Generic Recon Chain and Toolboxes

demo and inline debug

Hui Xue

NHLBI, NIH

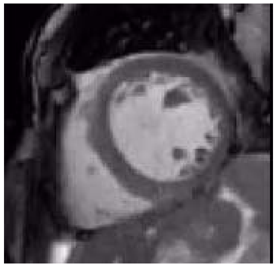
July 2020

In this session

- Motivation of generic chain
- MRD dimensions
- Triggering and on-the-fly recon
- Core data structures
- Walk through of key gadgets
- Interactive session - set up Gadgetron in Ubuntu 20.04 from scratch and inline debug

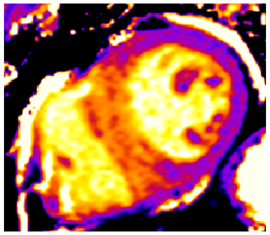
Why the generic chain is needed

- **To support common recon tasks**
- **To support deployment of Gadgetron**
- **Provide default implementation for reconstruction components**
- **Enable on-the-fly recon**
- **Demo and tutorial**



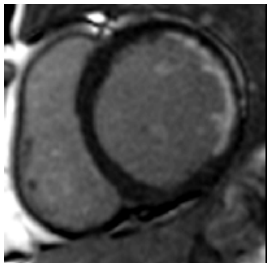
Cine

RO, E1, CHA, PHS, SLC



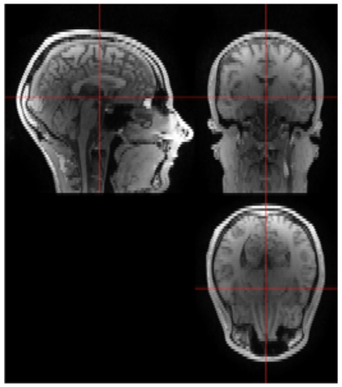
T1

RO, E1, CHA, SET, SLC, REP



Late Enhancement

RO, E1, CHA, SET, SLC



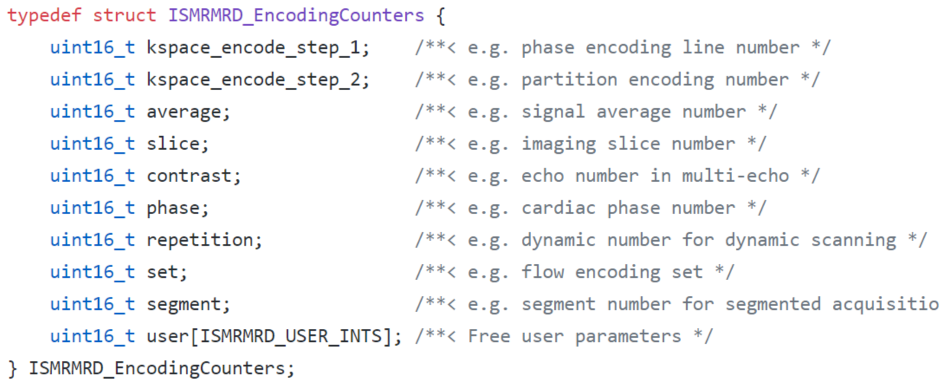
RO, E1, E2, CHA, SLC

MRD data dimensions

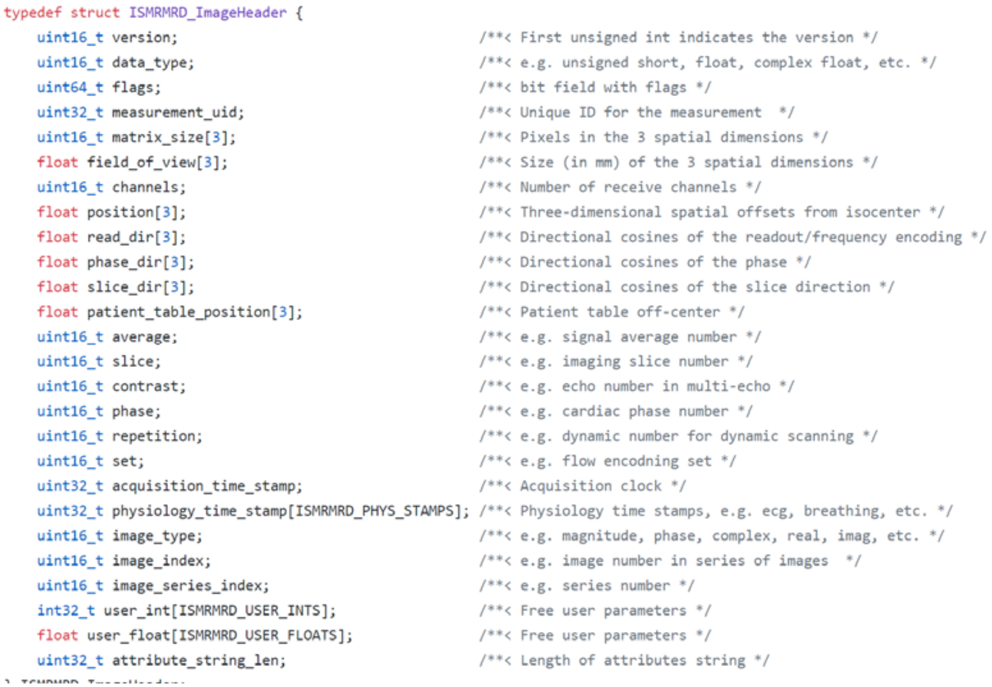
- **We don’t want to write a separate chain for every imaging app**

MR kspace data acquisition is sequential in general (e.g. compared to 3D ultrasound)

MRD data format encodes this data acquisition scheme



```c
typedef struct ISMRMRD_EncodingCounters {
    uint16_t kspace_encode_step_1;    /**< e.g. phase encoding line number */
    uint16_t kspace_encode_step_2;    /**< e.g. partition encoding number */
    uint16_t average;                 /**< e.g. signal average number */
    uint16_t slice;                   /**< e.g. imaging slice number */
    uint16_t contrast;                /**< e.g. echo number in multi-echo */
    uint16_t phase;                   /**< e.g. cardiac phase number */
    uint16_t repetition;              /**< e.g. dynamic number for dynamic scanning */
    uint16_t set;                     /**< e.g. flow encoding set */
    uint16_t segment;                 /**< e.g. segment number for segmented acquisitio
    uint16_t user[ISMRMRD_USER_INTS]; /**< Free user parameters */
} ISMRMRD_EncodingCounters;
```



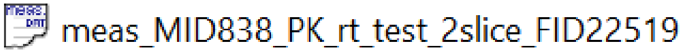
```c
typedef struct ISMRMRD_ImageHeader {
    uint16_t version;                                       /**< First unsigned int indicates the version */
    uint16_t data_type;                                     /**< e.g. unsigned short, float, complex float, etc. */
    uint64_t flags;                                         /**< bit field with flags */
    uint32_t measurement_uid;                               /**< Unique ID for the measurement  */
    uint16_t matrix_size[3];                                /**< Pixels in the 3 spatial dimensions */
    float field_of_view[3];                                 /**< Size (in mm) of the 3 spatial dimensions */
    uint16_t channels;                                      /**< Number of receive channels */
    float position[3];                                      /**< Three-dimensional spatial offsets from isocenter */
    float read_dir[3];                                      /**< Directional cosines of the readout/frequency encoding */
    float phase_dir[3];                                     /**< Directional cosines of the phase */
    float slice_dir[3];                                     /**< Directional cosines of the slice direction */
    float patient_table_position[3];                        /**< Patient table off-center */
    uint16_t average;                                       /**< e.g. signal average number */
    uint16_t slice;                                         /**< e.g. imaging slice number */
    uint16_t contrast;                                      /**< e.g. echo number in multi-echo */
    uint16_t phase;                                         /**< e.g. cardiac phase number */
    uint16_t repetition;                                    /**< e.g. dynamic number for dynamic scanning */
    uint16_t set;                                           /**< e.g. flow encodning set */
    uint32_t acquisition_time_stamp;                        /**< Acquisition clock */
    uint32_t physiology_time_stamp[ISMRMRD_PHYS_STAMPS];    /**< Physiology time stamps, e.g. ecg, breathing, etc. */
    uint16_t image_type;                                    /**< e.g. magnitude, phase, complex, real, imag, etc. */
    uint16_t image_index;                                   /**< e.g. image number in series of images  */
    uint16_t image_series_index;                            /**< e.g. series number */
    int32_t user_int[ISMRMRD_USER_INTS];                    /**< Free user parameters */
    float user_float[ISMRMRD_USER_FLOATS];                  /**< Free user parameters */
    uint32_t attribute_string_len;                          /**< Length of attributes string */
} ISMRMRD_ImageHeader;
```

National Heart, Lung, and Blood Institute

Order of acquisition to fill multi-dimensional array does matter

- **E.g. Cine imaging – RO, E1, CHA, PHS, SLC**

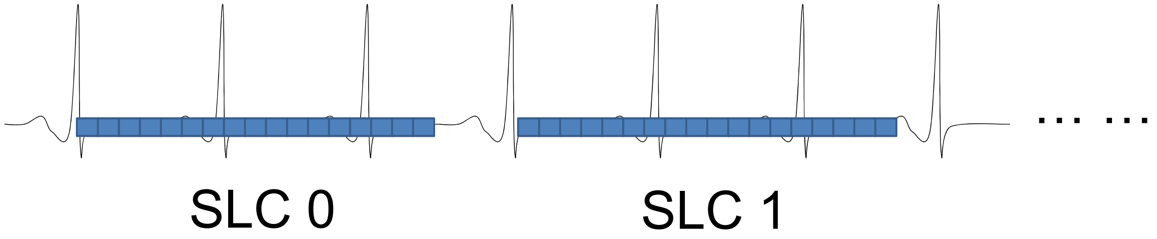
Let’s look at an example …

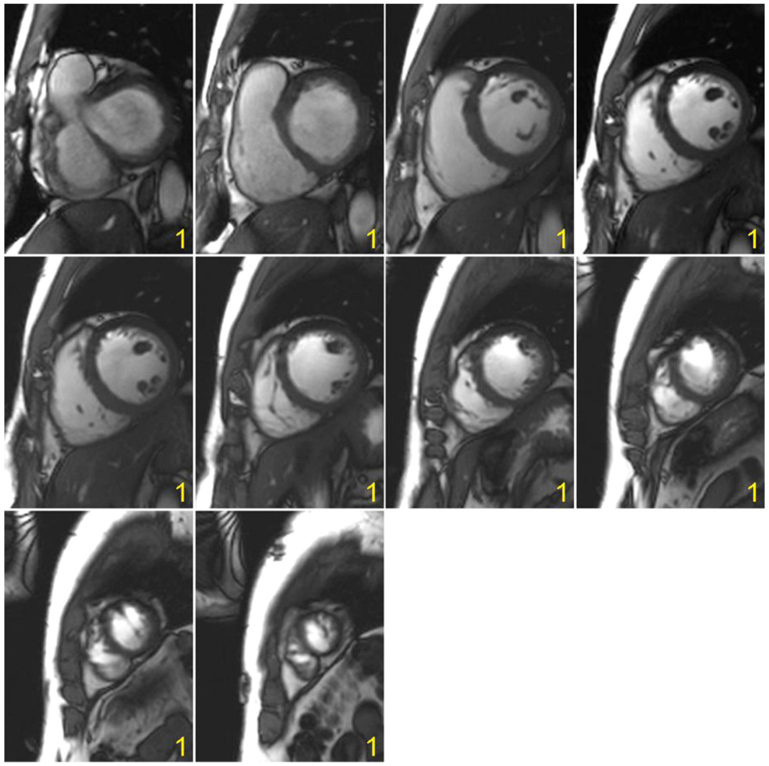


meas_MID838_PK_rt_test_2slice_FID22519

On-the-fly recon

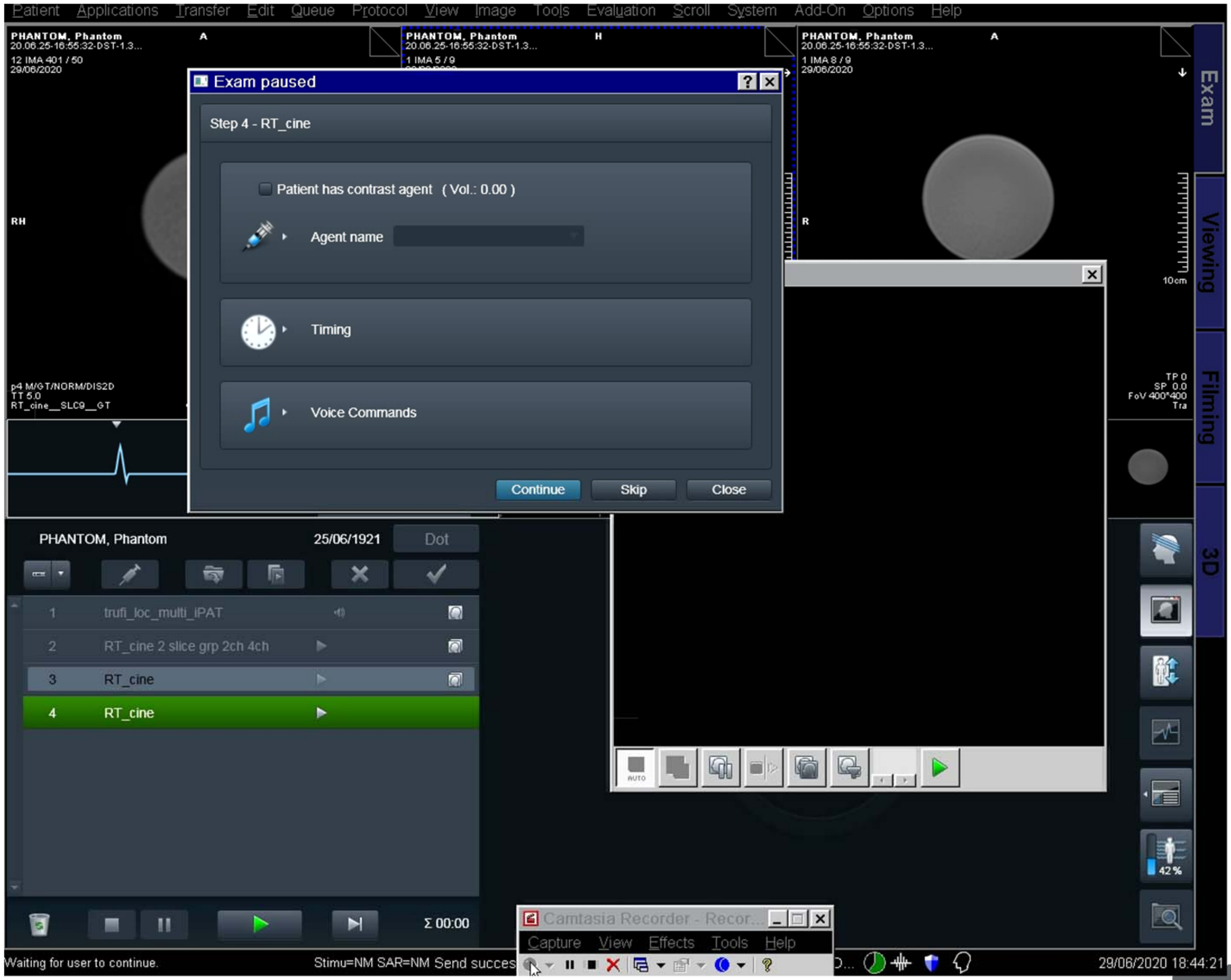
- No need to wait, but start when data was available
- Should be flexible





Consider the inline behavior

On-the-fly recon is popular among MR technicians.
Waiting time : 2s with on-the-fly, 20s without on-the-fly


Patient Applications Transfer Edit Queue Protocol View Image Tools Evaluation Scroll System Add-On Options Help
Exam
Viewing
Filming
3D
Continue
Skip
Close
Camtasia Recorder - Recor...
Capture View Effects Tools Help

Exam paused

Step 4 - RT_cine

□ Patient has contrast agent (Vol.: 0.00)

Agent name

Timing

Voice Commands

PHANTOM, Phantom 25/06/1921 Dot

1	trufi_loc_multi_iPAT	
2	RT_cine 2 slice grp 2ch 4ch	▶
3	RT_cine	▶
4	RT_cine	▶

Σ 00:00

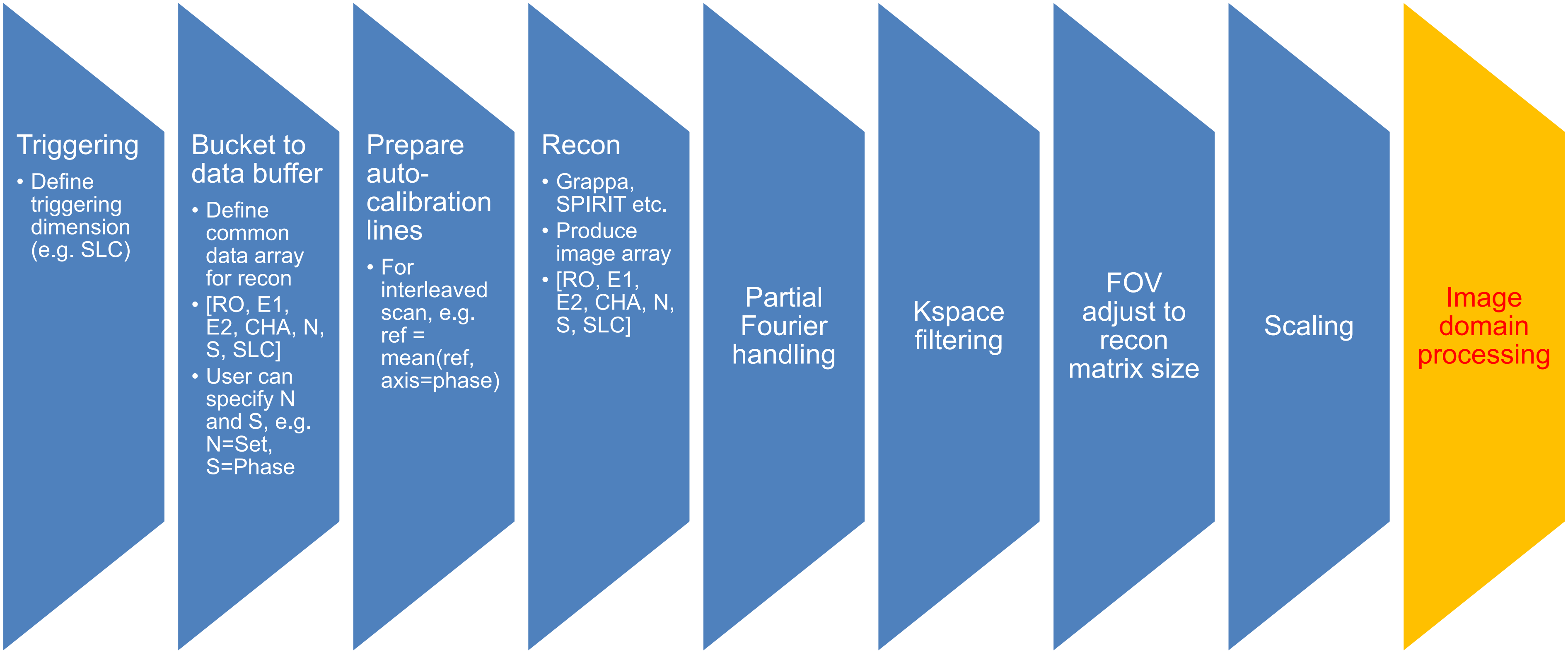
Waiting for user to continue.

Stimu=NM SAR=NM Send succes

29/06/2020 18:44:21

Generic recon chain

Key idea/features:



Triggering
- Define triggering dimension (e.g. SLC)

Bucket to data buffer
- Define common data array for recon
- [RO, E1, E2, CHA, N, S, SLC]
- User can specify N and S, e.g. N=Set, S=Phase

Prepare auto-calibration lines
- For interleaved scan, e.g. ref = mean(ref, axis=phase)

Recon
- Grappa, SPIRIT etc.
- Produce image array
- [RO, E1, E2, CHA, N, S, SLC]

Partial Fourier handling

Kspace filtering

FOV adjust to recon matrix size

Scaling

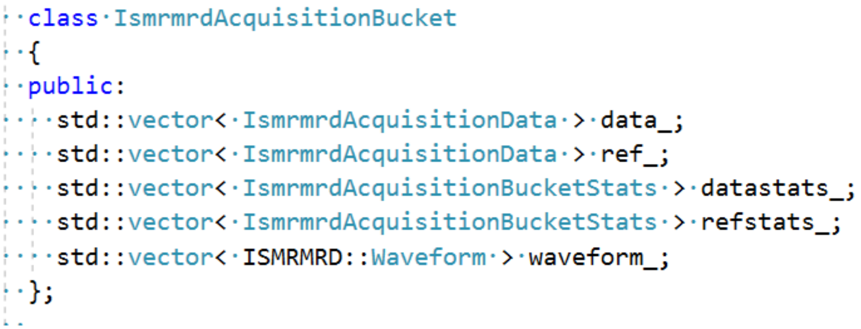
Image domain processing

National Heart, Lung, and Blood Institute

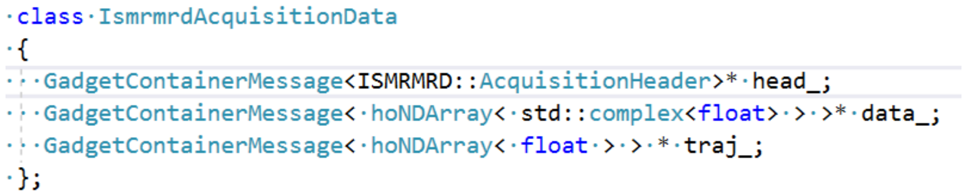
Generic recon chain: core data structure

IsmrmrdAcquisitionBucket

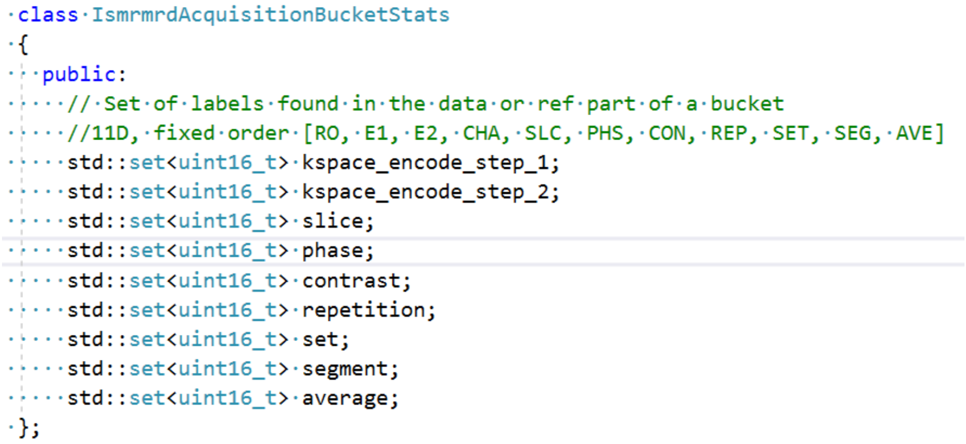
gadgetron\toolboxes\mri_core\mri_core_acquisition_bucket.h



```
 class IsmrmrdAcquisitionBucket
 {
 public:
    std::vector< IsmrmrdAcquisitionData > data_;
    std::vector< IsmrmrdAcquisitionData > ref_;
    std::vector< IsmrmrdAcquisitionBucketStats > datastats_;
    std::vector< IsmrmrdAcquisitionBucketStats > refstats_;
    std::vector< ISMRMRD::Waveform > waveform_;
 };
```



```
 class IsmrmrdAcquisitionData
 {
   GadgetContainerMessage<ISMRMRD::AcquisitionHeader>* head_;
   GadgetContainerMessage< hoNDArray< std::complex<float> > >* data_;
   GadgetContainerMessage< hoNDArray< float > > * traj_;
 };
```



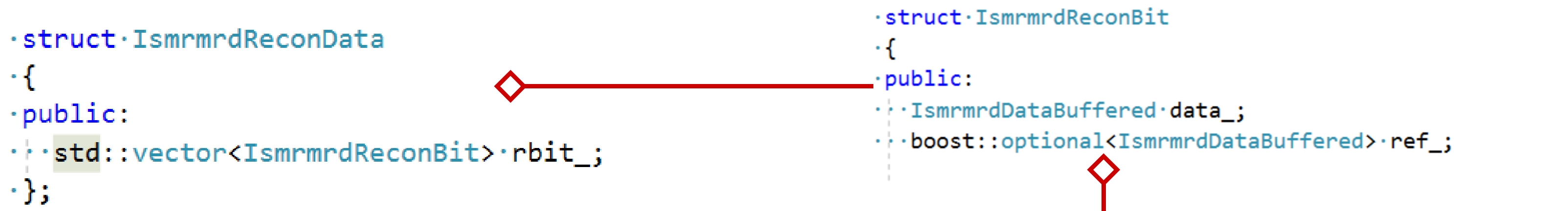
```
 class IsmrmrdAcquisitionBucketStats
 {
   public:
     // Set of labels found in the data or ref part of a bucket
     //11D, fixed order [RO, E1, E2, CHA, SLC, PHS, CON, REP, SET, SEG, AVE]
     std::set<uint16_t> kspace_encode_step_1;
     std::set<uint16_t> kspace_encode_step_2;
     std::set<uint16_t> slice;
     std::set<uint16_t> phase;
     std::set<uint16_t> contrast;
     std::set<uint16_t> repetition;
     std::set<uint16_t> set;
     std::set<uint16_t> segment;
     std::set<uint16_t> average;
 };
```

- Un-ordered buffer of incoming kspace lines, subject to sorting_dimension

Generic recon chain: core data structure

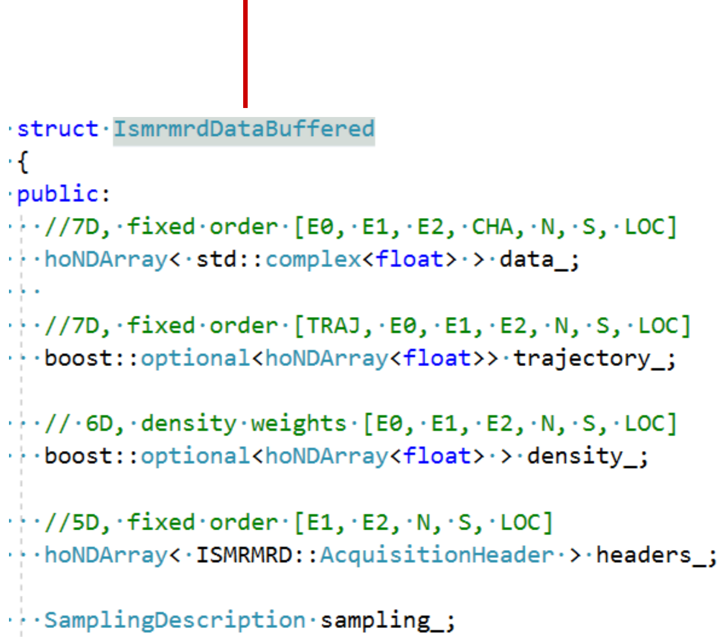
IsmrmrdReconData

An IsmrmrdAcquisitionBucket is converted to one or more IsmrmrdReconData seven dimensional array: [RO, E1, E2, CHA, N, S, SLC]



```
struct IsmrmrdReconData
{
public:
    std::vector<IsmrmrdReconBit> rbit_;
};
```

```
struct IsmrmrdReconBit
{
public:
    IsmrmrdDataBuffered data_;
    boost::optional<IsmrmrdDataBuffered> ref_;
```

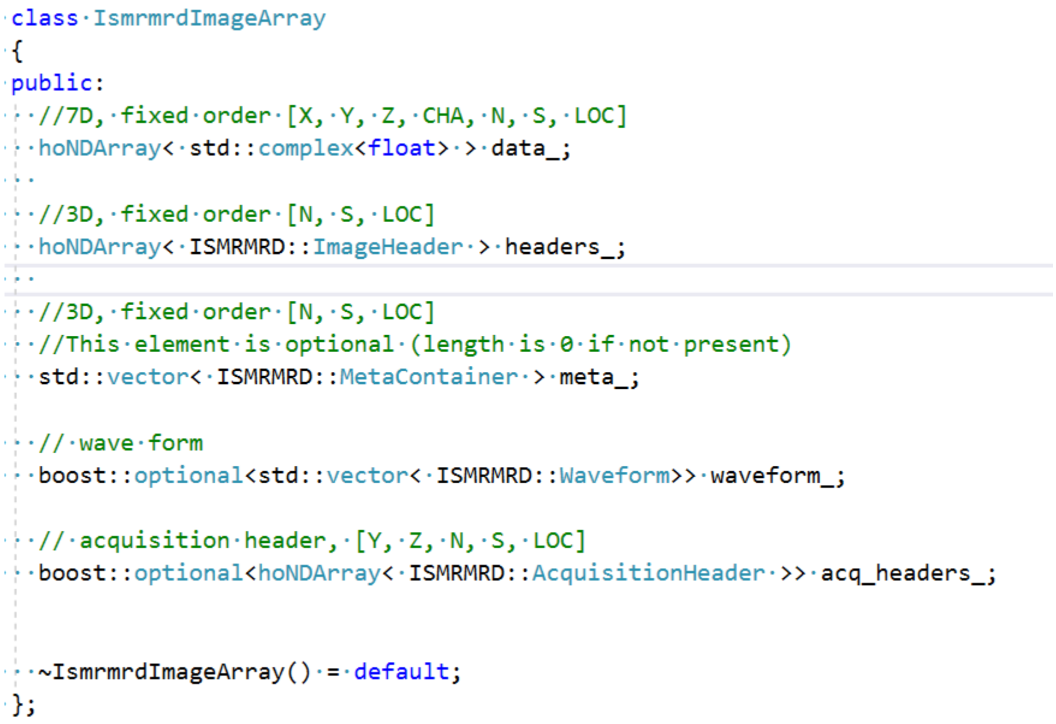


```
struct IsmrmrdDataBuffered
{
public:
    //7D, fixed order [E0, E1, E2, CHA, N, S, LOC]
    hoNDArray< std::complex<float> > data_;
    
    //7D, fixed order [TRAJ, E0, E1, E2, N, S, LOC]
    boost::optional<hoNDArray<float>> trajectory_;

    // 6D, density weights [E0, E1, E2, N, S, LOC]
    boost::optional<hoNDArray<float> > density_;

    //5D, fixed order [E1, E2, N, S, LOC]
    hoNDArray< ISMRMRD::AcquisitionHeader > headers_;

    SamplingDescription sampling_;
```

- Every encoding space corresponds to a ReconBit
- A ReconBit has data and ref
- Data and ref has 7D arrays for kspace/traj/density/headers…

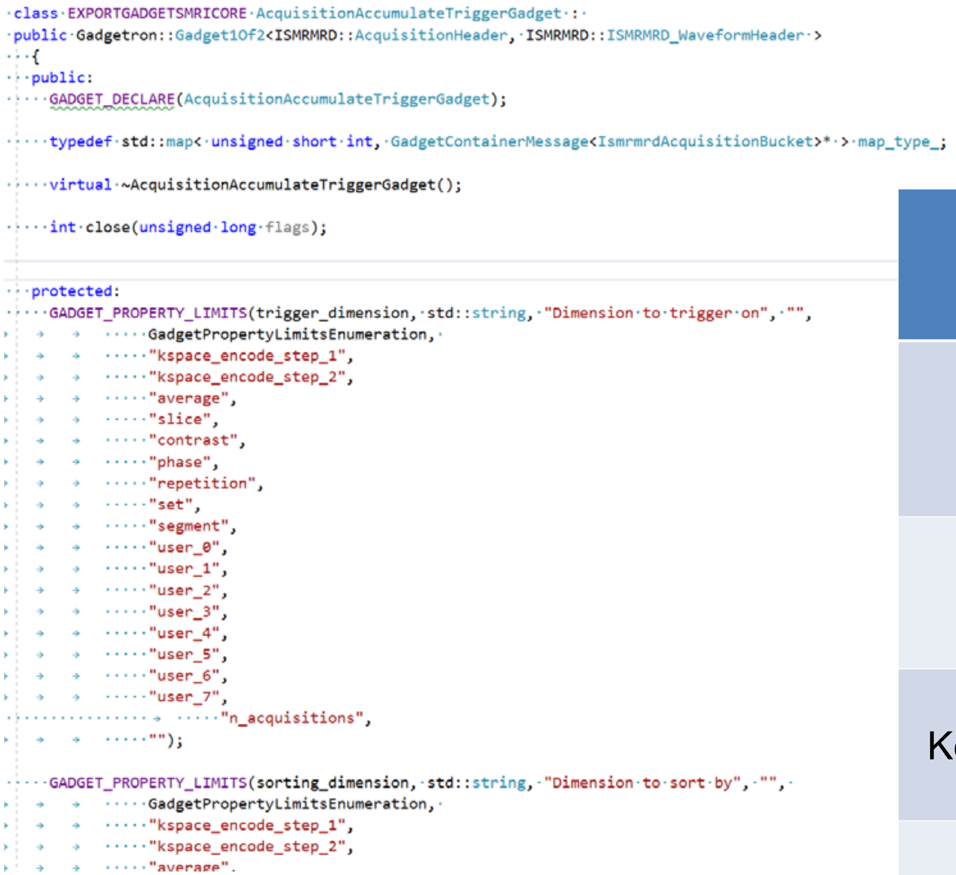
Generic recon chain: core data structure

IsmrmrdImageArray
data, header, meta fields
Each [RO, E1, E2, CHA] shares one header

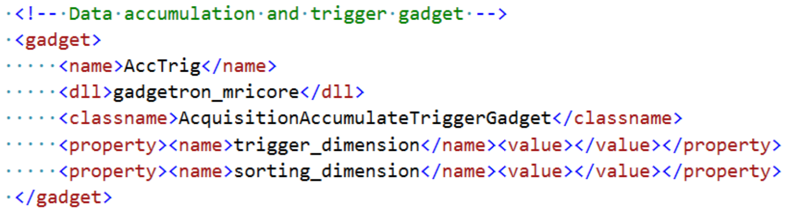


```
class IsmrmrdImageArray
{
public:
  //7D, fixed order [X, Y, Z, CHA, N, S, LOC]
  hoNDArray< std::complex<float> > data_;
  
  //3D, fixed order [N, S, LOC]
  hoNDArray< ISMRMRD::ImageHeader > headers_;
  
  //3D, fixed order [N, S, LOC]
  //This element is optional (length is 0 if not present)
  std::vector< ISMRMRD::MetaContainer > meta_;

  // wave form
  boost::optional<std::vector< ISMRMRD::Waveform>> waveform_;

  // acquisition header, [Y, Z, N, S, LOC]
  boost::optional<hoNDArray< ISMRMRD::AcquisitionHeader >> acq_headers_;


  ~IsmrmrdImageArray() = default;
};
```

Key gadgets: Triggering



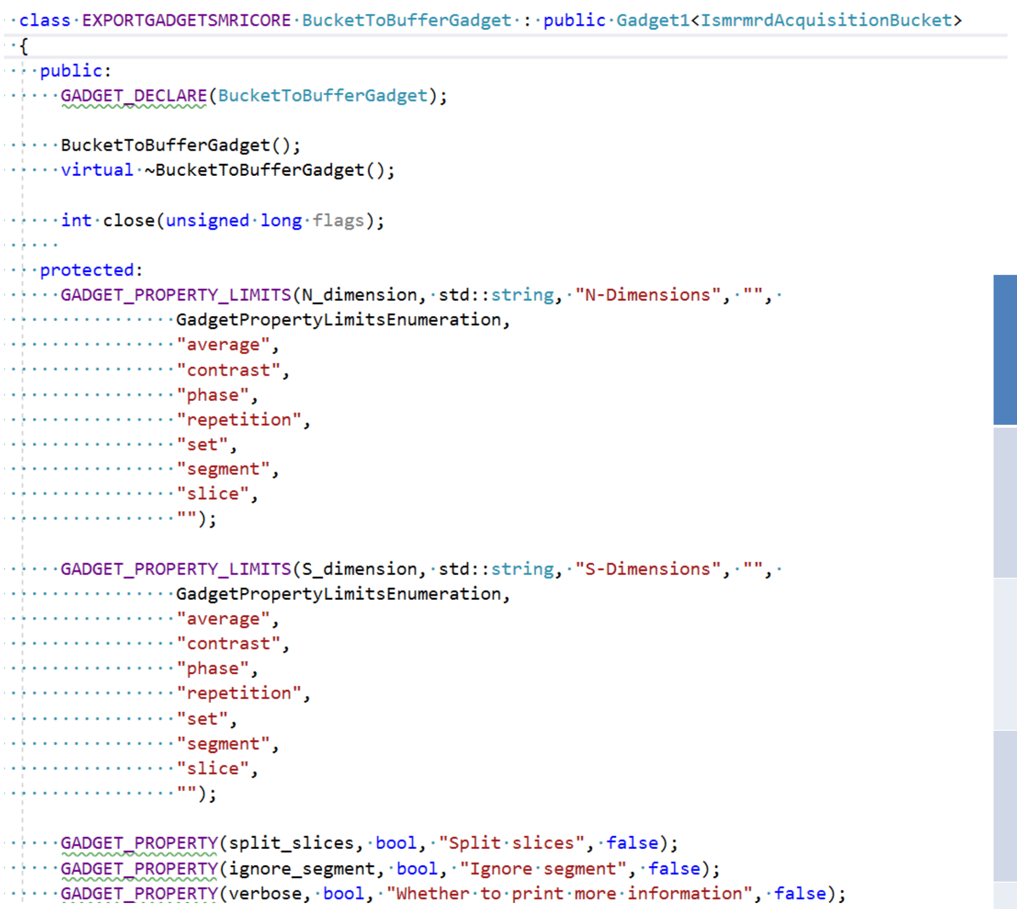
```
 class EXPORTGADGETSMRICORE AcquisitionAccumulateTriggerGadget : 
 public Gadgetron::Gadget1Of2<ISMRMRD::AcquisitionHeader, ISMRMRD::ISMRMRD_WaveformHeader >
   {
   public:
     GADGET_DECLARE(AcquisitionAccumulateTriggerGadget);

     typedef std::map< unsigned short int, GadgetContainerMessage<IsmrmrdAcquisitionBucket>* > map_type_;

     virtual ~AcquisitionAccumulateTriggerGadget();

     int close(unsigned long flags);

   protected:
     GADGET_PROPERTY_LIMITS(trigger_dimension, std::string, "Dimension to trigger on", "",
                     GadgetPropertyLimitsEnumeration, 
                     "kspace_encode_step_1",
                     "kspace_encode_step_2",
                     "average",
                     "slice",
                     "contrast",
                     "phase",
                     "repetition",
                     "set",
                     "segment",
                     "user_0",
                     "user_1",
                     "user_2",
                     "user_3",
                     "user_4",
                     "user_5",
                     "user_6",
                     "user_7",
                          "n_acquisitions",
                     "");

     GADGET_PROPERTY_LIMITS(sorting_dimension, std::string, "Dimension to sort by", "", 
                     GadgetPropertyLimitsEnumeration, 
                     "kspace_encode_step_1",
                     "kspace_encode_step_2",
                     "average",
```



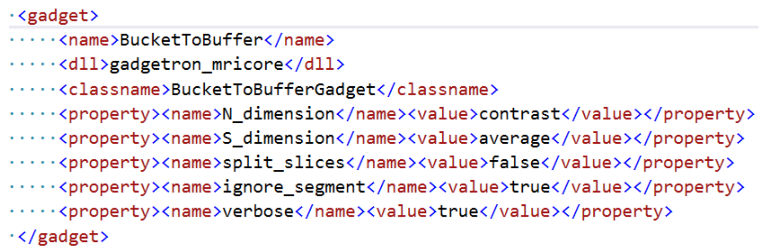
```
 <!-- Data accumulation and trigger gadget -->
 <gadget>
     <name>AccTrig</name>
     <dll>gadgetron_mricore</dll>
     <classname>AcquisitionAccumulateTriggerGadget</classname>
     <property><name>trigger_dimension</name><value></value></property>
     <property><name>sorting_dimension</name><value></value></property>
 </gadget>
```

gadgets\mri_core\ AcquisitionAccumulateTriggerGadget	
Input	Kspace line as ISMRMRD::AcquisitionHeader and hoNDArray
Output	IsmrmrdAcquisitionBucket
Key paras	trigger_dimension : once this dimension changes, send the buffered data to next gadget
Notes	• If trigger_dimension is not set, trigger only once in close(…) to send all data to next • sorting_dimension, if set, sort the buffered kspace line before sending

Key gadgets: BucketToBuffer



```
class EXPORTGADGETSMRICORE BucketToBufferGadget : public Gadget1<IsmrmrdAcquisitionBucket>
{
  public:
    GADGET_DECLARE(BucketToBufferGadget);

    BucketToBufferGadget();
    virtual ~BucketToBufferGadget();

    int close(unsigned long flags);
    
  protected:
    GADGET_PROPERTY_LIMITS(N_dimension, std::string, "N-Dimensions", "", 
               GadgetPropertyLimitsEnumeration,
               "average",
               "contrast",
               "phase",
               "repetition",
               "set",
               "segment",
               "slice",
               "");

    GADGET_PROPERTY_LIMITS(S_dimension, std::string, "S-Dimensions", "", 
               GadgetPropertyLimitsEnumeration,
               "average",
               "contrast",
               "phase",
               "repetition",
               "set",
               "segment",
               "slice",
               "");

    GADGET_PROPERTY(split_slices, bool, "Split slices", false);
    GADGET_PROPERTY(ignore_segment, bool, "Ignore segment", false);
    GADGET_PROPERTY(verbose, bool, "Whether to print more information", false);
```



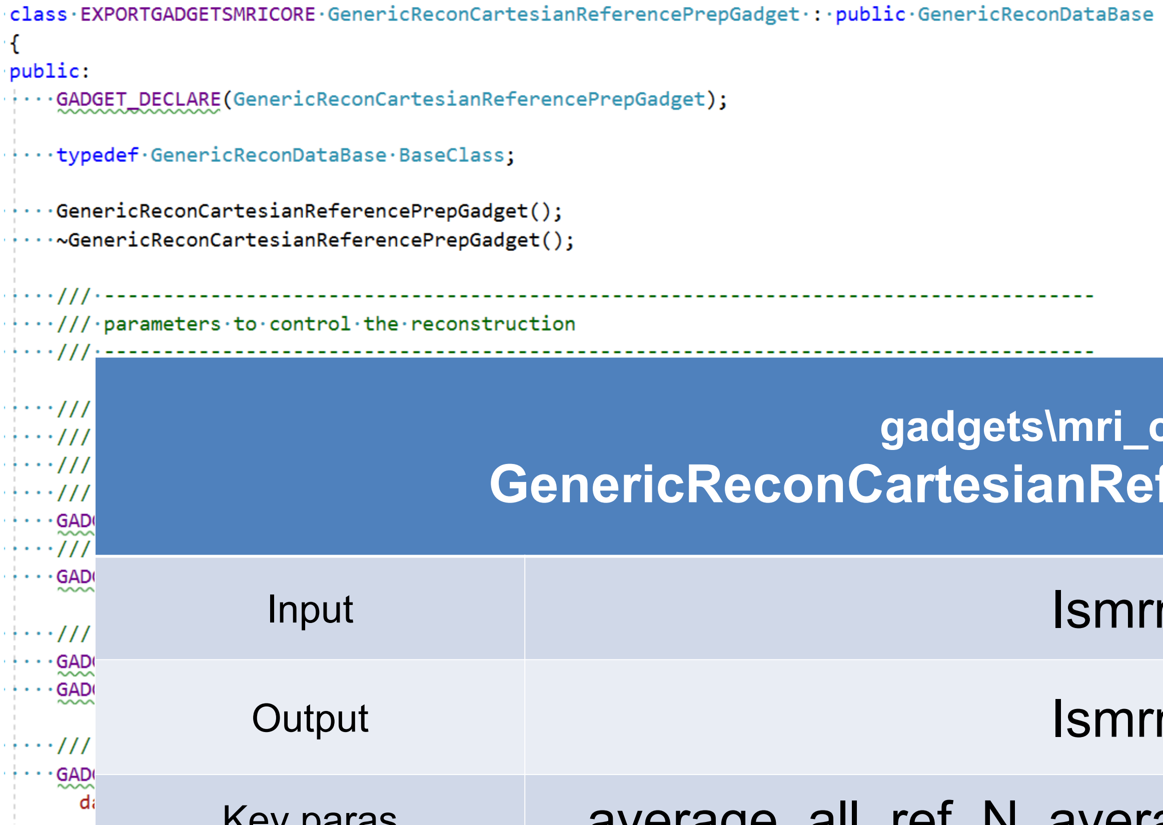
```
<gadget>
    <name>BucketToBuffer</name>
    <dll>gadgetron_mricore</dll>
    <classname>BucketToBufferGadget</classname>
    <property><name>N_dimension</name><value>contrast</value></property>
    <property><name>S_dimension</name><value>average</value></property>
    <property><name>split_slices</name><value>false</value></property>
    <property><name>ignore_segment</name><value>true</value></property>
    <property><name>verbose</name><value>true</value></property>
</gadget>
```

gadgets\mri_core\ BucketToBufferGadget	
Input	IsmrmrdAcquisitionBucket
Output	One or more IsmrmrdReconData
Key paras	N_dimension, S_dimension: [RO, E1, E2, CHA, N, S, SLC]
Notes	• Every incoming bucket is split into one or more ReconData • Separated into imaging and ref arrays • Optionally, include trajectory, density func • Include acquisition headers

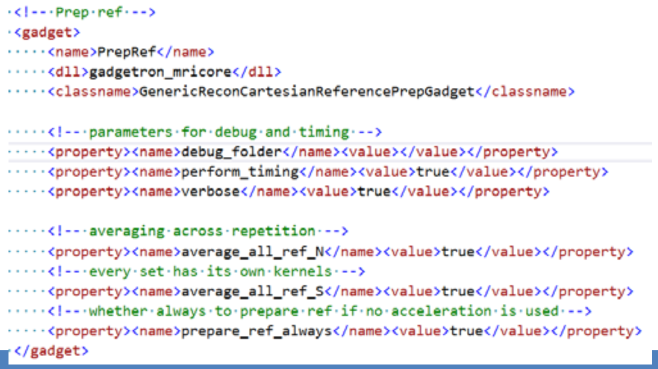
Bucket has AVE=3, N is Phase and S is SET

Then three ReconData will be sent out for ave=0,1,2

Key gadgets: Prepare reference



```
class EXPORTGADGETSMRICORE GenericReconCartesianReferencePrepGadget : public GenericReconDataBase
{
public:
    GADGET_DECLARE(GenericReconCartesianReferencePrepGadget);

    typedef GenericReconDataBase BaseClass;

    GenericReconCartesianReferencePrepGadget();
    ~GenericReconCartesianReferencePrepGadget();

    /// --------------------------------------------------------------------------------
    /// parameters to control the reconstruction
    /// --------------------------------------------------------------------------------

    ///
    ///
    ///
    ///
    GAD
    ///
    GAD

    ///
    GAD
    GAD

    ///
    GAD
      da
```

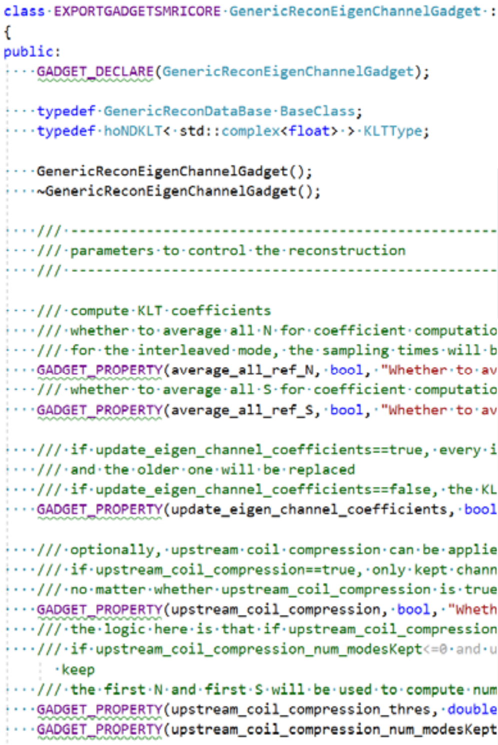


```
<!-- Prep ref -->
<gadget>
    <name>PrepRef</name>
    <dll>gadgetron_mricore</dll>
    <classname>GenericReconCartesianReferencePrepGadget</classname>

    <!-- parameters for debug and timing -->
    <property><name>debug_folder</name><value></value></property>
    <property><name>perform_timing</name><value>true</value></property>
    <property><name>verbose</name><value>true</value></property>

    <!-- averaging across repetition -->
    <property><name>average_all_ref_N</name><value>true</value></property>
    <!-- every set has its own kernels -->
    <property><name>average_all_ref_S</name><value>true</value></property>
    <!-- whether always to prepare ref if no acceleration is used -->
    <property><name>prepare_ref_always</name><value>true</value></property>
</gadget>
```

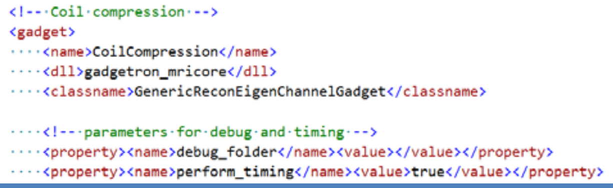
gadgets\mri_core\ GenericReconCartesianReferencePrepGadget	
Input	IsmrmrdReconData
Output	IsmrmrdReconData
Key paras	average_all_ref_N, average_all_ref_S, N_for_ref, S_for_ref
Notes	• Handle interleaved mode, R=1 mode • User can supply their classes for special needs • debug_folder, verbose, perform_timing

National Heart, Lung, and Blood Institute

Key gadgets: Coil compression



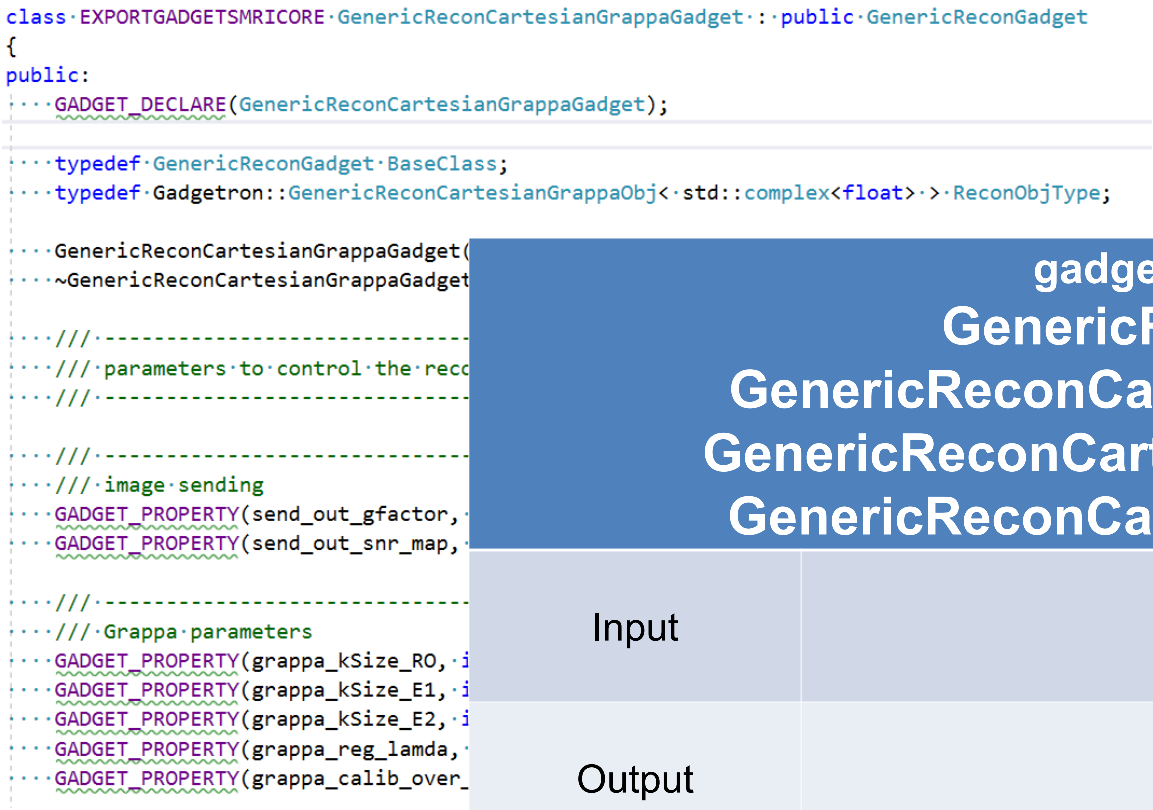
```
class EXPORTGADGETSMRICORE GenericReconEigenChannelGadget : public GenericReconDataBase
{
public:
    GADGET_DECLARE(GenericReconEigenChannelGadget);

    typedef GenericReconDataBase BaseClass;
    typedef hoNDKLT< std::complex<float> > KLTType;

    GenericReconEigenChannelGadget();
    ~GenericReconEigenChannelGadget();

    /// -----------------------------------------------------
    /// parameters to control the reconstruction
    /// -----------------------------------------------------

    /// compute KLT coefficients
    /// whether to average all N for coefficient computatio
    /// for the interleaved mode, the sampling times will b
    GADGET_PROPERTY(average_all_ref_N, bool, "Whether to av
    /// whether to average all S for coefficient computatio
    GADGET_PROPERTY(average_all_ref_S, bool, "Whether to av

    /// if update_eigen_channel_coefficients==true, every i
    /// and the older one will be replaced
    /// if update_eigen_channel_coefficients==false, the KL
    GADGET_PROPERTY(update_eigen_channel_coefficients, bool

    /// optionally, upstream coil compression can be applie
    /// if upstream_coil_compression==true, only kept chann
    /// no matter whether upstream_coil_compression is true
    GADGET_PROPERTY(upstream_coil_compression, bool, "Wheth
    /// the logic here is that if upstream_coil_compression
    /// if upstream_coil_compression_num_modesKept<=0 and u
       keep
    /// the first N and first S will be used to compute num
    GADGET_PROPERTY(upstream_coil_compression_thres, double
    GADGET_PROPERTY(upstream_coil_compression_num_modesKept
```



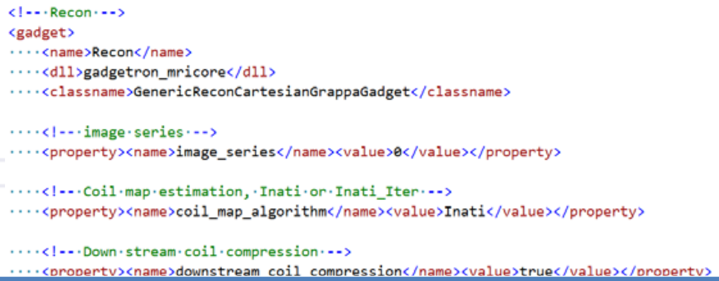
```
<!-- Coil compression -->
<gadget>
    <name>CoilCompression</name>
    <dll>gadgetron_mricore</dll>
    <classname>GenericReconEigenChannelGadget</classname>

    <!-- parameters for debug and timing -->
    <property><name>debug_folder</name><value></value></property>
    <property><name>perform_timing</name><value>true</value></property>
```

gadgets\mri_core\ GenericReconEigenChannelGadget	
Input	IsmrmrdReconData
Output	IsmrmrdReconData
Key paras	Control how to compute coil compression coefficients
Notes	• Compute coefficients on ref data • Control how many channels to keep with thres and num_modesKept • Different coefficients for different SLC • Output kspace are always in eigen channel (high energy first, along CHA)

Key gadgets: Recon



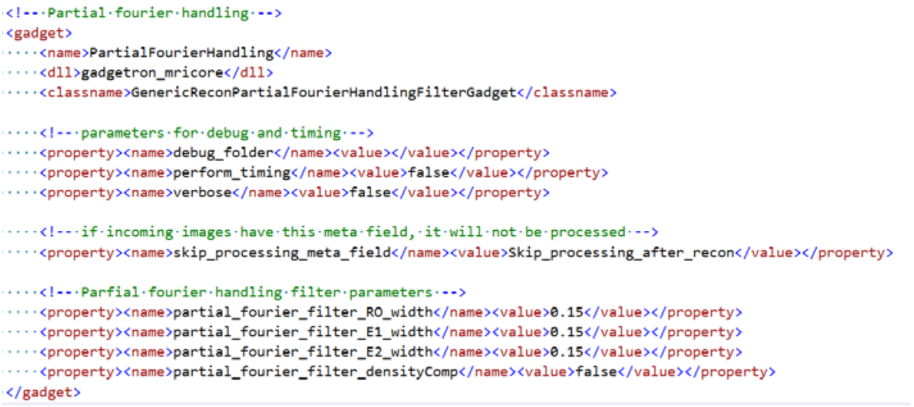
```
class EXPORTGADGETSMRICORE GenericReconCartesianGrappaGadget : public GenericReconGadget
{
public:
    GADGET_DECLARE(GenericReconCartesianGrappaGadget);

    typedef GenericReconGadget BaseClass;
    typedef Gadgetron::GenericReconCartesianGrappaObj< std::complex<float> > ReconObjType;

    GenericReconCartesianGrappaGadget(
    ~GenericReconCartesianGrappaGadget

    /// --------------------------------
    /// parameters to control the reco
    /// --------------------------------

    /// --------------------------------
    /// image sending
    GADGET_PROPERTY(send_out_gfactor,
    GADGET_PROPERTY(send_out_snr_map,

    /// --------------------------------
    /// Grappa parameters
    GADGET_PROPERTY(grappa_kSize_RO, i
    GADGET_PROPERTY(grappa_kSize_E1, i
    GADGET_PROPERTY(grappa_kSize_E2, i
    GADGET_PROPERTY(grappa_reg_lamda,
    GADGET_PROPERTY(grappa_calib_over_
```



```
<!-- Recon -->
<gadget>
    <name>Recon</name>
    <dll>gadgetron_mricore</dll>
    <classname>GenericReconCartesianGrappaGadget</classname>

    <!-- image series -->
    <property><name>image_series</name><value>0</value></property>

    <!-- Coil map estimation, Inati or Inati_Iter -->
    <property><name>coil_map_algorithm</name><value>Inati</value></property>

    <!-- Down stream coil compression -->
    <property><name>downstream_coil_compression</name><value>true</value></property>
```

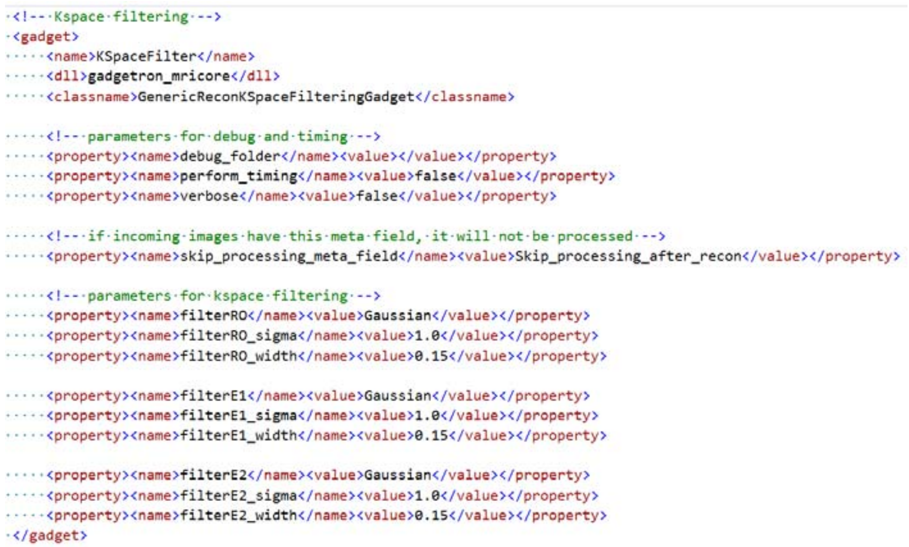
gadgets\mri_core\ GenericReconGadget, GenericReconCartesianGrappaGadget, GenericReconCartesianGrappaAIGadget, GenericReconCartesianSpiritGadget …	
Input	IsmrmrdReconData
Output	IsmrmrdImageArray
Key paras	Parameters to control recon, e.g. grappa, kernel size, regularization strength …
Notes	• Key task is to convert ReconData to array of images • Follow 7D convention • User should implement their recon methods here

Key gadgets: Partial Fourier handling, Kspace filter



```xml
<!-- Partial fourier handling -->
<gadget>
    <name>PartialFourierHandling</name>
    <dll>gadgetron_mricore</dll>
    <classname>GenericReconPartialFourierHandlingFilterGadget</classname>

    <!-- parameters for debug and timing -->
    <property><name>debug_folder</name><value></value></property>
    <property><name>perform_timing</name><value>false</value></property>
    <property><name>verbose</name><value>false</value></property>

    <!-- if incoming images have this meta field, it will not be processed -->
    <property><name>skip_processing_meta_field</name><value>Skip_processing_after_recon</value></property>

    <!-- Parfial fourier handling filter parameters -->
    <property><name>partial_fourier_filter_RO_width</name><value>0.15</value></property>
    <property><name>partial_fourier_filter_E1_width</name><value>0.15</value></property>
    <property><name>partial_fourier_filter_E2_width</name><value>0.15</value></property>
    <property><name>partial_fourier_filter_densityComp</name><value>false</value></property>
</gadget>
```

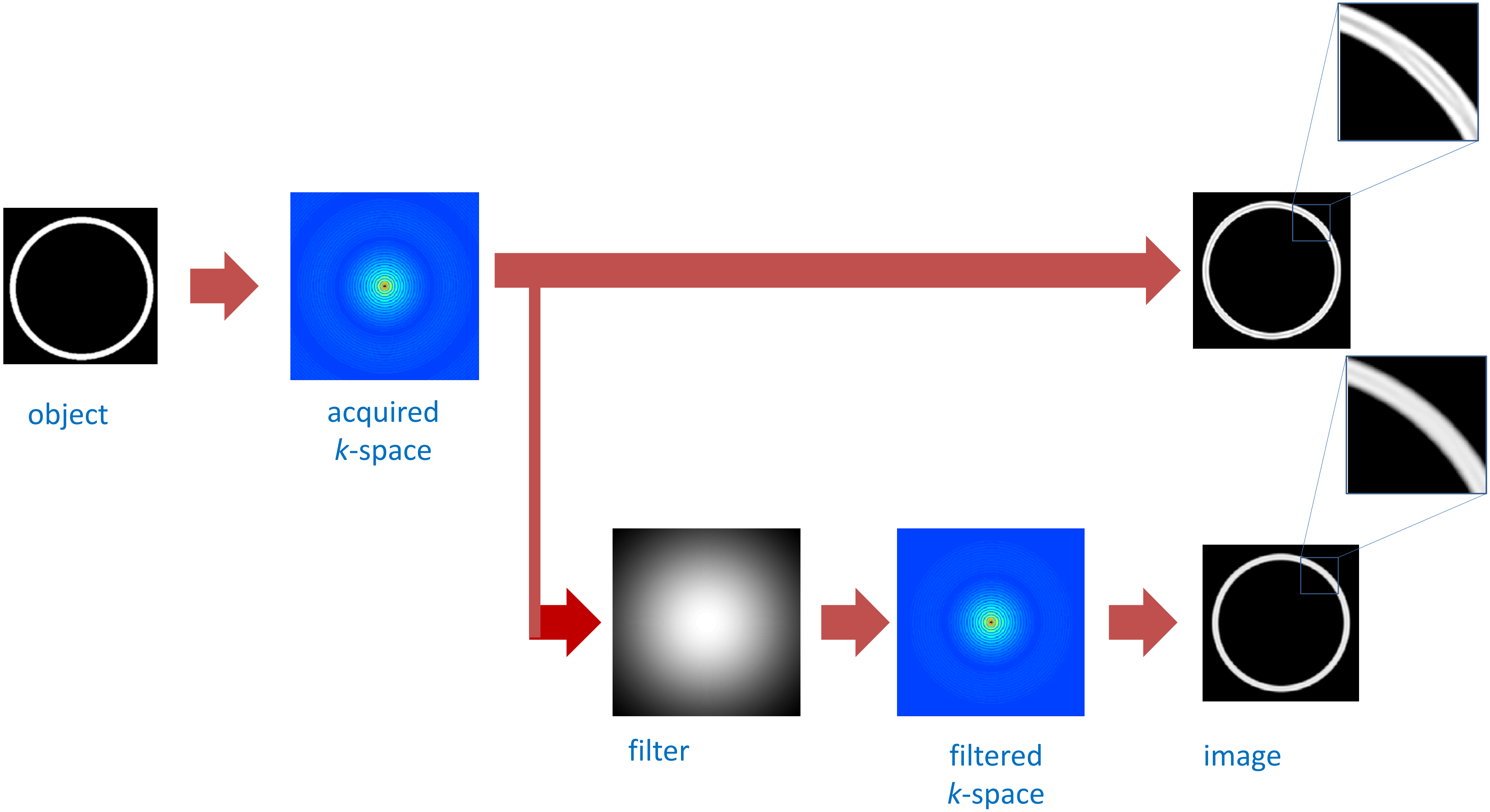
gadgets\mri_core\ GenericReconPartialFourierHandlingGadget GenericReconPartialFourierHandlingFilterGadget GenericReconPartialFourierHandlingPOCSGadget	
Input	IsmrmrdImageArray
Output	IsmrmrdImageArray
Key paras	Decide which gadget to use, filter strength, POCS
Notes	Support 2D and 3D Detect Partial Fourier from encoding space parameters



```xml
<!-- Kspace filtering -->
<gadget>
    <name>KSpaceFilter</name>
    <dll>gadgetron_mricore</dll>
    <classname>GenericReconKSpaceFilteringGadget</classname>

    <!-- parameters for debug and timing -->
    <property><name>debug_folder</name><value></value></property>
    <property><name>perform_timing</name><value>false</value></property>
    <property><name>verbose</name><value>false</value></property>

    <!-- if incoming images have this meta field, it will not be processed -->
    <property><name>skip_processing_meta_field</name><value>Skip_processing_after_recon</value></property>

    <!-- parameters for kspace filtering -->
    <property><name>filterRO</name><value>Gaussian</value></property>
    <property><name>filterRO_sigma</name><value>1.0</value></property>
    <property><name>filterRO_width</name><value>0.15</value></property>

    <property><name>filterE1</name><value>Gaussian</value></property>
    <property><name>filterE1_sigma</name><value>1.0</value></property>
    <property><name>filterE1_width</name><value>0.15</value></property>

    <property><name>filterE2</name><value>Gaussian</value></property>
    <property><name>filterE2_sigma</name><value>1.0</value></property>
    <property><name>filterE2_width</name><value>0.15</value></property>
</gadget>
```

gadgets\mri_core\ GenericReconKSpaceFilteringGadget	
Input	IsmrmrdImageArray
Output	IsmrmrdImageArray
Key paras	filterRO, filterE2, filterE3
Notes	• Separable filter for RO/E1/E2 • Default Gaussian filter • Good trade-off of main lobe width and remote lobe suppression

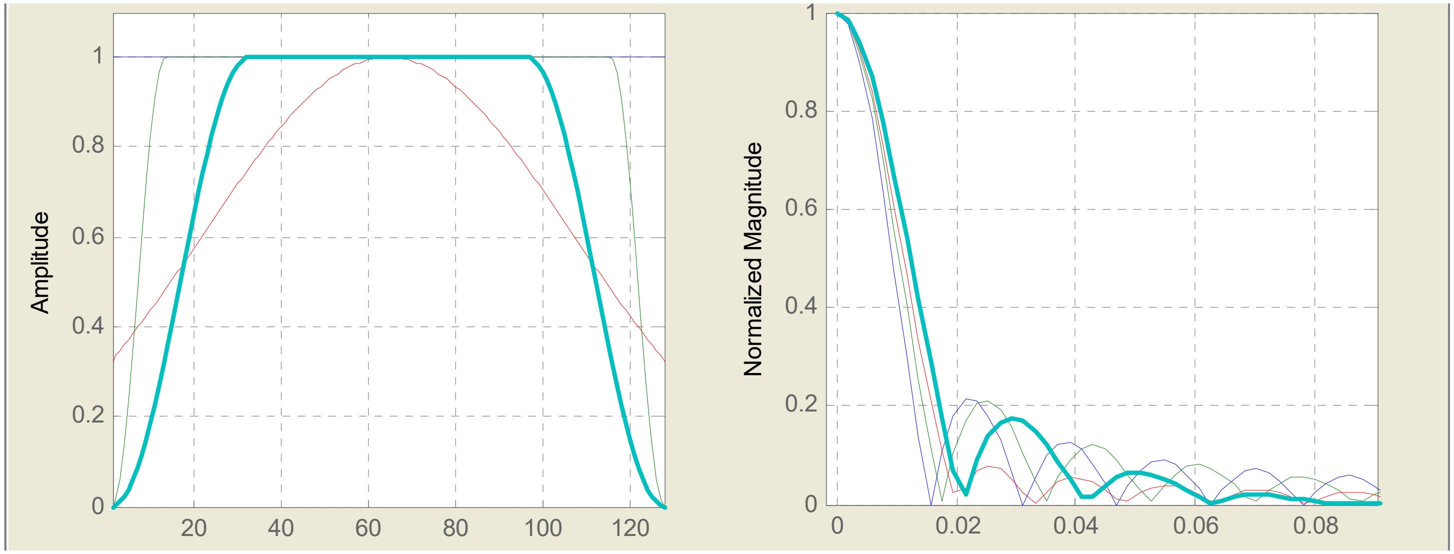
Kspace filter



Slides from Dr. Peter Kellman, NHLBI, NIH

NIH National Heart, Lung, and Blood Institute

Kspace filter: Comparison



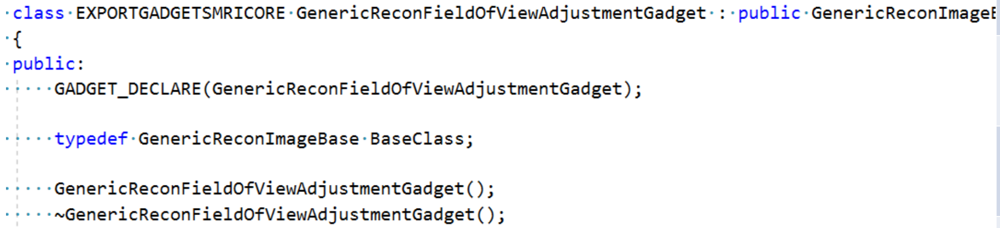
Gaussian, sigma=1.5pixel

Turkey, weak on the scanner

Turkey, medium on the scanner

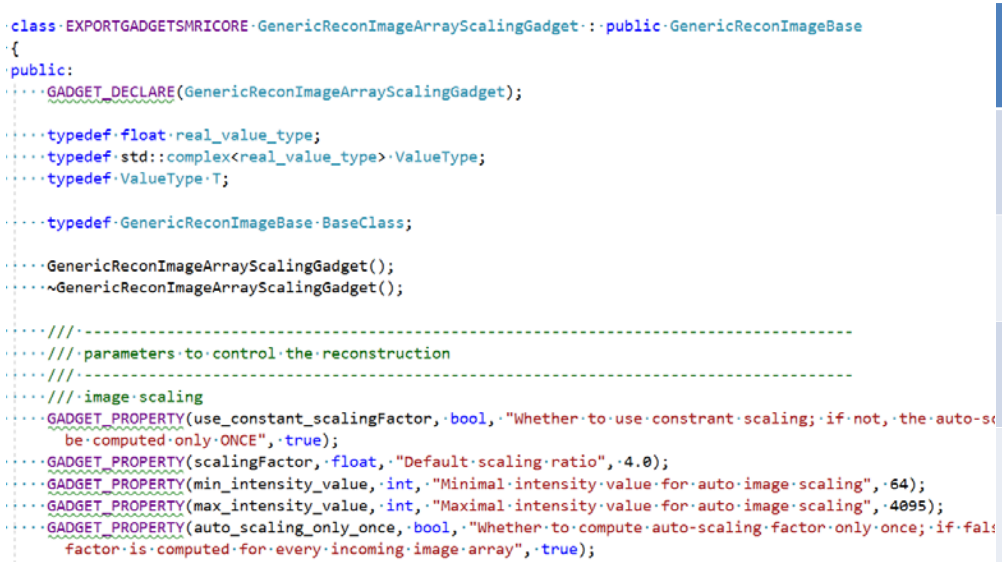
Slides from Dr. Peter Kellman, NHLBI, NIH

Key gadgets: FOV adjust and scaling



```
class EXPORTGADGETSMRICORE GenericReconFieldOfViewAdjustmentGadget : public GenericReconImageE
{
public:
    GADGET_DECLARE(GenericReconFieldOfViewAdjustmentGadget);

    typedef GenericReconImageBase BaseClass;

    GenericReconFieldOfViewAdjustmentGadget();
    ~GenericReconFieldOfViewAdjustmentGadget();
```

gadgets\mri_core\ GenericReconFieldOfViewAdjustmentGadget	
Input	IsmrmrdImageArray
Output	IsmrmrdImageArray
Key paras	Ismrmrd protocol
Notes	Support 2D and 3D Make sure output images are in correct size/FOV

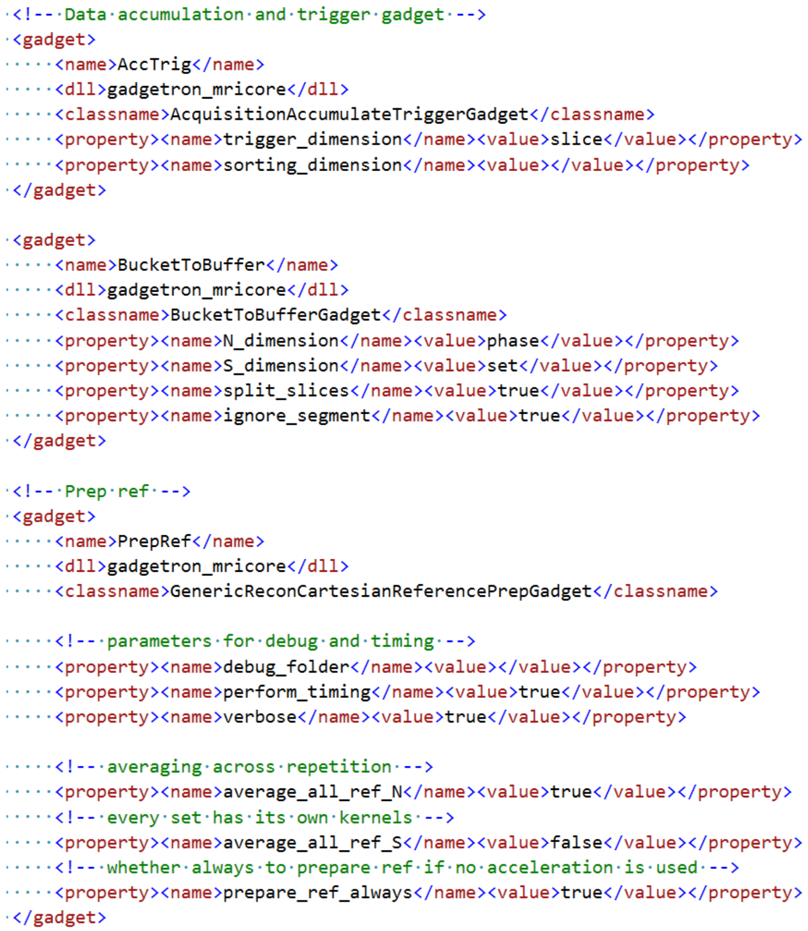


```
class EXPORTGADGETSMRICORE GenericReconImageArrayScalingGadget : public GenericReconImageBase
{
public:
    GADGET_DECLARE(GenericReconImageArrayScalingGadget);

    typedef float real_value_type;
    typedef std::complex<real_value_type> ValueType;
    typedef ValueType T;

    typedef GenericReconImageBase BaseClass;

    GenericReconImageArrayScalingGadget();
    ~GenericReconImageArrayScalingGadget();

    /// ------------------------------------------------------------------------------------
    /// parameters to control the reconstruction
    /// ------------------------------------------------------------------------------------
    /// image scaling
    GADGET_PROPERTY(use_constant_scalingFactor, bool, "Whether to use constrant scaling; if not, the auto-s
      be computed only ONCE", true);
    GADGET_PROPERTY(scalingFactor, float, "Default scaling ratio", 4.0);
    GADGET_PROPERTY(min_intensity_value, int, "Minimal intensity value for auto image scaling", 64);
    GADGET_PROPERTY(max_intensity_value, int, "Maximal intensity value for auto image scaling", 4095);
    GADGET_PROPERTY(auto_scaling_only_once, bool, "Whether to compute auto-scaling factor only once; if fal
      factor is computed for every incoming image array", true);
```

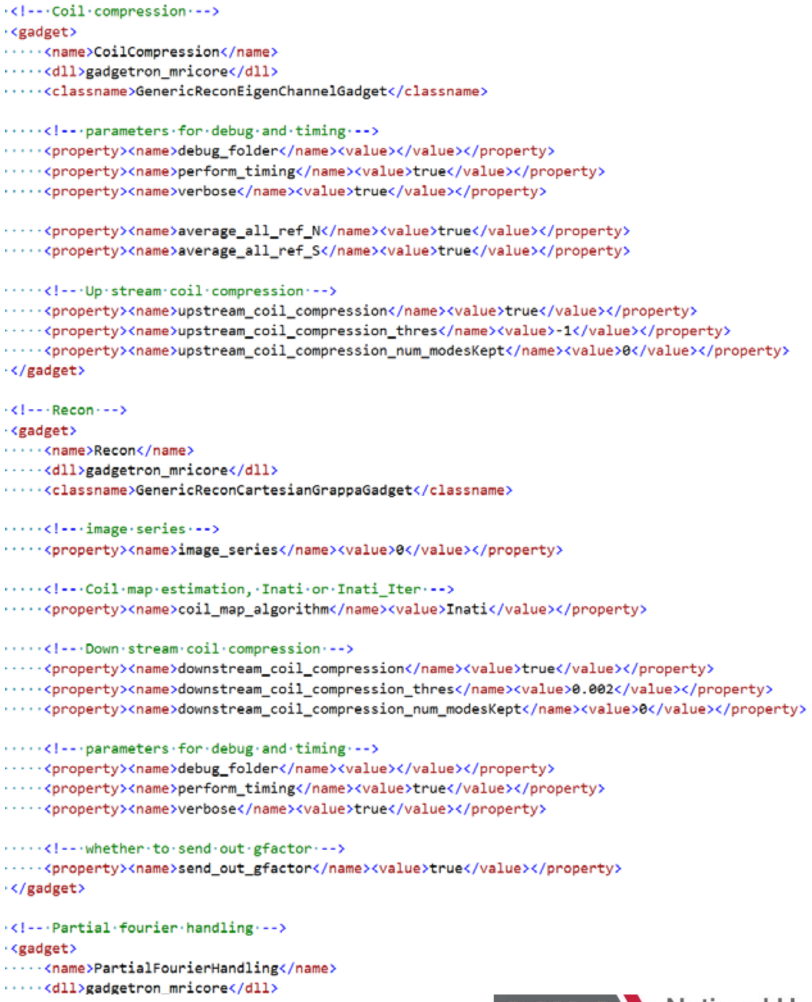
gadgets\mri_core\ GenericReconImageArrayScalingGadget	
Input	IsmrmrdImageArray
Output	IsmrmrdImageArray
Key paras	use_constant_scalingFactor, auto_scaling_only_once
Notes	Recommendation • Use constant scaling if SNR unit recon • Only scale once for multiple image array in one scan

Example: RealTime Cine

gadgets/mri_core/config/Generic_Cartesian_Grappa_RealTimeCine.xml



```xml
 <!-- Data accumulation and trigger gadget -->
 <gadget>
     <name>AccTrig</name>
     <dll>gadgetron_mricore</dll>
     <classname>AcquisitionAccumulateTriggerGadget</classname>
     <property><name>trigger_dimension</name><value>slice</value></property>
     <property><name>sorting_dimension</name><value></value></property>
 </gadget>

 <gadget>
     <name>BucketToBuffer</name>
     <dll>gadgetron_mricore</dll>
     <classname>BucketToBufferGadget</classname>
     <property><name>N_dimension</name><value>phase</value></property>
     <property><name>S_dimension</name><value>set</value></property>
     <property><name>split_slices</name><value>true</value></property>
     <property><name>ignore_segment</name><value>true</value></property>
 </gadget>

 <!-- Prep ref -->
 <gadget>
     <name>PrepRef</name>
     <dll>gadgetron_mricore</dll>
     <classname>GenericReconCartesianReferencePrepGadget</classname>

     <!-- parameters for debug and timing -->
     <property><name>debug_folder</name><value></value></property>
     <property><name>perform_timing</name><value>true</value></property>
     <property><name>verbose</name><value>true</value></property>

     <!-- averaging across repetition -->
     <property><name>average_all_ref_N</name><value>true</value></property>
     <!-- every set has its own kernels -->
     <property><name>average_all_ref_S</name><value>false</value></property>
     <!-- whether always to prepare ref if no acceleration is used -->
     <property><name>prepare_ref_always</name><value>true</value></property>
 </gadget>
```



```xml
 <!-- Coil compression -->
 <gadget>
     <name>CoilCompression</name>
     <dll>gadgetron_mricore</dll>
     <classname>GenericReconEigenChannelGadget</classname>

     <!-- parameters for debug and timing -->
     <property><name>debug_folder</name><value></value></property>
     <property><name>perform_timing</name><value>true</value></property>
     <property><name>verbose</name><value>true</value></property>

     <property><name>average_all_ref_N</name><value>true</value></property>
     <property><name>average_all_ref_S</name><value>true</value></property>

     <!-- Up stream coil compression -->
     <property><name>upstream_coil_compression</name><value>true</value></property>
     <property><name>upstream_coil_compression_thres</name><value>-1</value></property>
     <property><name>upstream_coil_compression_num_modesKept</name><value>0</value></property>
 </gadget>

 <!-- Recon -->
 <gadget>
     <name>Recon</name>
     <dll>gadgetron_mricore</dll>
     <classname>GenericReconCartesianGrappaGadget</classname>

     <!-- image series -->
     <property><name>image_series</name><value>0</value></property>

     <!-- Coil map estimation, Inati or Inati_Iter -->
     <property><name>coil_map_algorithm</name><value>Inati</value></property>

     <!-- Down stream coil compression -->
     <property><name>downstream_coil_compression</name><value>true</value></property>
     <property><name>downstream_coil_compression_thres</name><value>0.002</value></property>
     <property><name>downstream_coil_compression_num_modesKept</name><value>0</value></property>

     <!-- parameters for debug and timing -->
     <property><name>debug_folder</name><value></value></property>
     <property><name>perform_timing</name><value>true</value></property>
     <property><name>verbose</name><value>true</value></property>

     <!-- whether to send out gfactor -->
     <property><name>send_out_gfactor</name><value>true</value></property>
 </gadget>

 <!-- Partial fourier handling -->
 <gadget>
     <name>PartialFourierHandling</name>
     <dll>gadgetron_mricore</dll>
```

Demo session
Generic chain in ubuntu

- Let’s start from scratch
  - Install dependencies
  - Clone Gadgetron, ismrmrd etc.
  - cmake/ccmake
  - Compile debug version and install
  - Inline debug/code walk through with Eclipse
  - Walk through key gadgets in generic chain

For step-by-step instructions:

https://github.com/gadgetron/gadgetron/wiki/Visual-debug-Gadgetron-in-Ubuntu-using-Eclipse

NIH National Heart, Lung, and Blood Institute