

How the command XML code is formed

How to form a command in XML to send to AMQ is described in the document 0700A-009-1 XML & Comm Scheme.pdf, page 5-7. It gives an example of the Guider target command. Other commands are formed in a similar manner, it is just the content of the XML code is different, specific to what is needed for each specific system of the TCS.

AMQ + TCS command overview

WritetoXML

GiderClass (control)

The screenshot displays a LabVIEW project with several windows and annotations:

- Project Libraries:** A file explorer on the left shows a hierarchical structure of libraries including `TCSAtmosphereDispersionStatus.lvlib`, `TCSCommand.lvlib`, `TCSGiderClass.lvlib`, `TCSMountControlStatus.lvlib`, `TCSPublicStatus.lvlib`, `TCSStateValues.lvlib`, `NewScienceTarget.lvlib`, `AccessMethods.lvlib`, `OffsetStatus.lvlib`, `ReadHandsetOff1.lvlib`, `ReadHandsetOff2.lvlib`, `ReadOffsetType.lvlib`, `ReadUserOff1.lvlib`, `ReadUserOff2.lvlib`, `WriteHandsetOff1.lvlib`, `WriteHandsetOff2.lvlib`, `WriteOffsetType.lvlib`, `WriteUserOff1.lvlib`, `WriteUserOff2.lvlib`, `readFromXML.lvlib`, `writeToXML.lvlib`, `TCSTcsStatus.lvlib`, `CurrentTimes.lvlib`, `Limits.lvlib`, `PointingPositions.lvlib`, and `TCSTimeComponentCommand.lvlib`.
- WritetoXML:** A red arrow points to the `writeToXML.vi` block in the `OffsetStatus.lvlib` front panel. The block is annotated with the text "WritetoXML".
- GiderClass (control):** A red circle highlights the `GiderClass` control in the `OffsetStatus.lvlib` front panel. The control is annotated with the text "GiderClass (control)".
- Read from XML:** A red circle highlights the `readFromXML.vi` block in the `OffsetStatus.lvlib` front panel. The block is annotated with the text "Read from XML".
- XML out:** A text box displays the XML output generated by the `writeToXML.vi` block:

```
<offsetType>SIMPLE</offsetType>
<userOff1>0</userOff1>
<userOff2>0</userOff2>
<handsetOff1>0</handsetOff1>
<handsetOff2>0</handsetOff2>
```

- XML in:** A text box displays the XML input for the `readFromXML.vi` block:

```
<tcsErrorResponse>
<code>0</code>
<source></source>
<status>false</status>
</tcsErrorResponse>
<offsetDef>
<num1>Handset</num1>
<off1>10.0</off1>
<off2>30.0</off2>
<offsetType>SIMPLE</offsetType>
</offsetDef>
</scienceTargetOffset>
```

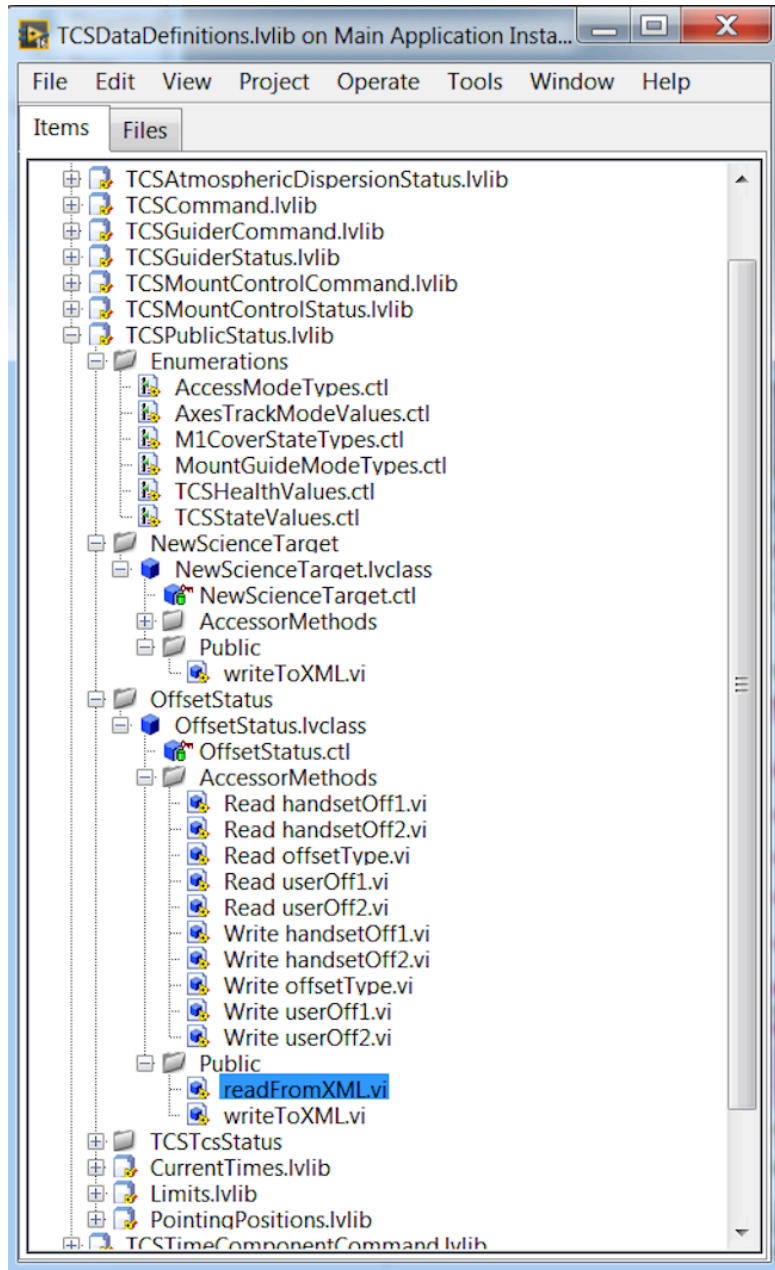
- error in (no error):** A control for the `readFromXML.vi` block, showing a status code of 0 and a source of 9305.
- error out:** A control for the `readFromXML.vi` block, showing a status code of 0 and a source of 9305.

How is it done in LV?

In LV it is done in two steps:

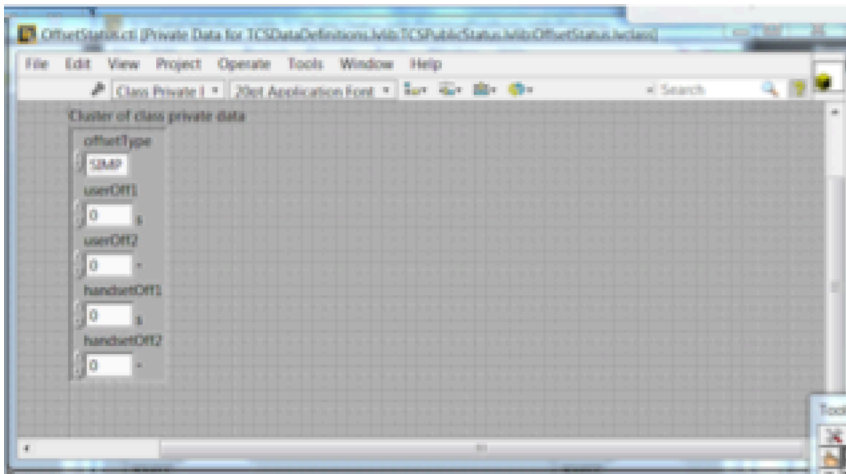
- (1) Form the XML command (specific to the TCS subsystem).
- (2) Send (or publish?) to AMQ.

Library structure



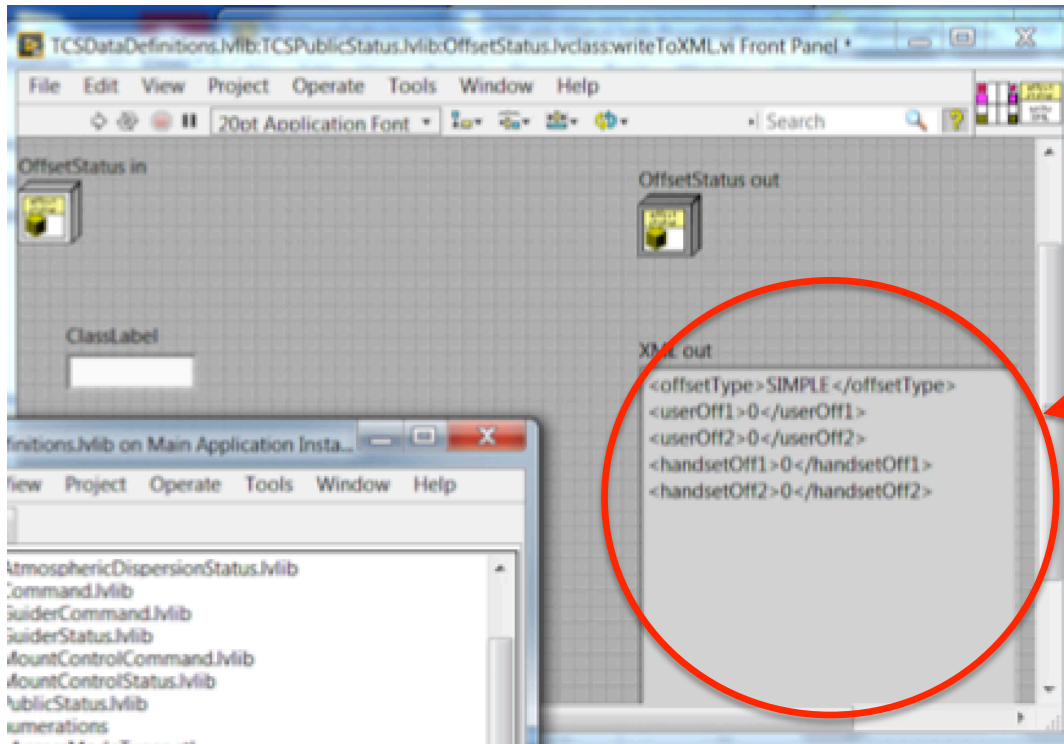
- The project that contains all the necessary Vis to create XML code is TCS DataDefinitions.lvlib.
- Project structure has numerous sections that are responsible for various shared variables (SV in their notation) in the TCS. The folder that is expanded for illustration is TCSPublicStatus.lvlib with OffsetStatus folder as an example. It has a number of Vis that are all doing XML conversion from and to specific variables. The variables that they use are defined as LV classes. For each TCS SV there is corresponding LV class (at least that is what I understood from their structure). None of these Vis sends or receives the commands to the AMQ, all they do is to convert (flatten/unflatten) XML code specific variables into the class.
- Suppose we need to create an XML code that we would send to the AMQ.
 - First we would use one (or many) of the Vis from the AccessMethods to input the required values of the SVs in the OffsetStatus guider class.
 - Then we would have to use WriteToXML.vi to produce XML code.

LV class example



- Here is an example of LV Class that is used for the Guider. It is basically a Control (a cluster), that has a number of variables specific for the Guider in this case. In case of some other components of the telescope it can be different variables, but it remains a cluster with a number of strings, numbers etc.

Forming XML code



- To produce the XML code from the LV Class variable we would use Write to XML.VI
- Its output is the piece of XML code that goes to (in this case) to the Guder.

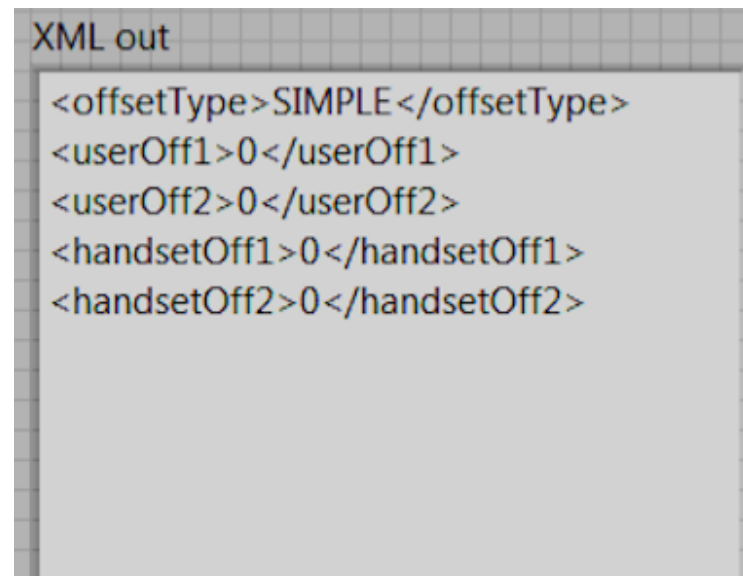
Submitting the command to AMQ

- After the command is formed it needs to be sent to AMQ.
- To send it to AMQ we use SimpleWriter that we used before.

Issues that are not understood

- Although the logic overall is understood, there are few things that are not clear:
 - if you compare the XML code that is created by the XML Writer with the XML example code that Geordi provided to us you will see that there are some lines that are missing. The LV Writer makes the body of the XML code that has all the content for the LV Class, but it doesn't have the header (for a lack of better word) that describes the command itself. At this point I do not know how to add that header. It is not to prepend/append that kind of code, so we could have it embedded in our Vis, but it may be good to see if they already have it implemented somewhere.

```
<scienceTargetOffset>
  <commandID>750237349</commandID>
  <tcsErrorResponse>
    <code>0</code>
    <source></source>
    <status>false</status>
  </tcsErrorResponse>
  <offsetDef>
    <num1>Handset</num1>
    <off1>10.0</off1>
    <off2>30.0</off2>
    <offsetType>SIMPLE</offsetType>
  </offsetDef>
</scienceTargetOffset>
```

A screenshot of a code editor window titled "XML out" showing XML code. The code is as follows:

```
<offsetType>SIMPLE</offsetType>
<userOff1>0</userOff1>
<userOff2>0</userOff2>
<handsetOff1>0</handsetOff1>
<handsetOff2>0</handsetOff2>
```


Issues that are not understood /2

- In his message Georgi says that the message ID should be different for each command. I am not sure I understand what it means. Does it mean that every time we subscribe to the Topic it should be different? Or does it have to be different for each of the topics we subscribe to? And if it is different, who keeps track of it?

Hi Alexander ,

With access to the ActiveMQ web page, you can try sending, for example, offsets to the TCS, and you can look at the response using the appropriate topic. For example, to test sending a SIMPLE offset (RA, Dec), you can do this:

-- Click on Topics
-- Select TCS.TCSharedVariables.TCSubData.TCSTcsCommandSV
-- Paste into Message Body this XML code:

```
<scienceTargetOffset>
  <commandID>750237349</commandID>
  <tcsErrorResponse>
    <code>0</code>
    <source></source>
    <status>false</status>
  </tcsErrorResponse>
  <offsetDef>
    <num1>Handset</num1>
    <off1>10.0</off1>
    <off2>30.0</off2>
    <offsetType>SIMPLE</offsetType>
  </offsetDef>
</scienceTargetOffset>
```

Please note that the commandID number must be different for every command.

Commands with the same ID will be ignored.

You can look at the results using the following topics:

TCS.TCSharedVariables.TCSHighLevelStatusSV.TCSTcsStatusSV
TCS.TCSharedVariables.TCSLowLevelStatusSV.TCSTcsCommandResponseSV
which give you information about possible errors, what the offsets were, etc.
Eventually, of course, this should be done by your software, which should send the appropriate XML packets to the ActiveMQ broker, which will then send them to the MCB and the TCS.
Georgi

Issues that are not understood /3

- There are other issues that are there related to the subscription to AMQ topics which are my ignorance with AMQ, may be Kenichi can answer those. For instance, I am not sure if we need to subscribe to the topic every time we run SimpleReader.vi We do AMQ_Init.vi and then AMQ_subscribe.vi inside of the reader now and then AMQ_Close at the end. If we do that for every topic that we use, will it work? Or can we (or should) init AMQ and then subscribe to a variety of topics, all the topics that we need and then just run them until the end of the session?

Reading from the Topics

- Reading would be done in the reverse order:
 - (1) read from AMQ Topic, receiving the message in XML
 - (2) use ReadFrom XML **using specific for that SV reader and unbundle by name from the cluster into individual variables that we can use.**

This is pretty simple and we are very close to have the whole process working, it is mostly figuring out the variable and the Vis that are used for them

One very important piece of it is in the document 0720S-005-8 TCS Interface Definitions.pdf. Paragraph 3.5 (pages 30-35, highlighted text). These are practically all the variables that we would need to use. What is not clear to me how we subscribe and read to them. From Georgi's example it looked like we are getting only one number for that TCS.TCSSharedVariables.TCSLowLevelStatusSV.TCSTcsCommandResponseSV, Whereas there are many of those variables in that topic. May be I am missing something here, let me know if you understand how to do it.