

# Slicer User's Guide

*Current version of document : 1.4*

*Last modification le 26/01/2006*

*History of modifications*

Date	Revision	Description	Author
15/12/05	1.0	Initial Version	Q. Siraut
15/12/05	1.1	Added device status descriptions	Q. Siraut
16/12/05	1.2	Took out some properties	Q.Siraut
20/12/05	1.3	Rolled-back + added dynamic attribute support	Q.Siraut
20/01/06	1.4	Added multiple attributes extraction	Q.Siraut

## Table of contents

<b>1</b>	<b>Applicative goals and main functionalities of the DeviceServer .....</b>	<b>2</b>
1.1	Introduction .....	2
1.2	Some vocabulary .....	2
<b>2</b>	<b>Technical scheme of the application .....</b>	<b>3</b>
2.1	Software Architecture .....	3
2.2	Design considerations/Implementation issues .....	3
2.2.1	Initialization and dynamically created attribute .....	3
2.2.2	Extraction process .....	5
<b>3</b>	<b>TANGO software interfaces .....</b>	<b>6</b>
3.1	Properties .....	6
3.2	Attributes .....	6
3.3	Commands .....	7
3.4	Exceptions .....	7
<b>4</b>	<b>Software Setup .....</b>	<b>8</b>
<b>5</b>	<b>Limitations and problems .....</b>	<b>8</b>
5.1	Current known limitations .....	8
5.2	Troubleshooting .....	8

## 1 Applicative goals and main functionalities of the DeviceServer

### 1.1 Introduction

The purpose of this “Slicer” DeviceServer is to extract specific data from the data produced by other DeviceServers. In fact, the slicer is able to extract data from several attributes, those attributes being part of the same DeviceServer or of different ones. This means the slicer can have multiple inputs and multiple outputs, which will be dynamically generated.

So, a “Slicer” DeviceServer will be seen as a client of other DeviceServers (acting as “servers”).

The data produced by the servers will always be arrays of numerical values that are stored in an attribute of the given server, this attribute being readable by the slicer.

The slicer is able to extract either a single value (scalar), either multiples values (spectrum), from either a single-dimension array (vector), either a bi-dimension array (matrix). Conceptually, those arrays will both be considered as bi-dimension arrays (a vector is nothing more than a matrix which one of its dimension is null), from which the slicer will be able to extract either a single value, either a full column, either a full line.

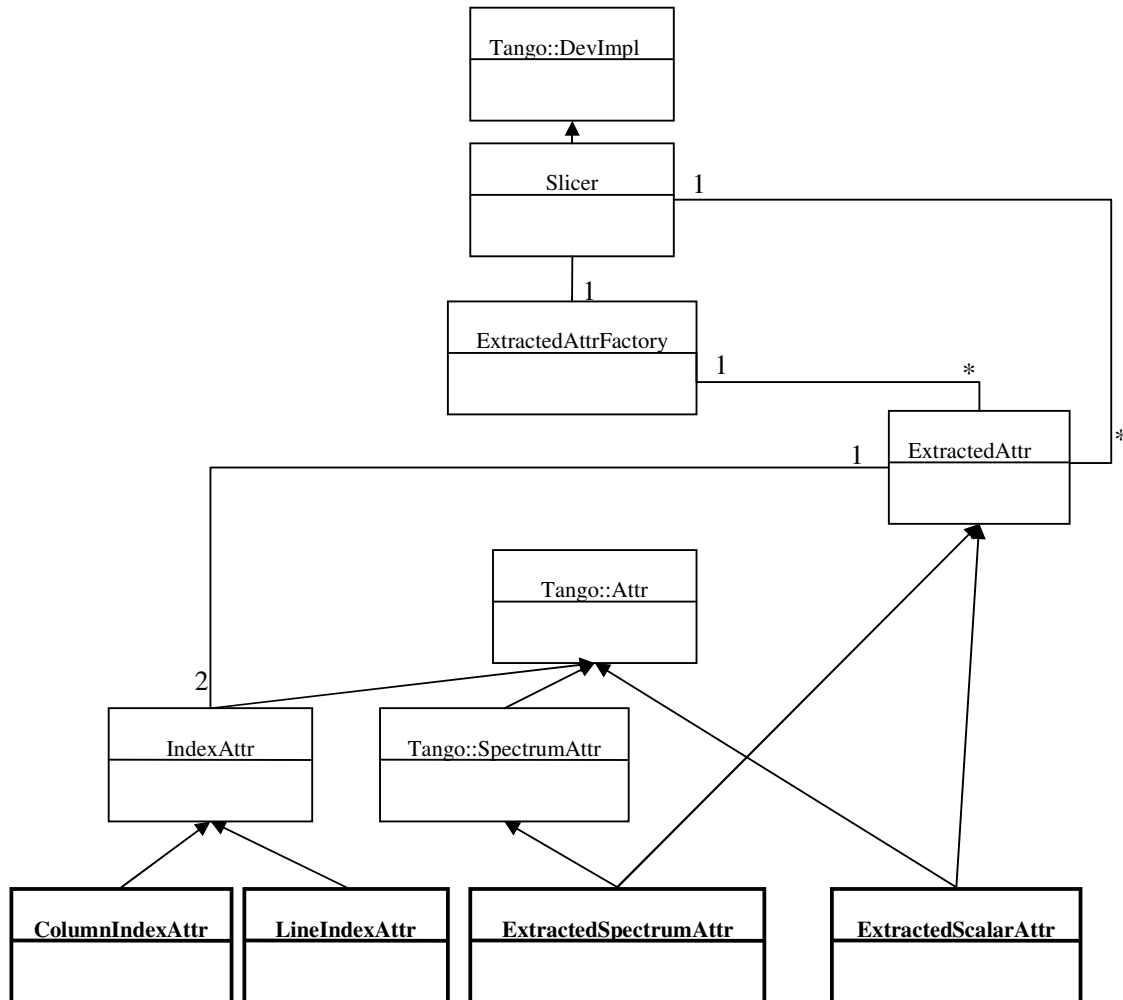
For each input attributes, the client will specify the data to be extracted by the slicer through two attributes: a column index, and a line index. Setting the line index or the column index to “-1” means respectively extracting a full column or a full line. Specifying both values as different from “-1” implies a single value extraction.

### 1.2 Some vocabulary

- Server: A device from which the slicer will extract specific data.
- Client: An application (i.e. an ATKPanel application or another device) which wants to read one of the slicer’s attributes (i.e. the extracted values from one of the input attributes)

## 2 Technical scheme of the application

### 2.1 Software Architecture



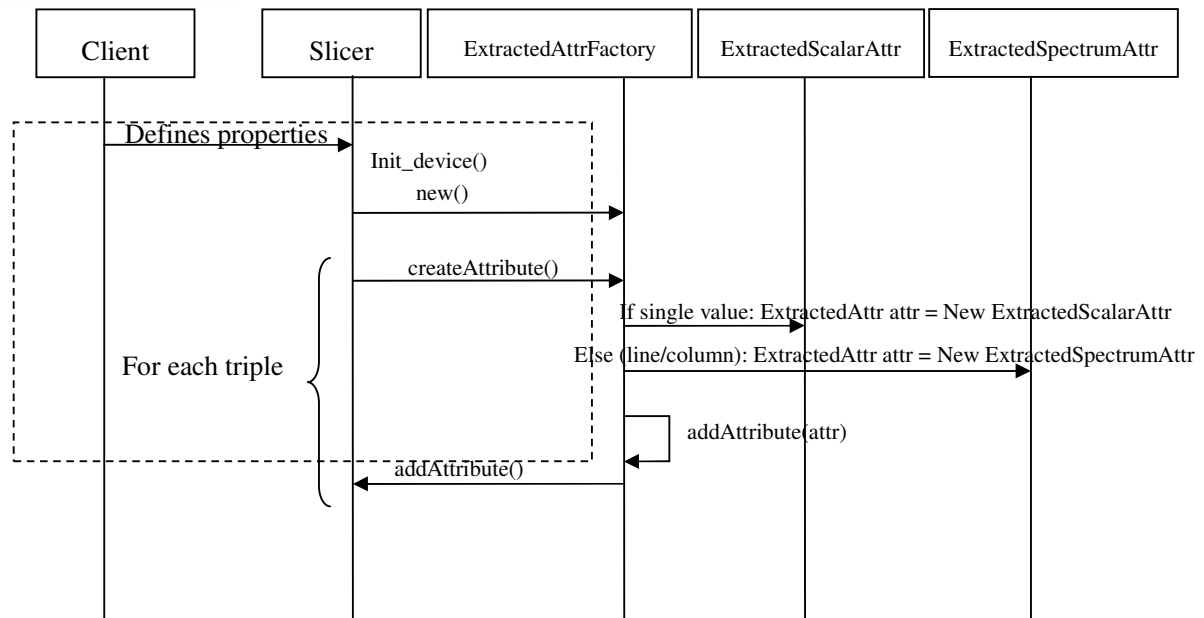
### 2.2 Design considerations/Implementation issues

#### 2.2.1 Initialization and dynamically created attributes

We specified earlier that the slicer is able to extract data (outputs) from several attributes (inputs) at the same time. The number of couples {input, output} is defined by the initial properties : the list of servers' attributes to read (inputs), the list of slicer's attributes to create (outputs) and the list of the couples (column index, line index) which defines which data and what kind (spectrum/scalar) should be extracted from each of the inputs.

Thus, we have to create the slicer's attributes dynamically: before initialization, we don't know how many attributes there will be, and what will be their respective types (spectrum/scalar).

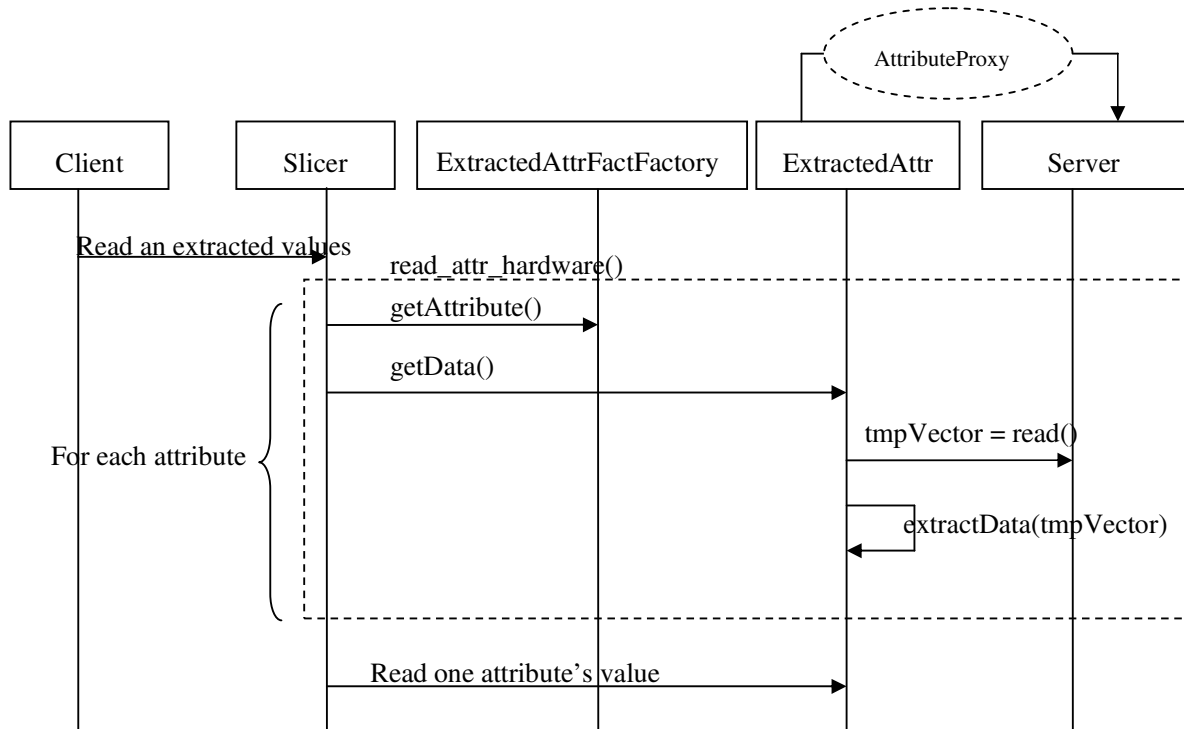
The creation of the different attributes is done during the initialization phase of the slicer (`init_device()`) :



The slicer splits the different properties lists (inputs, outputs, indexes) and check that everything is fine (same number of inputs, outputs, couples of indexes). Then, for each triple {input, output, couple of indexes}, the slicer will use the ExtractedAttrFactory to create the corresponding attribute, and to add it to its list of attributes. The ExtractedAttrFactory will create the correct type of attribute (scalar/spectrum), and create the proxy to the corresponding server/attribute.

When you use POGO to generate some device code, it will create a specific class per attribute in the SlicerClass.h file. In our case, as the default POGO generated classes had to be slightly modified, they were written in another .h file (SlicerClassExt.h and ExtractedAttr.h). Thanks to this, you won't overwrite the code if you generate the source code again with POGO.

### 2.2.2 Extraction process



The extraction process has to be described for a better understanding of this device:

- This process is triggered when a client wants to read one of the dynamic output attributes (each of them storing the data extracted from a given input attribute). Trying to read any attribute – reading several attributes at the time can occur - first implies calling the `read_attr_hardware()` method of the slicer. That's why this method will trigger the extraction process for each of the attributes that need to be read. So, for each of them (the slicer will access each of them through the `ExtractedAttrFactory`) :
  - The slicer will first get all the data stored in the corresponding server's attribute (`ExtractedAttr` → `get_data()` method) and store them in a temporary vector. The type of this vector (i.e. `vector<Tango::DevShort>`, `vector<Tango::DevBoolean>`, etc.) depends on the type of the data stored in the server's attribute (on the server side).
  - Then the slicer will extract the needed data (depending on the values of `columnIndex` and `lineIndex`) from this temporary vector (through the `ExtractedAttr` → `extractData()` template method), and assign them to `ExtractedAttr` → `extractedValues`.
- Then, for each of the attributes that need to be read, the `Tango::Attr` → `read()` method will be call. This method will return the corresponding `ExtractedAttr` → `extractedValues` previously stored.

### 3 TANGO software interfaces

#### 3.1 Properties

[POGO generated documentation](#)

InputAttributes	List of attributes which the slicer should extract data from Example: tests/tangotest/quentin/uchar_spectrum tests/tangotest/quentin2/boolean_image
OutputNames	List of names which will be given to the different dynamically generated output attributes Example: charS booleanI
Indexes	List of indexes couples (line index, column index) which define what data the slicer should extract from each of the input attributes. “-1” line index value means that : <ul style="list-style-type: none"> <li>- If the attribute is a matrix, the slicer should extract the columnIndex column (spectrum)</li> <li>- If the attribute is a spectrum, the slicer should extract the columnIndex value (scalar)</li> </ul> Whereas, a “-1” columnIndex value means : <ul style="list-style-type: none"> <li>- If the attribute is a matrix, the slicer should extract the lineIndex line (spectrum)</li> <li>- If the attribute is a spectrum, the slicer should extract the lineIndex value (scalar)</li> </ul> <b>Other than “-1” cases, indexes values start with “1” (and don’t start with “0” as it is in common development languages!).</b> Example: -1,1 1,-1

Note: All properties are arrays of <string>. That means using jive that the different inputAttributes, OutputNames, and Indexes couples, should be specify **on different lines!**

Example:

tests/tangotest/quentin/uchar\_spectrum

tests/tangotest/quentin2/short\_image

#### 3.2 Attributes

No static attributes where defined. All attributes are dynamically generated.

### 3.3 Commands

[POGO generated documentation](#)

Init	Initializes the device.  Check the “Design considerations/Implementation issues” section for details.
State	Returns the device state.  Possible states are RUNNING, FAULT (initialization failed), ALARM(extraction issue on at least one of the input attributes).
Status	Returns the device status.

### 3.4 Exceptions

When reading an extracted values attribute, several exceptions can be thrown (state set to ALARM, status set to “Extraction failed on at least one device”):

- The server’s attribute type is not supported (i.e. a scalar string attribute as the slicer can only extract numerical values). The following exceptions will be sent to the client :

*Extraction Error [Unsupported Data Type]*

- lineIndex and/or columnIndex values don’t fit with the data dimensions extracted from the server. Example: the server’s attribute is a spectrum (so that’s a matrix with a null Y-dimension) and the line index was set to any value different from -1. In this case, the client will receive such an exceptions:

*Extraction error [Out of range : ColumnIndex(1) >= DimY(0) ...]*

- both index values (line index and column index) are set to -1. This doesn’t make any sense (that would mean we don’t want to extract any value). So, the following exceptions will be sent to the client :

*Extraction error [Wrong indexes configuration- Hint: at least one index should be > -1]*

- Column index and line index values don’t match the kind of data the slicer was originally configured for, for this attribute: extract a scalar OR extract a spectrum (depending on the {line index, column index} couple in the Indexes property. At runtime, the client can modify the values of the column index and the line index attributes corresponding to this attribute only IF the new values don’t change the fact that the slicer was supposed to extract a scalar or a spectrum. If they do, the following exception will be thrown :

*Device was set up to extract a [spectrum/scalar], but it's not the case anymore.*

- One of the index was set to “0”. As mentioned before, indexes normal values should start with “1”. If “0” was specified the following exception will be thrown :

*Extraction error [Wrong indexes configuration - Hint: index value 0 is not a correct value. Indexes start with 1 value.]*

- The communication is lost between the slicer and the server (i.e. the server is not running anymore). The following exceptions will be sent to the client :

*Connection failed [Could not read from tests/tangotest/quentin/short\_scalar]*

## 4 Software Setup

- In most cases you would like to have some servers from which the slicer will extract data, so you need to create and/or launch the different DeviceServers which will be used as servers for the slicer.
- Create a DeviceServer for the slicer (class: Slicer) using jive. Create and assign values to the three needed properties: InputAttributes, Indexes, OutputNames.
- Launch the slicer executable.
- To test your configuration, you can use jive “Monitor device” panel to read any of the extracted values, or set any of the corresponding {line index, column index} couples.

## 5 Limitations and problems

### 5.1 Current known limitations

- The slicer can only extract numerical values (Tango::DevShort, Tango::DevLong, Tango::DevDouble, Tango::DevUChar, Tango::DevUShort, Tango::DevBoolean, Tango::DevFloat).

### 5.2 Troubleshooting

N/A so far.