with CR + LF + EOT
Feasa.Run('ENABLEEOT')
Feasa.SetStringToWait('[CRLF EOT]')
serial_number = Feasa.Run('GETSERIAL')
print serial_number
```

External Trigger

If External trigger has been previously enabled and configured, then an external trigger condition can be detected using the following command:

Syntax: `string WaitForTrigger(int Timeout, bool WaitForResponse)`

Arguments:

- `Timeout`: numeric value for the Timeout in ms, which is the maximum amount of time that the function will wait for the External Trigger condition to occur.
- `WaitForResponse`: this parameter has to be set to true if the Feasa device has been configured to output the results when the capture is completed after the trigger.

Returned value: string with the response obtained by the Feasa device after the trigger.

Examples:

```
Feasa.Run('SETEXTERNALCAPTURE06')
Feasa.Run('SETEXTERNALOUTPUTNONE')
if len(Feasa.WaitForTrigger(15000, false))>0:
    Feasa.Print('Capture completed!')
    Feasa.Print(Feasa.Run('GETHSI01'))

Feasa.Run('SETEXTERNALCAPTURE13')
Feasa.Run('SETEXTERNALOUTPUTHSI')
if len(Feasa.WaitForTrigger(15000, true))>0:
    Feasa.Print('Done!')
```

2.3 Timing functions

Print Current Time

The function **PrintTime()** prints on the Messages window, the current Time of the day; the message appears in the default output stream, in console mode.

Syntax: `PrintTime()`

Arguments: none

Returned value: none

Examples:

```
Feasa.Print('Time:')  
Feasa.PrintTime()
```

Measure Time

The function **GrabTime()** allows to save a time-stamp and the functions **GetElapsed()** and **GetElapsedSeconds()** measure the time elapsed since the last **GrabTime()**.

Syntax: `GrabTime()`

Arguments: none

Returned value: none

Syntax: string `GetElapsed()`

Arguments: none

Returned value: string; elapsed time represented through the standard time notation hh:mm:ss,ms

Syntax: string `GetElapsedSeconds()`

Arguments: none

Returned value: float; numeric value with the number of seconds elapsed, with a decimal part representing the number of milliseconds.

Examples:

```
Feasa.GrabTime()
Feasa.Delayms(450)
Feasa.GetElapsed()
elapsed = Feasa.GetElapsedSeconds()
seconds = Int(elapsed)
milliseconds = (elapsed - seconds) * 1000
Feasa.Print('Elapsed time: ' + seconds + 's ' + milliseconds +
'ms')
```

Delays

There are two available to generate a delay or wait time, “**Delay()**” and “**Delayms()**”, using one or the other depending on if the time is expressed in seconds or milliseconds (respectively).

Syntax: `Delay(int value)`

Arguments:

- `value`: numeric value indicating the number of seconds of the delay

Returned value: none

Examples:

```
Feasa.Delay(2)    # 2 seconds delay
```

Syntax: `Delayms(int value)`

Arguments:

- `value`: numeric value indicating the number of milliseconds of the delay

Returned value: none

Examples:

```
Feasa.Delayms(250)    # 250ms delay
```

2.4 External communications

Open port

The function **SerialOpen()** is used to open a communication through the given Serial port.

Syntax: bool SerialOpen(string PortName, int Baudrate, int Parity, int DataBits, int Handshake, int StopBits)

Arguments:

- **PortName**: string with the name of the Serial port, not the port number.
- **Baudrate**: numeric value indicating for the communication speed (baud rate).
- **Parity**: 1 for odd parity, 2 for even and 0 to disable the parity.
- **DataBits**: Number of bits to be transmitted. Possible values: 5, 6, 7 or 8.
- **Handshake**: use 1 for Hardware, 2 for Software and 0 to disable the Handshake / flow control.
- **StopBits**: use 1 for one stop bit, 2 for one and a half stop bit, three for two stop bits.

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
if Feasa.SerialOpen('COM12', 57600, 0, 8, 0, 1):  
    # do actions  
    Feasa.SerialClose('COM12')
```

Send data through the port

The command **SerialSend()** can be used to send ASCII data over a given port.

Syntax: bool SerialSend(string PortName, string DataToSend)

Arguments:

- **PortName**: string with the name of the Serial port, not the port number.
- **DataToSend**: string containing the set of ASCII characters that will be sent over the port.
Wild-cards like [CR], [LF], [CRLF], [EOT] and [CRLF EOT] can be used, that will be transformed to \r, \n, \r\n, \x04, \r\n\x04 respectively.

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
Feasa.SerialSend('COM12', 'SETLED0')
```

Receive data from the port

The function **SerialWaitForResponse()** can be used to receive data from the given serial port and the reception is finished when a certain pattern or fixed string is received.

Syntax: `string SerialWaitForResponse(string PortName, string ExpectedResponse, int Timeout)`

Arguments:

- `PortName`: string with the name of the Serial port, not the port number.
- `ExpectedResponse`: string defining the format of the response, in order to describe when the reception should stop. The character '*' can be used to indicate "anything", but can be used only to precede a set of characters; ex: if all responses end with the character '!', then the expected response could be composed as "*!". Wild-cards like [CR], [LF], [CRLF], [EOT] and [CRLF EOT] can be used, that will be transformed to \r, \n, \r\n, \x04, \r\n\x04 respectively.
- `Timeout`: numeric value indicating the number of milliseconds that the application will wait until any character is received, or the maximum time between two consecutive characters.

Returned value: string; the data returned by the remote device. Empty string in case of Timeout or Syntax Error; in this situation the class Feasa has two flags: "TimeoutError" and "SyntaxError" which will be set in case that any of those situations occur, this way there is a better control of the errors.

Examples:

```
Feasa.SerialSend('COM12', 'SETLED0')
buffer = Feasa.SerialWaitForResponse('COM12', '*[CR]', 1000)
Feasa.Print(buffer)
```

The function **SerialWaitForResponseLength()** can be used to receive data from the given serial port and the reception is finished when a certain number of characters is received.

Syntax: `string SerialWaitForResponseLength(string PortName, int ExpectedResponseLength, int Timeout)`

Arguments:

- `PortName`: string with the name of the Serial port, not the port number.
- `ExpectedResponseLength`: integer to indicate the number of characters that will be received from the port.



- `Timeout`: numeric value indicating the number of milliseconds that the application will wait until any character is received, or the maximum time between two consecutive characters.

Returned value: string; the data returned by the remote device. Empty string in case of Timeout or Syntax Error; in this situation the class Feasa has two flags: “TimeoutError” and “SyntaxError” which will be set in case that any of those situations occur, this way there is a better control of the errors.

Examples:

```
Feasa.SerialSend('COM12', 'SETLED0')
buffer = Feasa.SerialWaitForResponseLength('COM12', 5, 1000)
Feasa.Print(buffer)
```

Close communications

The command **SerialClose()** is used to close the communication with a given serial port.

Syntax: `SerialClose(string PortName)`

Arguments:

- `PortName`: string with the name of the Serial port, not the port number.

Returned value: none

Examples:

```
Feasa.SerialClose('COM12')
```

See also example in `SerialOpen()`

2.5 Chromaticity diagrams

CIE 1931

There are several commands that allow to operate with the CIE 1931 diagram:

The function CIE1931_Show() open a window that displays the CIE 1931 diagram.

Syntax: `CIE1931_Show()`

Arguments: none

Returned value: none

The function CIE1931_Plot() is used to plot points/markers on the CIE 1931 diagram.

Syntax: `CIE1931_Plot(float x, float y)`

Arguments:

- x: float representing the x coordinate of the Chromaticity Diagram
- y: float representing the y coordinate of the Chromaticity Diagram

Returned value: none

The function CIE1931_PlanckianLocus() is used to show or hide the Planckian Locus.

Syntax: `CIE1931_PlanckianLocus(bool status)`

Arguments:

- status: boolean value indicating whether the Planckian Locus will be plotted or no

Returned value: none

The function CIE1931_Clear() is used to clear any point/marker previously plotted on the diagram.

Syntax: `CIE1931_Clear()`

Arguments: none

Returned value: none

The function CIE1931_Close() is used to close the CIE diagram window.

Syntax: `CIE1931_Close()`

Arguments: none

Returned value: none

Examples:

```
Feasa.CIE1931_Show()  
Feasa.CIE1931_PlanckianLocus(True)  
Feasa.CIE1931_Plot(0.3215, 0.3812)  
Feasa.Close()
```

CIE 1976

There are several commands that allow to operate with the CIE 1976 diagram:

The function CIE1976_Show() open a window that displays the CIE 1976 diagram.

Syntax: `CIE1976_Show()`

Arguments: none

Returned value: none

The function CIE1976_Plot() is used to plot points/markers on the CIE 1976 diagram.

Syntax: `CIE1976_Plot(float up, float vp)`

Arguments:

- `up`: float representing the u' coordinate of the Chromaticity Diagram
- `vp`: float representing the v' coordinate of the Chromaticity Diagram

Returned value: none

The function CIE1976_PlanckianLocus() is used to show or hide the Planckian Locus.

Syntax: `CIE1976_PlanckianLocus(bool status)`

Arguments:

- `status`: boolean value indicating whether the Planckian Locus will be plotted or no

Returned value: none

The function CIE1976_Clear() is used to clear any point/marker previously plotted on the diagram.

Syntax: `CIE1976_Clear()`

Arguments: none

Returned value: none

The function CIE1976_Close() is used to close the CIE diagram window.

Syntax: `CIE1976_Close()`

Arguments: none

Returned value: none

Examples:

```
Feasa.CIE1976_Show()  
Feasa.CIE1976_PlanckianLocus(True)  
Feasa.CIE1976_Plot(0.3215, 0.3812)  
Feasa.Close()
```

2.6 Other useful functions

Clear output/screen

The command **Cls()** allows to clear the existing messages in the default output, messages box in visual mode or console window in console mode.

Syntax: `Cls()`

Arguments: none

Returned value: none

Examples:

```
Feasa.Print('This message will be cleared in two seconds')
Feasa.Delay(2)
Feasa.CLS()
```

Execute external programs

There is a way to run external programs from Feasa Terminal Python scripts using the function **ShellExec()**.

Syntax: `bool ShellExec(string ProgramPath, string Arguments, bool WaitToComplete)`

Arguments:

- `ProgramPath`: string containing the path to the program to be executed. Some wild-cards, that refer to system paths are accepted for the file path: “[MYDOCUMENTS]” and “[DESKTOP]”.
- `Arguments`: optional string; here could be placed the program arguments, if there were any.
- `WaitToComplete`: boolean; use True to force the Terminal to wait until the execution of the called program is complete or closed. Use False to continue the execution of the Python script immediately after.

Returned value: boolean; the value returned is False if any error occurs, otherwise is True.

Examples:

```
Feasa.LogOn('[MYDOCUMENTS]\\logging.txt', False, True)
Feasa.Log('This is the Logging file')
Feasa.ShellExec('c:\\windows\\notepad.exe', '[MYDOCUMENTS]\\logging.txt', False)
```

Exit program

If you wish to close the Feasa Terminal program just after the execution of the Python script, you can call the function **Quit()**. Please, note that this function does not close the program immediately.

Syntax: `Quit()`

Arguments: none

Returned value: none

Examples:

```
import sys
if Feasa.AbortBox('Would you like to exit?'):
    Feasa.Quit()
    sys.exit()
```

2.7 Error Handling

In case of error, the Feasa class has several flags and variables that store extended information for that precise error. This situation occurs when a function returns False (as error condition) or empty results. The variables are:

- [TimeoutError](#): this flag is set to True when a Timeout occurs while sending data to the Feasa Device or external serial port.
- [SyntaxError](#): this flag is set to True when a Syntax Error condition is returned by the Feasa Device (command Run)
- [Error_Description](#): contains the description of the last error occurred.

3. Examples

A set of examples can be found in the folder “scripts/”, in the program directory.

4. Quick language reference

Command	Info	Parameters	Return
AbortBox	Shows OK/Cancel dialog box	string message	bool
CIE1931_Clear	Clears all markers on the CIE diagram, which have been previously drawn	none	none
CIE1931_Close	Closes the CIE diagram	none	none
CIE1931_Plot	Plot a marker on the CIE diagram based in the given coordinates	float x, float y	none
CIE1931_Show	Shows the CIE 1931 diagram	none	none
CIE1931_PlanckianLocus	Plots the Planckian Locus on the CIE diagram	bool status	none
CIE1976_Clear	Clears all markers on the CIE diagram, which have been previously drawn	none	none
CIE1976_Close	Closes the CIE diagram	none	none
CIE1976_Plot	Plot a marker on the CIE diagram based in the given coordinates	float up, float vp	none
CIE1976_Show	Shows the CIE 1976 diagram	none	none
CIE1976_PlanckianLocus	Plots the Planckian Locus on the CIE diagram	bool status	none
Cls	Clears output	none	none
Delay	Performs a delay in seconds	int value	none
Delayms	Performs a delay in milliseconds	int value	none
EmptyLog	Empty previously defined Log file	none	bool
GetElapsed	Gets elapsed time in text format	none	string
GetElapsedSeconds	Gets elapsed time in seconds	none	float
GrabTime	Stores current time (timestamp)	none	none
Log	Saves a string in the Log file	[string int float double decimal] message	bool
LogOff	Disables logging to Log file	none	none
LogOn	Enables logging and define Log file	string DestinationFilePath, bool LogTx, bool LogRx	bool
Message	Shows dialog message box in graphical mode or simple message in console	string message	none
Print	Prints out a message in default output (graphical/console)	[string int float double decimal] data [, color]	none
PrintTime	Prints out current time in default output (graphical/console)	none	none
Quit	Forces the Terminal to exit once execution of script finishes	none	none
Run	Sends command to Feasa Device and receive response	string command [, int Timeout]	string

Feasa Python scripts · reference guide

by R&D Engineer Carles Martínez Rius, rev 4, © Feasa Enterprises Ltd.



SerialClose	Closes communication of a previously opened serial port	string PortName	none
SerialOpen	Opens communication in serial port	string PortName, int Baudrate, int Parity, int DataBits, int Handshake, int StopBit	bool
SerialSend	Sends string data through a previously opened serial port	string PortName, string DataToSend	bool
SerialWaitForResponse	Receives response from a previously opened serial port	string PortName, string ExpectedResponse, int Timeout	string
SerialWaitForResponseLength	Receives response from a previously opened serial port	string PortName, int ExpectedResponseLength, int Timeout	string
SetResponseTimeout	Sets default timeout for reception in communication with Feasa Devices or external serial ports	int Timeout	none
SetStringToWait	Sets End-Of-Response string condition (reception) for communication with Feasa Devices	string value	bool
ShellExec	Executes external application	string ProgramPath, string Arguments, bool WaitToComplete	bool