DeviceServer User’s Guide

***Current version of document: 1.5***

***Last modification the 04/02/2019***

***History of modifications***

|  |  |  |  |
| --- | --- | --- | --- |
| **Date** | **Revision** | **Description** | **Author** |
| 06/07/05 | 1.0 | Initial Version | S. Postollec |
| 19/07/05 | 1.1 | Revision after G. Abeille and V. Hardion remarks | S. Postollec |
| 22/07/05 | 1.2 | Version validated by G. Abeille and V. Hardion | S. Postollec |
| 06/01/16 | 1.3 | New fittings functions are added | L. Klenov |
| 08/10/18 | 1.4 | New fitting function KnifeEdge(x) is added | Eric LOMBARD |
| 28/01/19 | 1.5 | Correction on function KnifeEdge(x)  ROI on Experimental Data X is added  Some text corrections. | Eric LOMBARD |

*Table of contents*

[1 Applicative goals and main functionalities of the DeviceServer 2](#_Toc189536)

[1.1 Introduction 2](#_Toc189537)

[2 Technical scheme of the application 3](#_Toc189538)

[2.1 Software Architecture 3](#_Toc189539)

[2.2 Design considerations/Implementation issues 3](#_Toc189540)

[3 TANGO software interfaces 4](#_Toc189541)

[3.1 Attributes 4](#_Toc189542)

[3.2 Commands 12](#_Toc189543)

[3.3 Exceptions 12](#_Toc189544)

[4 Software Setup 14](#_Toc189545)

[4.1 Environment requisites 14](#_Toc189546)

[4.2 Installation of particular libraries, software or run time environments 14](#_Toc189547)

[5 Limitations and problems 16](#_Toc189548)

[5.1 Current known limitations 16](#_Toc189549)

[5.2 Troubleshooting 16](#_Toc189550)

[6 Development plan 16](#_Toc189551)

# Applicative goals and main functionalities of the DeviceServer

## Introduction

A common light beam measure consists in:

* Moving a detector.
* Collect a data which represent the beam intensity on this detector.

Thus, we get vectors (or spectrums) of values describing spatially the intensity of the beam:

A way to qualify the acquired spectrum is to “fit” the datas by known functions like Gaussian or Lorentzian, to get different pieces of information like the width of the peak, the half height intensity, etc. . Thanks to these pieces of information it will be possible to calibrate the beam in the best way (maximum energy).

Thus the aim of the ‘fit” is to find (if it is possible) the function which best-fit the experimental points set by estimating parameters describing this function.

In our case we want two things, the first one is to minimise the error (distance) between the set of experimental data values and the fitting function chosen and the second one is to find the best parameters which describe the fitting function.

Let N the number of replicate of the experiment. For each observation, let  the observed values (experimental data),  the predicated value (the fitting function chosen),  the measured parameters and  the estimated parameters (of the fitting function chosen). Thus the residual is the difference between the actual and expected values:



And the aim is to find all the which minimise:



The algorithm used to fit this non-linear function is the Marquardt-Levenberg Algorithm.

Levenberg-Marquardt is a popular alternative to the Gauss-Newton method of finding the minimum of a function that is a sum of squares of nonlinear functions:



with in our case 

For more precision about Non linear Least Square Fitting

<http://mathworld.wolfram.com/NonlinearLeastSquaresFitting.html>

<http://www.aip.org/tip/INPHFA/vol-9/iss-2/p24.html>

and for more precision about the Levenberg-Marquardt algorithm

<http://www.answers.com/topic/levenberg-marquardt-algorithm>

# Technical scheme of the application

## Software Architecture



## Design considerations/Implementation issues

### In order to add new functions in the simplest way, all the functions have to implement an abstract interface called “fittingFunction”. The device is composed of three main parts:

* The first part is the GSL (GNU Scientific Library). This library provides the Marquardt-Levenberg fitting algorithm to fit non-linear functions. This library is compiled in a static library which must be included in the device server compilation scheme.
* The second part is the classes above GSL. These classes are done to simplify the use of the GSL library. Thus if a programmer want to add a new function, he will not see the GSL code but only theses classes. These classes provide a frontage pattern in front of GSL.
* The third part is the Device Server which instantiate objects from the second part and provide the graphical display of the attributes and the results (values and graphics) of the fit.

All the “intelligence” is encapsulated in the second part.

# TANGO software interfaces

## Attributes

[Please insert POGO generated documentation link here](file:///F:\_LOMBARD\PROJETS\TANGO_SOLEIL_7\Local%20Settings\Temporary%20Internet%20Files\Local%20Settings\Temp\Rar$DI08.l30\Attributes.html)

|  |  |  |
| --- | --- | --- |
| **Scalar Attributes** | | |
| **Attribute name** | **Data Type** | **R/W Type** |
| **deviceName** ( Describes the name of the instance of the device which provides datas to be fitted) | DEV\_STRING | WRITE |
| **deviceAttributeNameX** ( Name of the attribute which will be fitted by this DataFittingDevice. It may be : a) a fully qualified attribute name . For instance : - D13-1\_C07/DT/IMAGEUR.1-ECR/PROFILX b) an alias (if defined in the TANGO Control System): PROFILX ) | DEV\_STRING | WRITE |
| **deviceAttributeNameY** ( Name of the attribute which will be fitted by this DataFittingDevice. It may be : a) a fully qualified attribute name . For instance : - D13-1\_C07/DT/IMAGEUR.1-ECR/PROFILY b) an alias (if defined in the TANGO Control System): PROFILY ) | DEV\_STRING | WRITE |
| **deviceAttributeNameSigma** ( Name of the attribute which will be fitted by this DataFittingDevice. It may be : a) a fully qualified attribute name . For instance : - D13-1\_C07/DT/IMAGEUR.1-ECR/PROFILSIGMA b) an alias (if defined in the TANGO Control System): PROFILSIGMA ) | DEV\_STRING | WRITE |
| **fittingFunctionType** ( Type of the function used for fitting (gaussian, lorenzian, sigmoid, knifeedge, with or without background) ) | DEV\_STRING | WRITE |
| **nbIterationMax** ( Number of maximum iteration for the fitting algorithm. If the algorithm doesn't go to a final solution after nbIterationMax it is stopped to avoid infinite loop. ) | DEV\_LONG | WRITE |
| **epsilon**  ( This value is the stopping value. When the difference between the values of two consecutives steps is under this value, the algorithm stops and succeeds. ) | DEV\_DOUBLE | WRITE |
| **fittedDataSameSizeAsData**  This attribute is used to indicate if the FittedData are of the same size as the experimental data.If TRUE the attribute startingX, resolutionX, nbPointToGenerate are not used and the startingX value used is the first X value of the ROI experimental data, the number of points generated is the number of points of ROI experimental data | DEV\_BOOLEAN | WRITE |
| **startingX** ( Starting value for X to generate the fittedDataX spectrum values. ) | DEV\_DOUBLE | WRITE |
| **resolutionX** ( Resolution according the X axis to calculate the values of the dataFittedY spectrum. ) | DEV\_DOUBLE | WRITE |
| **nbPointsToGenerate** ( Number of points to generate for the fitted functions. ) | DEV\_LONG | WRITE |
| **fitMode**  (type of mode used for fitting).  0 : Automatic Mode (data are fitted continuously)  1 : Manual Mode . You need to do the StartFit command to launch fit calculations. | DEV\_BOOLEAN | WRITE |
| **nbParameters** ( Number of parameters according the choosen fitting function. ) | DEV\_LONG | READ |
| **initialsParametersMode** ( Set the mode for choosing the initials parameters.  FALSE = Manual 🡪 the initials parameters (initialParameterPos, initialParameterWidth, initialParameterHeight, initialParameterBackground, initialBackgroundA, initialBackgroundB) are used to initialize the algorithm.  TRUE = Automatic 🡪 initials values are choosen automatically by finding the max value in the Y. This value gives the height. The X value associated to this Y value gives the position. Width is taken at a value of 1.0 and background is given by the first Y value. ) | DEV\_BOOLEAN | WRITE |
| **experimentalDataX\_Min** Minimum range value of ExperimentalDataX, used for calculation. ***Note***: If the value is the same that "ExperimentalDataX\_Max", all ExperimentalDataX spectrum is used. | DEV\_DOUBLE | READ/WRITE |
| **experimentalDataX\_Max** Maximum range value of ExperimentalDataX, used for calculation. ***Note***: If the value is the same that "ExperimentalDataX\_Min", all ExperimentalDataX spectrum is used. | DEV\_DOUBLE | READ/WRITE |

|  |  |  |
| --- | --- | --- |
| **Spectrum Attributes** | | |
| **Attribute name** | **Data Type** | **X Data Length** |
| **knownFittersList**  (The list of the names of the known functions used for fitting) | DEV\_STRING |  |
| **experimentalDataX** ( Data copied from original X spectrum attribute ) | DEV\_DOUBLE | 100000 |
| **experimentalDataY** ( Data copied from original Y spectrum attribute ) | DEV\_DOUBLE | 100000 |
| **experimentalDataSigma** ( Data copied from original Sigma spectrum attribute ) | DEV\_DOUBLE | 100000 |
| **fittedFunctionParameters** ( This spectrum is composed of the differents calculated parameters of the function used to fit. The number of parameters depends on the function. ) | DEV\_DOUBLE | 10 |
| **fittedDataX** ( Spectrum providing the fitted data according the X axis. These datas are calculated thanks to the fitted parameters and the resolutionX, nbPointToGenerate and startingX values. ) | DEV\_DOUBLE | 1000000 |
| **fittedDataY** ( Spectrum providing the fitted data according the Y axis. These datas are calculated thanks to the fitted parameters and the resolutionX, nbPointToGenerate and startingX values. ) | DEV\_DOUBLE | 100000 |

### FittingFunctionType

This attribute is a WRITE String attribute which contains the name the type of the function used for fitting.

The possible values to enter are:

1. “gaussian”,
2. “gaussianb”,
3. “lorentzian”,
4. “lorentzianb”,
5. “sigmoid”,
6. ”sigmoidb”,
7. “gaussianbf”,
8. “lorentzianbf”
9. "knifeedgeb".

The basic functions ("**gaussian**", "**lorentzian**", and "**sigmoid**") uses only 3 parameters :

* position (= μ) : The x position for the height of the curve peak.
* width (= σ) : The width of the curve peak.
* height : The height of the curve peak.

The first extended type of functions ("**gaussianb**", "**lorentzianb**", "**sigmoidb**", and "**knifeedgeb**") consist in adding the parameter "background" (indicated by a final "b" in their names):

* background : The offset of the curve.

The second extended type of functions ("**gaussianbf**" and "**lorentzianbf**") consist in adding the parameters "a" and "b" of the "background function" *a.x + b* (indicated by a final "bf" in their names). It replaces the previous background parameter:

* background\_a: The gain of the curve.
* background\_b: The offset of the curve.

The descriptions of these functions are thereafter. The optional background values are highlighted.

#### Gaussian FunctionType

“**gaussian**”: make a fit with a Gaussian function (background = 0),

“**gaussianb**”: make a fit with a Gaussian function with background as ,

“**gaussianbf**”: make a fit with a Gaussian function with background as .

*The "full" Gaussian function is defined by:*

*The Gaussian derivative is defined by:*

*The Gaussian partial derivatives are defined by:*

* (for functions “**gaussianb**”and “**gaussianbf**”)
* (for function “**gaussianbf**”)

*The FWHM is defined by:*

#### Lorentzian FunctionType

“**lorentzian**”: make a fit with a Lorentzian function (background = 0),

“**lorentzianb**”: make a fit with a Lorentzian function with background as ,

“**lorentzianbf**”: make a fit with a Gaussian function with background as .

*The "full" Lorentzian function is defined by:*

*The Lorentzian derivative is defined by:*

*The Lorentzian partial derivatives are defined by:*

* (for functions “**lorentzianb**”and “**lorentzianbf**”)
* (for function “**lorentzianbf**”)

The FWHM is defined by:

#### Sigmoid FunctionType

“**sigmoid**”: make a fit with a Sigmoïd function

“**sigmoidb**”: make a fit with a Sigmoïd function with background.

*The Sigmoïd function is defined by:*

*The Sigmoïd derivative is defined by:*

*The Sigmoïd partial derivatives are defined by:*

* (for function “**sigmoidb**”)

*The FWHM is defined by:*

#### KnifeEdge FunctionType

“**knifeedgeb**”: make a fit with a KnifeEdge function with background.

*The KnifeEdge function is defined by:*

* + *(To have more details on erf function :* [*https://en.wikipedia.org/wiki/Error\_function*](https://en.wikipedia.org/wiki/Error_function)*, https://fr.mathworks.com/help/matlab/ref/erf.html)*

*The KnifeEdge derivative is defined by:*

*The KnifeEdge partial derivatives are defined by:*

*The FWHM is defined by:*

### FittedFunctionParameters

This attribute is a read-write spectrum.

The write part of this attribute is used to give the initials values for each parameters of the function to be estimated. (Initials values for pos, width, height and background of the function).

If the write Boolean attribute “**InitialParametersMode**” is set to Manual (unchecked), the initials values entered are used by the program for starting the algorithm. If the “**InitialParametersMode**” is set to Automatic (checked) the initials parameters are computed automatically like explained in the scalar attribute arrays description at the beginning of this section.

Moreover if the fit succeed, the estimated values for each parameters of the function are put in the read part of the spectrum of this attribute. So the initials values entered are not lost by the estimated ones. The read part contains the estimated and the write part the values entered by the user.

The spectrum value order follows the same order than the parameters function order (pos, width, height, background).

This order gives the order of the parameters when you want to enter initials values.

For instance, when you entered 1.0; 2.0; 3.0; 4.0 in the device for the initials values of the fittedFunctionParameters spectrum that means that the pos initial value is 1.0, the width initial value is 2.0, the height initial value is 3.0 and the background initial value is 4.0 (background is constant).

If you entered 1.0; 2.0; 3.0; 4.0; 5.0 in the device for the initials values of the fittedFunctionParameters spectrum that means that the pos initial value is 1.0, the width initial value is 2.0, the height initial value is 3.0 and the background initial value is 4.0 and 5.0 (background is **a straight line**).

It’s the same for reading estimated parameters.

N.B: The number of initials parameters values entered for the function chosen to fit must be the same as the number of parameters describing the function to fit

For instance if you entered 1.0;2.0;3.0 and you want to fit by a function with background (gaussianb or gaussianbf) which requires four (or five) parameters, the fourth (gaussianb) or fifth and fourth (gaussianbf) initial parameters doesn’t have values.

Thus, the choice in that case is that the initials values are automatically computed without taking into consideration the values entered in the **FittedFunctionParameters** spectrum attribute like if the **initialParameterMode** attribute was set to Automatic.

### experimentalDataSigma

This spectrum sigma is not necessary to run. This spectrum can be used if it is possible to get an error value for each for each couple (X, Y). (For instance if the sensors give an error of position due to their sensibility, this value can be entered in the sigma spectrum).

This error (when exists) is taken into consideration by the fitting algorithm to provide an error range for each estimated parameters.

The attributes which contain the results are

* **fittedFunctionParameters**
* **fittedDataX**
* **fittedDataY**

The three attributes (**startingX**, **resolutionX**, **nbPointsToGenerate**) are used to generate the fitted function if the algorithm finds a solution for the fit. If the fit succeed, the estimated parameters allow getting an expression of the fitted function and it is possible to compute values for this function.

* + The “**startingX**” parameter gives the beginning value of X
  + the “**resolutionX**” gives the step between two X values
  + “**nbPointToGenerate**” the desired number of points.

**fittedDataSameSizeAsData** attribute:

* If set to false, the fittedDataX spectrum (used to generate fittedDataY and derivedFittedDataY spectra) is computed from the 3 values above.
* If set to true, the fittedDataX spectrum is associated to the ROI extracted from experimentalDataX.

## Commands

[Please insert POGO generated documentation link here](F:\\_LOMBARD\\PROJETS\\TANGO_SOLEIL_7\\Local Settings\\Temporary Internet Files\\Local Settings\\Temp\\Rar$DI08.l30\\DevCommands.html)

The command “startFit” starts the fit.

If the “**FitMode”** boolean attribute is set to Automatic, this command is repeated automatically if new data arrives. **FitMode** in Automatic avoid the user to push the start button to start the fit. The fitting algorithm is repeated when a new scan arrives.

In order to start a fit, the values which are required are the Spectrums which contain the data to be fitted (**experimentalDataX** and **experimentalDataY**).

Moreover, if the **InitialsParametersMode** is set to MANUAL (unchecked) you have to enter the initials values in the **fittingFunctionParameters** attribute (NB: Until now, writing on a spectrum attribute is not possible with ATKPanel. It can be done thanks to the “TestDevice “interface under JIVE by giving the values separated by a coma in the argin textbox).

There is no “stop” or “abort” Tango command but the algorithm is stopped automatically if the fit algorithm reaches the maximum number of iterations or if the fit failed due to abnormal values for the parameters describing the function.

The time calculation is fast (Less than the second). The computing time may increase if the number of experimental data points is increased.

## Exceptions

The possible initializing errors that can occur are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ERROR TYPE** | **STATE** | **QUALITY** | **MESSAGE** | **Result/Explanation** |
| We can’t get data from the beam line device server. | ALARM | All the spectrums attributes are set to an INVALID quality. | "ERROR READING DATA TO BE FITTED!!!”. | The fit is not possible |
| The X and Y spectrum size of the data provided by the beam line are not the same | ALARM | All the spectrums attributes are set to an INVALID quality. | "ERROR DATAS X,Y,SIGMA DIFFERENTS SIZES !!!”. | The fit is not possible |
| The function can’t be initialized (different size for the datas spectrums, can’t reach the beam line server…) | FAULT | Let the previous quality. | "ERROR: FITTING FUNCTION INITIALISATION FAILURE” | The fit is not possible |
| The string entered for the fitting function is not compliant with the implemented functions. | FAULT | Let the previous quality. | "ERROR: FITTING FUNCTION UNKNOWN” | The fit is not done. |

The different cases for the results of a fit are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **FIT RESULT TYPE** | **STATE** | **QUALITY** | **MESSAGE** | **Result/Explanation** |
| The fitter has found a solution | The device state comes back to STANDBY | VALID | “Fit Succeeded!” | Success |
| The fit fails | ALARM | All the fitted spectrums attributes are set to an INVALID quality. | "ERROR FITTING FAILURE!!!” | Failure |
| The fit doesn’t converge | ALARM | Let the previous quality. | "BECAREFUL FITTING DOESN'T CONVERGE!!! FITTED DATAS ARE PROBABLY WRONG!!!” | It doesn’t mean that the fit fails but only that the number of iterations reaches the **NbMaxIteration** attribute value. Thus, the algorithm stops. The result can be quite good or not if the fit diverge |
| An unknown kind of exception is thrown | ALARM | All the fitted spectrums attributes are set to an INVALID quality. | "ERROR UNKNOWN!!!” | Failure |

# Software Setup

## Environment requisites

The programs work under Microsoft Windows 2000, the ATK jars must be at least:

* ATKCore version 2.1.24
* ATKWidget version 2.1.24
* ATKPanel version 1.16
* GSL library version 1.6.

## Installation of particular libraries, software or run time environments

### The library used is GSL and is compiled as a static library. This library is located in the lib directory of the DataFitter device.

### Example of Data Fitting

Creation of a device providing data (the beam line or the DataFittingTestDevice for instance)

Creation of the DataFitterDevice

Using Jive: Create a Dserver DataFitter and register it. (eg: tmp/fitter/1)

No property is needed to start

Start the device server executable

Test the device (right click on tmp/fitter/1 and click on “Test device”or “monitor device”) select ping command, the device responds ALIVE)

You have now the DataFitter Device installed and running.

**Under the ATK panel you can press StartFit to start the fit.**

**If you want to enter initial parameters, you need to do it under the Test Device(since writing on spectrum is not possible with ATKPanel for the moment) for the attribute fittingFunctionParameters and to enter in the argin textbox the initials values separated by a coma.**

**If you want to use a ROI, you can define the bounds on the Experimental Data X values with the attributes "experimentalDataX\_Min" and "experimentalDataX\_Max". Note that if those values are the same, the whole Experimental Data X values is used for the fit.**

### Modus Operandi

In order to do a fit, you need two arrays, one containing the X values and another with the Y values. These two arrays give the set of points to be fitted. These values come from another device. So you need to enter the name of the device (**deviceName** attribute) to reach and to give the two attributes names providing the X (**deviceAttributeNameX** attribute) and Y (**deviceAttributeNameY** attribute) datas spectrums (and sigma if exists (**deviceAttributeNameSigma** attribute)).

Then you choose the fitting function by giving a value to the **fittingFunctionType** attribute. After that, you need to give initials values for each parameter describing the fitting function chosen by filling the **fittedFunctionParameters** spectrum **attribute.** It is necessary to give the same number of parameters than the fitting function chosen.

This step can be overcome if the **initialsParametersMode** attribute is set to automatic. In this case, the initials values are calculated automatically.

Then you use the **StartFit** command in order to launch the fitting process.

If the fit succeed, the estimated parameters are entered in the **fittedFunctionParameters** spectrum (same order than the description of the function) and a fitted function is computed in the two spectrums **fittedDataX** and **fittedDataY**.

The curves can be compared.

If the fit fails because the algorithm diverge (give erroneous values), an error message is displayed, the device status is set to ALARM, the **fittedFunctionParameters** are not given and the fitted curve is not computed and not displayed.

If the fit fails because the algorithm reach the maximum of step before stopping the algorithm, an error message is displayed, the status is set to ALARM, but the **fittedFunctionParameters** are given and the fitted curve is computed and displayed even if the result might be wrong.

# Limitations and problems

## Current known limitations

*Device/Algorithm limitations:*

* The current limitation is the number of fitting functions (“gaussian”, “gaussianb”, “gaussianbf”, “lorentzian”, “lorentzianb”, “lorentzianbf”, “sigmoid”, ”sigmoidb”, ”knifeedgeb”)
* Another limitation is that the initials values need to be near the real solution. If not, the result can be erroneous; the function diverges or finds a local minimum. But this limitation can’t be corrected; it’s an inner limitation due to the algorithm.

*ATKPanel limitations:*

* Moreover, you have to give “clever” values for the **startingX**, **resolutionX** and **nbPointToGenerate** attribute in order to get a fitted curve realistic and well drawn.

🡪 The main thing is that you have to give the same **startingX** value than the experimentalData from the beam line. If not, the experimental curve and the fitted curve will be shifted from the variation between the first X value of the experimental datas and the curve startX value. But the parameters for the function are good and well calculated. The effect is only visual.

* The estimated parameters are displayed like a spectrum. Thus, you have to uncheck the graph display check button to display the value in an array and not like a graph.
* The display is only done for 1D spectrum. It is not possible to watch a real Y=f(X) function for the moment. The two experimental spectrums **experimentalDataX**, **experimentalDataY** and the two fitted spectrums **fittedDataX**, **fittedDataY** are uncorrelated and displayed separately.
* The experimentalDatas and the fittedDatas curves are not displayed on the same graphics.

## Troubleshooting

TODO…

# Development plan

New fitting function can be added.

A future development will concern the Graphical User Interface.

For the moment, the estimated parameters are displayed like a spectrum. The aim will be to watch the values directly.

Another improvement will be to give the possibility to display the X and the Y spectrum in the same graph thus to display a real Y=f(X) curve.

In the same idea, displaying the two curves (experimental and fitted) in the same graphic.

In a first time, new fitting functions could be added. It was already the case with the sigmoid function which was not planned since the beginning.

To finish, another improvement will be to give the possibility of entering directly the description of the function to fit. This improvement is much more difficult because it needs parsing a string with mathematical symbols and parameters. (Therefore, the algorithm requires the derivative function, and one partial derivative per parameter to optimize.)