LA-UR-16-24272
Approved for public release; distribution is unlimited.

Title: Echo™ User Manual

Author(s): Harvey, Dustin Yewell

Intended for: Software documentation

Issued: 2016-06-17

Echo™ User Manual

Version 0.2880

June 6, 2016

Contents

1 What is Echo?

Echo™ is a MATLAB-based software package designed for robust and scalable analysis of complex data workflows. An alternative to tedious, error-prone conventional processes, Echo is based on three transformative principles for data analysis: self-describing data, name-based indexing, and dynamic resource allocation. The software takes an object-oriented approach to data analysis, intimately connecting measurement data with associated metadata. Echo operations in an analysis workflow automatically track and merge metadata and computation parameters to provide a complete history of the process used to generate final results, while automated figure and report generation tools eliminate the potential to mislabel those results. History reporting and visualization methods provide straightforward auditability of analysis processes. Furthermore, name-based indexing on metadata greatly improves code readability for analyst collaboration and reduces opportunities for errors to occur. Echo efficiently manages large data sets using a framework that seamlessly allocates resources such that only the necessary computations to produce a given result are executed. Echo provides a versatile and extensible framework, allowing advanced users to add their own tools and data classes tailored to their own specific needs. Applying these transformative principles and powerful features, Echo greatly improves analyst efficiency and quality of results in many application areas.

End users may not distribute this software. The most recent release version is available at http://int.lanl.gov/echo for internal use. External users may submit a request to EchoSupport@lanl.gov to obtain the software.

Contact EchoSupport@lanl.gov to request software, report issues, provide feedback, or inquire about training.

2 Requirements and Installation

Echo requires MATLAB R2015b or later. Some operations may depend on additional toolboxes.

To install Echo, simply download the latest version as a zip file and uncompress the contents in your desired location. To use Echo with a MATLAB instance, run `startEcho.m` from the root Echo folder. To permanently install Echo in MATLAB, run `startEcho.m` from the root Echo folder then run `savepath`.

3 Getting Started

Echo utilizes an object-oriented approach to data analysis. In addition to basic MATLAB knowledge, users will benefit from an understanding of object-oriented programming including the concepts of classes, meth-

ods, handles, and inheritance. The following sections provide a starting point for new Echo users to become acquainted with the software.

3.1 The Echo record

The core of the Echo data analysis approach is the record. All activities in Echo are focused on records. Each piece of data to be used in an analysis is encapsulated in a record. Records are Echo objects of an appropriate class for the type of data. For example, scalar data uses the class ScalarRecord and a time history uses the class TimeRecord. Echo provides many record classes for different types of data, and users can add their own classes to support their specific needs.

In addition to the data itself, a record is self-describing including many other pieces of information about the data, i.e. metadata, as well as information about the record itself. Record metadata includes units, time stamps, coordinate system information, and textual metadata called labels. Additionally, records track various information about their creation and preferences for how the data should be displayed in a figure. The encapsulation of all knowledge about a piece of data within a single object provides many benefits to analysts including:

- Ease of performing analysis
- Scalability of analysis tasks
- Improved code readability
- Fast, error-free reporting of results
- Effective collaboration with other analysts

Record classes are part of the Echo inheritance tree shown below. Each class inherits all of the methods and properties from its superclasses, or parents. For example, TimeRecord includes all of the methods and properties of OneD, ArrayData, Data, Record, Plottable, Labeled, CustomArray, and handle.

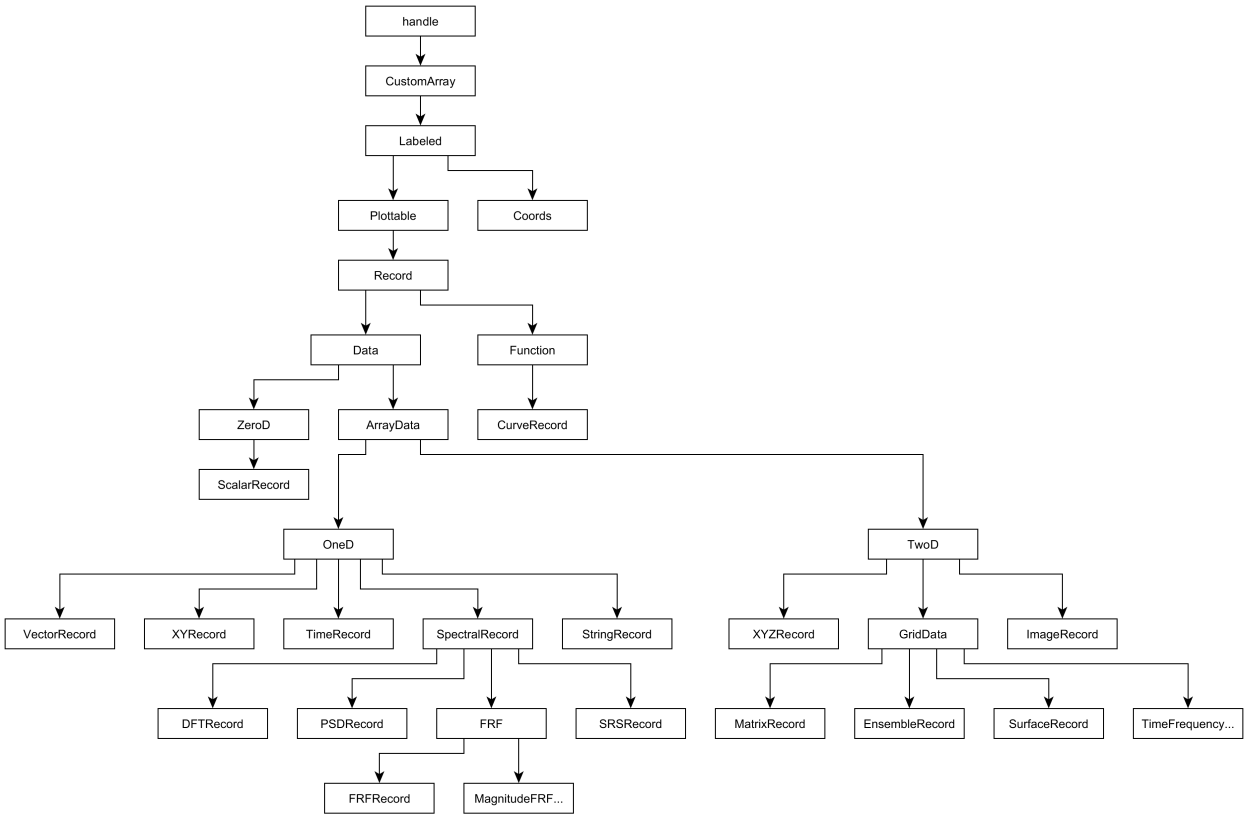

Figure 1: Echo Inheritance Tree

3.2 Performing an analysis

Each record class provides various analysis and reporting methods appropriate for the type of data it is intended to manage. For example, the TimeRecord class has methods for filtering time histories and estimating various spectral quantities. Each analysis step in Echo is called an operation. Operations act on source records and parameters to produce a result record.

Let's create a record and perform an operation. Records are created by calling the constructor of the appropriate record class. The code below constructs a time history record with some random data as the variable `A`.

```
A = TimeRecord(1:10,rand(10,1));
```

`A` is now a single TimeRecord with a time vector, `t`, of the integers 1 through 10 and data of 10 random values. We can assign metadata such as units through an operation. The `assignUnits` operation allows for units to be assigned to each set of values in the record, called *datums*.

```
B = A.assignUnits('t','day','data','ft');
```

'B' is now the result of using `A` as a source and assigning units of days to the `t` datum and feet (ft) to the `data` datum. All Echo operations follow a similar process of calling an operation method on the source record or records along with additional argument parameters. To visualize the record `B`, simply enter

```
B.plot();
```

The resulting figure is shown below. The figure is automatically populated with appropriate labels on the axes defined by the TimeRecord combined with the units assigned to the data through the `assignUnits` operation. The plot can be further customized by storing preferred plot options within a record using the `setPlotOptions` method, or by passing plot options directly to the `plot` command.

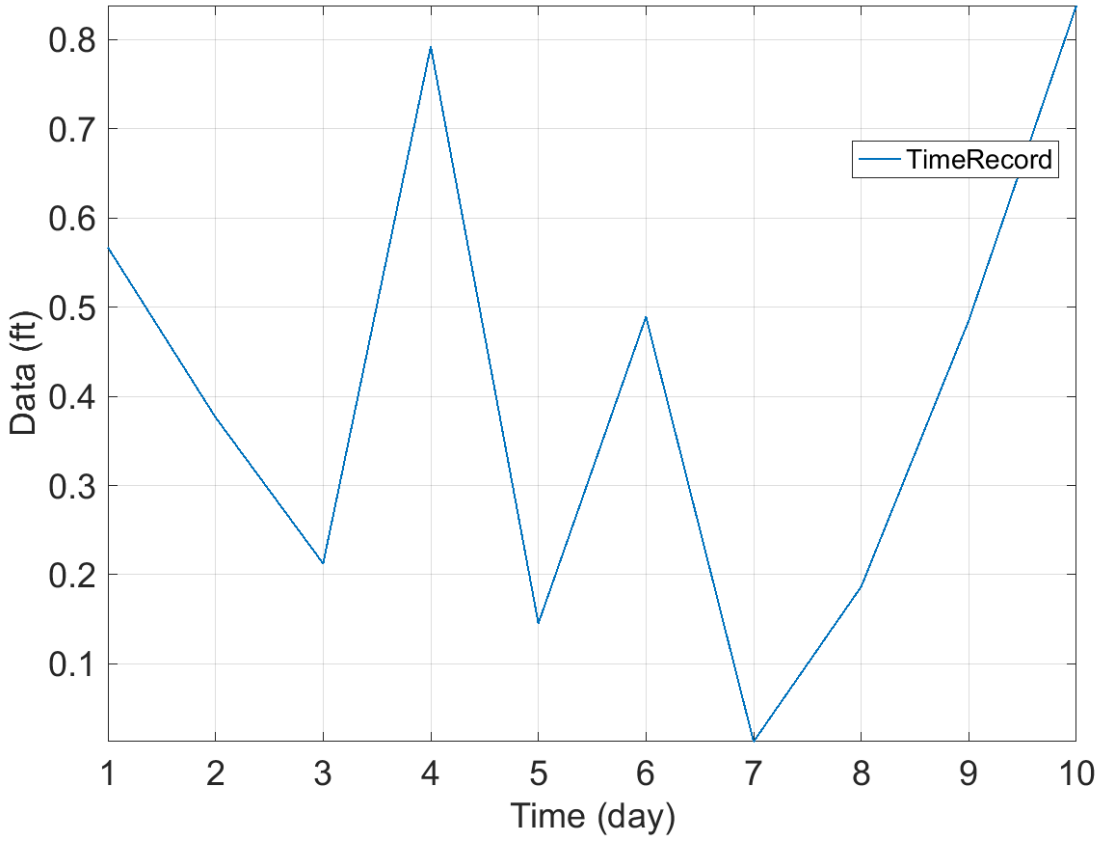

Figure 2: Plot of TimeRecord, B

3.3 History

Records have a powerful capability to track their creation process. For example, record `B` created in the previous section automatically stores the name, parameters, and source record `A` of the operation used to produce it. The `display` method for record `B` shows:

```
B =
1 TimeRecord object with properties:

           coords: struct
          hasData: 1
      hasOperator: 1
     hasSavedData: 0
          isFixed: 0
        keepsData: 0
           labels: struct
   operationNotes: []
      plotOptions: struct
 recordAttributes: struct
         recordID: v863b54f30e24904...
       sourceFile: []
        timeStamp: []
avgSampleInterval: 1
             data: [10x1 double] (Data, data[t])
       data_units: ft
        isUniform: 1
       sampleRate: 1
                t: [10x1 double] (Time)
          t_units: day

Computed from assignUnits with parameters:

sources: [1 TimeRecord]
 Datum1: t
 Units1: day
 Datum2: data
 Units2: ft
```

The top section of the display shows various properties of record `B` that store the data, metadata, and record information. The bottom section contains the name of the operation used and the parameters entered including `sources` which is the original source record `A`. Because each record stores and provides access to the source records used to create it, the structure of operations and sources can be recursively traversed to explore the entire pedigree or history of a record. The inclusion of history information in records is a powerful feature for review of past analyses and collaboration with other analysts.

The entire history of a record can be quickly visualized with the `echo` method. The history of `B` is shown below.

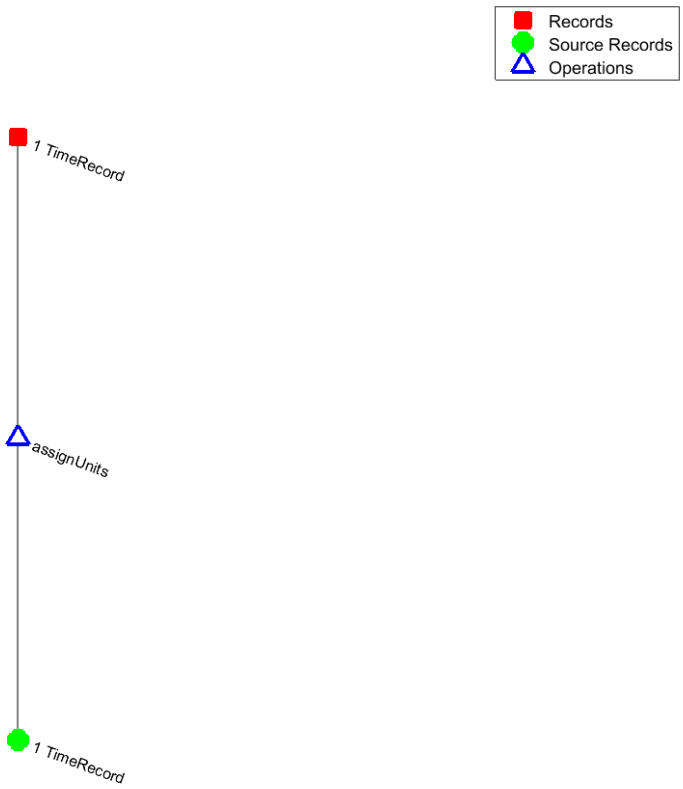

Figure 3: History of TimeRecord, B

3.4 Working with multiple records

Records of the same type can be stored in arrays to be analyzed together. The following code produces an example array of 14 TimeRecords, R.

```
R = importCountryData;
```

Textual metadata called labels are used in Echo to identify records and to sort and index record arrays. Labels are stored as name/value pairs where the label name identifies what piece of information is represented by the corresponding value. Labels are added to records using the `label` method. The existing label values for an array can be quickly viewed by producing a label report as below. This code produces a label report for 2 of the labels in the array.

```
R.labelReport('labelNames',{'Country','DataSource'});
```

The resulting report, shown below, includes a column for each of the labels selected in the array and a row for each record in the array.

```
       Country      DataSource
-------------    ------------
        Canada      Population
        Canada    Unemployment
        France      Population
        France    Unemployment
         Japan      Population
         Japan    Unemployment
        Norway      Population
        Norway    Unemployment
   South Korea      Population
   South Korea    Unemployment
United Kingdom      Population
United Kingdom    Unemployment
 United States      Population
 United States    Unemployment
```

To index a subset of a record array, the `pull` method is called with the desired label values. For example, to create an array `Rsubset` containing the 7 records identified with a DataSource of *Unemployment*, the command is:

```
Rsubset = R.pull('DataSource','Unemployment');
```

Labels can also be used to control plots. The plot option `legendLabels` specifies a set of label names to use to populate legend entries. To plot the unemployment data with a legend showing the name of the country, we can use

```
Rsubset.plot('legendLabels','Country');
```

to produce the figure below.

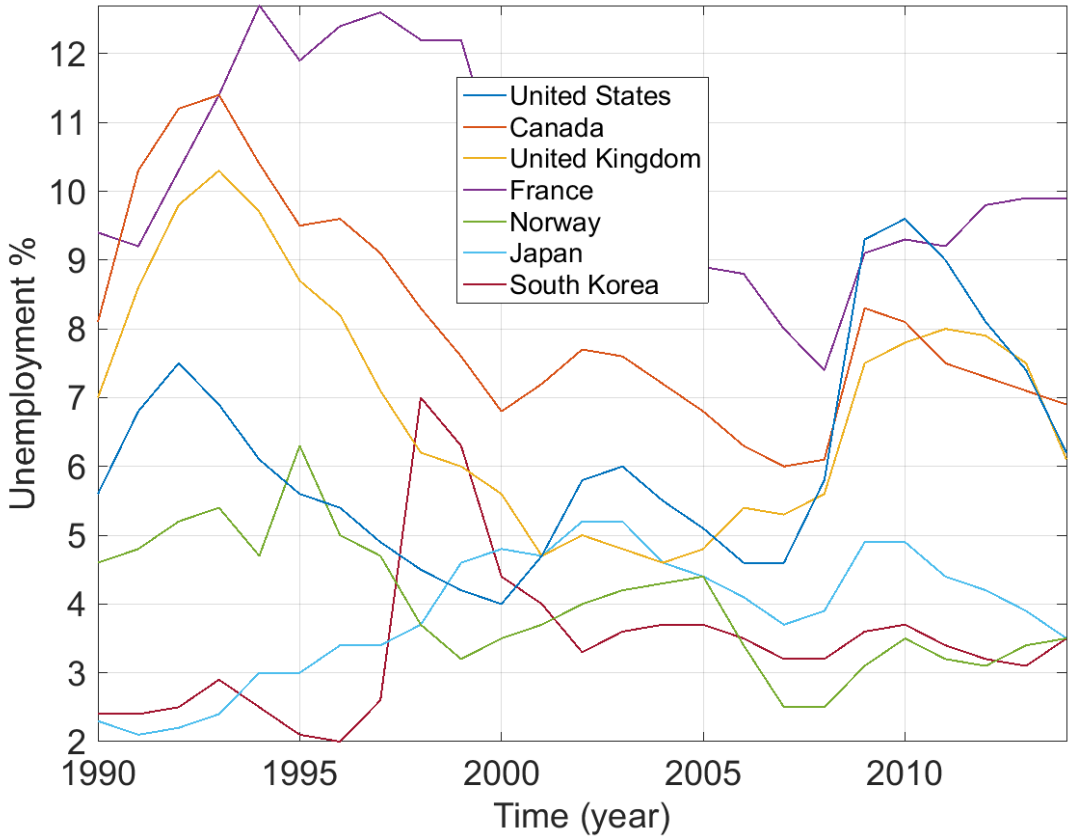

Figure 4: Plot of record array, R

3.5 Saving and loading records

Echo record arrays are saved as HDF5 files using the `save` method. Each record is written to a separate file containing all of the data, metadata, history, source records, and other associated record information. The Echo file format is readable by any program or application with HDF5 file support.

Additionally, a file called ***_LoadRecordArray.mat*** is saved in the same folder as the record files. This file is used by Echo to keep track of all of the records in the saved array. The entire array can be loaded in MATLAB using the `load` function on the folder path or the ***_LoadRecordArray.mat*** file. When the array is loaded, warnings are provided if any record files are missing from the array or if new record files have been added to the directory. This check maintains integrity of record arrays in file copy and transfer operations.

4 Help and Documentation

To find help within MATLAB, enter `help echo` in the command window to open Echo help menu. Follow the provided links to access help topics. Individual echo classes, methods, and functions contain documentation which can be accessed through the usual `help` and `doc` functions in MATLAB.

Tutorials and exercises can be found in the Documentation/Tutorials folder starting with Record Tutorial.mlx. Additional Echo documentation currently in progress will include the following:

- Complete, searchable reference of all method and function documentation
- Developer guide for adding new operations and record classes

5 User Extensions and Upgrades

Echo is designed to be highly extensible allowing users to customize existing features and add to the framework to support their own needs. Users can add new operations to existing record type classes and develop entirely new record classes to better support specific types of data and analyses.

When new Echo versions become available, the upgrade process is simply to overwrite existing Echo files with the new versions. Therefore, any customization by a user could be lost if not managed properly. The Echo team strongly recommends maintaining the Echo files within a version control system such as GIT to help manage updates and avoid loss of any user customization. To make the update process easier, it is recommended that edits to original Echo files are minimized. For example, when adding methods to existing classes, it is preferable to write the new method in a separate m-file within the class folder then only edit the class definition m-file with the new method declaration.

6 Frequently Asked Questions

(1) When upgrading to a newer version of Echo, will I lose my modifications or additions?

See section on user extensions and upgrades.

(2) Does Echo use parallel computing?

Yes, Echo is designed to seamlessly make use of the MATLAB Parallel Computing Toolbox, if available, for tasks such as saving and loading records and performing computations. If the Parallel Computing Toolbox is not installed or a license is not available, all tasks are performed in serial.