Scientific File Repository - Toolbox

The SFR Toolbox for Matlab.

The Scientific File Repository toolbox provides an intuitive method for accessing continuous multichannel scientific data from various file-formats. It does this by embedding the file-type and the locations of the raw data files into an object and transparently calling the correct getter method to return the data, and associated meta-data.

October 5, 2012

J.B. Wagenaar

Introduction

Scientific data is collected in a wide variety of data formats. Some data is stored using flat binary files, whereas others are stored in complex structures with embedded meta-information which need dedicated read methods. This toolbox targets all file-formats that store continuous multichannel data with or without associated meta-information.

To access data and meta-data from various different scientific file-formats, a user typically needs to use vastly different syntax per file-format and provide specific methods for each file-type. This often leads to incorrect handling of the native data-format, and inefficient data processing,

This toolbox addresses these issues by representing a dataset (consisting of 1 or more files) by a single object. The user specifies which files belong to the dataset and which data-format is used in the files. The user can then access the raw data and the meta-data using the properties 'data' and 'attr' of the created object. The toolbox automatically calls the correct 'getMethods' for the associated file-format without any additional input from the user.

Support for various file-formats is included with the toolbox and is easily extendable by the user. In fact, only two files are required to add support to a new format. An INFOMethod should return meta-information about the data in the form of a Matlab structure and provide information during object construction. A GETMethod should return the data in a 2D array [indices channels] or matlab structure, using the information (i.e. filenames) in the objects.

Additional features allow the user to improve efficiency by caching information from the GETMethod, allowing the user to add meta-information directly to the object, and provide an optional CLEANmethod that is called whenever the object is deleted.

Finally, because a dataset is now represented as an object, it is very easy to write standardized methods that act on the data. This can vastly increase the reusability of code and promotes a structured methodology for data-analysis.

Installation

Installing the SFRepos toolbox is as simple as downloading the project and adding it to the matlab path. In order to do this, follow the following steps:

1) Go to the project webpage and download the project.
   a. Download the project as a zip-file: This will give you the latest version but you will have to re-download newer versions from the project page.
   b. Clone the repository: This will make a copy of the latest version on your local drive and you will be able to automatically update in case newer versions are available. This requires you to install Github on your local machine (which is very easy).
   c. Fork the repository: This will do the same thing as b) but you also have the option to make changes to the code and upload them to the project (per Pull-request). This also requires you to install Github.
2) Add the following folders to you matlab path using the 'setpath' method in the 'file'-menu of matlab or using the ADDPATH method in Matlab. Remember to save the path and that the ADDPATH method does not remember the added path when you exit Matlab:
   a. The project folder, which is the folder that contains this document and the README file.
   b. The 'lib' folder that is located in the project folder.
   c. (optional) The 'tools' folder which contains methods that support the toolbox.

This is all you need to do to install the toolbox. You can test whether it is correctly installed using the WHICH method. This should return the location of the 'SFRepos.m' file (see below).

Type:

```
>> which SFRepos
.../Documents/GIT/SFR-Toolbox/@SFRepos/SFRepos.m  % SFRepos constructor
>>
```

The Location Library File

The Location Library File is an XML file that needs to be specified for each individual user of the SFR-Toolbox. It describes the absolute paths to the data repositories for the particular user by means of a set of XML-tags. Before the SFRepos toolbox can be used, each user should copy the 'sample_ReposFile.xml' file from the Toolbox, save it to a different location and edit the contents to reflect the correct settings. This process is very easy and explained below.

Contents of the 'locationLibrary.xml' file:

```
<?xml version="1.0" encoding="utf-8"?>
<SFR_REPOSFILE user="userName">
   <LOC id="Work">
     <REPOS id="someProject" path="/Users/Projects/Someproject"/>
     <REPOS id="HumanData" path="/Users/Data"/>
   </LOC>
   <LOC id="Home">
     <REPOS id="someProject" path="G:/tempData/Someproject"/>
     <REPOS id="HumanData" path="G:/tempData/Users/Data"/>
   </LOC>
</SFR_REPOSFILE>
```

In this example, we defined two locations where we access the data (Work, and Home). Depending on the location, the absolute path to the data differs (i.e. you access the data from a NAS at work and from an external disk at home). Using the SFRSETLOCATION method, you can use the same SFRepos objects even if the absolute path to the data differs between the two scenarios.

The XML file can be edited in the Matlab editor or any other text-editor. It contains three different tags which are explained below:

Tag	*Description*
SFR_REPOSFILE	This is the main tag which contains all other tags. It has one attribute: "user" which can be used to identify the user for which these settings are generated.
LOC	The LOC tag is used to identify the location where the computer is used. It is possible that the path to the location of the raw data files depends on the location of the computes. For example, whether the data is accessed from home or from work.

Tag	*Description*
REPOS	This tag identifies where a repository with a particular identity is located. The attribute “id” is a unique identifier that corresponds with the absolute path specified in the “path” attribute (the root-location). The path should contain the absolute path to the repository identified by “id” for the current location. Note that this path does not necessarily need to point to the specific location of the files themselves but to the root of a repository. The path to the files relative to the root-location can be specified in the object itself. It is safest to use forward slashes (‘/’) between folders as Matlab interprets them correctly in both windows and mac.

Creating SFR Objects

SFRepos files are created by calling the SFRepos constructor with a specific set of arguments. These arguments can not be changed after the object has been created as the validity of the arguments is checked during the object creation. In this section, we will briefly explain the various inputs to the constructor method. See the tutorial for an example on how to create a valid SFRepos object. The constructor is specified as follows:

OBJECT = SFRepos(TYPE, ROOTID, SUBPATH, FILES, TYPEATTR, OBJATTR);

where:

Variable	*Description*
TYPE (String)	The TYPE input specifies the file-format of the files that are linked in this object. This string forms the basis for the name of the getMethod and infoMethod. For example, if the TYPE input is specified as "SimpleBinary", the method that returns data of this type should be named "getSimpleBinary" and the info-method should be called "infoSimpleBinary". For more information about the specific methods, see the next section.
ROOTID (String)	The ROOTID input identifies the repository location of the data linked in the object. This string should match one of the "id" attributes of the REPOS tag in the XML file. The object will look for the path of the repository location that matches the specified ROOTID. This means, you need to update the XML file prior to constructing the objects.
SUBPATH (String)	The SUBPATH inputs specifies the path to the folder where the files are located relative to the location of the path associated with the ROOTID. This allows the user to specify a single repository root location that is shared by multiple datasets. For example, you can set the ROOTID to a folder called 'HumanData' and use the SUBPATH to specify the location of a specific data set (i.e. 'Patient1/Session1').
FILES (Cell-array)	The FILES input is used to specify all the filenames that are associated with the dataset in the object. In general, there are three variations that are allowed: 1) 1D vector of file names where each file contains the data for a single channel, 2) 1D vector of file names where subsequent names contain subsequent blocks of data on all channels, and 3) 2D array of filenames where each row contains filenames associated with data for a single channel and the columns are associated with a different block of data for that single channel.

Variable	*Description*
TYPEATTR *(optional, 1D Cell-Array)*	The infoMethod determines whether the constructor method needs extra information in the form of the TYPEATTR input (see below). Type attributes contain information that is required for the getMethod to load the data. For example, the "BinByChannel" data type needs to know the precision of the data (i.e. 'double' 'single' or 'uint') in order to return the data. This information in not embedded in the datafiles and should therefore be provided by the user. The general syntax of the TYPEATTR is: TYPEATTR = {'attributeName' attributeValue 'attributeName' attributeValue ...};
OBJATTR *(optional, 1D Cell-Array)*	The OBJATTR input defines additional attributes that are NOT required by the getMethod. These attributes could include anything that is associated with the object and are completely optional. You can also add optional attributes after creating the object using the ADDATTR method. For example, channel names are not required by the getMethod but the user might like to associate them with the object. Optional attributes follow the same general syntax as the required attributes.

Using SFR Objects

SFR-Object act similar to a normal Matlab structure but have some additional features to streamline data access and increase the flexibility to support custom access methods. This section will outline the ways that you can interact with a SFRepos object from the Matlab console.

OBJECT REPRESENTATION

A typical SFRepos object will be displayed in the Matlab console as shown in the figure below (This object was generated using the tutorial).

```
SFRepos:

    typeId: BinaryByChannel
    rootId: 'SFRroot'
   subPath: 'SFRExample/data'
     files: {'Channel1.dat'  'Channel2.dat'  'Channel3.dat'}
  typeAttr: [1x1 struct]
      data: [100000x3 double]
      attr: [1x1 struct]

Methods, Location, getOptions
```

There are a couple of links that can be used to obtain more information about the object:

1) SFRepos : This link will show the help-section of the SFRepos class, which contains a summarized version of this manual.
2) BinaryByChannel : The value of the 'typeId' property is a hyperlink to the help-section of the 'info'-method for the file-format that is associated with the SFRepos object. In this case, the link will display the help-section of the 'infoBinaryByChannel.m' file in the @SFRepos folder.
3) Methods : This link will display the available methods for objects of class SFRepos.
4) Location : This will combine the information from the location-XML file and the subpath to indicate the absolute path to the files associated with the object.
5) getOptions : This link shows the attributes that can be used for the associated filetype. These attributes are defined in the 'info'-Method for the particular file-type, and can be used to modify the behavior of the 'data'-property (see below).

THE DATA PROPERTY

The “data” and “attr” properties of an SFRepos object are dynamically linked to information. This means that the data in these properties is loaded from the files when the properties are accessed rather than when the object is loaded. This allows the object to access very large datasets with minimal memory requirements and fast access-times.

The SFR-Toolbox supports two ways in which data can be returned through the data-property.

1) As an array which mimics the way that the data is represented in the object.
2) As a structure with a “data”-property that contains the array of data. This seems counter-intuitive but it provides some flexibility for users to implement custom get-data methods. For example, the returned structure can include information about the continuity of the returned dataset or other related information. Note that the object representation will always show the contents of the “data”-property as an array.

To force the SFR-Toolbox to return either a structure or an array, you can use the ‘asArray’ and ‘asStruct’ attributes (see below).

The default way to access data is the same as accessing the property with a set of indices and channels:

```
>> out = repos.data(1:1000, 1:5);
```

This will return the a 1000x5 array of values or a structure with a property “data” that contains the array of values. To force the return of an array, or struct, use:

```
>> outArray = repos.data(1:1000, 1:5, ‘asArray’);
>> outStruct = repos.data(1:1000, 1:5, ‘asStruct’);
```

If the default behavior of the getdata method is to return a structure, the ‘asArray’ attribute will only return the “data” property of this array. If the default behavior of the getdata method is to return an array, the “asStruct” attribute will create a struct with a single “data” attribute containing the data.

OPTIONAL DATA PROPERTY ATTRIBUTES

The previous section already showed the use of attributes for accessing data using through the "data" property. The mentioned attributes "asStruct" and "asArray" are available by default, but additional attributes can be made available for each implementation of a getdata method for a specific filetype. The available attributes are defined in the infoMethod (see next section).

The general form of adding attributes is:

```
>> out = repos.data(indices, channels, 'attrName', 'attrValue', ...);
```

If no attribute value is provided (or the second attribute input is another attribute name), the value of the attribute is set to TRUE. This means that the value of a custom attribute cannot be a string that matches another attribute name as this will be interpreted as two attributes with value TRUE.

For each filetype, the optional attributes for the "data"-property are defined in the "optionalAttr" variable. For example, the MEF-format has a number of optional attributes that change the way that the data is returned: "getByTime" allows the user to use the indeces as start and stop times of the requested data, and "padNan" will pad non-continuous data with Nan's.

Example syntax for object containing MEF-files:

```
>> out = repos.data(1:1000, 1:16, 'asArray', 'padNan');
```

NOTE: The Info-method for a particular file-type also specifies the "requiredAttr" variable (see below). This variable specifies the required attributes that need to be specified during object creation. Without these attributes, the Get-method cannot correctly return the data.

THE ATTR PROPERTY

The "attr" property of the SFRepos object dynamically calls the info method for the associated file-type. It returns a structure with the attributes associated with the object by concatenating attributes declared in the object itself and attributes that are returned by the info-method. The info-method typically reads some header-information from the associated files.

Additional attributes can be added to the SFRepos object using the ADDATTR method. These attributes will be visible as properties of the SFRepos object and are also returned by the "attr" property.

There are two attributes that are automatically defined by the SFRepos object. These attributes are automatically returned by the "attr" property but can be manually set using the ADDATTR method. They are:

Attribute	*Description*
Units	(Default: "unit") This string identifies the units of the data associated with this object.
Gain	(Default: 1) This value can be used to multiply the stored data to match the provided "Units" attribute. This is usually 1, but is sometimes needed when the data is stored with on an arbitrary scale.

FileFormat Specific Methods

The strength of the SFRepos Toolbox lies with the standardized GET, INFO, and CLEAN methods. For each file-type, there should be one GET and one INFO method available. The CLEAN method is optional. In this section, we will discuss the inputs and outputs of these methods and show how easy it is to add support for a new file-type by creating these three methods.

METHOD NAMES AND HELP SECTION

The name of the GET and INFO methods is a simple extension of the TYPE attribute of the object. The method that is called to access data is simply the value of the TYPE attribute prepended by either 'get' or 'info'. For example, the methods associated with the 'BinaryByChannel' TYPE attribute are: 'getBinaryByChannel.m' and 'infoBinaryByChannel.m'. These files should be stored inside the '@SFRepos' folder of the toolbox.

Information about the file-type should be entered as commented lines following the function declaration of the infoMethod. This information is automatically displayed when the user clicks on the 'type' hyperlink when the object is displayed in the Matlab command window. It is recommended that the user adheres to standard help notation in Matlab. See below for an example:

Help example:

```
function out = infoBinaryByChannel(obj, locPath, option)
  %BINARYBYCHANNEL  Flat Binary file format with single file per channel.
  %
  %   The BINARYBYCHANNEL format handles flat binary files with no header that
  %   have a single file for each of the channels.
  %
  %   Required attributes:
  %     'Format':   This string indicates the class of the values in the file.
  %                 Set this to 'double', 'single', 'uint16' or any other
  %                 default type of data.
  %     'SwapBytes':This Boolean determines whether the bytes in the file should
  %                 be swapped. This is sometimes necessary depending on the
  %                 platform that was used to save the data.
  %   Optional attributes:
  %     none
```

THE INFOMETHOD

The infomethod serves two purposes: 1) Returning information about the requirements of the constructor inputs during object initialization, and 2) Returning meta-information that can be extracted from the included files.

The template for the INFOMETHOD is given below and is also available as a ".m"file in the toolbox folder. Use this template to specify the INFOMETHOD for additional file-types. Do not change the template format unless you make sure that the same arguments are returned for the specified options!

The template contains 3 sections:

1) The function declaration: The INFOMETHOD should always contain a single output and the specified 3 inputs. The inputs are the object, the location to the files of the object and the option input which determines what is returned in the output. The name of the function should always start with "info" followed by the filetype it serves.
2) The "init"-case: The returned variable is a structure with 4 properties as described in the code. You should add code that populates the properties correctly.
   a. requiredAttr: This cell-array of strings indicate the names of the type-attributes that are required to be specified during object creation. These should be the names of attributes whose content is necessary for the GETMETHOD to function properly.
   b. optionalAttr: This cell-array of strings indicate the names of the type-attributes that are optional inputs for the GETMETHOD. The values of these attributes are not required for the GETMETHOD but might alter its behavior. These attributes are defined when the user requests the data. For example:

```
>> data = obj.data(1:1000, 1:15, 'optAttrName', 'optAttrValue');
```

   c. size: This [1x2] vector indicates the number of samples and the number of channels respectively.
   d. format: This string indicates the precision of the returned data (i.e. "single" "double" or "uint64").

3) The "info"-case: The returned variable is a structure with with zero or more properties. This structure can have any number of properties depending on the information that is embedded in the files associated with this object. When the user accesses the "attr" property of the object, the returned results of this method are concatenated with any attributes that are specified in the object itself. You can use this to return header information embedded in the files.

The infoTemplate.m file:

```
function out = infoTemplate(obj, locPath, option)
  %INFOTEMPLATE  Short one-sentence summary of file-type.
  %
  %   Additional information.

  assert(nargin == 3, 'SciFileRepos:infoMethod', ...
    'Incorrect number of input arguments for infoMethod.');

  switch option
    case 'init'
      % Required output structure for case 'init'.
      out = struct(...
        'requiredAttr', [], ...
        'optionalAttr', [], ...
        'size', [], ...
        'format', [] ...
        );

      % -- Set the contents of the out structure here --
      % ADD CONTENT

    case 'info'
      out = struct([]);

      % -- Set the contents of the out structure here --
      % ADD CONTENT

    otherwise
      error('SciFileRepos:getattr','Incorrect option: %s', option);
  end
end
```

THE GETMETHOD

The GETMETHOD is used to return data that is requested by accessing the "data" property of the object. This method has a mandatory structure which is shown below which is also available in the toolbox folder as a template (getTemplate.m).

1) The function declaration: The GETMETHOD can either return an array or a structure with a property 'data' that contains an array. The array should represent the data that is requested. The GETMETHOD has 5 inputs:
   a. obj: This is a handle to the object that requested the data. All properties of the object are available to be used by the GETMETHOD. Specifically, the properties "userData" and "fetchCache" can be used as needed to make the GETMETHOD more efficient. How this can be used is discussed below.
   b. channels: This vector of numerics indicate the channels that are requested.
   c. indices: This vector indicates the indices that are requested.
   d. filePath: This string is the path to the folder where the files are located.
   e. options: This is a structure that includes the required attributes that were set during the object construction and the optional properties that are set in the call to the "data"-property.
2) Data retrieval: The body of the GETMETHOD is used to fetch the data from the files associated with the object. The filenames are retrievable from the object using: "obj.files". The method that is used to fetch the data from the files is dependent on the specific file-format.

```
function data = getTemplate(obj, channels, indices, filePath, options)
  % GETTEMPLATE  See INFOTEMPLATE

  % The returned data should follow one of the following formats.
  % data = zeros(length(indices),length(channels),format);
  % data = struct('data', zeros(length(indices),length(channels),format));

  % -- Set the contents of the data array here --
  % ADD CONTENT
```

THE CLEANMETHOD

The CLEAN method is an optional method that can be used to specify specific behavior when the object is deleted or when the CLEANUP function is used. This can be useful in situations where you might want to use temporary files as part of the GET or ATTR methods. You can correctly delete those files when the object is being deleted from memory using this method.

There is no special syntax required for the CLEAN method; it will run as long as you adhere to the standard naming convention. Some additional information can be found in the 'cleanTemplate.m' file. As you can see below, the method should have a single input which is the object itself.

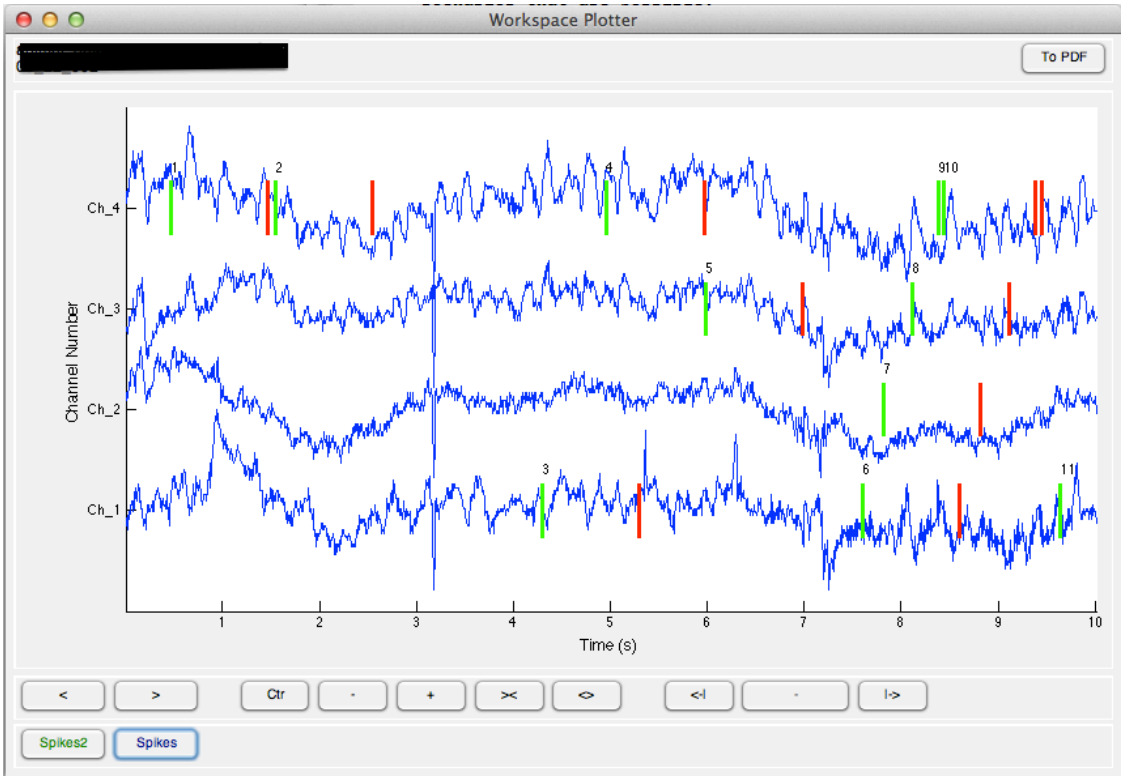
```
function cleanTemplate(obj)
  % CLEANTEMPLATE  See infoTemplate

  % Place any code that you want to execute befor deleting the object or
during
  % cleanup here.

  % NOTE: If you are debugging this method when an object is being
  % deleted, the command window will not be able to show the contents of the
  % object (it shows: "Deleted object"). However, the object is actually NOT
  % deleted yet and the individual properties of the objects are still
  % accessible. This is the result of Matlab setting the isvalid property to
  % false before this function is executed.

end
```

SFR-Toolbox Tools

The SFRToolbox is developed to simplify and standardized access data. The idea is that if datasets can be represented by a single object independent of the underlying file-format, it is easy to make methods and scripts that work on any dataset.

A viewer tool is provided with the SFR-Toolbox. This tool allows you view and browse through the multichannel data in your object. It also provides support for showing one, or more annotation layers using a standardized annotation-struct. Please review the help sections of the SFRVIEWER and the ANNOTATIONSTRUCT methods of the SFR-Toolbox. They are located in the SFR-Toolbox/tools folder.

A screenshot of the SFR-Viewer is provided below:



SFRepos Tutorial

The SFRepos example shows how easy it is to create a SFRepos object that provides access to data in a set of files. The completed example object can be loaded from the 'repos.mat' file in the 'SFRexample' folder of this Toolbox. However, in this quick tutorial we will create the object from scratch and show how it loads the sample data from the '/SFRexample/data' folder.

The SFRepos object in the 'repos.mat' file was created as follows:

```
>> sfrsetlocation('Work', 'Path\To\XML\File.xml');
>>
>> files = {'Channel1.dat' 'Channel2.dat' 'Channel3.dat'};
>> rootID = 'SFRroot';
>> typeID = 'BinaryByChannel';
>> subPath = 'SFRExample/data';
>> typeAttr = {'Format' 'double' 'SwapBytes' false};
>> repos = SFRepos(typeID, rootID, subPath, files, typeAttr)

repos =

  SFRepos:

      typeId: BinaryByChannel
      rootId: 'SFRroot'
     subPath: 'SFRExample/data'
       files: {'Channel1.dat'  'Channel2.dat'  'Channel3.dat'}
    typeAttr: [1x1 struct]
        data: [100000x3 double]
        attr: [1x1 struct]

  Methods, Location, getOptions

>>
```

The typeID indicates the file-format of the associated files. This determines which getMethod and infoMethod is used for this object. The RootID is an identifier that point to a specific path in the userLocation.xml file for a given location. The subPath is the relative path from the rootPath to the location of the files. The files is a cell-array of strings with the filenames that are associated with this object and the typeAttr are the required and optional attributes that will be passed to the getMethod. Which typeAttr attributes are required is set in the infoMethod.

Data can be accessed using the 'data' property:

```
>> aux = repos.data(1:5,:)

aux =

   0.000999999833333   0.865524970855107   1.000000000000000
   0.001999998666667   0.865023672400875   1.000000000000000
   0.002999995500002   0.864521508923044   1.000000000000000
   0.003999989333342   0.864018480923776   1.000000000000000
   0.004999979166693   0.863514588906098   1.000000000000000

>>
```

Attributes can be accessed using the 'attr' property:

```
>> repos.attr

ans =

         size: [100000 3]
       format: 'double'
    SwapBytes: 0

>>
```

Attributes returns a structures with the merged attributes from 1) the 'typeAttr' property, 2) attributes added with ADDATTR (see below), and 3) the attributes that are returned by the infoMethod.

Additional attributes can be added to the object using ADDATTR:

```
>> addattr(repos,'ExpName','Experiment1','SessionName', ...
'Session1','sampleFreq',2713)

ans =

  SFRepos:

         typeId: BinaryByChannel
         rootId: 'SFRroot'
        subPath: 'SFRExample/data'
          files: {'Channel1.dat'  'Channel2.dat'  'Channel3.dat'}
       typeAttr: [1x1 struct]
           data: [100000x3 double]
           attr: [1x1 struct]
     sampleFreq: 2713
    SessionName: 'Session1'
        ExpName: 'Experiment1'

  Methods, Location, getOptions

>>
```

The added attributes can contain information that is not available within the associated files. In other words, information that is not returned by the infoMethod. Examples are ‘channel Names’, ‘Session Name’, ect.