This confluence page is a summary of what is expected at this point. On day 3 we will have 30 minutes to catch up and then we will follow to the next steps of the development of our fist component!

- [ ] Have the Virtual Machine Working
- [ ] Configure Visual Studio Code
- [ ] Prepare your Environment
- [ ] Install IDL Interfaces to your INTROOT
- [ ] Prepare your Module
- [ ] ITS Representative to Push to GIT Group's module base files
- [ ] Testing Environment Basics
- [ ] Dummy Component Files

Visual Studio Code will help a lot to work in parallel, and for this reason, it is a very useful tool for this workshop. For the ones having trouble to share and join sessions due to the connection hanging when trying to sign in, Mario prepared the following instructions. The short trick is the following:

- For all that have issues with VSCode LiveShare sign-in:
    - To fix it you need to press CTRL + P, write "> Live Share: Disable VSCode Account authentication", select from auto-complete and press enter. After that, go to the bottom-left corner to the "Live Share" button and do again the sign-in process
    - Consider that depending on your combination of host and guest OS, you may need to use something from:
        - **Ctrl+Shift+P, Cmd+Shift+P, F1 or Fn+F1**
- Mario was kind enough to prepare a small document with more detailed instructions
    - Live Share Sign-in Fix

Preparing your environment

The first steps you would need to do is to checkout the repository to a convenient location. As an example, we will use ~/Repos directory:

```
> mkdir -p ~/Repos
> cd ~/Repos
```

We retrieve the repository

```
> git clone https://bitbucket.alma.cl/scm/acsws/awv.git
```

We make sure we have an INTROOT:

```
> echo $INTROOT
```

If this is empty, please make sure you it is defined and pointing to something useful. For instance ~/introot:

```
> export INTROOT=~/introot
```

Check that this directory exists and has the right directory structure:

```
> file $INTROOT
/home/almamgr/introot: directory
```

If the file doesn't exist, create the directory structure:

```
> getTemplateForDirectory INTROOT $INTROOT
```

Check the contents:

```
> ls -1 $INTROOT
ALARMS
app-defaults
bin
bitmaps
CDT
config
ERRORS
idl
include
lib
LOGS
man
responsible
rtai
sounds
Sources
templates
vw
```

Now, make sure this INTROOT variable gets automatically loaded to your environment, by adding it to your .bashrc

~/.bashrc

```
...
#End of original content

export INTROOT=~/introot
source /alma/ACS-2020JUN/ACSSW/config/.acs/.bash_profile.acs
```

This is to ensure, the libraries, executables and other files installed to the INTROOT are available for compilation and runtime. After this point close all your consoles and start with new ones in order for them to be properly configured.

Fill the INTROOT with the interface data

Go to the repository directory:

```
> cd ~/Repos/awv/ICD/src
> make all install
```

Check you have some key files in your INTROOT:

```
> ls -1 $INTROOT/lib/lib*Stubs.so
/home/almamgr/introot/lib/libCameraStubs.so
/home/almamgr/introot/lib/libConsoleStubs.so
/home/almamgr/introot/lib/libDataBaseStubs.so
/home/almamgr/introot/lib/libInstrumentStubs.so
/home/almamgr/introot/lib/libSchedulerStubs.so
/home/almamgr/introot/lib/libStorageStubs.so
/home/almamgr/introot/lib/libSYSTEMErrStubs.so
/home/almamgr/introot/lib/libTelescopeControlStubs.so
/home/almamgr/introot/lib/libTelescopeStubs.so
/home/almamgr/introot/lib/libTypesStubs.so

> ls -1 $INTROOT/lib/python/site-packages/*_idl.py
/home/almamgr/introot/lib/python/site-packages/Camera_idl.py
/home/almamgr/introot/lib/python/site-packages/Console_idl.py
/home/almamgr/introot/lib/python/site-packages/DataBase_idl.py
/home/almamgr/introot/lib/python/site-packages/Instrument_idl.py
/home/almamgr/introot/lib/python/site-packages/Scheduler_idl.py
/home/almamgr/introot/lib/python/site-packages/Storage_idl.py
/home/almamgr/introot/lib/python/site-packages/SYSTEMErr_idl.py
/home/almamgr/introot/lib/python/site-packages/TelescopeControl_idl.py
/home/almamgr/introot/lib/python/site-packages/Telescope_idl.py
/home/almamgr/introot/lib/python/site-packages/Types_idl.py

> ls -1 $INTROOT/lib/*.jar
/home/almamgr/introot/lib/Camera.jar
/home/almamgr/introot/lib/Console.jar
/home/almamgr/introot/lib/DataBase.jar
/home/almamgr/introot/lib/Instrument.jar
/home/almamgr/introot/lib/Scheduler.jar
/home/almamgr/introot/lib/Storage.jar
/home/almamgr/introot/lib/SYSTEMErr.jar
/home/almamgr/introot/lib/TelescopeControl.jar
/home/almamgr/introot/lib/Telescope.jar
/home/almamgr/introot/lib/Types.jar
```

Prepare your module

Go to the repository and add your module to the directory structure. For instance I will choose the following name for the Telescope implementation in Java:

```
> cd ~/Repos/awv
> getTemplateForDirectory MODROOT_WS jTelescope
```

Check the contents of the directory are correct:

```
> ls -1 jTelescope/
bin
ChangeLog
config
doc
idl
include
lib
LOGS
man
object
rtai
src
test
```

Right now the directory ~/Repos/awv should look something like:

```
> ls -1 ~/Repos/awv/
EXTERNAL
ICD
ITS
jTelescope
```

To put an example, once it is integrated with other groups' modules, it could see similar to:

```
> ls -1 ~/Repos/awv/
EXTERNAL
ICD
ITS
jTelescope
pyScheduling
cppInstruments
pyConsole
jDatabase
...
```

Bear in mind that there's some freedom with the name convention and these names are purely referential.

Pushing to Git

The person that was appointed in your group to work for ITS (Integration and Testing System) needs to upload your module (and during the workshop, the updates), to Git. First commit!

```
cd ~/Repos/awv
git pull
git add jTelescope/src/Makefile
git commit -m "First commit for jTelescope module"
git push
```

That way, other people in your team, and the integration team, can see the changes you've been working on during the workshop.

Testing Environment

You can prepare your testing environment to save some time later on:

```
> cd ~/Repos/awv/jTelescope/test
> cp ../src/Makefile .
> cp -r ~/Repos/awv/ITS/test/SIM/CDB CDB
```

This will give you an environment to tweak later on for testing your component's interaction with other components

Checking the testing environment

You can test everything is alright in the following way:

```
> export ACS_CDB=~/Repos/awv/jTelescope/test
> acsStart

> acsStartContainer -py pySimContainer

> objexp
```

Alternatively, you can use the ACS Command Center GUI, but make sure you define the ACS_CDB path correctly there.

While using objexp, retrieve some component(s) and check some of their methods. You should get random values for the methods you interact with.

Starting Component Development

The initial development of the components should have started, although no serious development was expected for the first day session. Basically the expectations were to have the files in place and maybe an initial prototype of the classes.

I will assume for this example that we have pyDatabase, jTelescope and cppInstrument modules, to show the file structure for each programming language:

Python

The following documents were provided to use as assistance:

- ACSPython.pdf
- ACS Workshop - Python Hands On

The expectation for day 2 were to have just the files:

```
> cd ~/Repos/awv/pyDatabase/src
> mkdir ws
> touch ws/db.py
```

And the Makefile configuration:

Makefile

```
...
PY_PACKAGES = ws
...
```

Java

The following documents were provided to use as assistance:

- ACS Java Component Tutorial
- ACS Workshop - Java Hands On

The expectation for day 2 were to have just the files:

```
> cd ~/Repos/awv/jTelescope/src
> mkdir -p workshop/telescope
> touch workshop/telescope/TelescopeImpl.java
> cp ~/Repos/awv/ICD/src/acsws/TELESCOPE_MODULE/TelescopeImpl/TelescopeComponentHelper.java.tpl workshop
/telescope/TelescopeComponentHelper.java
```

Modify the helper class slightly:

```
<-- package acsws.TELESCOPE_MODULE.TelescopeImpl;
--> package workshop.telescope;
...
<--- import acsws.TELESCOPE_MODULE.TelescopeImpl.TelescopeImpl;
--> import workshop.telescope.TelescopeImpl;
```

And the Makefile configuration:

Makefile

```
...
JARFILES = jTelescope
jTelescope_DIRS = workshop/telescope
...
```

C++

The following documents were provided to use as assistance:

- BACI Device Server Programming Tutorial
- ACS Workshop - C++ Hands On

The expectation for day 2 were to have just the files:

```
> cd ~/Repos/awv/cppInstrument/src
> touch ../include/InstrumentImpl.h
> touch InstrumentImpl.cpp
```

And the Makefile configuration:

Makefile

```
...
INCLUDES = InstrumentImpl.h
...
LIBRARIES = cppInstrumentImpl
cppInstrumentImpl_OBJECTS = InstrumentImpl
cppInstrumentImpl_LIBS = InstrumentStubs
...
```

ITS Tasks

At the end of all this work (it's not that little!). We would like for the ITS representative of each group to push their changes to git:

```
> cd ~/Repos/awv
> git pull
> git add jTelescope/test/Makefile
> git add jTelescope/test/CDB
> git add jTelescope/src/workshop/telescope/TelescopeImpl.java
> git add jTelescope/src/workshop/telescope/TelescopeComponentHelper.java
> git add jTelescope/src/Makefile


> git commit -m "Adding initial implementation and testing files for jTelescope module!"
> git push
```