From Variant Attributes to Sets and Maps
(New in 2019)

Christian Altenbach, PhD
Research Ophthalmologist
Stein Eye, UCLA
LabVIEW Champion, Knight of NI, CLD
LabVIEW 2019 introduces two new data types: sets and maps. You can use these to achieve simpler, more intuitive solutions to common programming tasks and replace existing code based on variant attributes. Learn how to adapt older code and explore innovative examples where these new data types really shine.
LabVIEW 2019 introduces two new, never seen datatypes (Set & Map)

- What is a Set? What is a Map?
- What we did before sets and maps (variant attributes, etc.)
- What’s in the new palettes?
- Why are maps and sets so brilliant?
- How easy is it to adapt legacy code to the new tools? (hint: It’s very easy!)
- Examples, Examples!
The New **Collection Palettes**

<table>
<thead>
<tr>
<th>Functions</th>
<th>Programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structures</td>
<td>Array</td>
</tr>
<tr>
<td>Boolean</td>
<td>Cluster, Class, &amp; Variant</td>
</tr>
<tr>
<td>Numeric</td>
<td>String</td>
</tr>
<tr>
<td>Comparison</td>
<td>Waveform</td>
</tr>
<tr>
<td>File I/O</td>
<td>Dialog &amp; User Interface</td>
</tr>
</tbody>
</table>

**Map**
- Build Map
- Insert Into Map
- Remove From Map
- Look In Map
- Read Map Max & Min Keys
- Map Constant
- Collection Size
- Empty Collection?
- Convert Map To Array
- In Place Map Access

**Set**
- Build Set
- Insert Into Set
- Remove From Set
- Element of Set?
- Read Set Max & Min
- Set Constant
- Collection Size
- Empty Collection?
- Convert Set To Array
- Set Union.vim
- Intersection.vim
- Set Cartesian Product.vim
- Set Difference.vim
- Set Symmetric Difference.vim
Organizing information for efficient access is hard

- Unsorted phone book is useless
- Reprinting with each data change is expensive

- Elevator panel at old UCLA hospital poorly designed.
  - Numbers not sorted vertically (*Key/map: We know the floor >> which button?*)
  - Number labels equidistant between two buttons requires double-take.
Key concepts

- Having information is useless unless we can find it efficiently.

- Sets and Maps are optimized for storing information and answering questions:
  - Does this entry exist? (sets: element, maps: key)
  - If it exists, what is its value (maps: value)

- Sets and maps are optimized for dynamic changes in their data
  - Add entry (maps, sets)
  - Remove entry (maps, sets)
  - Replace value (IPE (3 modes), maps)
Variant attributes

- A collection of “Attribute name (= key) <> Value” pairs. “Name” is a string, value is a variant. *(Note that waveform attributes are basically the same. There are some names used by NI drivers to relay metadata).*
- If we don’t care about the variant or waveform, we can use either interchangeably. We can even mix&match (within limits) and things continue to work just fine:

This motley code works!
Variant attributes

Variant irrelevant, can be empty!
Compare Maps and variant attributes

Similarities
- Key -> Value pairs (Key must be unique!)
- High performance lookup, insert, delete
- Hierarchical possibilities (Value can be map, set, or variant, etc.)

Differences
Maps
- Key can be of any datatype
- No reserved values
- All values must share datatype
- First class datatype

Variant attributes
- Key (=name) must be string
- Key cannot be empty string
- Can mix value datatypes
- Tacked onto (often irrelevant) variants
Set

- Colloquial use of the word “set” is poorly defined

- Use in LabVIEW is well defined:
  - It is an unordered collection of elements, all sharing the same datatype and in which duplicate elements cannot exist. The items need to be sortable in some way.
  - Datatype can be almost anything.
  - Seems restrictive, but these limitations allows for efficient operations!
Map

- Colloquial use is poorly defined

- Use in LabVIEW is well defined:
  - It is an unordered collection of keys sharing the same datatype and in which duplicate keys cannot exist. Each key has an associated value.
  - Duplicate values are allowed.
  - A map is basically a set where each element contains extra data.
  - Value can be of almost any datatype, accessed by knowing the Key.
Maps (Other terms for the same thing!)

- The functionality as implemented in LabVIEW is known under many names, all deal with Key <> value pairs.

- Map
- Associative array
- Symbol table
- Dictionary
- ...

- A classic computer science problem is to design data structures to efficiently do operations (lookup, insert, delete, etc.)
Good arrangement of data can facilitate operations

- Determining if an entry (map key, set element) exists can be improved by storing entries in sorted order. Assume a set with 1024 entries and we must prove that a certain element does not exist:
  - If we have an unsorted array, we need to inspect all 1024 elements
  - If we know that the array is sorted, we need to inspect 10 elements, each inspection eliminates half of the remaining entries. \( \log_2 N \) vs \( N \)

- Problem:
  - If entries are added or removed, re-sorting is needed.
  - The solution is **efficient data structures** where rebalancing is cheap.
One possibility: Balanced Binary trees

- Computer scientists have devised data structures that exhibit efficient operations. There are many flavors and research is ongoing.
- Examples: red-black tree, AVL tree, 2-3-4 tree, etc. etc.
- Story: Robert Sedgewick (Princeton) first published the red-black tree structure in 1978 and it took him 30 years and contributions from others to improved it (left leaning red-black tree, improvement is fewer lines of code!). There is critique that optimizing for code size might not optimize for speed (Eddie Kohler (Harvard): “Left-Leaning Red-Black Trees Considered Harmful”)

- Fortunately we can ignore all that!
- LabVIEW Sets and Maps use standard public libraries under the hood.
- Well tested and optimized!
Sets and Maps on the diagram
Defining Sets and Maps

Place Set/Map constant and drop desired datatype

**Good idea to name items**
All Set functions

Explore the palettes!
Advanced Set functions

**Set Union.vim**

Example:
```
set 1
set 2
```

**Union**
Computes the union of two sets. The union is all elements that belong to either of the two sets.

**Set Intersection.vim**

Example:
```
set 1
set 2
```

**Intersection**
Computes the intersection of two sets. The intersection is all the elements that belong to both sets.

**Set Cartesian Product.vim**

Example:
```
set 1
set 2
```

**Cartesian product**
Computes the Cartesian product of two sets. The Cartesian product is a set of 2-element clusters which covers all possible combinations of elements from the two sets.

**Set Difference.vim**

Example:
```
set 1
set 2
```

**Difference**
Computes the difference of two sets. The difference is the elements from the first set that are not included in the second set.

**Set Symmetric Difference.vim**

Example:
```
set 1
set 2
```

**Symmetric difference**
Computes the symmetric difference of two sets.
All Map functions

Registration map (Map where value is a set!) Look at shipping example …
Get all data into classic structures

Indexing on a FOR loop: Ingenious!

- Set: autoindexing yields all elements **sorted**
- Map: autoindexing yields cluster of key/value pairs, **sorted by key**.

Unbundle by name requires Items to be named. Highly recommended for Self-documenting code!
“In Place Element” structure for Map operations

Map Get / Replace Value

Use this border node to access and modify a value in a map without copying the value out of the map. Use the border node on the left side of the structure to retrieve a value in a map. Use the border node on the right side of the structure to modify or discard the retrieved value.

3 action modes (enum)
Some actions are not really “in place”, but compiler knows what to do!
Warning: default action (unwired!) will not add new entries!
New array function in array palette (implemented as vim)

- Remove duplicates while keeping elements in order of first occurrence

Uses a Set!
Operating on Sets and Maps (cont.)

Saving/loading:
- Just use binary file IO. There is no data overhead at all.
Set example

- Generate lotto numbers until we find one that is a duplicate
- We need to keep all entries and compare each new one with all old ones.
- Worst case lookup because entry is not found until we stop

- Four implementations:
  1. Linear search (search expensive, append fast)
  2. Binary search (search fast, smart insert to keep it sorted)
  3. Variant attributes (search fast, insert fast)
  4. Set (search fast, insert fast)

All four implementations are similar in code. Compare the differences!
The four implementations

Flatten each new entry and **append** to the end of a string array
Search if entry exists

Do a binary search for entry **Insert** entry at right place to keep array sorted
(subVI code complicated, see next slide!)

Use **variant** attributes with flattened entry as attribute name, Ignore value
Add new entries until an entry already exists

Use **Set** with 1D U8 array as datatype
Add new entries until an entry already exists
A single binary search answers two questions:
- Does the entry exists, and if so where
- If it does not exist, where to insert it to keep array sorted
Set Example
Lottery Summary

- Brainless “append” implementation is slow and scales steeply with the number of tries. Impossible for larger N.

- Simple home-made “insert” implementation using binary search and “smart insert” pretty competitive, somewhat nonlinear. Steeper at large N. Lots of work to adapt to different datatypes. Not practical, even as vim.

- Variant attribute and set scale well and are quite similar!
- Set outperforms variant attributes over the whole range but they are within a few % of each other. (Exact differences slightly processor dependent).

- “Set” implementation has cleanest code!

Winner: Set!
Recap of 2018 talk

- At NI Week 2018, I talked about the usefulness of Variant Attributes and showed some typical applications.

- Let’s revisit these and see …

- What needs to be done to change from variant attributes to maps
- Compare code side-by-side
Map Examples
Static table

- (existing old forum example, using variant attributes)
1. Replace variant with map
2. Change map type to string/string
3. Replace write function
data
4. Replace read function
5. Voila!

Rewritten to use a set
example update demo
If data is really (really!) static, convert to map constant!
Count Duplicates

- Count duplicate words in a space delimited string and sort by frequency
- (Old variant attribute based forum example)
Variant attribute based

Rewritten to use a map
Layers

- Variants with attributes inside variant attributes (forum example)
  (find all unique entries in column #1
  …and for each, all unique values from columns #2)
Variants with attributes inside variant attributes …

… replace with Map where value is a Set!

Convert set to array!
Fixed size cache (presented in 2012 @ NI Week)

- Parallel computation of all needed spectra
- Reassemble result from all cached parts
- (See also)
Application example: MultiComponent

- Parallelization and caching allows linear scaling with the number of CPU cores
- Runs well on typical hardware, but even better on high end machines (See also)

AMD is competitive again!
Preliminary benchmarks

Upgrade of cache code from variant attributes to Map possible within minutes

Similar performance.
(1K size result shown)
Upgrading Variant Attribute Code to Map code
Should we upgrade?

- If the Variant Attribute based code is working well, there is no real need to upgrade to Sets/Maps.
- If the old code is clumsy (complicated datatypes flattened to strings, exception handling to deal with empty string keys, etc.) upgrade might be worth it.
- Upgrading is typically very quick and inexpensive. Not a huge barrier!
- (Still, all modified code needs to be tested and verified for correct operation!)
- Use Maps and Sets for all (or most) new code.
- Don’t upgrade if code potentially needs to be converted to LabVIEW 2018 or lower. Maps and Sets do not exist there.
- If not all variant attributes have the same datatype, make map value a variant!
Summary: Upgrading Variant attribute based code

- Replace Variant with set or map
  - Variant attributes where *we don’t care about the value*: Replace with Set
  - Variant attributes where *we care about the value*: Replace with Map
- If key (i.e. attribute name) is cast to a string, consider using primary datatype of key directly
- Replace functions with equivalents (nearly pin compatible!)
- Don’t forget to invert the boolean output (look in map vs. Get variant attribute)
  - Variant: TRUE if found
  - Map/Set: TRUE if not found (More consistent with typical error handling!)

![Diagram](image-url)
Guidelines and Caveats

- In order to find a certain entry, the keys need to be **binary identical**.
- (except for NaN, see below).
- We don’t get a “closest match”, “most similar” or any kind of interpolation.
- Be careful with floating point data as **keys** (SGL, DBL, CDB, etc.), i.e. any datatype where equal comparisons are dangerous.
- NaN is allowed and is used normalized and equal to itself.
- There are no built-in limits on hierarchical structures, but our brain has such limits. Keep it simple (A map of maps of maps of sets is NOT simple).
- Keep the data on the diagram (Map and Set controls/indicators mostly useful as subVI connectors).
- Encapsulate Set/Map in single subVI, don’t shovel it around. (action engine!)
Guidelines and Caveats

- Could keep Set/Map in DVR if needed in many places. (not really recommended)
- Reverse lookup not possible because values don’t need to be unique
- Could implement two maps with key/value reversed and update both, checking for duplicates when adding entries, etc.
Summary

- Sets and Maps are a highly welcome addition to LabVIEW.
- They help solve many common programming tasks with:
  - Simpler code!
  - Easier to understand code!
  - Faster code!

- This presentation only touched the surface of possible applications.
- I encourage anyone to dive in and upgrade to LabVIEW 2019.

- There are plenty of other new features in LabVIEW 2019, but Sets and Maps alone make upgrading worthwhile!
Before you go, take the survey.
Stay Connected During and After NIWeek

ni.com/niweekcommunity

facebook.com/NationalInstruments

twitter.com/niglobal

youtube.com/nationalinstruments