Current Value Table (CVT) Reference Library

Publish Date: Dec 13, 2013

Overview

Many machine and industrial control applications require a mechanism to store and manage data in one location in the application so that different parts of the application have access to the current value of I/O channels and others variables. The Current Value Table (CVT) is a set of LabVIEW VIs that developers use to store and retrieve data asynchronously from different parts of an application. The CVT is based on functional global variables, also called LabVIEW 2 style global variables, and can be used on most LabVIEW targets.

Table of Contents

1. Downloads

Install Library: This link will launch VI Package Manager and install the latest CVT package.

Alternative Download: If you require a direct download, you can access the VIP file from our FTP directory. We recommend the VIP file with the highest version number of most recently modified date.

Software Requirements

- **Application Software: LabVIEW Development System 2010 SP1**
- **Toolkits and Addons: VI Package Manager (Community)**

2. Purpose

Machine control, automation and monitoring applications are typically developed as a number of independent processes running on one or more systems. Each process performs a separate task such as communication to other systems, I/O, control logic, data logging, etc. These independent processes share a set of common data or variables which allows them to work together to accomplish the tasks of the application. Sharing one common set of data enables centralized I/O operations with the I/O data being shared by many processes.
The Current Value Table (CVT) serves as the central data component and can be used in a wide range of applications. It allows other application components to share a common data repository an have direct access to the most up-to-date value of any variable used between components. Using this architecture one component in the application can handle all of the I/O operations and share th I/O data variables or tags with the rest of the application components. Application operations such as alarm detection, user interface updates, process logic, etc. are all handled by separate processes that share the same data repository.

Figure 1: Placing the CVT in context of the application functional blocks

3. Implementation

The Current Value Table is implemented in a two layer hierarchy consisting of core VIs and API VIs. The core contains all of the functionality of the CVT, including the data storage mechanism and additional service functions. The API VIs provide access to the CVT functionality in a simple interface. There are three groups of API functions that provide slightly different access to the CVT and vary in flexibility and performance, as well as easy-of-use.

Figure 2: CVT VI Hierarchy

Across the core and API implementations, the CVT manages different data types in different sets of VIs. For all of the core and API VIs there are separate implementations for each CVT supported data type Users can extend the supported data types in the CVT by using the current VIs as a template and adding VIs for additional data types. Booleans, 32-bit integers, double precision floating-point numbers and strings are currently supported.

Core

The core VIs consist of two VIs for each data type supported by the CVT. Each of these VIs is configured to provide a set of functions or methods to their callers and uses shift registers to store data between calls to the VI.

The first of these VIs is the data storage VI called MemBlock. It contains a shift register which stores all of the CVT data for a single data type. It contains a few basic functions (Init, Read, Write) tha provide access to the data. Different variables of the same data type are stored in an array in the MemBlock VI. To access an individual variable the caller needs to know the index of the desired dat item in the array.

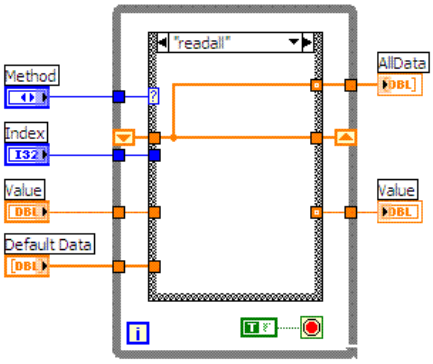
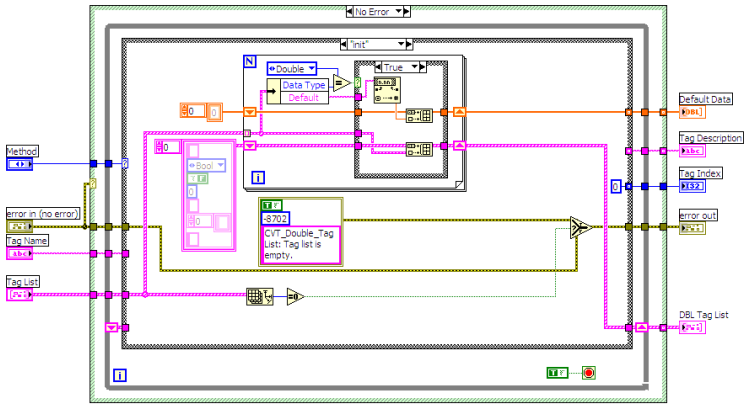
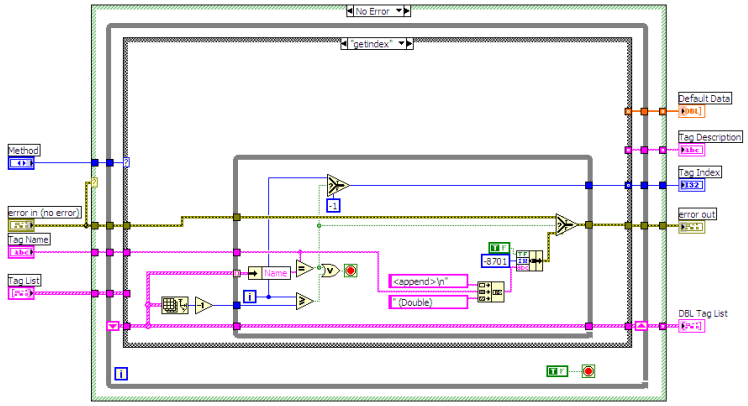


Figure 3: Implementation of the CVT MemBlock Core VI

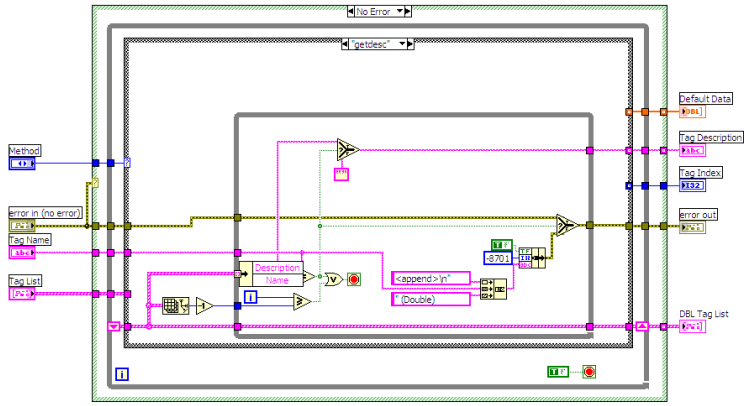
The second core VI (TagList) maintains a list of properties for each of the values stored in the MemBlock VI including the name and index of each value. This VI enables an application to retrieve the index in the storage array using the variable name. It also stores a string description for each value.



[+] Enlarge Image



[+] Enlarge Image



[+] Enlarge Image

[+] Enlarge Image

Figure 4: Implementation of the CVT TagList Core VI

API - Using the CVT

The CVT contains three sets of API functions to provide different interfaces that can be chosen according to the needs of an individual application. The basic API provides simple write and read functionality. Two additional APIs provide a higher performance interface to the CVT, but place some restrictions on the application when using the CVT. All three APIs share the same common core VIs and can be used in conjunction with one another, so you can choose the appropriate function for each individual access to the CVT.

Initialize

Before the CVT can be used to store and retrieve data it needs to be initialized to allocate memory in the MemBlock and define the variable properties in the TagList. The initialization of the CVT is defined using a cluster array that contains the attributes of all variables used by an application. This cluster array is defined as a TypeDef (CVT_TAG.ctl) in the CVT and can be created as a constar on the diagram or loaded from a file (see below).

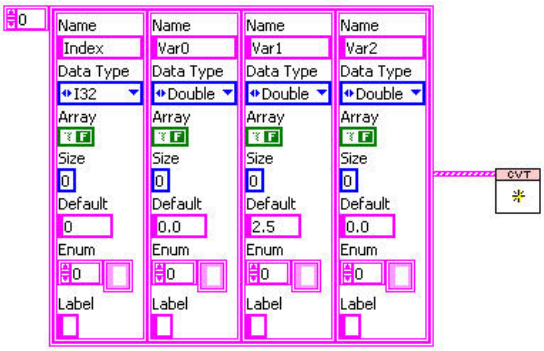


Figure 5: Initializing the CVT using a constant cluster array

Load From File

Rather than specifying the TagList as a constant or control, it is generally desirable to load the configuration from a file, such as a tag configuration file created using the Tag Configuration Editor (TCE). Using a file allows you to modify the available tags and the properties of tags without having to modify the code of your program. An editor such as the TCE also reduces the effort required create and configure large tag lists. The CVT API includes the CVT Load Tag List VI to load a TCE generated tag configuration file. Note that one tag configuration file may store tags for multiple targets, so this VI requires the current target name (as configured in the TCE) as an input.

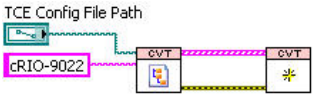


Figure 6: Loading the CVT TagList from a file

Basic API

The basic API contains functions to read and write data items in the CVT using the variable name as an identifier. Using the basic API the variable name can be a static string or can be created at runtime using the LabVIEW string functions. In the API call, the variable name is used to lookup the index of the variable and then the value is accessed in the MemBlock using the index.

Figure 7: Example accessing different CVT variables using the basic API

All of the basic API functions, including the initialization VI, are contained in the main CVT function palette which is installed in the User Libraries function palette in LabVIEW.

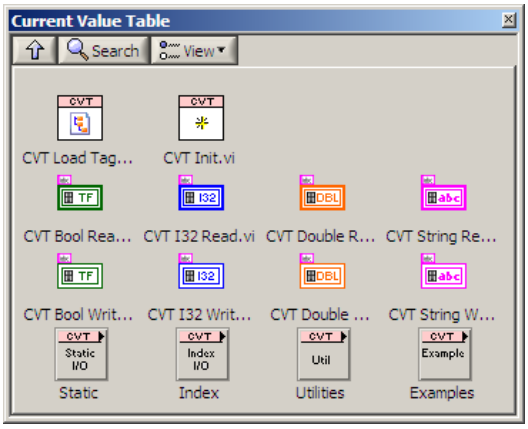


Figure 8: CVT Basic API Function Palette

Static API

The Static API (CVT_xxx_StaticWrite.vi, CVT_xxx_StaticRead.vi) is very similar to the basic API. However it requires that you use a static name (constant string) in your application when accessing variable. This restriction allows each instance of the Static API to only perform the variable index lookup operation once on the first call to each instance. It then buffers the index for subsequent accesses providing a significant performance advantage when repeatedly reading or writing the same variable in the CVT.

The Static API functions are available in a subpalette of the main CVT function palette.

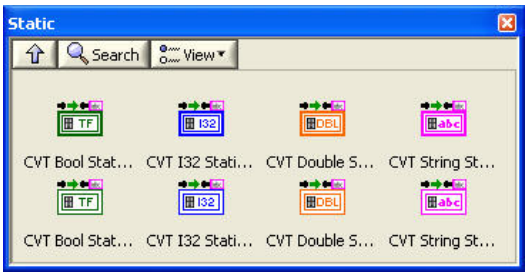


Figure 9: CVT Static API Function Palette

Index API

The Index API is an extension of the Static API and provides even better performance. Using the Index API you use two VIs to access a value in the CVT. The first VI is used to retrieve the index of variable (CVT_xxx_GetIndex.vi) in the CVT. This step is only performed once for each variable used. Then the index is used to access the variable value (CVT_xxx_IndexWrite.vi, CVT_xxx_IndexRead.vi). This provides the thinnest possible API to repeatedly access the same value in the CVT.

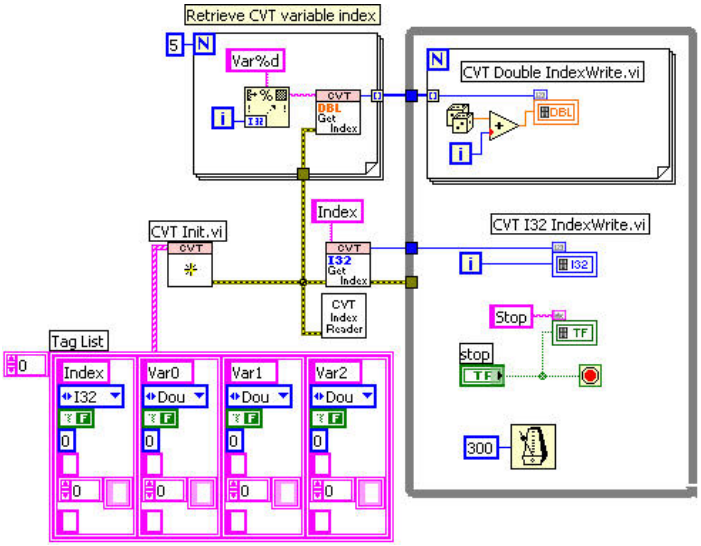


Figure 10: Example using the CVT Index API

The Index API functions are available in a subpalette of the main CVT function palette.

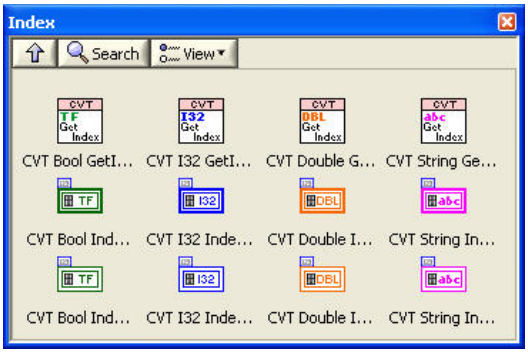


Figure 11: CVT Index API Function Palette

Utilities Functions

In addition to the VIs that access the stored variable values, the CVT also provides utility VIs to access additional properties or attributes of each variable. The current Utilities API provides the ability to retrieve the variable description. The description is a string attribute associated with each variable that may be used to store arbitrary information that relates to the variable.

The Utilities API functions are available in a subpalette of the main CVT function palette.

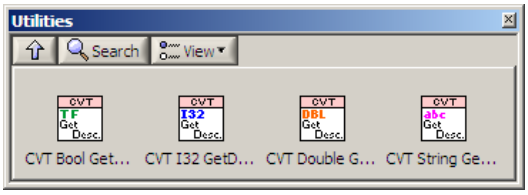


Figure 12: CVT Utilities API Function Palette

These properties are stored in the TagList core VI for each data type. In order to add additional variable properties to the CVT, a developer must add the new property to the CVT_Tag control TypeDef and then add a new case to the TagList core VIs to provide a method to retrieve the new property from the TagList. API functions can be added following the template of CVT_xxx_GetDescription.vi.

4. Example

Using the CVT is very simple in most applications. The first step is to define the variables that will be used by the rest of the application and shared through the CVT. The variables to be used need t be defined in the tag configuration array, either as a constant on the diagram or using another method.

In the code of your application the CVT needs to be initialized once by passing the tag configuration array to the CVT_Init VI. Once the CVT is initialized you can write and read any of the variables i the CVT using the three previously described APIs.

If you try to access a variable not defined in the CVT, the access function returns an error using a standard LabVIEW error cluster.

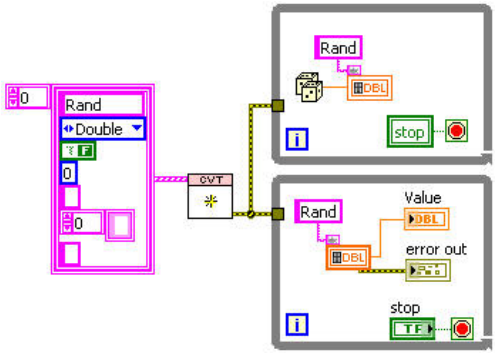


Figure 13: Example using the CVT functions to share data between two loops

5. Where to Go From Here

The following is a map of other documents that describe the machine control reference architecture. You can click on the image to navigate directly to each document.

[Your user agent does not support frames or is currently configured not to display frames. However, you may visit the related document.]

6. Feedback

This reference application was created by the NI Systems Engineering group.

We **do not** regularly monitor Reader Comments posted on this page.

Please submit your feedback in the Current Value Table (CVT) discussion forum so that we can improve this component for future applications.

Please direct support questions to NI Technical Support.