Paul Scherrer Institut

EPICS V4 Archiver Service and Matlab client

Timo Korhonen

Overview

- ArchiverService
  - To access Channel Archiver data using pvAccess RPC
  - Written by David Hickin (Diamond)
- Client code to access the ChannelArchiver service from Matlab
  - Written by me to
    - Have a tool to access the service
    - Learn how to write client code
  - Used Matlab because
    - Matlab is a central tool for our SwissFEL project
    - Java API can be directly used
    - Quick cycle for testing (scripting)
  - Some Qt (C++) code to do the same thing
    - At the moment (slightly) less sophisticated
    - Not enough time to show in this talk, ask me for a demo if interested

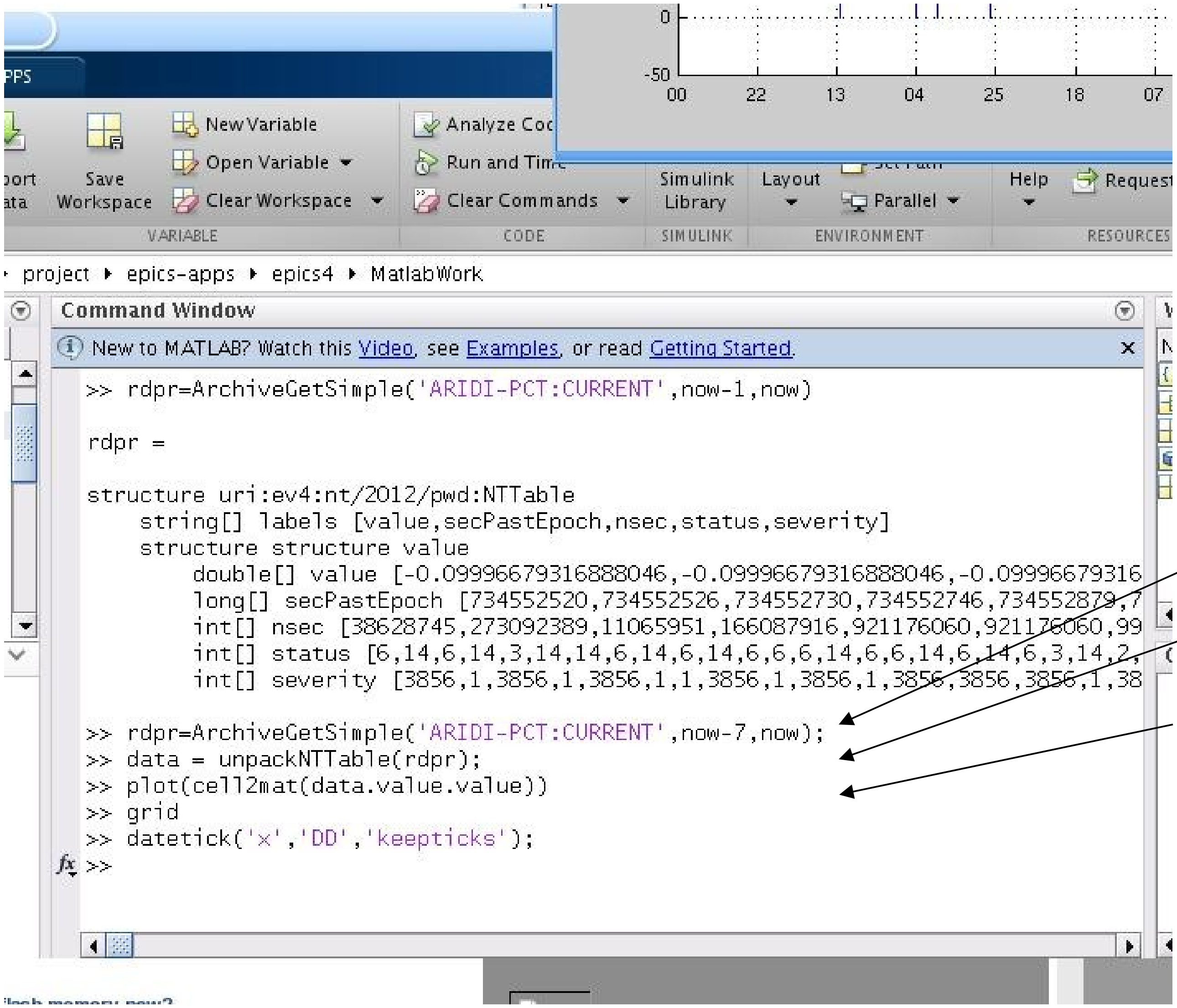
Archive Service

- ArchiverService
  - Many sites are still using the Channel Archiver (PSI and Diamond at least)
  - Direct access to the data would be valuable
  - Implement access to the data as a V4 service
  - One of the first services that have been developed and deployed
    - There are a few more (RDB service, etc.)
- Basic mode of operation:
  - Use the RPC method that pvAccess provides (new in V4)
  - Client sends a query to a server, with parameters
  - Server fetches the data, packs it up and sends to the client
  - Client receives the data, unpacks the structure and (in this case) returns the data as a Matlab native structure
  - The Java API can be used natively in Matlab
    - No wrappers in between
    - Some conversions between Java structures and Matlab structures required, however

Anatomy of a RPC (Remote Procedure Call) in V4

- RPC is a pvAccess operation that can take parameters
  - In the archive service case:
    - Channel to be retrieved
    - From <start time> to <end time>
  - These parameters are sent to the server as a structure
    - The rule is to use the NTURI normative type http://epics-pvdata.sourceforge.net/alpha/normativeTypes/normativeTypes.html#nturi
    - The client creates this structure and sends it to server
  - Server advertises a channel name that the client connects to
    - Basic connection mechanism is similar to channel access:
      - Search broadcast, server that has the name, replies, etc.
      - After that the differences start....(introspection, etc.)
  - Server receives the structure from client and
    - unpacks the parameters, fetches the data from archiver
    - Packs the data into another normative type structure (NTTable) and sends
      - This will probably change to use a more appropriate structure
  - Client receives the data and unpacks it

The code...



```
>> rdpr=ArchiveGetSimple('ARIDI-PCT:CURRENT',now-1,now)

rdpr =

structure uri:ev4:nt/2012/pwd:NTTable
    string[] labels [value,secPastEpoch,nsec,status,severity]
    structure structure value
        double[] value [-0.09996679316888046,-0.09996679316888046,-0.09996679316
        long[] secPastEpoch [734552520,734552526,734552730,734552746,734552879,7
        int[] nsec [38628745,273092389,11065951,166087916,921176060,921176060,99
        int[] status [6,14,6,14,3,14,14,6,14,6,14,6,6,6,14,6,6,14,6,14,6,3,14,2,
        int[] severity [3856,1,3856,1,3856,1,1,3856,1,3856,1,3856,3856,3856,1,38

>> rdpr=ArchiveGetSimple('ARIDI-PCT:CURRENT',now-7,now);
>> data = unpackNTTable(rdpr);
>> plot(cell2mat(data.value.value))
>> grid
>> datetick('x','DD','keepticks');
fx >>
```

What the user sees...

ArchiveGet call

Unpacking the data

Plotting, etc. or whatever the user then wants to do with the data

Let us look inside these functions

PAUL SCHERRER INSTITUT PSI

The code...

```
function rdpr = ArchiveGetSimple( pvname, starttime, endtime )
%ArchiveGetSimple get data from archiver service into a pvData structure
%   Detailed explanation goes here

import('org.epics.pvaccess.*')
import('org.epics.pvaccess.easyPVA.*')
import('org.epics.pvdata.*')
%
request.scheme='pva';
request.path='SLS-LT'; %hardcode for now - replace later
request.query={'starttime',starttime;'endtime',endtime;'entity',pvname};

%start the EasyPVA factory
easy = EasyPVAFactory.get();
pvr=BuildRPC(request);
% now do the query
rdp=easy.createChannel(request.path).createRPC();
%created an EasyRPC, now connect
rdp.connect();
% do the request. Result is a PVStructure object
rdpr = rdp.request(pvr);
%now the result is in structure rdpr.
end
```

This is just a wrapper around pvAccess and pvData calls

Import the Java classes

Create a Matlab structure for the request

The actual pvAccess things are here

The request call returns a NTTable (Java structure) ; rdpr
This is returned to the caller

The code: creating a RPC query structure

```
function pvr = BuildRPC( request )
%BuildRPC Build a PVStructure for making a RPC call (EPICS 4)
%   pvr = BuildRPC(request)
%   request is a Matlab struct that contains the query data
%   namely: scheme, path and query
%     scheme: pva
%     path: the service name (EPICS 4 PV name)
%     query: query parameters, service-dependent
%   pvr is the NTURI PVStructure
% For RPC queries, the NTURI normative type is used.

if(isfield(request,'scheme') && isfield(request,'path') &&
isfield(request,'query') )

    % uses pvdata
    import('org.epics.pvdata.*')
    %convenience number for possible time calculations
    epicsepoch = datenum(1990,1,1);
…<code continues>
```

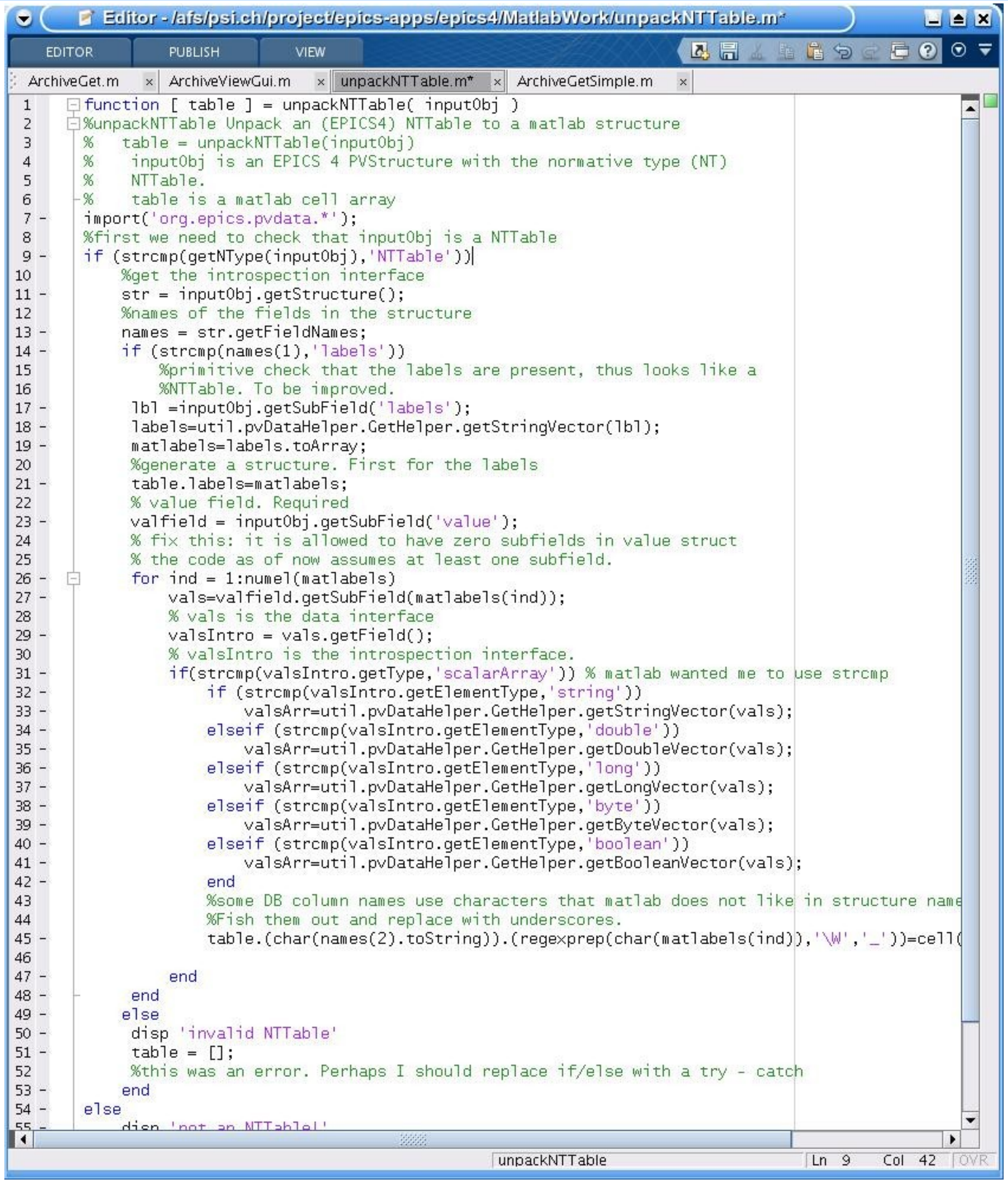
BuildRPC creates the NTURI structure for a query

A bit too long to be shown on a slide (82 lines of code, with comments)

-takes data from a matlab structure

-this routine can be used for any service (only specialty here is how to handle EPICS times: times have to be converted from the EPICS epoch to times that Matlab understands.)

The code: data unpacking



```
function [ table ] = unpackNTTable( inputObj )
%unpackNTTable Unpack an (EPICS4) NTTable to a matlab structure
%   table = unpackNTTable(inputObj)
%    inputObj is an EPICS 4 PVStructure with the normative type (NT)
%    NTTable.
%    table is a matlab cell array
import('org.epics.pvdata.*');
%first we need to check that inputObj is a NTTable
if (strcmp(getNType(inputObj),'NTTable'))
    %get the introspection interface
    str = inputObj.getStructure();
    %names of the fields in the structure
    names = str.getFieldNames;
    if (strcmp(names(1),'labels'))
        %primitive check that the labels are present, thus looks like a
        %NTTable. To be improved.
     lbl =inputObj.getSubField('labels');
     labels=util.pvDataHelper.GetHelper.getStringVector(lbl);
     matlabels=labels.toArray;
     %generate a structure. First for the labels
     table.labels=matlabels;
     % value field. Required
     valfield = inputObj.getSubField('value');
     % fix this: it is allowed to have zero subfields in value struct
     % the code as of now assumes at least one subfield.
     for ind = 1:numel(matlabels)
         vals=valfield.getSubField(matlabels(ind));
         % vals is the data interface
         valsIntro = vals.getField();
         % valsIntro is the introspection interface.
         if(strcmp(valsIntro.getType,'scalarArray')) % matlab wanted me to use strcmp
             if (strcmp(valsIntro.getElementType,'string'))
                 valsArr=util.pvDataHelper.GetHelper.getStringVector(vals);
             elseif (strcmp(valsIntro.getElementType,'double'))
                 valsArr=util.pvDataHelper.GetHelper.getDoubleVector(vals);
             elseif (strcmp(valsIntro.getElementType,'long'))
                 valsArr=util.pvDataHelper.GetHelper.getLongVector(vals);
             elseif (strcmp(valsIntro.getElementType,'byte'))
                 valsArr=util.pvDataHelper.GetHelper.getByteVector(vals);
             elseif (strcmp(valsIntro.getElementType,'boolean'))
                 valsArr=util.pvDataHelper.GetHelper.getBooleanVector(vals);
             end
             %some DB column names use characters that matlab does not like in structure name
             %Fish them out and replace with underscores.
             table.(char(names(2).toString)).(regexprep(char(matlabels(ind)),'\W','_'))=cell(

         end
     end
    else
     disp 'invalid NTTable'
     table = [];
     %this was an error. Perhaps I should replace if/else with a try - catch
    end
else
    disp 'not an NTTable!'
```

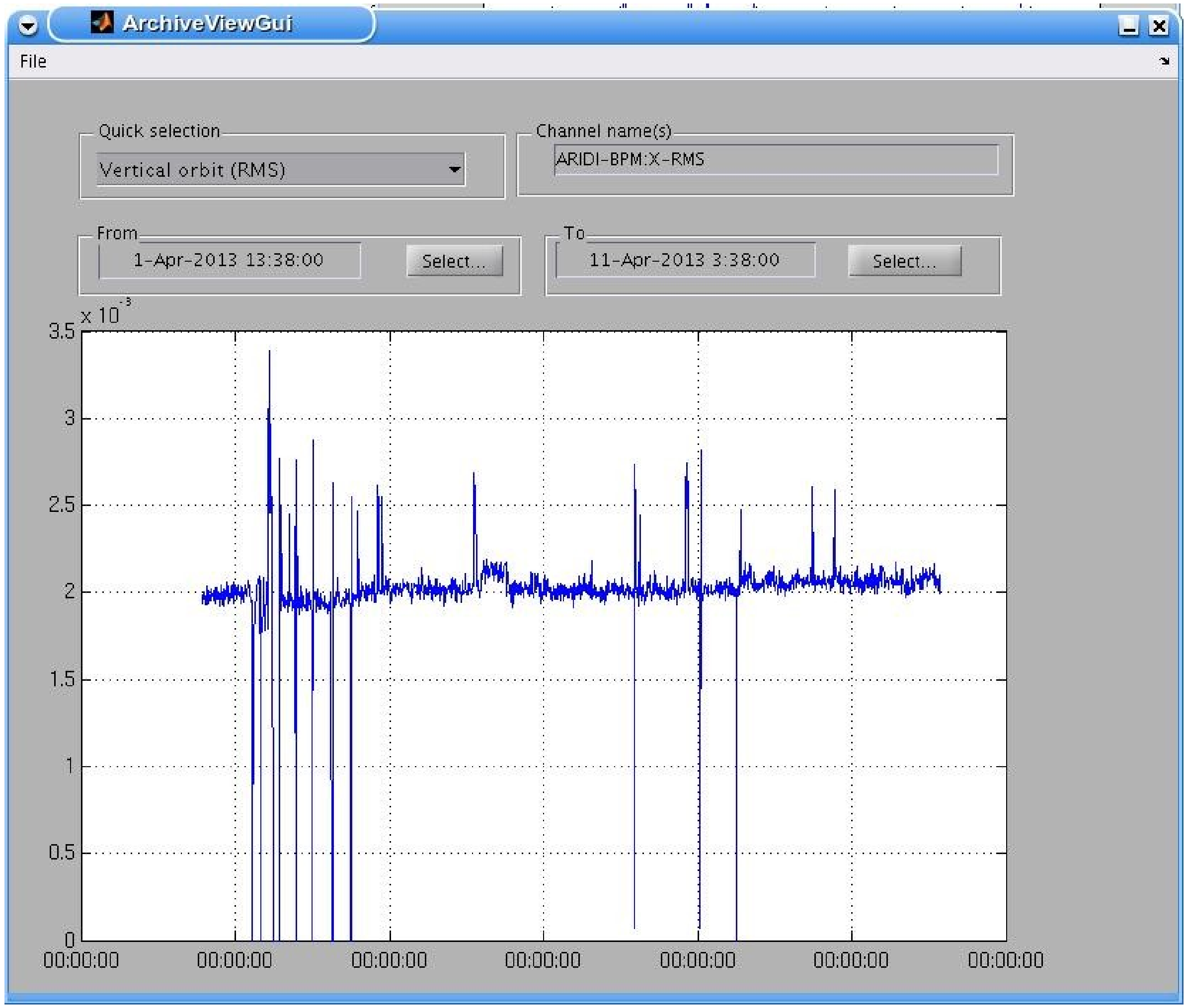
Another helper routine: unpackNTTable

-again generic, not specific to any service

-returns the data in a matlab structure for easy manipulation (plotting, calculations, etc.)

About 60 lines of code (with comments, 40 without)

Some screenshots



Quick selection: Vertical orbit (RMS)

Channel name(s): ARIDI-BPM:X-RMS

From: 1-Apr-2013 13:38:00

To: 11-Apr-2013 3:38:00

Beginnings of a GUI (matlab)

Fetch the data from archive and plot

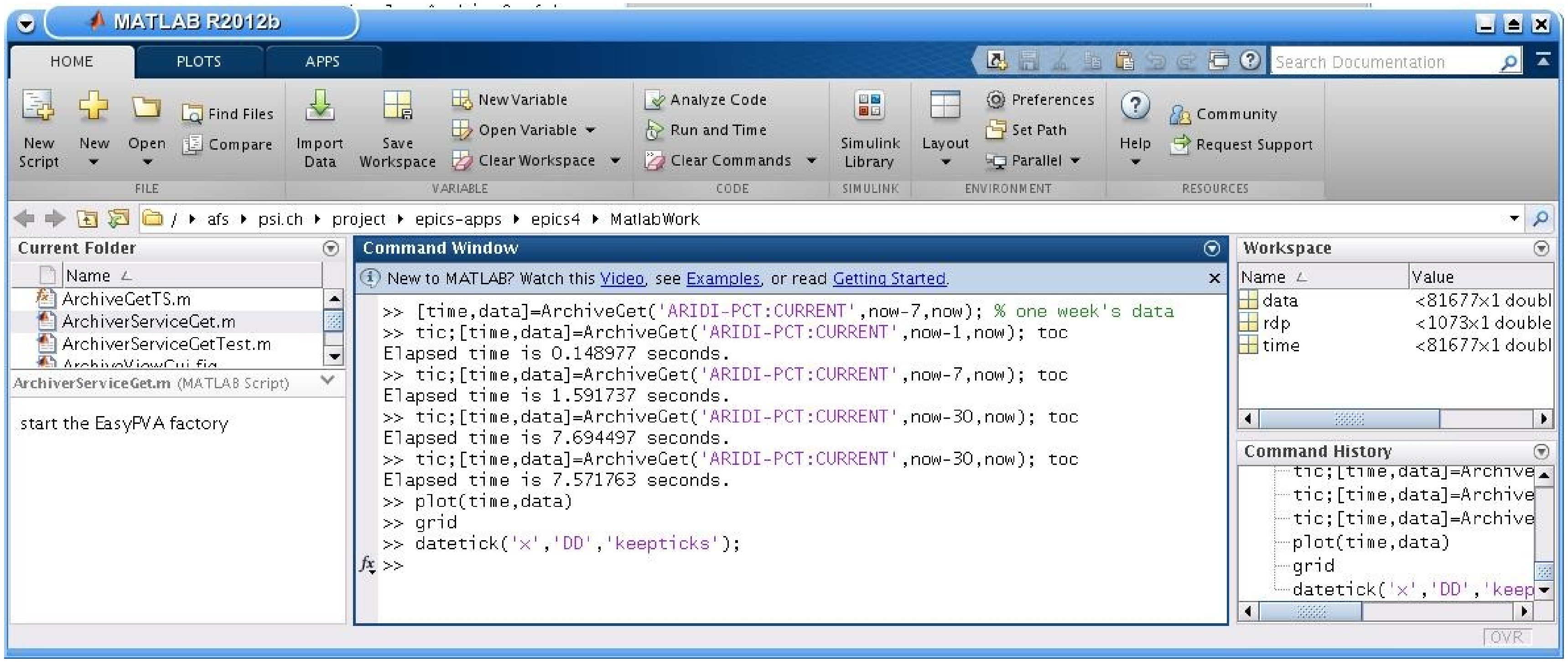
This is still at a primitive state, but nevertheless fun to play with

Define a channel, or select one from a predefined set
*the idea is to get the channel names from a service – not implemented yet

Define start and end times

(demo would be nice, but using Matlab remotely can be risky – and slow)

Some screenshots



```
>> [time,data]=ArchiveGet('ARIDI-PCT:CURRENT',now-7,now); % one week's data
>> tic;[time,data]=ArchiveGet('ARIDI-PCT:CURRENT',now-1,now); toc
Elapsed time is 0.148977 seconds.
>> tic;[time,data]=ArchiveGet('ARIDI-PCT:CURRENT',now-7,now); toc
Elapsed time is 1.591737 seconds.
>> tic;[time,data]=ArchiveGet('ARIDI-PCT:CURRENT',now-30,now); toc
Elapsed time is 7.694497 seconds.
>> tic;[time,data]=ArchiveGet('ARIDI-PCT:CURRENT',now-30,now); toc
Elapsed time is 7.571763 seconds.
>> plot(time,data)
>> grid
>> datetick('x','DD','keepticks');
>>
```

Name	Value
data	<81677x1 doubl
rdp	<1073x1 double
time	<81677x1 doubl

```
tic;[time,data]=Archive
tic;[time,data]=Archive
tic;[time,data]=Archive
plot(time,data)
grid
datetick('x','DD','keep
```

Some timings:

-get one day's data (beam current): **0.15 seconds**
-one week's data: **1.5 seconds**
-one month's data (81677 values): around **7 .5 seconds**

-most of the time is Matlab structure manipulation (I have not profiled the code, however)

Summary

- ArchiverService
  - Works very well (stable, fast)
  - Needs still some extensions (add waveforms, display information)
- Programming with V4 pvData, pvAccess
  - There is a learning curve, can be steep at times
  - But: when you get familiar with the programming, it is very efficient and productive
  - Opens up **a lot** of new possibilities
  - Normative types are a key aspect: even if they do not look very sexy in the beginning, you will eventually love them :-)
- Services programming
  - Once you have learned how to do one, creating more services becomes easy
  - This is a very efficient way of data integration
    - One set of tools for all data
    - Combining data from different sources becomes easy
- Final disclaimer
  - The code shown is from a beginner – anybody interested is welcome to have it, but it is by no means production-ready. Use at your own risk.