Observatory Sciences Ltd
www.observatorysciences.co.uk

LabVIEW Attributes and Tables

Issue: 1.0

Date: 14.10.2010

OSL Author(s): A. Greer ……………………………………………….

Name Date Signature

CHANGE RECORD

ISSUE	DATE	SECTION/PARA. AFFECTED	REASON/INITIATION DOCUMENTS/REMARKS
1.0	14.10.2010	All	First issue

TABLE OF CONTENTS

1 Introduction 4
1.1 Scope 4
1.2 Intended Audience 4
1.3 Abbreviations and Acronyms 4
2 Overview 5
3 Installation 7
3.1 Prerequisites 7
3.2 Installation Instructions 7
3.3 Demo Application 7
4 Description 9
4.1 CreateAttribute.vi 9
4.2 CreateAttributeTable.vi 9
4.3 InsertIntoAttributeTable.vi 10
4.4 GetAttributeFromTable.vi 10
4.5 GetValueFromAttribute.vi 11
4.6 FormatAttributeTable.vi 12
5 Future Development 13

1 Introduction

1.1 Scope

This document describes the use of LabVIEW Attribute and AttributeTable VIs allowing flexible data storage whilst maintaining the same overall structure of the data type.

1.2 Intended Audience

The intended audience of this document is software architects, software managers, designers and potential integrators of LabVIEW systems.

1.3 Abbreviations and Acronyms

1D	One Dimensional
2D	Two Dimensional
VI	Virtual Instrument

2 Overview

Attributes are (name, value) pairs where the value can be either an integer, string, double or boolean. As well as single values attributes can be either 1 or 2 dimensional arrays of values. Internally the values are stored as strings, thus keeping the structure of the Attribute unchanged no matter which data type is stored within. The Attribute also keeps count of the array size in the case of 1D and 2D arrays to ensure the same array is passed out that is passed in. The string storage allows quick presentation of the attribute for logging or displaying. A later version under development is looking at storing attributes as arrays of unsigned bytes. This greatly reduces the size of the Attribute and is designed for high speed transfer of Attributes across a network. For more information see section 5.

Sets of logically-related Attributes are collected into AttributeTables. For example, all the information that is required to perform a specific action might be collected into a single AttributeTable. AttributeTables are unordered collections of Attributes. Only a single Attribute with a given name can exist in an AttributeTable. Locating an Attribute within an AttributeTable is performed using the name of the Attribute, AttributeTables can be thought of as hash maps.

The structure of an AttributeTable will never change even though the data contained within the Attributes can be entirely different between tables. This provides a completely flexible method of data transport and storage. At Observatory Sciences AttributeTables and Attributes are used throughout all of our LabVIEW projects and we have developed various services built on top of them. The figures below shows the structure of an Attribute and an AttributeTable.

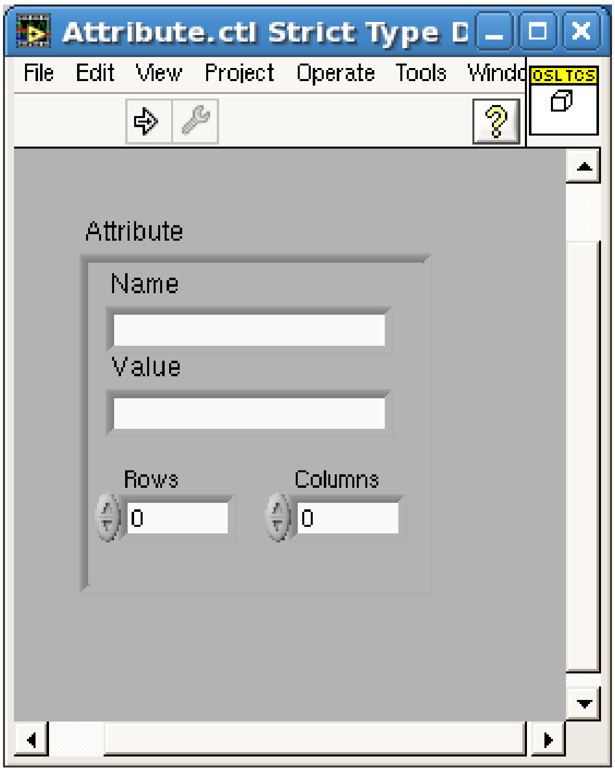


Figure 1 Attribute control.

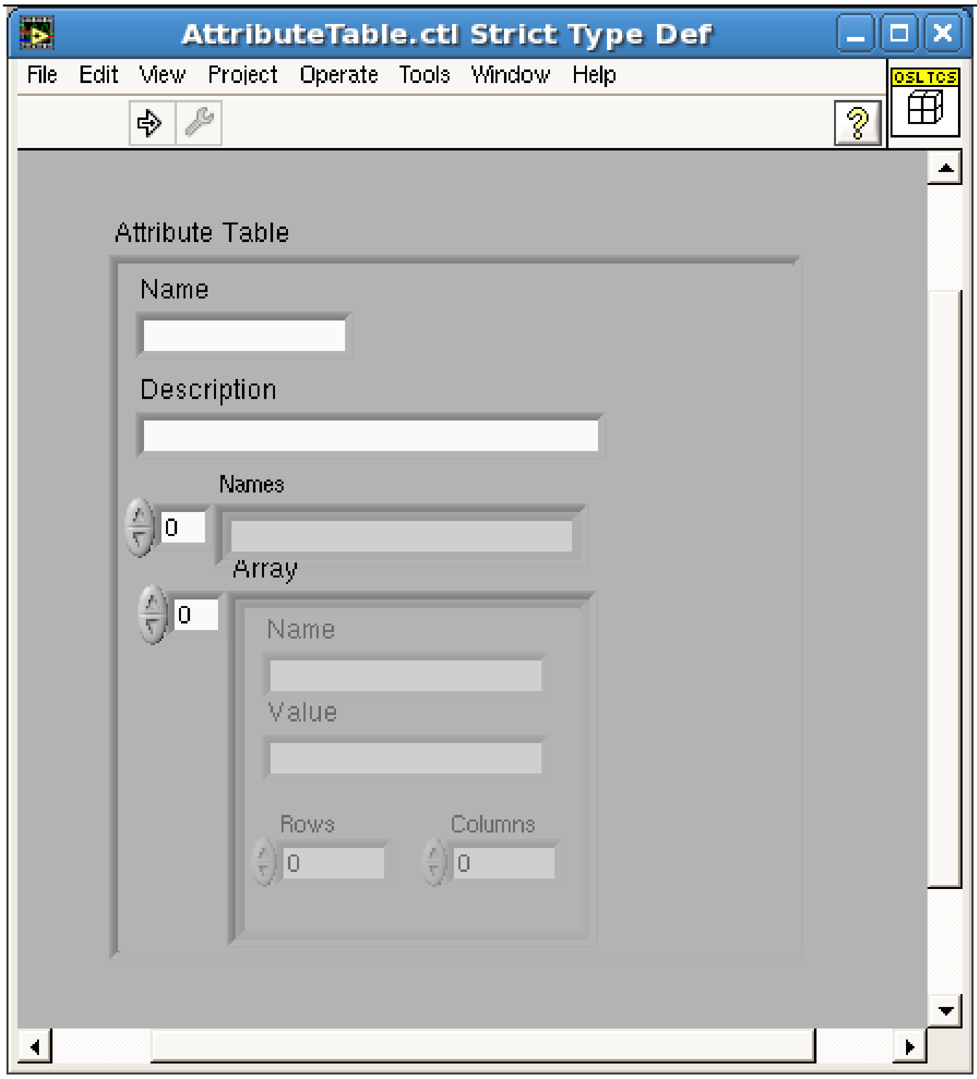


Figure 2 AttributeTable control.

3 Installation

This section details installation of the AttributeTable VIs.

3.1 Prerequisites

The software has been developed and tested on Windows XP and Linux RHEL 5. The software should work on any platform supported by LabVIEW version 2009.

3.2 Installation Instructions

To use the VIs simply unpack the archive to wherever required. The VIs are separated into two directories, the main directory containing the public VIs and a _private directory. The _private directory contains the internal VIs and is named so that the whole set of VIs can be placed in the user.lib folder of a LabVIEW installation and only the public VIs will be visible from the menu.

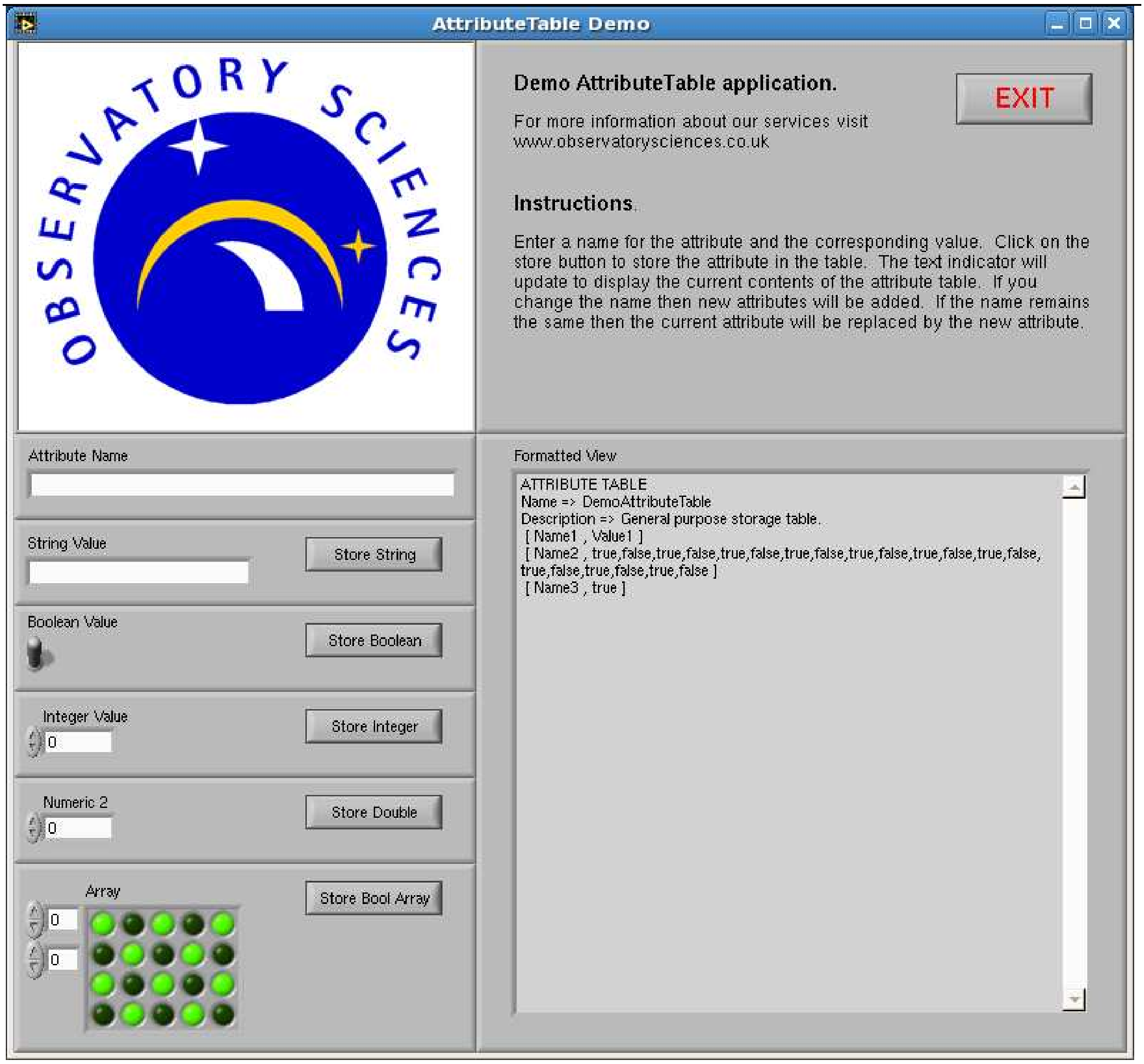
Once the VIs have been added to a project file they can be dragged into any other open VI. Use of LabVIEW projects is beyond the scope of this document.

3.3 Demo Application

There is a demonstration VI that creates and displays an AttributeTable. This can be found at

```
<unpacked_location>/Attribute/AttributeDemoProgram.vi
```

The VI can be run to demonstrate how AttributeTables are constructed and manipulated. The figure below show the demonstration program. Once running Attributes can be added to the AttributeTable by entering a name into the “Attribute Name” text entry control and selecting a value for the Attribute and then clicking on the corresponding button to store that Attribute in the table. The “Formatted View” text indicator shows the current AttributeTable in a user friendly string format. As many Attributes can be added to the AttributeTable as is required; simply change the name and value and click on the corresponding button. If the name is already present in the AttributeTable then the current Attribute is replaced by the new Attribute. There is an example array control to show how 2 dimensional arrays of values can be stored. Click on the LEDs to obtain the desired pattern and then insert the array into the AttributeTable. Note that the dimensions of the array are stored within the attribute even though the displayed value is a linear representation. This ensures the same 2D array can be retrieved from the table.



Demo AttributeTable application.

For more information about our services visit www.observatorysciences.co.uk

Instructions.

Enter a name for the attribute and the corresponding value. Click on the store button to store the attribute in the table. The text indicator will update to display the current contents of the attribute table. If you change the name then new attributes will be added. If the name remains the same then the current attribute will be replaced by the new attribute.

```
ATTRIBUTE TABLE
Name => DemoAttributeTable
Description => General purpose storage table.
 [ Name1 , Value1 ]
 [ Name2 , true,false,true,false,true,false,true,false,true,false,true,false,true,false,
true,false,true,false,true,false ]
 [ Name3 , true ]
```

Figure 3 AttributeTable Demonstration VI

4 Description

The following section describes in detail each public Attribute and AttributeTable VI. There are no restrictions placed on any application using the Attribute VIs (other than LabVIEW version restrictions that are beyond our control). The VIs use only low level LabVIEW nodes present in both Linux and Windows versions and so should be completely compatible. We already use Attributes to send information from Applications running on Linux across the network to applications running on Windows.

4.1 CreateAttribute.vi

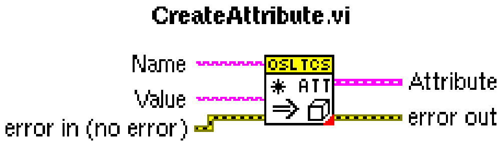


Figure 4 CreateAttribute VI

Inputs: Name. The name of the Attribute to create (string).
Value. The value of the Attribute (polymorphic).
Standard error in
Outputs: Attribute. The constructed Attribute item.
Standard error out
Notes: This VI is polymorphic. It accepts the following values:
boolean
1D array of booleans
2D array of booleans
string
1D array of strings
2D array of strings
integer
1D array of integers
2D array of integers
double
1D array of doubles
2D array of doubles

4.2 CreateAttributeTable.vi

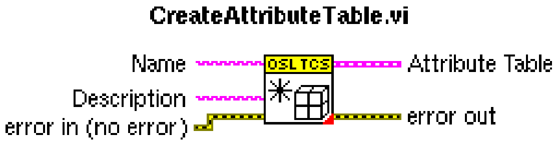


Figure 5 CreateAttributeTable VI

Inputs: Name. The name of the AttributeTable to create (string).
Description. A textual description of the AttributeTable (string).
Standard error in.

Outputs: AttributeTable. The constructed AttributeTable item.
Standard error out.

Notes: This creates an empty AttributeTable with the given name and description ready to accept Attributes.

4.3 InsertIntoAttributeTable.vi

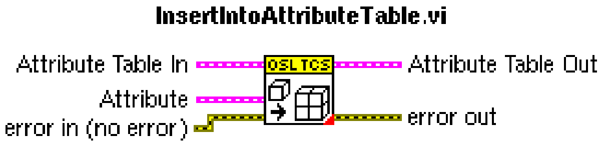


Figure 6 InsertIntoAttributeTable VI

Inputs: Attribute Table In. The AttributeTable that the Attribute(s) are inserted into.
Attribute. This is polymorphic. Either a single Attribute or an array of Attributes are accepted.
Standard error in

Outputs: Attribute Table Out. The AttributeTable with the Attribute(s) added.
Standard error out

Notes: The VI inserts the given Attributes into the AttributeTable. If any of the Attributes already exist within the AttributeTable then they are replaced, otherwise they are simply added. The VI is polymorphic, accepting single Attributes or arrays of Attributes.

4.4 GetAttributeFromTable.vi

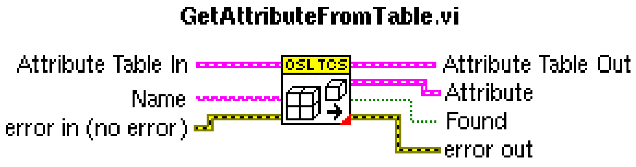


Figure 7 GetAttributeFromTable VI

Inputs: Attribute Table In. The AttributeTable that the Attribute(s) are inserted into.
Name. The name of the Attribute to retrieve from the table (string).
Standard error in

Outputs: Attribute Table Out. The same AttributeTable that was passed in.
Attribute. The retrieved Attribute. If no Attribute was found with the name supplied this is an empty (default) Attribute.
Found. This is true if the Attribute was found in the AttributeTable (boolean).
Standard error out.

Notes: Call this VI to retrieve an Attribute from the AttributeTable by name. The "Found" value should always be checked to ensure the Attribute existed, if it did not then the returned Attribute will be empty. This does not delete the Attribute from the AttributeTable.

4.5 GetValueFromAttribute.vi

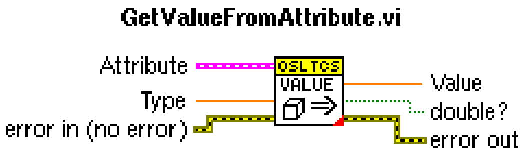


Figure 8 GetValueFromAttribute VI

Inputs: Attribute. The Attribute to retrieve the value from.
Type. This VI is polymorphic and can accept multiple types. The value is not used, only the type is important.
Standard error in

Outputs: Value. The value retrieved. The type of this output will be the same as the input "Type" parameter.
Validity flag (double?). Many of the Get VIs have this flag to dictate if the retrieved value can indeed be what was requested. For example if the Attribute was actually a boolean then attempting to get a double would result in this flag being set to false.
Standard error out

Notes: This VI will convert an Attribute into a value if possible. It is polymorphic and can accept the following "Type" values

boolean
1D array of booleans
2D array of booleans
string
1D array of strings
2D array of strings
integer
1D array of integers
2D array of integers
double
1D array of doubles
2D array of doubles

4.6 FormatAttributeTable.vi

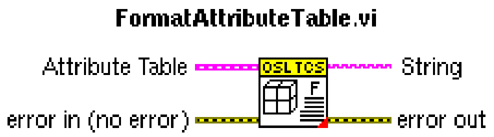


Figure 9 FormatAttributeTable VI

Inputs: Attribute Table In. The AttributeTable that the Attribute(s) are inserted into.
Standard error in.

Outputs: String. A nicely formatted string representation of the AttributeTable, used for display and debugging.
Standard error out.

Notes: This is a helper VI. We use this VI for debugging and logging of AttributeTables when required.

5 Future Development

There are planned upgrades for the Attribute and AttributeTable VIs but currently until it is seen as a necessity to implement the new versions they will stay as they are. Below is a list of possible upgrades.

1) Currently sending AttributeTables across a network is inefficient as they are stored as strings. We would like to reproduce the AttributeTable structure with internal storage as arrays of unsigned bytes. A streamlined method for storage has been devised, but the implementation has not yet been attempted.
2) Is it possible to store any LabVIEW structure as an Attribute using the variant type? We think it is.
3) We already use a publish-subscribe mechanism for sending AttributeTables between different LabVIEW applications on different operating systems for our own internal projects. It would be nice to re-implement this as a modular set of code using the new Attribute structure discussed in point 1.

As well as the AttributeTables Observatory Sciences have developed the following sets of VIs:

1) SNMP trap server
2) GPS Bancom Time card VIs (TAI, UTC, GPS)
3) Event server (publish/subscribe for attributes)
4) Command/Action/Response for command control
5) TCP server coupled with above for issuing commands over TCP
6) Health service
7) EPICS LabVIEW interface VIs (no library required, implemented at channel access level using UDP & TCP)
8) Delta Tau PMAC motion controller VIs (ethernet interface, no library required).

Whilst these are currently only used for our internal projects, we would be happy to discuss any requirements relating to these VIs or any other LabVIEW projects. Please visit us at www.observatorysciences.co.uk.

__oOo__