Converting an R Script to a DataSHIELD function
Loading modified versions of the DataSHIELD packages

1. Client functions

The process for testing client functions is easier than the process for testing server functions. This is because we don't need to modify the VMs to test changes to the client.

Typically it is your own computer that acts as the DataSHIELD client. The aim is to momentarily replace your installed client package with a client package that contains your modified version of functions; so that the R session on your computer can use these modified versions instead.

Process
(1) If you are not already on the branch that holds the changes you want to test, checkout that branch.
(2) Start R and use the devtools package to load the changed package from your computer.

(1) Checkout the code (for example, dsBaseClient)
Navigate to your local copy of the package repository, and make sure you have checked out the branch which holds the changes you want to test.

You can checkout the branch that contains those changes through GitHub for Windows, or through the git shell as follows:

```
$ cd /path/to/dsBaseClient
$ git checkout <branch with changes to test>
```

NOTE: If you want to test someone else's changes, then you will either need to get a copy of the repositiory that contains their changes, or update your local copy of that repository.

If you don't already have a cloned version of the dsBaseClient repo on your computer, then clone it from github:

```
$ git clone
https://github.com/datashield/dsBaseClient.git
```

If you do have a cloned version of the dsBaseClient repo on your computer then make sure you are up to date. Either by pressing 'sync' in GitHub for Windows or through git shell:

```
$ git pull origin
```

Once you know you have a local copy of the new changes you want to test, you can checkout the branch that contains those changes.

(2) Load the changes
By checking out the appropriate branch of the package repository you have made sure that the code you want to test is available in a folder on your computer. Now you can start R and use the devtools library to load the version of package in that folder instead of the version that was installed with 'install.packages()'.

```
R
> library(devtools)
>
> # Tell devtools where to find the modfied
package
> devtools::load_all('/path/to/dsBaseClient/')
>
> # Load the other client packages as you
normally would
> library(dsStatsClient)
> library(dsGraphicsClient)
> library(dsModellingClient)
```

When you now use DataSHIELD the changes to dsBaseClient will be available to you.

If you subsequently run library(dsBaseClient) then you will revert to the official installed version of dsBaseClient. However, you can use devtools::load_all() to keep re-loading the modified package.

You can repeat this 'load, test, change' cycle until you are happy. Then you can commit your changes to git, and push to GitHub.

2. Server functions

Testing server functions involves some extra steps, and there are few different methods to make functions available on the VMs.

First, you can input an R script through the Opal web interface on each VM.

Second, you can install a modified version of the server side DataSHIELD packages on each VM. This can be done in multiple ways.

2.1 Add a script, via the Opal web interface

The easiest way to get a function onto the VMs, so that you can test it is to use the 'Add method' functionality of the Opal web interface. As follows:

(1) Navigate to Administration > DataSHIELD.
(2) In the 'Methods' section select 'Aggregate' or 'Assign', depending on what kind of function you want to create.
(3) Click on the 'Add Method'.
(4) Give the new function a name. Add a suffix to the name to avoid conflicts with existing functions (e.g. meanDS.new).
(5) For 'type' choose R scirpt
(6) Paste your code into the window. (start the code with "`function(...) {…}`", the function will not work if you start in the more conventional way "`function_name <- function(...) {...}`").

2.2 Installing a modifed version of a server package

Installing a modified version of a server package is similar to the process for client packages: you need to replace

the official released versions of the DataSHIELD server packages with your modified versions. And you need to do this on each of the VMs.

There are at least two ways to achieve this. The simplest is to get your changes onto the VMs by going via GitHub. (This is possible because GitHub has a dual role for DataSHIELD. It is a tool for managing the source code, but also a component of the Opal and DataSHIELD infrastructure.)

You can only install code onto your VMs that is stored in a repository that is part of the DataSHIELD organisation on GitHub. So you cannot use the methods below to install modified package on the VMs from personal repositories, for example. This means that only developers who can commit to the DataSHIELD repositories can use GitHub as a way install modifed server side packages. *Please see the 'alternative method' below for instructions for circumventing this.*

Process
(1) Make and commit your changes to a new branch.
(2) Push your new branch to GitHub
(3) Pull that branch to the VMs

(1) Make and commit your changes to a new branch
If you want to edit a function in dsBase create an approptiate branch locally. Either through GitHub for Windows, or through the git shell, for example:

```
$ git checkout -b <function_mod> master
```

Make all your edits on this branch, and then commit your changes.

(2) Push your new branch to GitHub
Now that you have a branch containing the changes you want to put onto the VMs, you need to push that branch to GitHub so that the VMs can access it.

Within GitHub for Windows this pushing is called 'publishing', if the branch you're working on does not already exist on GitHub. If it does already exist, then it is refered to as 'syncing'. In both cases the button is in the same place in the interface, towards the top right. From the git shell, we simply push:

```
$ git push origin <function_mod>
```

(3a) Pull the branch onto the VMs – Method 1
With your modifications to a package pushed to a branch on GitHub, you can use the VM's Opal web interface to download and install the code from that branch.

The Opal web interface for each VM is available by at the IP address of the VM on port number 8080 (and 8443). Go to your web browser and type:

```
192.168.56.100:8080
```

You will see the Opal web interface log in screen. You can log in with:
username: administrator
password: password

Navigate to Administration > DataSHIELD and you will see a dashboard with information about the DataSHIELD packages that are installed. You can replace some or all of these packages with specific versions pulled from GitHub by doing the following:
* Remove the package.
* Select 'Add Package'.
* Select 'Install a specific package', and type 'dsBase' for example.
* In 'advanced options', give the name of the branch that holds your changes.
* Click add package.
* Remember to repeat this on each VM.

Note: using this method you don't just have to give a branch name, you could give a specific commit (from any branch, or a 'tag'). So for example you could downgrade your VMs to DataSHIELD v3.0.0 by typing "3.0.0", if you wanted to.

(3b) Pull the branch onto the VMs – Method 2
With your modifications to a package pushed to a branch on GitHub, you can use the opaladmin R package to download and install the code from that branch.

The opaladmin package allows you to run commands from a DataSHIELD client computer (i.e. your computer) to adminster opal. In this case, to install a modified version of a DataSHIELD server packages held on GitHub.

The advantage of this method is that you can issue commands to all your VMs at once (rather than one by one through the Opal web interface). The disadvantage is that we have found that it occasionally inexplicably fails to work.

```
R
> library(opal)
> library(opaladmin)
> opals <- datashield.login(logins=logindata)
>
> # remove the official package
> dsadmin.remove_package(opals, pkg='dsBase')
>
> # install the modifed package, where the
branch name
> # is specified in the ref= argument.
> dsadmin.install_package(opals, pkg='dsBase',
ref='<branch name>')
>
> # make the packages functions available
> dsadmin.set_package_methods(opals,
pkg='dsBase')
```

2.3 Alternative method - Add a modified package to Opal without access to the DataSIHIELD GitHub

It is possible – but even more convoluted – to add modified versios of server side packages to the VMs without using the DataSHIELD github repositories.

Essentialy, you will have to do manually what the Opal web interface and dsadmin commands above do for you.

Process
(1) Get your modified package onto each of the VMs
(2) Install the modified package on the VMs

(1) Get your modified package onto each of the VMs.
For example, linux users can use rsync:

```
$ rsync -a dsBase user@192.168.56.100:/home/user
```

Or, you can push the package to a personal GitHub repository, and then clone it on the VMs. For example, log in to the VM (either directly or via ssh) with:
username: administrator
password: password

```
$ git clone https://github.com/<my
username>/dsBase.git
```

(2) Install the modified package on the VMs
If you have'nt already, then log into the VMs. Next you need to start R, but as the 'rserver' user. This will make anything we install available in Opal.

```
$ sudo -u rserver R
```

```
R
> #Load the devtools package, which is already
installed
> library(devtools)
>
> # Install the package files that were copied
to the VM
> devtools::install('dsBase/', args="--
library='/var/lib/rserver/R/i686-pc-linux-gnu-
library/3.2/'")
```

The 'args' argument tells devtools to install the package in the rserver user's package library. The rserver user does not have permission to install packages anywhere else.

The new dsBase package should be visable and available through the Opal web interface. Remember to 'publish methods'. And to repeat this for each VM.

Note: this alternative method is particularly useful if you want to install a non-datashield R package on the VMs. For instance, if you want to write datashield functions that call functions from other analysis packages.

In that case, you can use the 'sudo -u rserver R' command to start an R session in which you can install packages using whatever method you need. e.g. from CRAN using install.packages(). Remember that you may need to set the library desitination, as above.