Follow these steps to integrate the Shared Data Access System in your Octave code. All the examples were successfully tested in MS Windows and Linux distributions like Fedora, Gentoo and Debian.

For Windows, we recommend using the installer.

Version 3.8 added Java support to the Octave core. If you are using an older version, please follow these intructions instead.

Windows

If your Octave doesn't automatically find your Java instalation you can try to set **JAVA_HOME** as a system variable:

On Windows 10, open the windows *Definitions,* on the start menu.

Search for *Environment Variables...*
In the system variables click *New...*
The Variable name is: **JAVA_HOME**
In the value field enter the full path to the folder where *jvm.dll* is installed (for example, for Java 14, the default is `C:\Program Files\Java\jdk-14\bin\server).`

Attention: The build architecture of Octave and Java must match (32 bits with 32 bits, 64 bits with 64 bits).

Linux

This issue hasn't been reported on Linux systems, but if Octave doesn't find your java instalatio, you can try setting **JAVA_HOME** as a system variable:

Supposing you have **Java** in `/usr/lib/jvm/default-java/`:

```
export JAVA_HOME="/usr/lib/jvm/default-java/"
```

Download the libraries

Download the following libraries into a folder of your system:

Apache XML-RPC
Apache Jakarta Commons

Since the server at baco computer uses an older version, if you are planing on accessing it server, you should download this version of SDAS Core Libraries and Client:

SDAS Core Libraries
SDAS Client

If you are using a different server, download these instead:

SDAS Core Libraries
SDAS Client

Add the SDAS libraries to the octave classpath

These example assume that you have saved all the jar files in the folder /home/user/sdas/

```
javaaddpath("/home/user/sdas/SDAS.jar")
javaaddpath("/home/user/sdas/SDASClient.jar")
javaaddpath("/home/user/sdas/commons-codec-1.3.jar")
javaaddpath("/home/user/sdas/xmlrpc-2.0.jar")
```

Beware that on Windows you must use / instead of the regular \. If you have saved all the jar files in the folder C:\sdas\, for example:

```
javaaddpath("C:/sdas/SDAS.jar")
javaaddpath("C:/sdas/SDASClient.jar")
javaaddpath("C:/sdas/commons-codec-1.3.jar")
javaaddpath("C:/sdas/xmlrpc-2.0.jar")
```

Get a connection to the sdas server

```
client = javaObject("org.sdas.core.client.SDASClient",
"baco.ipfn.ist.utl.pt", 8888);
```

Search events

```
found = client.searchDeclaredEventsByName('S');

found = client.searchDeclaredEventsByName('SHOT');

found = client.searchDeclaredEventsByName('SHOT', 'en');

found = client.searchDeclaredEventsByUniqueID('0x0000');

found = client.searchDeclaredEventsByDescription('Shot');

found = client.searchDeclaredEventsByDescription('Shot', 'en');

for i=1:1:size(found)
    found(i).toString
end

max = client.searchMaxEventNumber('0x0000')

min = client.searchMinEventNumber('0x0000')
```

Search events in a time window

NOTE: You can construct time with a resolution of picosseconds, just add to the example values for millis, micros, nanos and picos
NOTE 2: Date constructors have the months index to 0 (January is 0 and December is 11)

Search events in December 2005:

```
date_start = javaObject("org.sdas.core.time.Date", 2005, 11, 1);
date_end =  javaObject("org.sdas.core.time.Date", 2005, 11, 31);
tstart =  javaObject("org.sdas.core.time.TimeStamp", date_start);
tend =  javaObject("org.sdas.core.time.TimeStamp", date_end);
eventsFound = client.searchEventsByEventTimeWindow(tstart, tend);
for i = 1:1:size(eventsFound)
    eventsFound(i).toString
end
```

Search events in the 22 December 2005 between 5pm and 6pm:

```
date_start = javaObject("org.sdas.core.time.Date", 2005, 11, 22);
date_end = javaObject("org.sdas.core.time.Date", 2005,11,22);
time_start = javaObject("org.sdas.core.time.Time", 17, 0, 0);
time_end = javaObject("org.sdas.core.time.Time", 18, 0, 0);
tstart = javaObject("org.sdas.core.time.TimeStamp", date_start,
time_start);
tend = javaObject("org.sdas.core.time.TimeStamp", date_end, time_end);
eventsFound = client.searchEventsByEventTimeWindow(tstart, tend);
for i = 1:1:size(eventsFound)
    eventsFound(i).toString
end
```

Search parameters

```
parametersFound = client.searchParametersByName('DENS');

parametersFound = client.searchParametersByName('DENS', 'pt');

parametersFound = client.searchParametersByUniqueID('DENS');

parametersFound = client.searchParametersByDescription('current');


for i = 1:1:size(parametersFound)
    parametersFound(i).toString
end
```

Search data

This function returns the parameters unique identifiers where the data isn't null for the selected event:

```
dataFound = client.searchDataByEvent('0x0000', 17898);
for i = 1:1:size(dataFound)
    dataFound (i)
end
```

Get data

NOTE: The unique identifiers are CASE-SENSITIVE

The returned data structure gives you information about:

- start time
- end time
- time of the event
- mime_type
- the parameter unique identifier

Data for only one parameter

```
dataStructArray=client.getData('POST.PROCESSED.DENSITY','0x0000', 17898)
dataStruct=dataStructArray(1);
dens=dataStruct.getData;
for i=1:length(dens)
density(i)=dens(i);
end
tstart = dataStruct.getTStart;
tend = dataStruct.getTEnd;
```

Calculate the time between samples

```
tbs= (tend.getTimeInMicros - tstart.getTimeInMicros)/length(density);
```

Get the events associated with this data

```
events = dataStruct.getEvents;
```

The event time (I'm assuming the event I want is at the index 0, but I should check first...)

```
tevent = events(1).getTimeStamp;
```

The delay of the start time relative to the event time

```
delay = tstart.getTimeInMicros - tevent.getTimeInMicros
```

Finally create the time array

```
times = delay:tbs:delay+tbs*(length(density)-1);
```

And plot the data

```
plot(times, density);
```

Data for several parameters in the same event

```
dataStruct=client.getMultipleData({'POST.PROCESSED.DENSITY',
'POST.PROCESSED.IPLASMA'},'0x0000', 17898)
dataStructDens=dataStruct(1,1);
dataStructIP=dataStruct(2,1);
dens=dataStructDens.getData();
ip=dataStructIP.getData();
```

Data for several parameters in different events

```
dataStruct=client.getMultipleData({'POST.PROCESSED.DENSITY',
'POST.PROCESSED.IPLASMA'},{'0x0000','0x0000'}, [17898,17899])
dataStructDens=dataStruct(1,1);
dataStructIP=dataStruct(2,1);
dens=dataStructDens.getData();
ip=dataStructIP.getData();
```

Data for the same parameter in different events

```
dataStruct=client.getMultipleData('POST.PROCESSED.DENSITY',{'0x0000','0x0000'},
[17898,17899])
dataStructDens=dataStruct(1,1);
dataStructIP=dataStruct(2,1);
dens=dataStructDens.getData();
ip=dataStructIP.getData();
```

Data for the same parameter in different event numbers

```
dataStruct=client.getMultipleData('POST.PROCESSED.DENSITY', '0x0000',
[17898,17899])
```

More example code is available on the Octave Code page.