DCT Cross-platform Messaging and XML Data Format Design Description

DCT-0700A-009-1

July 29, 2010

Author: Daniel Greenspan

Approved by: Byron Smith

Lowell Observatory
1400 West Mars Hill Rd.
Flagstaff, AZ 86001
Tel: 928 774-3358

Revision History

Revision Number	Author	Date	Description of Changes	ECN #
A	D. Greenspan	11 Aug 2009	Initial Release	
1	D. Greenspan	29 July 2010	Added a description of empty XML value shorthand tags; corrected spelling and terminology.	347

Table of Contents

List of Figures

1. Acronyms

DCT	Discovery Channel Telescope
LIG	Lowell Instrument Group
JMS	Java Messaging System
CMS	C++ Messaging Service
OS	Operating System
W3C	World Wide Web Consortium
DLL	Dynamic Link Library

2. Introduction

DCT Project is an effort to design, construct and install a 4.3-meter telescope and support facility. The DCT incorporates a software system comprised of elements from multiple sources. This document describes one method for communication between disparate software systems within the DCT facility and also describes the data format scheme.

2.1 Description

Software components generated by the DCT group may not be capable of communicating directly with software from external organizations, so a method of enabling communication between dissimilar components is necessary.

The methods described in this document have been chosen to address a specific need to communicate between DCT and LIG software but are not limited to that application. The system described is intended for communication between any number and type of components.

DCT software is written primarily in LabVIEW; LIG software is written primarily in Java. The two languages do not share a message-passing protocol, so an intermediary is used to handle DCT/LIG communication. The intermediary is a JMS broker.

Although the broker addresses the lack of a common message-passing interface, it does not define the format of the messages. DCT has chosen an XML format for the messages, which is described in this document. The content of the messages is not described in this document (see DCT-0720D-010 for the data description).

2.2 Scope

This document specifies:

- A communications architecture for passing messages between DCT and other systems

- The XML schema used to represent objects passed within messages

2.3 Definitions

- Software objects that are composed of other objects are said to "contain" those objects.
- An XML element is a set of start and end tags surrounding one or more XML elements or a value.
- The term "object" refers to LabVIEW or Java software constructs; the term "element" refers to an XML representation of an object or value within an object.

2.4 References

Controlling Documents

Document Number	Document Title	Notes
DCT-0700A-003	DCT Software Architectural Design	
DCT-0700A-005	DCT Data Communications Design	
DCT-0700A-006	DCT Software Engineering Process	

Dependent Documents

Document Number	Document Title	Notes

Referenced Documents

Document Number	Document Title	Notes
DCT-0720D-010	TCS Data Definitions	
DCT-0000S-002	Top-Level Telescope & Facility Requirements	
DCT-0720S-005-I	TCS External Interface Definitions	
	http://activemq.apache.org/	ActiveMQ page
	http://www.w3.org/TR/REC-xml/	XML specification

3. Brokered Messaging System

3.1 Description

The cross-platform message broker employed by DCT is "ActiveMQ," an open-source JMS-compliant application supplied by the Apache Software Foundation. A detailed description of ActiveMQ and JMS is outside the scope of this document but a brief description is warranted.

In this implementation, ActiveMQ lies between the DCT and LIG systems, which do not directly communicate. Each system publishes its own data to the broker and subscribes to the other system's data via the broker. Messages are organized using "topics," which are string identifiers defined by each publisher. Subscribers need to specify the desired topic but do not define it. An arbitrary number of subscribers may exist for any published topic, with each one receiving a copy of each message. Only the broker is aware of which systems are publishing and subscribing to data. ActiveMQ decouples publishers and receivers and is stateless.

DCT will publish a topic but won't "know" that LIG components are subscribed to that topic; LIG components will likewise publish a topic and won't "know" that DCT is subscribed. The DCT, Broker and LIG applications are operating-system independent and can reside on one or more machines.

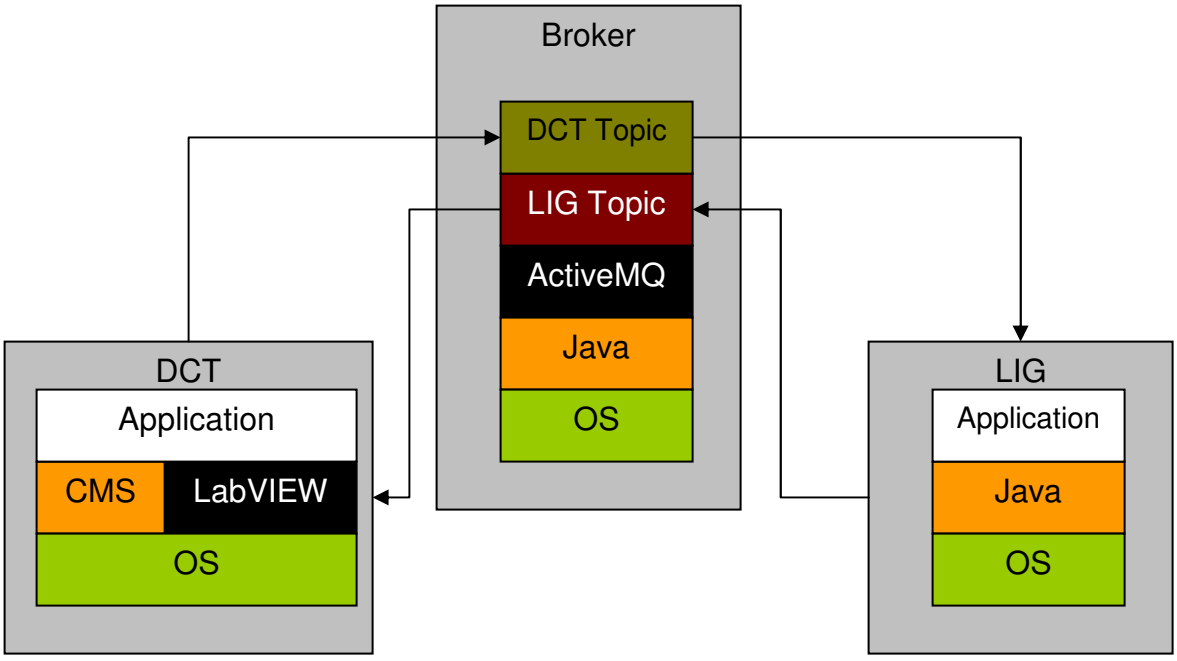

Figure 1: Brokered communications between DCT and LIG software using ActiveMQ.

3.2 Implementation

3.2.1 ActiveMQ Installation

The installation of ActiveMQ is simple as it does not require configuration. Installation instructions and system requirements are specified at the ActiveMQ web site (see referenced documents).

3.2.2 Java and ActiveMQ

ActiveMQ implements the native Java messaging system (ActiveMQ is written in Java); Java and ActiveMQ can communicate directly.

3.2.3 LabVIEW, ActiveMQ and CMS

LabVIEW does not have a native implementation of JMS. To overcome this incompatibility, the CMS library supplied by the Apache Software Foundation will be utilized. CMS is written in C++ and will be compiled into a Windows DLL that LabVIEW can use. CMS will generate and receive the JMS-compatible messages on behalf of the DCT LabVIEW application layer. CMS will notify LabVIEW of incoming messages by generating a LabVIEW event. LabVIEW will send configuration parameters and messages by passing them to the CMS library.

4. Data Format

The messages will contain XML representations of software objects. The format (not the content) of XML data is defined in paragraph 4.1. The XML implemented is not the full W3C standard but uses the style of that standard.

4.1 XML Representation

1. The atomic unit of data transmission is the JMS message. Each message contains only one object (the "top-level" object), which may encapsulate child objects.

2. Top-level objects are defined in the TCS external interface document (see referenced documents).

3. Objects are represented using XML elements.

4. Each XML element is composed of one or more child elements **or** (inclusively) a value.

5. All values are represented in ASCII.

6. The type of a value is not represented and must be interpreted appropriately by the XML consumer.

7. uninitialized or valueless objects may be represented using the shorthand of a single tag terminated with a slash, i.e. the following two lines are equivalent:

   <thing></thing>

   <thing/>

4.2 Whitespace

The following whitespace pattern is recommended but not required to maintain human readability:

- Top-level element tags are not indented.

- Child objects are indented using one space doublet (ASCII 0x20 0x20) added to the number of doublets used by the parent object. Peers share the same indent.

- If an object contains one or more objects, its start and end tags are terminated with a windows EOL (ASCII 0x0D 0x0A) sequence.

- If an object contains only a value, the start tag, value, and end tag are on one line terminated by a CRLF sequence.

These recommendations are for cosmetic purposes only. The actual whitespace used is irrelevant; End-Of-Lines can be of any type desired and arbitrary spaces and tabs (or none at all) can be used.

5. Examples

Figure 2 is the UML definition of a GuideTargetPosition object, which is composed of an XPosition and a YPosition object (GuideTargetPosition contains the XPosition and YPosition objects). The XPosition and YPosition position objects are peers; each contains a single value.

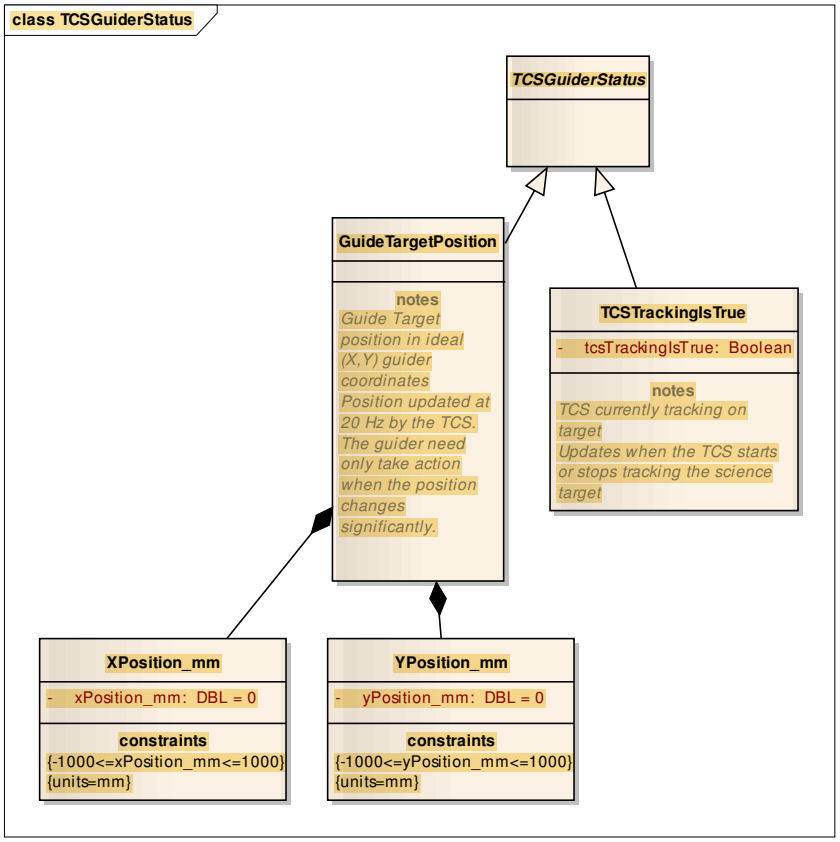

Figure 2: UML representation of GuideTargetPosition

The XML representation of GuideTargetPosition as represented in Figure 2: UML representation of GuideTargetPosition is

```
      <GuideTargetPosition>
        <XPosition_mm>
          <xPosition_mm>10</xPosition_mm>
        </XPosition_mm>
        <YPosition_mm>
          <yPosition_mm>2.2</yPosition_mm>
        </YPosition_mm>
      </GuideTargetPosition>
```

If an object inherits an attribute (value) from a parent class, the attribute will be a peer of any other member objects that may exist; the attribute and the member objects will be "peer children" of the parent object. A contrived example is shown below; let us suppose that `GuideTargetPosition` inherits an attribute named "Numeric" from its parent class `TCSGuiderStatus`. The definition and XML representations are almost the same as above; note that the TCSGuiderStatus node now contains the "Numeric" attribute.

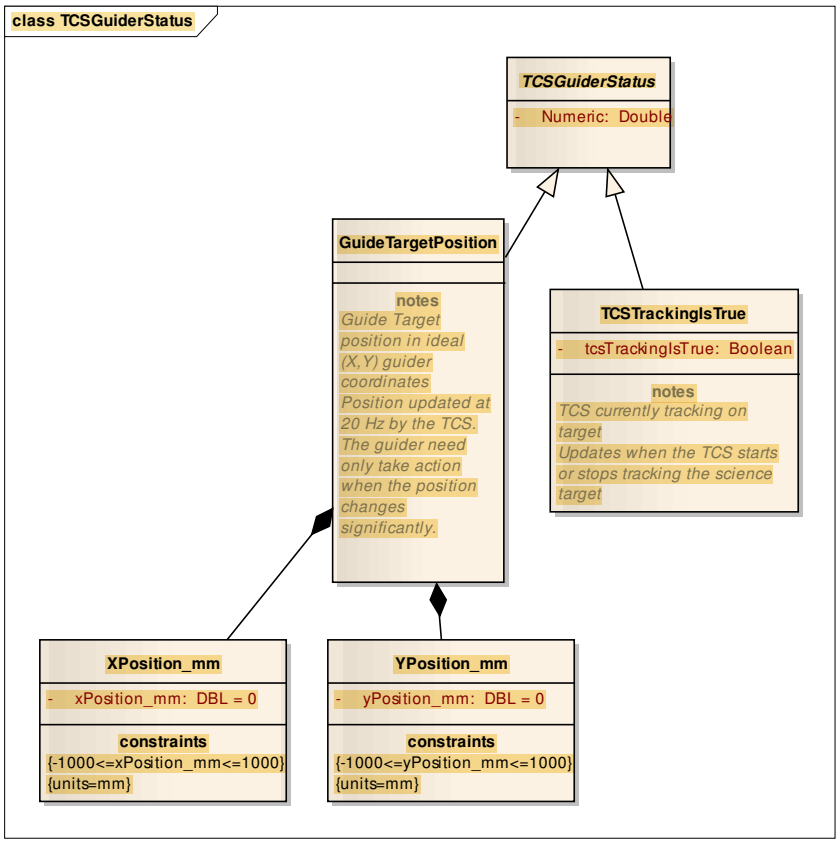

Figure 3: UML representation of GuideTargetPosition if it had a parent attribute

The XML representation of the UML representation in Figure 3: UML representation of GuideTargetPosition if it had a parent attribute is

```
<GuideTargetPosition>
  <Numeric>2.34</Numeric>
  <XPosition_mm>
    <xPosition_mm>10</xPosition_mm>
  </XPosition_mm>
  <YPosition_mm>
    <yPosition_mm>2.2</yPosition_mm>
  </YPosition_mm>
</GuideTargetPosition>
```