

# Pipeline de registro de PET y MRI en Xnat

El objetivo de este pipeline es procesar conjuntamente dos imágenes MRI y PET de un mismo sujeto adquiridas en las fechas más próximas posibles, siempre a menos de 6 meses (dos experimentos Xnat, en el mismo proyecto), registrarlas, calcular algunas estadísticas y someterlas a un proceso de validación manual.

El proceso se ha separado en tres partes que se han integrado, finalmente, en un único pipeline:

1. preprocesamiento de los experimentos (emparejamiento de imágenes PET con las MRI más próximas del mismo sujeto y conversión de ambas a NIFTI)
2. ejecución del pipeline de registro y cálculo
3. generación de una imagen de previsualización del registro y un proceso *user friendly* de validación del resultado

## Ejemplo de uso

El pipeline que se ejecuta automáticamente se halla en:

```
/nas/data/xnat/pipeline/catalog/RegisterPETwithMRI/MatchPETwithMRI/registerPETwithMRImatch.xml
```

pero conviene definir también los otros pipelines: MatchPETwithMRI, DicomToNifti\_Y y RegisterPETwithMRI.

	Applies To	Generates	Description	Path
Delete	PET Sessions		Match PET experiment with the nearest MRI one of the same subject	/nas/data/xnat/pipeline/catalog/RegisterPETwithMRI/MatchPETwithMRI.xml
Delete	PET Sessions		PET - MRI matching, registration and QA pipeline	/nas/data/xnat/pipeline/catalog/RegisterPETwithMRI/registerPETwithMRImatch.xml
Delete	PET Sessions		PET - MRI registration and QA pipeline	/nas/data/xnat/pipeline/catalog/RegisterPETwithMRI/registerPETwithMRI.xml

## Instalación del pipeline: argumentos de entrada

En el proyecto, se añaden los tres pipelines de registro y el pipeline de conversión a formato NIFTI:

Details			Access			Manage			Pipelines		
<i>Pipelines for TestMatch1</i>											
			Applies To	Generates	Name	Description					
	<a href="#">Edit</a>	<a href="#">Details</a>	PET Sessions		MatchPETwithMRI	Match PET experiment with the nearest MRI one of the same subject					
	<a href="#">Edit</a>	<a href="#">Details</a>	PET Sessions		RegisterPETwithMRI	PET - MRI registration and QA pipeline					
	<a href="#">Edit</a>	<a href="#">Details</a>	PET Sessions		RegisterPETwithMRImatch	PET - MRI matching, registration and QA pipeline					
	<a href="#">Edit</a>	<a href="#">Details</a>	All Datatypes		DicomToNifti_Y	Pipeline creates NIFTI files from DICOM files using dcm2nix (modified).					
<a href="#">Add More Pipelines</a>											

Pero sólo se pide que se arranque automáticamente tras la carga de imágenes RegisterPETwithMRImatch, que es el que llama a los otros en el orden adecuado.

PET - MRI matching, registration and QA pipeline

### Pipeline setup

Launch pipeline automatically when session is archived?

#### Project specific pipeline parameters:

Note: Parameters that take multiple values should be comma-separated

Name	Values	
scanIDs	xnat:petSessionData/scans/scan/ID	<a href="#">Help</a>
sessionID	xnat:petSessionData/ID	<a href="#">Help</a>
sessionName	xnat:petSessionData/label	<a href="#">Help</a>
projectID	xnat:petSessionData/project	<a href="#">Help</a>
subjectID	xnat:imageSessionData/subject_ID	<a href="#">Help</a>
sessionDate	xnat:imageSessionData/date	<a href="#">Help</a>
dcmT1tag	<input type="text" value="_tf13d1_16ns"/>	<a href="#">Help</a>
dcmFBBtag	<input type="text" value="_20min"/>	<a href="#">Help</a>

Las opciones dcmT1tag y dcmFBBtag indican, respectivamente, patrones en los atributos SequenceName de la secuencia MRI y ProtocolName de la secuencia PET. Pueden variar de un proyecto a otro.

**Nota:** Es necesario marcar la opción "Launch pipeline automatically when session is archived?" al añadir el pipeline; esta opción, aunque se editen luego las opciones del pipeline, no se guarda.

## Carga de los archivos MRI y PET: ejecución automática del pipeline

Dado que el pipeline RegisterPETwithMRI se ejecuta automáticamente al cargar un estudio PET, es imprescindible que antes se haya cargado el estudio MRI correspondiente al mismo sujeto, con el que se quiere emparejar.

**Nota:** Si se usa la utilidad xnatapic upload\_dicom, se debe indicar explícitamente que se ejecuten los pipelines automáticos al completar la carga de los estudios PET. P.ej.

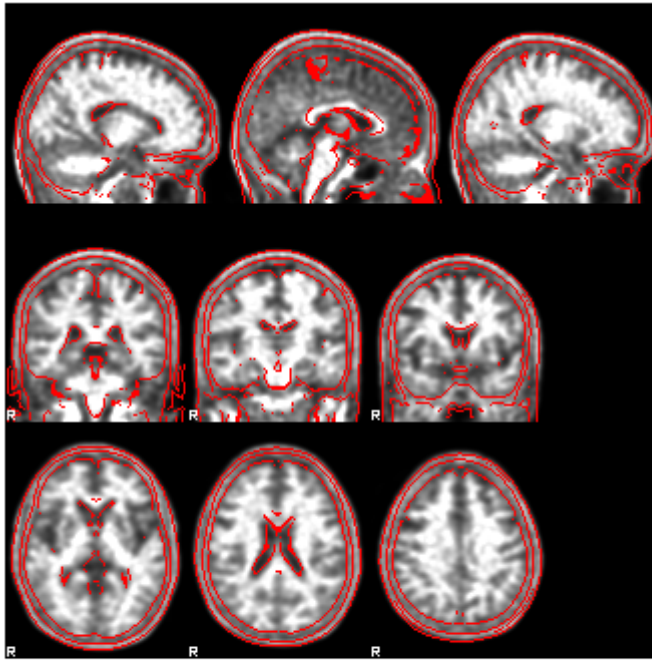
```
xnatapic upload_dicom --project_id TestMatch1 --subject_id FACEHBI_001 --  
pipelines facehbi/FACEHBI-F001B
```

## Notificación por e-mail y enlace a la validación

El proceso de cada estudio no es muy largo: unos 20 minutos. Al finalizar (si todo ha transcurrido sin errores), le llegará al usuario un correo electrónico como el que sigue:

Dear A.Admin,  
The PET - MRI registration pipeline completed without errors.  
Details for this analysis are available at [the XNAT website](#).  
You can perform the manual QA by clicking [here](#).  
XNAT Team.

El primer enlace sólo es un log de las operaciones. El segundo enlace, abre una ventana donde se muestra el resultado del registro. El usuario debe decidir si es correcto o no, y esa decisión se guardará en Xnat junto con el resto de información de registro y estadísticas.



XNAT5\_S00037\_fbb\_mni

Q&A success

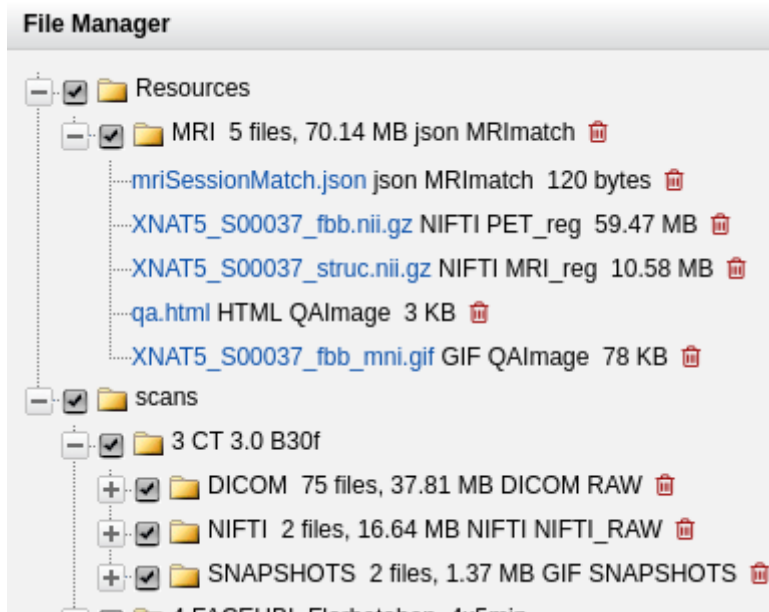
Q&A error

Quality Assessment

## Recursos generados

Como resultado del pipeline, en cada estudio PET procesado se crea una carpeta de recursos llamada MRI. En ella se guarda la siguiente información:

- un archivo JSON, llamado `mriSessionMatch.json`, con resultados del emparejamiento PET-MRI, estadísticas sobre el registro (SURV y CL) y sobre la evaluación de la calidad del mismo (QA)
- los archivos NIFTI, `_fbb` y `_struc`, generados durante el registro
- un archivo HTML llamado `qa.html` y una imagen GIF (obtenida superponiendo los bordes del `_struc` sobre la imagen `_fbb`) que son con los que se lleva a cabo la evaluación de calidad del apartado anterior



## Generación del informe

El script `get_registration_report` de la `xnatapic` genera un archivo CSV que resume estos valores para cada PET de cada individuo. Se debe invocar como:

```
xnatapic get_registration_report --project_id TestMatch1 --output report.csv
```

El archivo CSV de salida contiene las siguientes columnas:

- DCM\_SUBJECT: código DICOM del paciente en el estudio PET
- SUBJECT\_ID: código Xnat del paciente
- PET\_EXPERIMENT\_ID: código Xnat del estudio PET
- MRI\_EXPERIMENT\_ID: código Xnat del estudio MRI
- REGISTRATION\_QA: 0 si no se hizo o si no se consideró bueno el registro; 1 si se consideró bueno el registro
- STATISTICS\_SURV: estadístico SURV calculado al comparar el PET con la referencia
- STATISTICS\_CL: estadístico CL calculado al comparar el PET con la referencia

	A	B	C	D	E	F	G
	DCM_SUBJECT	SUBJECT_ID	PET_EXPERIMENT_ID	MRI_EXPERIMENT_ID	REGISTRATION_QA	STATISTICS_SURV	STATISTICS_CL
1	F001B	XNAT5_S00037	XNAT5_E00069	XNAT5_E00068	0	0,969622311660773	-5,25993739123739
2	F002B	XNAT5_S00038	XNAT5_E00071	XNAT5_E00070	1	1,08412209991661	12,3043301272075
3	F004B	XNAT5_S00040	XNAT5_E00074	XNAT5_E00073	1	1,07553072058275	10,9864125373941
4	F005B	XNAT5_S00041	XNAT5_E00076	XNAT5_E00075	0	0,95634102358525	-7,29728698202267
5	F006B	XNAT5_S00042	XNAT5_E00078	XNAT5_E00077	1	1,25395796509651	38,3571518458042
6	F007B	XNAT5_S00043	XNAT5_E00080	XNAT5_E00079	1	1,50605395337085	77,0286764470885
7	F008B	XNAT5_S00044	XNAT5_E00082	XNAT5_E00081	0	0,898553726266819	-16,16185839067
8	F009B	XNAT5_S00045	XNAT5_E00084	XNAT5_E00083	0	1,03051155722437	4,08047287821763
10	F003B	XNAT5_S00039	XNAT5_E00072	xnat_mrsessiondata0			
11	F010B	XNAT5_S00046	XNAT5_E00086	XNAT5_E00085	1	1,61738491378125	94,106845774043

**Nota:** Cuando un estudio PET no ha podido emparejarse con uno MRI, el usuario recibe un mensaje de error. Entonces, en el archivo CSV la columna de REGISTRATION\_QA aparece vacía (y en MRI\_EXPERIMENT\_ID un valor que no corresponde a ningún experimento).

## Detalle de la implementación

La implementación de RegisterPETwithMRI se ha hecho como un wrapper que ejecuta en orden tres pipelets independientes (esto es, que también se pueden ejecutar cada uno sobre un estudio PET, si se asume que ya ha sido preprocesado por los anteriores pipelines de la cadena).

### RegisterPETwithMRI: llamadas a los pipelets

El pipeline distribuye los datos en tres directorios en los que se ejecutarán procesos diferentes: el de datos de PET (pet), el de datos de MRI (anat) y el de registro (reg).

El registro, se ejecutará con el pipeline de emparejamiento MatchPETwithMRI. Éste, generará la primera versión del archivo JSON. Es la llamada al pipelet más sencilla, que tiene este código:

```
<step id="RUNMATCHPETWITHMRI" description="Finds the MRI matching this PET session" continueOnFailure="false">
  <pipelet location="RegisterPETwithMRI" name="MatchPETwithMRI">
    <parameters>
      <name>callerworkdir</name>
    <values><unique>~/Pipeline/parameters/parameter[name='regDir']/values/unique/text()^</unique></values>
      <description>child workdir</description>
    </parameters>
  </pipelet>
</step>
```

**Nota:** La forma de declarar un pipelet no está correctamente documentada en el [Xnat Pipeline Development Schema](#), donde parece indicarse que el bloque <parameters> tiene la misma estructura que los otros bloques del mismo tipo en el XML del pipeline; no es así, aquí, cada <parameters> actúa, en realidad, como cada elemento <parameter> de las otras secciones. Es posible que esto sea un error que corrijan en futuras versiones del *pipeline engine*.

El parámetro callerworkdir es el que se pasa a los pipelets para que construyan con él el directorio de trabajo; si no se les pasa este argumento (p.ej. si son llamados como pipelines independientes), entonces crean su propio directorio de trabajo.

**Nota:** callerworkdir no es un estándar, sino una decisión de diseño de este conjunto de pipelines.

El resultado del pipeline anterior es un archivo JSON (que también es cargado como recurso de la sesión PET en Xnat) con una la estructura similar a las respuestas de los servicios de Xnat, para poder hacer uso después de este mecanismo de consulta; p.ej.

```
{ "ResultSet": {
  "Result": [
    { "MRIsession": "XNAT03_E00005",
      "_": ""
    }
  ]
}
```

```
}
}
```

La conversión de DICOM a NIFTI se lleva a cabo ejecutando el pipeline `DicomToNifti_Y` dos veces: una, sobre el estudio PET, y otra sobre el estudio MRI identificado en el pipeline anterior. Al ejecutarse como pipelet, los archivos NIFTI se crean en los subdirectorios `pet/NIFTI` y `anat/NIFTI` (`pet` y `anat` son pasados como `callerworkdir`).

La llamada a `DicomToNifti_Y` sobre el estudio MRI se hace pasando un `id` obtenido mediante una llamada a `fileUtils:GetColumn`, una utilidad que permite extraer valores de respuestas JSON a consultas hechas a la API de Xnat en tiempo de ejecución del pipeline. De igual modo se pasan los identificadores de las imágenes del estudio, o `scanids`. En particular, en este segundo pipelet, se pasan los parámetros mediante este código:

```
<parameters>
  <name>id</name>
  <values><unique>^fileUtils:GetColumn(
/Pipeline/parameters/parameter[name='host']/values/unique/text(),
/Pipeline/parameters/parameter[name='user']/values/unique/text(),
/Pipeline/parameters/parameter[name='pwd']/values/unique/text(),
concat('data/experiments/',/Pipeline/parameters/parameter[name='sessionID']/
values/unique/text(),'/resources/MRI/files/mriSessionMatch.json'),
'MRIsession')^</unique></values>
  <description>MRI session ID</description>
</parameters>
<parameters>
  <name>scanids</name>
  <values><list>^fileUtils:GetColumnList(
/Pipeline/parameters/parameter[name='host']/values/unique/text(),
/Pipeline/parameters/parameter[name='user']/values/unique/text(),
/Pipeline/parameters/parameter[name='pwd']/values/unique/text(),
concat('data/experiments/',
fileUtils:GetColumn(/Pipeline/parameters/parameter[name='host']/values/unique/
text(), /Pipeline/parameters/parameter[name='user']/values/unique/text(),
/Pipeline/parameters/parameter[name='pwd']/values/unique/text(),
concat('data/experiments/',/Pipeline/parameters/parameter[name='sessionID']/
values/unique/text(),'/resources/MRI/files/mriSessionMatch.json'),
'MRIsession'), '/scans?format=JSON'), 'ID')^</list></values>
  <description>MRI scan IDs</description>
</parameters>
```

**Nota:** Los pipelines de Xnat pueden usar llamadas a funciones Java del `/pipeline engine/` para evaluar variables y expresiones y asignar valores a parámetros. En particular, `fileUtils:getColumn` y `fileUtils:GetColumnList`, permiten obtener valores de un campo o de una lista de valores de un `/ResultSet/` como el de arriba:

```
fileUtils:GetColumn(HOST, USER, PASSWORD, PATH, COLUMN)
fileUtils:GetColumnList(HOST, USER, PASSWORD, PATH, COLUMN)
```

Más detalles en el código fuente de [FileUtils](#).



**Nota:** La diferencia entre `fileUtils::GetColumn` y `fileUtils::GetColumnList` es que la primera devuelve un valor único (de ahí <unique>) y la segunda devuelve una lista (<list>). Estas listas, como las <csv> (valores separados por comas incluidos en el propio XML), se manejan en los pipelines mediante iteradores <loop>.

El último pipeline llamado es `RegisterPETwithMRI`. Se trata de una llamada semejante a la de `MatchPETwithMRI`, pasando el `callerworkdir` que, en este caso, es el mismo directorio de trabajo del pipeline `RegisterPETwithMRI`.

## MatchPETwithMRI: emparejado PET-MRI

El pipeline usa el script `matchPETwithMRI.sh` (representado mediante el resource `matchPETwithMRI.xml`).

Primero, hace la consulta a Xnat sobre los experimentos de modalidad (xsiType) `xnat:mrSessionData` que se encuentran del mismo sujeto dentro del proyecto. Esto se guarda como un archivo CSV (`mriSessions.csv`).

Después, ejecuta el script `matchPETwithMRI.sh` (representado por el recurso del mismo nombre), llamándolo como

```
matchPETwithMRI.sh --PETdate $DATE --MRIrecords mriSessions.csv --
outputMatch mriSessionMatch.json
```

donde `$DATE` es sustituida por el valor de la fecha de la sesión de PET (dato que Xnat guarda como `xnat:imageSessionData/date`).

[El código de este script se puede ver a continuación.](#)

### [matchPETwithMRI.sh](#)

```
#!/bin/bash -l

ARGS=( "$@" )

#parse options
HOST=""
USER=""
PASSWORD=""
PROJECT=""
SUBJECT=""
PET_EXPERIMENT=""
MATCH_OUT=""
for ((n=0; n<${#ARGS[@]}; n++)) ; do
  case "${ARGS[$n]}" in
    --project)
      let n=n+1
```



```

    PROJECT="${ARGS[$n]}"
    ;;
--subject)
    let n=n+1
    SUBJECT="${ARGS[$n]}"
    ;;
--PETdate)
    let n=n+1
    PET_DATE="${ARGS[$n]}"
    ;;
--MRIrecords)
    let n=n+1
    MRI_RECORDS="${ARGS[$n]}"
    ;;
--outputMatch)
    let n=n+1
    MATCH_OUT="${ARGS[$n]}"
    ;;
--version)
    echo "RegisterPETwithMRI: matchPETwithMRI.sh"
    echo "v20201208"
    exit 0
    ;;
--help)
    echo "--help          show help"
    echo "--PETdate         reference PET experiment"
    echo "--MRIrecords      MRI experiment records in CSV format"
    echo "--outputMatch     file with the matching experiment"
    echo "--version         script version"
    exit 0
    ;;
esac
done

if [ -z "$PET_DATE" ] || [ -z "$MRI_RECORDS" ] || [ -z "$MATCH_OUT" ] ;
then
    echo "Error: not enough parameters; use --help" >&2
    exit 1
fi

#helper function to read CSV values
function getCSVfield() {
    echo "$1" | sed 's/,/\n/g' | (
        n=1
        while read l ; do
            if [ $n == "$2" ] ; then echo "$l" ; fi
            let n=n+1
        done
    )
}

```

```
#PET date in seconds
PET_UPDATE=$( date -d "$PET_DATE" +%s )

#list of MRI experiments
cat "$MRI_RECORDS" | grep -v "URI$" | (

  #choose closest MRI image in time
  MRI_EXPER=""
  DT_MIN=0
  while read MRI_RECORD ; do
    MRI_DATE=$( getCSVfield "$MRI_RECORD" 4 )
    MRI_UPDATE=$( date -d "$MRI_DATE" +%s )
    DT=$(echo "$(( "$MRI_UPDATE" - "$PET_UPDATE" ))" | sed 's/^-//')
    if [ -z "$MRI_EXPER" ] || [ $DT -lt $DT_MIN ] ; then
      MRI_EXPER=$( getCSVfield "$MRI_RECORD" 1 )
      DT_MIN=$DT
    fi
  done

  if [ $DT_MIN -gt $((3600*24*30*6)) ] ; then
    MRI_EXPER=""
  fi

  #write MRI experiment name to file
  if ! [ -z "$MRI_EXPER" ] ; then
    (
      echo '{"ResultSet":{"Result":[{"
      printf '"MRIsession": "%s",\n' "$MRI_EXPER"
      echo '"_": ""}]}}'
    ) > "$MATCH_OUT"
  fi
)
```

El archivo JSON generado, `mriSessionMatch.json`, es después cargado dentro del recurso MRI (que es creado, si no existía previamente) de la sesión PET, p.ej. en

[http://localhost/data/experiments/XNAT03\\_E00008/resources/MRI/files/mriSessionMatch.json](http://localhost/data/experiments/XNAT03_E00008/resources/MRI/files/mriSessionMatch.json)

## DicomToNifti\_Y: conversión con dcm2niix

Este pipeline es una adaptación del `DicomToNifti_X` que se encuentra en `dcm2niix`. La adaptación consiste en:

- \* usar un formato de salida en los nombres de los ficheros NII distinto del formato por defecto (en particular, se usa: `%n_%s_%d`).
- \* usar el argumento `/ignore derived/ (-i)` [para esto se modificó el resource `dcm2niix`]
- \* recibir como parámetro el directorio de trabajo (`callerworkdir`), cuando es

llamado como pipelet.

**Nota:** previamente, ya se había modificado el recurso que llama a `dcm2niix` para incluir la opción `-i`.

## RegisterPETwithMRI: registro

Si este pipeline no recibe el parámetro `callerworkdir`, crea los tres directorios, `pet`, `anat` y `reg`, y descarga en los dos primeros las correspondientes series de imágenes de las sesiones, en formato NIFTI, y en el segundo directorio descarga el archivo `mriSessionMatch.json` de los recursos de la sesión PET. En caso de recibir el `callerworkdir`, es que esos datos ya existían en esos directorios y no hace falta volver a descargarlos.

El pipeline ejecuta tres scripts: `linkNIIwithTagValue.sh`, `registerPETwithMRI.sh` y `mkSlicesQAhtm.sh`.

### linkNIIwithTagValue.sh

Este script busca dentro del directorio de trabajo los archivos JSON asociados a archivos NIFTI que contienen un valor particular de un tag DICOM (es decir, un par clave-valor en el JSON).

[El código del script se puede ver a continuación.](#)

#### [linkNIIwithTagValue.sh](#)

```
#!/bin/bash

ARGS=( "$@" )

#parse options
DCMTAG=""
DCMVALUE=""
LINKBASE=""
for ((n=0; n<${#ARGS[@]}; n++)) ; do
  case "${ARGS[$n]}" in
    --tag)
      let n=n+1
      DCMTAG="${ARGS[$n]}"
      ;;
    --value)
      let n=n+1
      DCMVALUE="${ARGS[$n]}"
      ;;
    --link)
      let n=n+1
      LINKBASE="${ARGS[$n]}"
      ;;
    --version)

```

```
    echo "RegisterPETwithMRI: matchPETwithMRI.sh"
    echo "v20201208"
    exit 0
;;
--help)
    echo "--help    show help"
    echo "--tag     DICOM tag in JSON"
    echo "--value   value of DICOM tag"
    echo "--link    base name of the linked file (for NII and
JSON)"
    echo "--version script version"
    exit 0
;;
esac
done

if [ -z "$DCMTAG" ] || [ -z "$DCMVALUE" ] || [ -z "$LINKBASE" ] ; then
    echo "Error: not enough parameters; use --help" >&2
    exit 1
fi

find . -name '*.nii*' | while read fn ; do
    fnj="$(echo "$fn" | sed 's/\.nii.*$/\.json/')"
    if grep "\"$DCMTAG\"" "$fnj" | grep -q ".*$DCMVALUE" ; then
        rm -f "$LINKBASE".$EXT "$LINKBASE".json

        EXT=$(echo "$fn" | sed 's/^\.*\.(nii.*$)/\1/')
        ln -s "$fn" "$LINKBASE.$EXT"
        ln -s "$fnj" "$LINKBASE.json"

    fi
done

if ! [ -e "$LINKBASE.json" ] ; then
    exit -1
fi
```

El script se llamaría desde la línea de comandos de esta forma:

```
linkNIIwithTagValue.sh --tag SequenceName --value _tfl3d1_16ns --link sub-
XNAT5_S00037_T1w
```

El argumento `--link` indica el nombre de los enlaces suaves que se crearán en el directorio de trabajo apuntando a los archivos correspondientes (uno con extensión `nii.gz` y otro con extensión `.json`).

## registerPETwithMRI.sh

Este script usa las herramientas de la *FSL* y de *ANTS* para registrar las imágenes cuyos enlaces se crearon en el script anterior. Después, calcula dos valores estadísticos (usando la utilidad *fslstats*) comparando la imagen PET-MRI registrada con los atlas de referencia

*Centiloid\_Std\_VOI/nifti/2mm/voi\_ctx\_2mm.nii* y

*Centiloid\_Std\_VOI/nifti/2mm/voi\_WhlCbl\_2mm.nii*. Estos valores estadísticos los añade a un archivo JSON con la estructura del *mriSessionMatch.json* (si no existe, lo crea).

El código del script se puede ver a continuación.

### registerPETwithMRI.sh

```
#!/bin/bash -l

ARGS=( "$@" )

if [ -z "$PIPEDIR" ] ; then
    export PIPEDIR=/nas/software/neuro.dev
fi

if [ -z "$FSLDIR" ] ; then
    export FSLDIR="/usr/local/fsl"
    source ${FSLDIR}/etc/fslconf/fsl.sh
fi

if [ -z "$ANTS_PATH" ] ; then
    export ANTS_PATH="/nas/usr/local/opt/bin/ants/bin"
    export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/nas/usr/local/opt/bin/ants/lib
fi

#parse options
STUDY=""
SID=""
WDIR=""
PET=""
MRI=""
for ((n=0; n<${#ARGS[@]}; n++)) ; do
    case "${ARGS[$n]}" in
        --study)
            let n=n+1
            STUDY="${ARGS[$n]}"
            ;;
        --sid)
            let n=n+1
            SID="${ARGS[$n]}"
            ;;
        --wdir)
    
```

```
    let n=n+1
    WDIR="${ARGS[$n]}"
    ;;
--pet)
    let n=n+1
    PET="${ARGS[$n]}"
    ;;
--mri)
    let n=n+1
    MRI="${ARGS[$n]}"
    ;;
--out)
    let n=n+1
    OUT="${ARGS[$n]}"
    ;;
--version)
    echo "RegisterPETwithMRI: registerPETwithMRI.sh"
    echo "v20201208"
    exit 0
    ;;
--help)
    echo "--help      show help"
    echo "--study      study name (not used)"
    echo "--sid        subject ID"
    echo "--wdir       working directory"
    echo "--pet        PET NII file"
    echo "--mri        MRI T1 NII file"
    echo "--out        output"
    echo "--version    script version"
    exit 0
    ;;
esac
done

if [ -z "$SID" ] || [ -z "$WDIR" ] || [ -z "$PET" ] || [ -z "$MRI" ] || [ -z "$OUT" ] ; then
    echo "Error: not enough parameters; use --help" >&2
    exit 1
fi

${FSLDIR}/bin/fslreorient2std ${MRI} ${WDIR}/${SID}_struc

${ANTS_PATH}/antsRegistrationSynQuick.sh -d 3 -f
${WDIR}/${SID}_struc.nii.gz -m ${PET} -t s -o
${WDIR}/${SID}_movingToFixed_

${ANTS_PATH}/antsApplyTransforms -d 3 -r ${WDIR}/${SID}_struc.nii.gz -i
${PET} -t ${WDIR}/${SID}_movingToFixed_0GenericAffine.mat -t
```

```

${WDIR}/${SID}_movingToFixed_1Warp.nii.gz -o ${WDIR}/${SID}_fbb.nii.gz

${ANTS_PATH}/ANTS 3 -m CC[${FSLDIR}/data/standard/MNI152_T1_2mm.nii.gz,
${WDIR}/${SID}_struc.nii.gz, 1, 4] -r Gauss[0,3] -t Elast[1.5] -i
30x20x10 -o ${WDIR}/${SID}_fbb_t1_mni.nii.gz

${ANTS_PATH}/WarpImageMultiTransform 3 ${WDIR}/${SID}_fbb.nii.gz
${WDIR}/${SID}_fbb_mni.nii.gz -R
${FSLDIR}/data/standard/MNI152_T1_2mm.nii.gz
${WDIR}/${SID}_fbb_t1_mniWarp.nii.gz
${WDIR}/${SID}_fbb_t1_mniAffine.txt

CTX=$( ${FSLDIR}/bin/fslstats ${WDIR}/${SID}_fbb_mni.nii.gz -k
$PIPEDIR/lib/Centiloid_Std_VOI/nifti/2mm/voi_ctx_2mm.nii -M )
NORM=$( ${FSLDIR}/bin/fslstats ${WDIR}/${SID}_fbb_mni.nii.gz -k
$PIPEDIR/lib/Centiloid_Std_VOI/nifti/2mm/voi_WhlCbl_2mm.nii -M )

SURV=$(bc -l <<< "$CTX/$NORM")
CL=$(bc -l <<< "153.4 * $SURV - 154.0")
if [ -s "$OUT" ] ; then
    jq -c ".ResultSet.Result[0].qa=0 | .ResultSet.Result[0].surv=$SURV
| .ResultSet.Result[0].cl=$CL" ${OUT} > ${OUT}~
    mv ${OUT}~ ${OUT}
else
(
    cat <<.EOF
{"ResultSet":{"Result":[{"qa":0, "surv":$SURV, "cl":$CL}]}}
.EOF
) > "$OUT"
fi

```

El script se llamaría desde la línea de comandos como sigue:

```

registerPETwithMRI.sh --sid XNAT5_S00037 --wdir ./reg/ --pet ./pet/sub-
XNAT5_S00037_single_fbb.nii.gz --mri ./anat/sub-XNAT5_S00037_T1w.nii.gz --
out ./reg/mriSessionMatch.json

```

**Nota:** Para que se pueda ejecutar el script, además de ser visibles las bibliotecas de Freesurfer y de ANT, debe estar instalado en todos los nodos del cluster el programa jq [yum install jq].

**Nota:** para que el script pueda acceder a las imágenes de referencia en /nas/software/neuro.dev, el usuario xnat debe tener permisos de lectura en él.

## mkSlicesQAhtm.sh

Este script genera una página HTML interactiva con la que hacer la comprobación visual del registro. Utiliza la herramienta slices de la FSL que crea un archivo GIF con diferentes vistas del registro. La página HTML muestra esa imagen y, mediante llamadas AJAX a la API de Xnat, modifica el valor del



parámetro qa en el archivo `mriSessionMatch.json` del recurso MRI del estudio PET. En particular, el código Javascript para hacer esta modificación es:

```
//update registration status
regStatus.ResultSet.Result[0].qa=( $("input#QAssessOk").checked )?1:0;
fetch("mriSessionMatch.json?XNAT_CSRF="+csrfToken+"&overwrite=true&inbody=true&format=json&content=MRImatch", {
  "method": "PUT",
  "mode": "same-origin",
  "credentials": "same-origin",
  "headers": { 'Content-type': 'application/json; charset=UTF-8' },
  "body": JSON.stringify(regStatus) }
).then( ... );
```

**Nota:** Para interactuar con Xnat a través de una página web, se debe obtener un *token* CSRF (*Cross-Site Request Forgery*), que es un mecanismo de seguridad para detectar ataques. Este *token* se encuentra fácilmente en el código HTML de las páginas devueltas por el servidor Xnat (siempre que el usuario se haya identificado previamente en él) y hay que obtenerlo previamente; p.ej. descargando y reconociendo su patrón (`csrfToken=`) en la página de entrada del servidor Xnat.

El código completo del script se puede ver a continuación.

`mkSlicesQAhtm.sh`

```
#!/bin/bash -l

ARGS=( "$@" )

if [ -z "$FSLDIR" ] ; then
  export FSLDIR="/usr/local/fsl"
  source ${FSLDIR}/etc/fslconf/fsl.sh
fi

#parse options
SUBJECT_ID=""
DIRBASE=""
XNATBASE=""
for ((n=0; n<${#ARGS[@]}; n++)) ; do
  case "${ARGS[$n]}" in
    --sid)
      let n=n+1
      SUBJECT_ID="${ARGS[$n]}"
      ;;
    --wdir)
      let n=n+1
      DIRBASE="${ARGS[$n]}"
      ;;
  esac
done
```

```

--version)
    echo "RegisterPETwithMRI: mkSlicesQAhtm.sh"
    echo "v20201208"
    exit 0
    ;;
--help)
    echo "--help    show help"
    echo "--sub     subject ID"
    echo "--dir     slicedir generated directory"
    echo "--version script version"
    exit 0
    ;;
esac
done

if [ -z "$DIRBASE" ] || [ -z "$SUBJECT_ID" ] ; then
    echo "Error: not enough parameters; use --help" >&2
    exit 1
fi

#QA with slices
qaGIF=${SUBJECT_ID}_fbb_mni.gif
${FSLDIR}/bin/slices ${DIRBASE}/${SUBJECT_ID}_fbb_mni
${FSLDIR}/data/standard/MNI152_T1_2mm -o ${DIRBASE}/${qaGIF}

(
cat <<.EOF
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8" />
<title>Q&A $SUBJECT_ID</title>
<script>
window.onload=function () {
    function \$(q) { return document.querySelector(q); }
    function \$$ (q) { return document.querySelectorAll(q); }

    //download registration status and parse it
    var csrfToken=null;
    var regStatus=null;
    //hack to get Xnat csrfToken
    fetch("/").then(
        function(response) {
            if( response.ok )
                return response.text();
            throw "Error: "+response.statusText;
        }
    ).then(
        function(text) {
            var a=text.match(/csrfToken *= *'[^']+/);
            if( a ) {

```

```
        csrfToken=a[0].replace(/csrfToken *= *'/, "");
    }
    else
        throw "Error: "+"No CSRF token";
    }
);

//get MRI session registration info
fetch("mriSessionMatch.json").then(
    function(response) {
        if( response.ok )
            return response.json();
        throw "Error: "+response.statusText;
    }
).then(
    function(jresp) {
        //set input#QAssess accordingly
        if( jresp.ResultSet.Result[0].qa )
            \$("#input#QAssessOk").checked=true;
        else {
            jresp.ResultSet.Result[0].qa=0;
            \$("#input#QAssessErr").checked=true;
        }
        regStatus=jresp;
        \$("#input#QAssessBtn").disabled=false;
    }
);

//get images from Xnat server (served as binary streams?)
var imgs=\$\$("img[data-src]");
for(var i=0; i < imgs.length; i++) {
    (function (img) {
        fetch(img.getAttribute('data-src'))
        ).then(
            function(response) {
                if( response.ok )
                    return response.blob();
                throw "Error: "+response.statusText;
            }
        ).then(
            function(blob) {
                img.src=URL.createObjectURL(blob);
            }
        ).catch(
            function(e) {
                alert(e);
            }
        );
    })(imgs[i]);
}
```

```

    }

    \$("#input#QAssessBtn").onclick=function () {
        if( confirm("Please confirm:\nAre you sure PET registration is
"+(\$("#input#QAssessOk").checked?'correct':'wrong')+?"") ) {
            //update registration status
            regStatus.ResultSet.Result[0].qa=(
\$("#input#QAssessOk").checked )?1:0;
            fetch("mriSessionMatch.json?XNAT_CSRF="+csrfToken+"&overwrite=true&inbo
dy=true&format=json&content=MRImatch", {
                "method": "PUT",
                "mode": "same-origin",
                "credentials": "same-origin",
                "headers": { 'Content-type': 'application/json;
charset=UTF-8' },
                "body": JSON.stringify(regStatus) }
            ).then(
                function(response) {
                    if( ! response.ok )
                        throw "Error: "+response.statusText;
                }
            ).catch(
                function(e) {
                    alert(e);
                }
            );
        }
    };
}
</script>
</head>
<body>
.EOF

cat <<.EOF
<div id="$qaGIF"> $(echo "$qaGIF"
| sed 's/\.gif.*$//')</div>
<br />
.EOF

cat <<.EOF
<hr />
<form id="QAform" name="QAform">
<input type="radio" id="QAssessOk" name="QAssess" value="Ok" default
/>Q&A success
&nbsp;
<input type="radio" id="QAssessErr" name="QAssess" value="Err" />Q&A
error
&nbsp;
<input type="button" id="QAssessBtn" name="QAssessBtn" value="Quality
Assessment" disabled />

```

```
</form>
</body>
</html>
.EOF

) >>"$DIRBASE/qa.html"
```

La llamada al script desde la línea de comandos se haría como sigue:

```
mkSlicesQAhtm.sh --sid XNAT5_S00037 --wdir ./reg/
```

Como se ve, no necesita más información que el id del sujeto, porque el acceso al resto de la información se hará con rutas relativas a la carpeta de recursos MRI.

## Script para generación de informe final

Para facilitar el acceso a la información, se ha añadido a la `xnatapic` una herramienta de consulta `get_registration_report`, que genera un archivo CSV con los valores relevantes de los archivos `mriSessionMatch.json` de todas las sesiones PET de un proyecto.

El código completo de este script (que se debe copiar o bien en el directorio `share/xnatapic` de la instalación de `xnatapic`, o bien en el directorio `.xnatapic` del usuario), se puede ver a continuación.

`get_registration_report.sh`

```
#!/bin/bash

ARGS=( "$@" )
HELP="get_registration_report - gets PET registration results archived
in Xnat"

#global variables
[ -s "$XNATAPIC_APPS/xnat.conf" ] && . "$XNATAPIC_APPS/xnat.conf"
[ -s "$XNATAPIC_HOME_APPS/xnat.conf" ] && .
"$XNATAPIC_HOME_APPS/xnat.conf"

#local variables
PROJ_ID=""
OUTPUT=""

#parse arguments
for ((n=0; n<${#ARGS[@]}; n++)) ; do
    case "${ARGS[$n]}" in
        --help-short)
            echo "$HELP"
    esac
done
```

```

        exit 0
        ;;
    --help)
        echo "$HELP"
        cat <<.EOF
    --help: show this help
    --project_id: project (only required if experiment_id is a label)
    --output: name of the CSV file (defaults to stdout)
    --version: script version
.EOF
        exit 0
        ;;
    --version)
        echo "xnatapic get_fsresults.sh"
        echo "v20201208"
        exit 0;
        ;;
    --project_id)
        let n=n+1
        PROJECT_ID="${ARGS[$n]}"
        ;;
    --output)
        let n=n+1
        EXPERIMENT_ID="${ARGS[$n]}"
        ;;
    -*)
        echo "Warning: ignoring option or command ${ARGS[$n]}" >&2
        ;;
    esac
done

#experiment route
[ -z "$PROJECT_ID" ] && echo "Error: project_id is required" >&2
URIpath="/data/projects/$PROJECT_ID/experiments"

if [ -z "$OUTPUT" ]; then
    echo
    "DCM_SUBJECT,SUBJECT_ID,PET_EXPERIMENT_ID,MRI_EXPERIMENT_ID,REGISTRATIO
    N_QA,STATISTICS_SURV,STATISTICS_CL"
else
    echo
    "DCM_SUBJECT,SUBJECT_ID,PET_EXPERIMENT_ID,MRI_EXPERIMENT_ID,REGISTRATIO
    N_QA,STATISTICS_SURV,STATISTICS_CL" > "$OUTPUT"
fi

curl -f -X GET -u "$USER:$PASSWORD"
"$HOST$URIpath/?format=csv&modality=petSessionData" 2>/dev/null |\
grep 'xnat:petSessionData' |\
cut -d, -f 2 |\
while read PET_EXPERIMENT_ID ; do
    URIpath="/data/experiments/$PET_EXPERIMENT_ID"

```

```
DCM_SUBJECT_ID=""
SUBJECT_ID=""
MRI_EXPERIMENT_ID=""
REGISTRATION_QA=""
STATISTICS_SURV=""
STATISTICS_CL=""

curl -f -X GET -u $USER:$PASSWORD "$HOST$URIpath/?format=json"
2>/dev/null | \
sed 's/\[{\|}\|\\|,|\/\n/g' | grep '^"[a-zA-Z_]\+":' | (
  while read kv ; do
    k="$(echo "$kv" | sed 's/\s*:.*/' | sed 's//g')"
```

```
    v="$(echo "$kv" | sed 's/^\s*:\s*/' | sed 's//g')"
```

```
    #echo "K=$k, V=$v"
```

```
    case $k in
      dcmPatientId) DCM_SUBJECT_ID="$v" ;;
      subject_ID)   SUBJECT_ID="$v"   ;;
    esac
  done
  printf "%s", "%s", "%s", ' "$DCM_SUBJECT_ID" "$SUBJECT_ID"
"$PET_EXPERIMENT_ID"
)
```

```
curl -f -X GET -u $USER:$PASSWORD
"$HOST$URIpath/resources/MRI/files/mriSessionMatch.json" 2>/dev/null | \
sed 's/\[{\|}\|\\|,|\/\n/g' | grep '^"[a-zA-Z_]\+":' | (
  while read kv ; do
    k="$(echo "$kv" | sed 's/\s*:.*/' | sed 's//g')"
```

```
    v="$(echo "$kv" | sed 's/^\s*:\s*/' | sed 's//g')"
```

```
    #echo "K=$k, V=$v"
```

```
    case $k in
      MRIsession) MRI_EXPERIMENT_ID="$v" ;;
      qa)         REGISTRATION_QA="$v"  ;;
      surv)       STATISTICS_SURV="$v"  ;;
      cl)         STATISTICS_CL="$v"    ;;
    esac
  done
  printf "%s", %s, %s, %s\n' "$MRI_EXPERIMENT_ID"
"$REGISTRATION_QA" "$STATISTICS_SURV" "$STATISTICS_CL"
)
```

```
done | if [ -z "$OUTPUT" ] ; then
  cat
else
  cat >> "$OUTPUT"
fi
```

La llamada y el formato del resultado se pueden ver más arriba.



From:  
<https://cortafuegos.fundacioace.com/wiki/> - **Detritus Wiki**

Permanent link:  
[https://cortafuegos.fundacioace.com/wiki/doku.php?id=neuroimagen:xnat\\_pipelines\\_registro](https://cortafuegos.fundacioace.com/wiki/doku.php?id=neuroimagen:xnat_pipelines_registro)

Last update: **2020/12/11 16:25**

