GyroTools

www.gyrotools.com

User Manual

Recon Programming Environment for Scientists

Document Version 2.3, March 2015

INDEX OF CONTENTS

Preface

This manual describes the functionality and the instructions for installation and operation of the ReconFrame enviroment for Philips Achieva and Ingenia systems.

Disclaimer

Attention: ReconFrame is an investigational tool and is not certified for use in clinical diagnosis. Operation of ReconFrame on the scanner requires a clinical science research agreement with Philips Healthcare
The software comes with no warranties. In particular, image quality may be compromised by defects and improper use of the software. Any consequences of the use of this software are the sole responsibility of the end-user.

Legal Notices

Introduction and System Overview

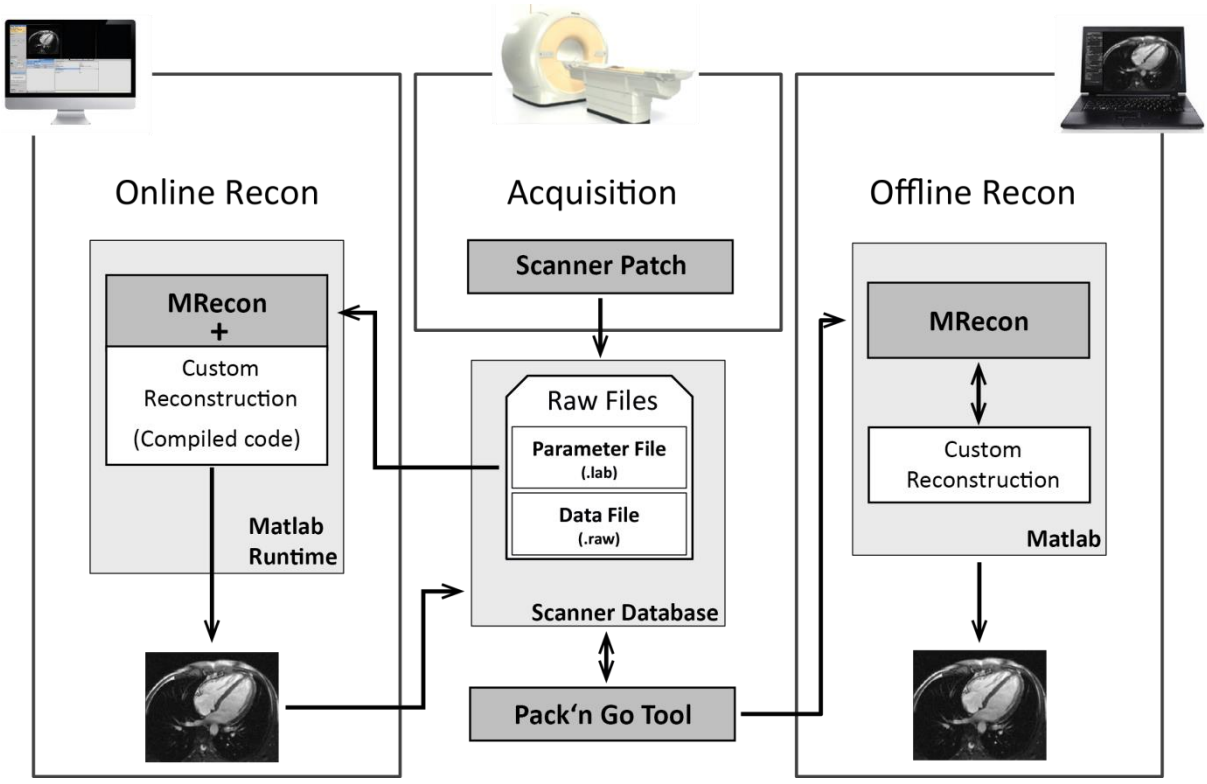
ReconFrame is a software platform providing various tools and all the functionality required to develop and perform a complete reconstruction of MR data. All available Philips MR data formats can be read, reconstructed and exported. Using Matlab as programming environment, data reconstruction procedures can be designed and tested offline. A compiled version of such reconstructions can also be installed directly on the scanner and run instead or in parallel to the standard Philips reconstruction. Images reconstructed with ReconFrame on the scanner are written back in the scanner patient database, allowing for a seamless integration of custom reconstructions into the scanner workflow.

MRecon: ReconFrame contains an extensive object-oriented Matlab library, implementing many common image and spectrum reconstruction task. The object-oriented design makes it easy to expand or alter the existing functionality. Code examples of complete reconstructions are provided. They can serve as a starting point for custom development.

Scanner Software Patch: On the scanner, ReconFrame installs a scanner software patch which for each scan will complement the standard raw data with all available GoalC parameter and objects. The raw data is then the only input to ReconFrame based reconstructions. Sequence parameters from the user interface and pulse programing enviroment are directly available in Matlab and can be used in reconstruction. The scanner software patch source is provided and can be merged automatically with any other patch of your choice.

New

Pack'n Go Tool: The pack and go tool is ReconFrame's independent patient database browser. Installed on the scanner, it allows accessing the database and exports raw and rec data files, using conveniently compressed file archives.



INSTALLATION

Prerequisites

Scanner:

- Access to either the gyrotest, MRService or MRResearch account is required. If the account is not available on your scanner, ask your local Philips representative to enable it.
- Download the *Scanner Software Patch* version that matches the scanner software release. Example: For the scanner software release R3.2.1 the installation file ReconFrame_r321.exe is required.
- For running compiled custom reconstructions on the scanner, the Matlab runtime libraries must be installed. Make sure to install the library version that matches the Matlab version the reconstruction was compiled with.

Offline Workstation:

- On the offline workstation, a full Matlab installation is required. The MRecon library is compatible with Windows, Linux and MacOS systems, both 32bit and 64bit. Recommended is Matlab 64bit along with a 64bit operating system and 8GB or more of installed RAM.
- Required Matlab Toolboxes: Image Processing Toolbox.
- To be able to compile custom reconstructions and run them on the scanner, a Windows Matlab installation is required along with the Matlab Compiler Toolbox.
- To integrate ReconFrame into custom scanner software patches, the full Philips Pulse Programing environment is required.

Downloadable Installation Files

▪ RECFRAME_rxxx_vx.exe	Scanner Software patch (compiled)
▪ ReconFrame_rxxx_Source.zip	Scanner Software patch (source code)
▪ MRecon-x.x.xxx.exe	Matlab recon library (Windows installer)
▪ MRecon-x.x.xxx.zip	Matlab recon library (zipped version for linux or OSX)
▪ gtPacknGo_vxxx_Installer.exe	Packn' Go tool
▪ gtPackGo_Manual.pdf	Packn'Go manual
▪ MCRInstaller.exe	Matlab runtime environment
▪ ReconFrame_Manual.pdf	This document

Some components of ReconFrame are intended for installation on the scanner, others on offline workstations only. Follow the instruction bellow for installing the various component of ReconFrame.

Scanner Software Patch

Compiled Version:

1. Copy the Software patch ReconFrame-rxxx-exe to a memory stick and plug it into the scanner
2. Double-click on ReconFrame-rxxx.exe
3. If asked to continue, press yes
4. If desired, change installation path and press unzip.

5. To start the patch do the following depending on your scanner software release:
    a. **Releases 2,3,4:** Start the patch by clicking on the Windows Start button → GyroTools → Start ReconFrame
    b. **Releases 5 and higher:** Start the patch by clicking "Select Patch" in the "MR Boot Configuration Manager" and then selecting "RECFRAME_rxxx".
6. Start the user interface and begin scanning. **Please make sure that "Save raw data" is enabled in the "postproc" tab for the scans you wish to reconstruct with ReconFrame.**
7. To stop the patch do the following depending on your scanner software release:
    a. **Releases 2,3,4:** When leaving the scanner, stop the patch via the Windows Start button → GyroTools → Stop ReconFrame
    b. **Releases 5 and higher:** Log out to the Philips start screen

Source Files:

New

1. Copy ReconFrame-rxxx-Source.zip to your VMware
2. Unpack the .zip file to your PPE source directory (e.g. G:\ppe\Rxxxxxxx\)
3. Navigate to the just extracted GyroTools folder (e.g. G:\ppe\Rxxxxxxx\GyroTools\)
4. Double click on install.exe
5. Compile and build the code
6. Copy the compiled patch to the scanner and start it
7. **Please make sure that "Save raw data" is enabled in the "postproc" tab for the scans you wish to reconstruct with ReconFrame.**

Pack'n Go Tool

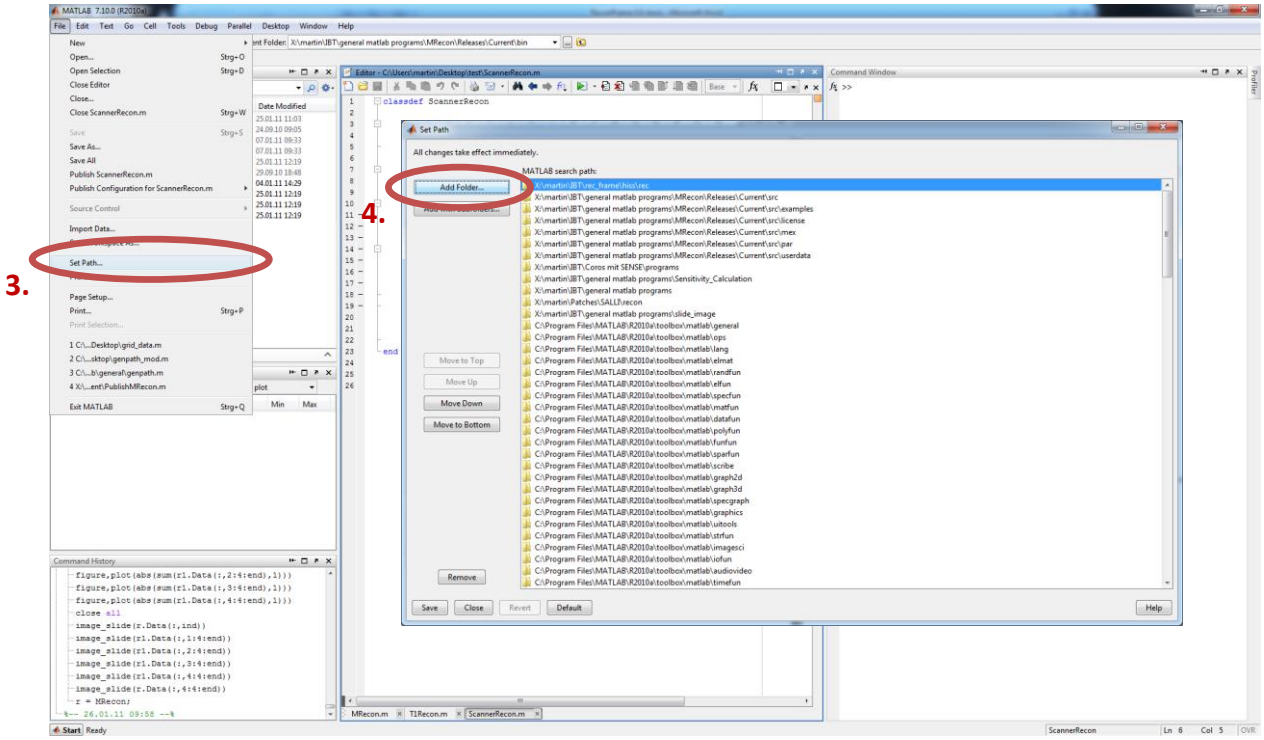
After scanning the raw data can be obtained via the Pack'n Go Tool:

1. Copy the Pack'n Go Tool PacknGo.exe to a memory stick and plug it into the Scanner
2. Double-click on PacknGo.exe
3. Select "Setup"
4. If asked to continue, press "yes"
5. If desired change the installation path and press "unzip"
6. After Completion the Matlab Compiler Runtime Enviroment 7.13 installer will start
7. Install the Matlab Compiler Runtime. If desired chose a different installation path
8. After installation start the Pack'n Go Tool via the Windows Start button → GyroTools → Pack'n Go Tool
9. Chose a Patient in the left window
10. Set the filter in the upper right corner to RAW to display only the raw data
11. Select a scan in the right window and Press "Pack&Go"
12. Choose a directory where to save the raw data and press "Save"
13. Obtain the zip file from the selected location

Matlab Library (MRecon)

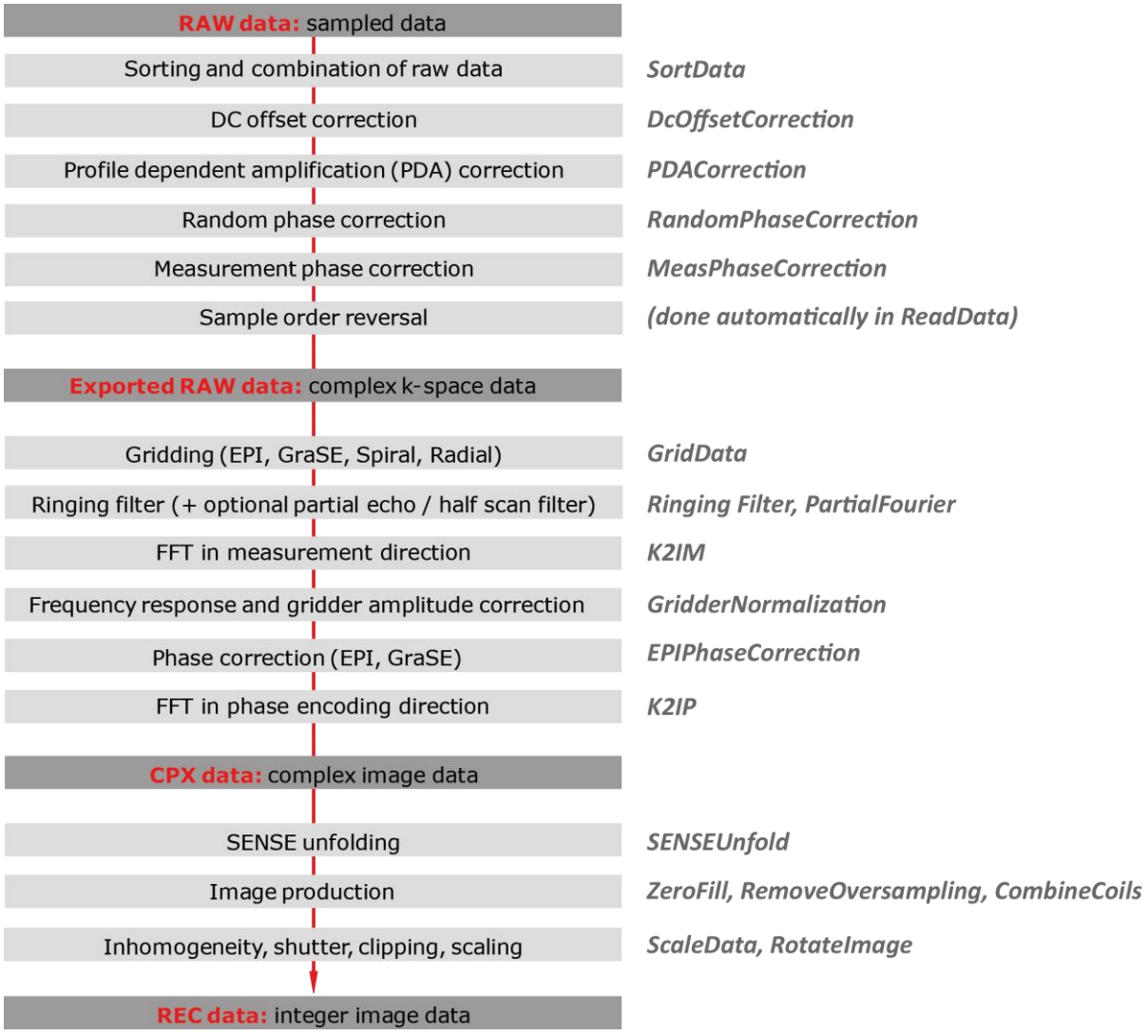
1. Double-click on MRecon-x.x.xxx.exe and install it to a directory of your choice
2. Open Matlab
3. Navigate to: File → Set Path… (see image below)
4. Add the MRecon installation directory to the Matlab path (see image below)
5. Execute the following command in the command line: MRecon('MachineID');

6. Send the displayed MachineID to GyroTools to obtain a license (martin.buehrer@gyrotools.com)
7. Copy the separately provided license file “license.lic” to the license directory *MRecon installation path*\license\ and overwrite the existing file in that directory.



RECONSTRUCTION OVERVIEW

An overview over the reconstruction can be observed in the chart below. The corresponding function names in MRecon are displayed on the right side in italic letters. Note that all the different datatypes (Raw, Exported Raw, Cpx, Rec), shown in the chart can be read and processed with MRecon. To get an overview over the matrix and data sizes during reconstruction please refer to the next section. Please refer to the section "MRecon Usage" for a description of how this reconstruction process is implemented in MRecon.

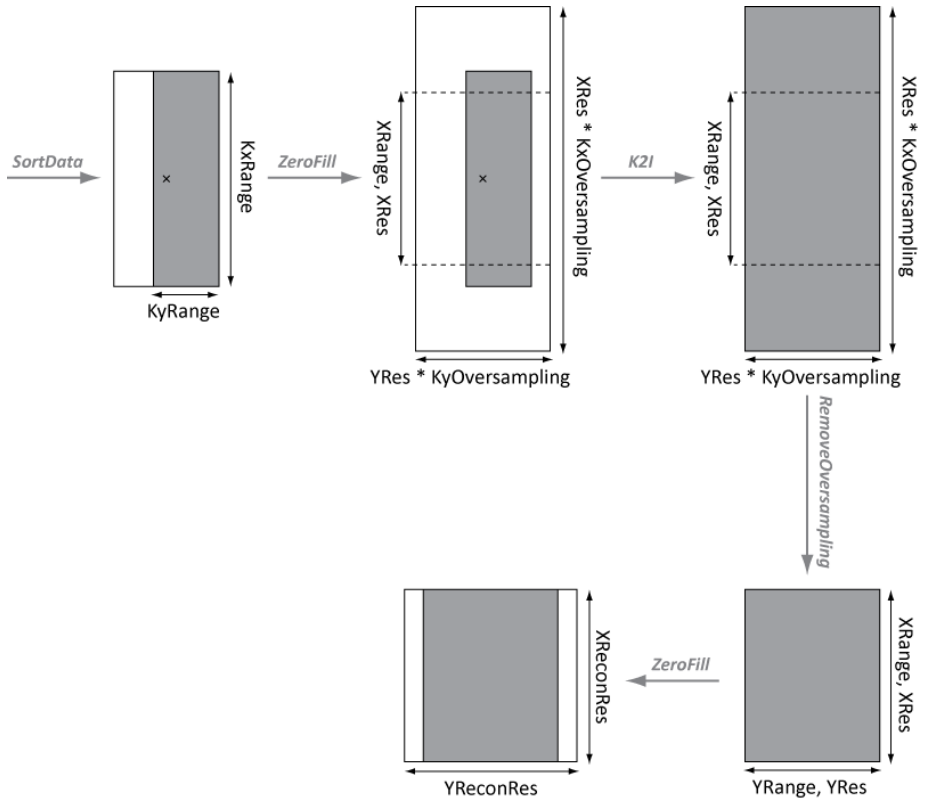


Matrices and Sizes

The data size and matrix dimensions are adapted several times during the reconstruction process. The sizes for the individual reconstruction steps are defined by the Parameter.Encoding struct. For illustration a graphical representation of the reconstruction process is shown below. Here a 2 dimensional cartesian dataset with the following parameters is reconstructed:

FOV:	320 x 250 mm
Acquired Voxel Size:	2 x 1.6 mm
Reconstructed Voxel Size:	1.25 x 1.25 mm
HalfScan:	Yes
Partial Echo:	No
KxRange:	-160, 159
KyRange:	-19, 77
XRange:	-128, 27
YRange:	-200, -1
XRes:	256
YRes:	200
XReconRes:	256
YReconRes:	256
KxOversampling:	2
KyOversampling:	1

Matrices are displayed as rectangles shaded grey in areas with values unequal zero (acquired data) and white in zero valued areas. The reconstruction steps in italic correspond to the function names in MRecon and the ranges to the Encoding struct fieldnames. The cross marks the center of k-space



MRecon Usage

Online Documentation

The online documentation is built into the Matlab help browser. To display the MRecon Documentation from Matlab type the following into the command prompt:

```
doc MRecon
```

Object Creation / Initialization

The first step is to create a reconstruction object by calling MRecon and passing the filename(s) of the data to be reconstructed. You can either pass both, the parameter and datafile:

```
r = MRecon( 'raw_data_file.lab', 'raw_data_file.raw' );
```

or you can pass just one of these files and MRecon tries to find the corresponding data/parameter file automatically:

```
r = MRecon( 'raw_data_file.raw' );
```

or

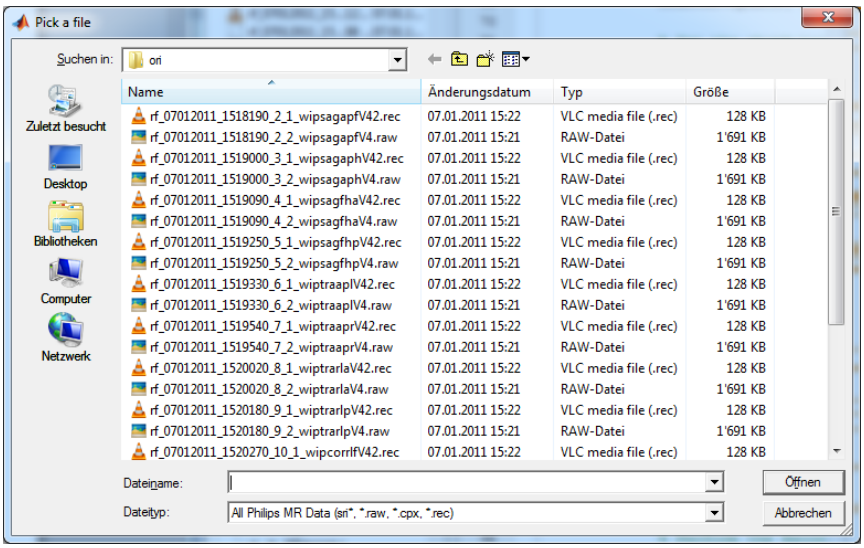
```
r = MRecon( 'raw_data_file.lab' );
```

Please note that when you pass both filenames, the parameter file always has to be the first argument. The above commands create a reconstruction object of a raw-dataset. Thereby MRecon automatically detects the data format of the dataset. Therefore, to read and process reconstructed .REC datasets the same command can be used:

```
r = MRecon( 'rec_data_file.rec' );
```

Calling MRecon without arguments will open a file dialog where the MR-datafile can be chosen manually. The initial path of the file dialog will be the current directory of Matlab:

```
r = MRecon;
```



Calling MRecon with a directory as input will open the same file dialog, but its initial path will be the specified directory:

```
r = MRecon( 'C:\MyData\MyRawData\' );
```

Upon creation, MRecon reads the associated parameter file (.lab, .cpx, .par) and fills its Parameter property.

Displaying r yields the following:

```
r =

  MRecon handle

  Properties:
    Parameter: [1x1 MRparameter]
         Data: []

  Methods, Events, Superclasses
```

We can observe the two properties of MRecon: Parameter (already filled) and Data (still empty). The Data property will be filled the next time ReadData is called.
Clicking on the Methods link displays the functions of the reconstruction object MRecon:

```
Methods for class MRecon:

CombineCoils            Perform                 ZeroFill
DcOffsetCorrection      RandomPhaseCorrection   addlistener
EPIPhaseCorrection      ReadData                delete
ExportLabels            RemoveOversampling      eq
GridData                RingingFilter           findobj
GridderNormalization    RotateImage             findprop
I2K                     SENSEUnfold             ge
K2I                     ScaleData               gt
K2IM                    SortData                isvalid
K2IP                    WriteCpx                le
MRecon                  WriteExportedRaw        lt
MeasPhaseCorrection     WritePar                ne
PDACorrection           WriteRec                notify
PartialFourier          WriteXMLPar
```

In the next sections we will explain how to use these functions.

Reading & Sorting Data

Before loading the data into Matlab, we can specify what to read. For that purpose we have a look at the Parameter2Read struct by typing the following command :

```
r.Parameter.Parameter2Read

ans =

  Parameter2ReadPars handle
```

```
  Properties:
      typ: [3x1 uint8]
      mix: 0
      dyn: 0
     card: 0
     echo: 0
     loca: 0
     chan: [8x1 double]
    extr1: 0
    extr2: 0
       ky: [97x1 double]
       kz: 0
     aver: 0
     rtop: 0

  Methods, Events, Superclasses
```

In the initial state, the Parameter2Read struct tells us exactly what data to expect in the raw-file. In this case we have a dataset with three different image types (standard data, phase correction data, noise data), 8 coils and 97 phase encoding profiles.
Every time data is read, MRecon checks the Parameter2Read struct and only loads the data specified in it. Thus, to read data selectively we change the struct fields before calling the read function. For example, let's assume that we only want to read the first 2 of the available 8 coils. To do this we set the chan (channels) field to:

```
r.Parameter.Parameter2Read.chan = [ 0, 1 ];
```

Please note that all the values, except the typ attribute, start from 0. To actually load the data we call the reading function:

```
r.ReadData;
```

This loads the data and updates the Data Property of MRecon. r has now the following form:

```
r =

  MRecon handle

  Properties:
    Parameter: [1x1 MRparameter]
         Data: {5x1 cell}

  Methods, Events, Superclasses
```

We see that the Data property is now set to a cell array, with 5 rows:

```
r.Data

ans =

    [  320x194 single]
    []
    [  320x2   single]
    []
    [19840x2   single]
```

The reason for this is that we have read different image types (Parameter2Read.typ). Every image type is written in its own cell element, as they usually differ in size and dimension. In this case our dataset contains of standard data (first row), phase correction data (third row) and noise data (fifth row). Let's assume we only want to reconstruct standard imaging data. To do so we modify the typ field and read the data again:

```
r.Parameter.Parameter2Read.typ =1;
r.ReadData;

r =

  MRecon handle

  Properties:
    Parameter: [1x1 MRparameter]
         Data: [320x194 single]

  Methods, Events, Superclasses
```

Since we now read only one image type, the Data property is no longer a cell but a simple 2 dimensional matrix. The first dimension corresponds to the measurement direction, featuring 320 samples. The second dimension corresponds to the number of loaded profiles which are sorted in the order they have been measured.

To sort the data into a matrix where each dimension corresponds to an imaging parameter we call the sort function:

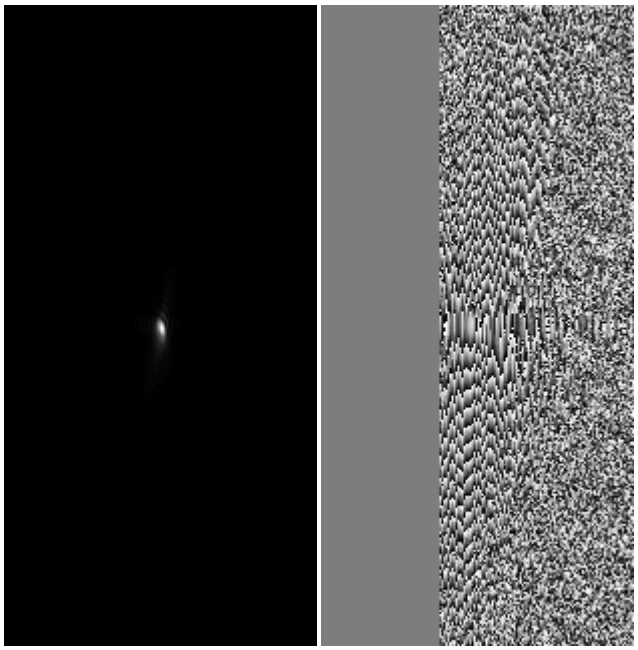
```
r.SortData;

r =

  MRecon handle

  Properties:
    Parameter: [1x1 MRparameter]
         Data: [4-D single]

  Methods, Events, Superclasses
```

The Data property has now been reshaped to a multidimensional array with size :

```
size( r.Data )

ans =

   320   156     1     2
```

The dimensions of the array are predefined as: ***x – y – z – coils – dynamics - cardiac phases – echoes – locations – mixes – extr1 – extr2 – averages***
This means our dataset contains 320 samples, 156 phase encoding profiles, and 2 coils (we only loaded the first 2 coils as specified earlier in the Parameter2Read struct). You might wonder why we got 156 phase encoding profiles instead of the 97 as shown in the Parameter2Read struct. The reason for that

is that Halfscan was enabled in the current scan and the sort function already places the data correctly in the k-space (and fills the rest with zeros). This is seen best if we visualize the data:



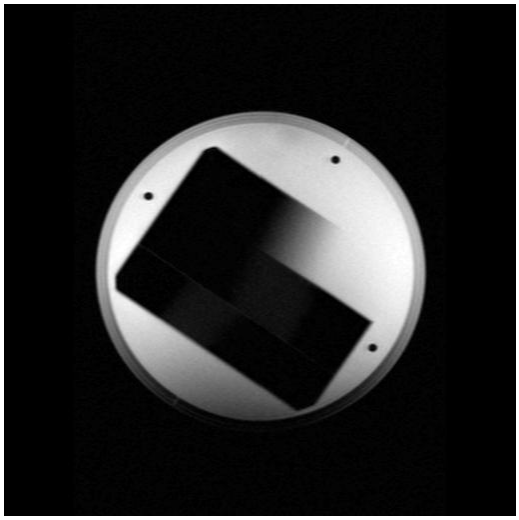
As we can see from the plot above, we actually measured 97 profiles, asymmetrically in k-space. *Please note that the recframe patch has to be used for MRecon to know the correct sampling pattern!*
Also note that the data is still unusable in this stage. To reconstruct meaningful images several corrections have to be performed. Most importantly the Random Phase Correction. Please refer to the Reconstruction Overview section for more information about the reconstruction steps.

Performing a Complete Reconstruction

To perform a complete reconstruction we more or less follow the reconstruction steps shown in the section Reconstruction Overview. The easiest way to do so is to call the Perform function of MRecon which executes a compete reconstruction. However before calling the function we reset the Parameter property of MRecon to undo all the changes we did in the last section (e.g. reconstruct all coils not just the first two):

```
r.Parameter.Reset;
r.Perform;

r =

  MRecon handle

  Properties:
    Parameter: [1x1 MRparameter]
         Data: [256x256 single]

  Methods, Events, Superclasses
```

After calling the Perform function the Data corresponds to the final squared image with the reconstructed dimension of 256 pixels. If we visualize the image we get:



The code for the Perform function can be found below. As you see we pretty much follow the scheme which is shown in the reconstruction overview. Note that we reconstruct the data in chunks and save the images of every reconstructed chunk in a temporary file on the disk.

```
function Perform( MR )
    switch MR.Parameter.DataFormat
        case{'ExportedRaw', 'Raw', 'Bruker' }

            %Reconstruct only standard (imaging) data
            MR.Parameter.Parameter2Read.typ = 1;
            MR.Parameter.Parameter2Read.Update;

            % Check if enough memory is available to reconstruct
            % the whole file at once. Otherwise reconstruct the
            % data in chunks
            if ispc
                [MemoryNeeded, MemoryAvailable, MaxDataSize] = ...
                  MR.GetMemoryInformation;
                if MemoryNeeded > MemoryAvailable
                    MR.Parameter.Chunk.Def = {'kx', 'ky', 'kz', 'chan' };
                end
            end

            % Switch off for performance reasons (after recon it is
            % switched back to original state again)
            AutoUpdateStatus = MR.Parameter.Recon.AutoUpdateInfoPars;
            MR.Parameter.Recon.AutoUpdateInfoPars = 'no';

            % Define new counter
            counter = Counter( 'Performing Recon --> Chunk %d/%d\n');

            % Loop over all chunks
            for cur_loop = 1:MR.Parameter.Chunk.NrLoops

                % Update Counter
                if strcmpi( MR.Parameter.Recon.StatusMessage, 'yes')
                   counter.Update( {cur_loop ,MR.Parameter.Chunk.NrLoops});
                end

                % Set the chunk-loop which automatically determines the
```

```
% image parameter to read for the current chunk
MR.Parameter.Chunk.CurLoop = cur_loop;

% ------------------------------------------------------
% Perform the Reconstruction for the Current Chunk (Start)
% ------------------------------------------------------

% spectro begin ----------------------------
if MR.Parameter.Labels.Spectro
    MR.ReadData;
    MR.RandomPhaseCorrection;
    MR.PDACorrection;
    MR.DcOffsetCorrection;
    MR.MeasPhaseCorrection;
    MR.SortData;
    MR.Average;
    MR.RemoveOversampling;
    MR.RingingFilter;
    MR.ZeroFill;
    MR.SENSEUnfold;
    MR.EddyCurrentCorrection;
    MR.CombineCoils;
    MR.GeometryCorrection;
    MR.K2I;
    %MR.RotateImage;
else
% spectro end ------------------------------
    MR.ReadData;
    MR.RandomPhaseCorrection;
    MR.RemoveOversampling;
    MR.PDACorrection;
    MR.DcOffsetCorrection;
    MR.MeasPhaseCorrection;
    MR.SortData;
    MR.GridData;
    MR.RingingFilter;
    MR.ZeroFill;
    MR.K2IM;
    MR.EPIPhaseCorrection;
    MR.K2IP;
    MR.GridderNormalization;
    MR.SENSEUnfold;
    MR.PartialFourier;
    MR.ConcomitantFieldCorrection;
    MR.DivideFlowSegments;
    MR.CombineCoils;
    MR.Average;
    MR.GeometryCorrection;
    MR.RemoveOversampling;
    MR.ZeroFill;
    MR.FlowPhaseCorrection;
    MR.RotateImage;
end

% The current chunk is now reconstructed. If the data is
% reconstructed in more than one chunk write the result to
% a temporary file on the disk.
if MR.Parameter.Chunk.NrLoops > 1
    [exported_datafile, exported_listfile] = ...
  MR.WriteExportedRaw( [MR.Parameter.Filename.Data,...
  '_temp.data'], MR.Parameter.Parameter2Read );
end
```

```
                % ------------------------------------------------------
                % Perform the Reconstruction for the Current Chunk (End)
                % ------------------------------------------------------
            end

            % If data has been written to a temporary file read it
            % again
            if MR.Parameter.Chunk.NrLoops > 1
                r_temp = MRecon(exported_datafile);
                r_temp.ReadData;
                r_temp.SortData;
                MR.Data = r_temp.Data;
                fclose all;
                delete(exported_datafile);
                delete(exported_listfile);
                clear r_temp;
            end
            if strcmpi( MR.Parameter.Recon.StatusMessage, 'yes')
                fprintf('\n');
            end
            MR.Parameter.Recon.AutoUpdateInfoPars = AutoUpdateStatus;
            MR.Parameter.Reset;
            case 'ExportedCpx'
            MR.ReadData;
            MR.SortData;
            MR.CombineCoils;
        case 'Cpx'
            MR.ReadData;
            MR.CombineCoils;
        case 'Rec'
            MR.ReadData;
            MR.RescaleREC;
            MR.CreateComplexREC;
        otherwise
            error( 'Error in Perform: Unknown data format' );
    end
end
```

Subclasses and Function Overloading

Even though the MRecon class contains all the basic reconstruction steps, its functionality is far from being complete to cope for all possible reconstructions which are available or presented in the MR community. If you want to implement your own reconstruction which differs from the Perform function above, but still want to use the basic functionalities from MRecon, then deriving a subclass is an elegant and easy way.
Deriving a class means creating your own reconstruction object, which has access to all functions and properties of its superclass (MRecon). Furthermore own functions can be implemented in the new derived class extending the functionality of its superclass. And last but not least existing functions can be overwritten by defining one with the same name in the derived subclass. This procedure is called function overloading. Such a modular design makes it very easy to create new, or extend an existing reconstruction without the need to rewrite or copy-paste existing code.

For a better illustration of this concept we have a look at an example where we create a new reconstruction object (My_Recon) and extend it by implementing an additional ringing (Hamming) filter. Furthermore we overwrite the existing Perform function of the MRecon class with our own one. The new Perform function should only deliver filtered k-space data instead of performing a full reconstruction:

```
% Create an own reconstruction object My_Recon by deriving it from MRecon
classdef My_Recon < MRecon
    properties
        % No additional properties needed
    end

    methods
        function MR = My_Recon( filename )
            % Create an MRecon object MR upon creation of My_Recon
            MR = MR@MRecon(filename);
        end

        % Overload (overwrite) the existing Perform function of MRecon
        function Perform( MR )
            %Reconstruct only standard (imaging) data
            MR.Parameter.Parameter2Read.typ = 1;
            % Produce k-space Data (using existing MRecon functions)
            MR.ReadData;
            MR.DcOffsetCorrection;
            MR.PDACorrection;
            MR.RandomPhaseCorrection;
            MR.MeasPhaseCorrection;
            MR.SortData;
            MR.GridData;

            % Filter k-space data using an own function defined in My_Recon
            MR.HammingFilter;
        end

        % Definition of the Hamming-filter function
        function HammingFilter( MR )
            % Prerequisites: Check if the data has the right form
            if ~MR.Parameter.ReconFlags.isread
                error( 'Please read the data first' );
            end
            if ~MR.Parameter.ReconFlags.issorted
                error( 'Please sort the data first' );
            end

            % MR.Data can be a cell or an array depending on the dataset.
            % To make our function work with all kinds of data, we convert
            % MR.Data to a cell in any case and then let the hamming filter
            % work on every cell element (using cellfun)
            MR.Data = MR.Convert2Cell( MR.Data );

            % Filter the Data by calling the filter function on every cell
            % element
            MR.Data = cellfun( @(x)MR.filter_image( x ), MR.Data, …
            'UniformOutput', 0 );

            % Convert MR.Data back to oroginal state
            MR.Data = MR.UnconvertCell( MR.Data );
        end

    end     % End of public methods


    % These functions are Hidden to the user
    methods (Static, Hidden)
        % Actual code for the Hamming filter (called by cellfun)
        function data = filter_image( data )
```

```
            if ~isempty( data )
                % Obtain the number of images in MR.Data
                data_size = size(data);
                nr_images = prod( data_size( 3:end) );
                % Define filter
                hfilter = (hamming(size(data,1))*hamming(size(data,2))');
                hfilter = hfilter/max(hfilter(:));
                % Filter every single image
                for i = 1:nr_images
                    data(:,:,i) = data( :,:,i) .* hfilter;
                end
            end
        end

    end           % End of hidden methods
end         % End of My_Recon class
```

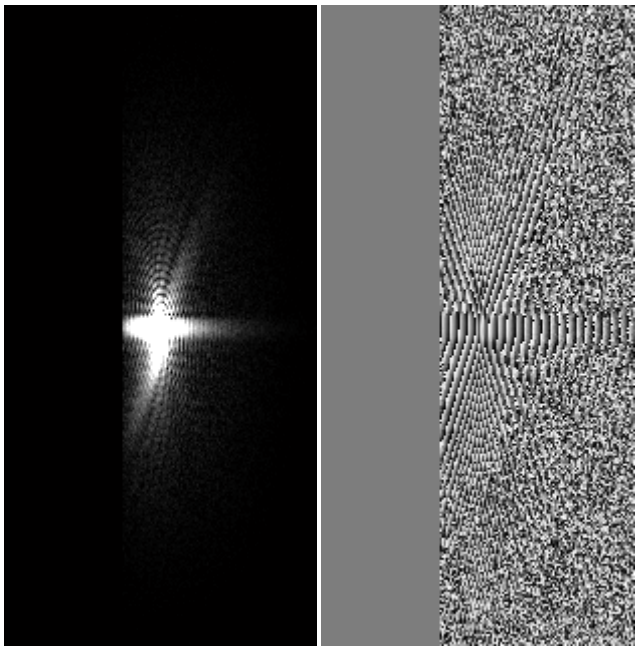
After implementing the code above, we can create our own reconstruction object by :

```
r = My_Recon( 'raw_file_name' ) ;
```

and perform the recon with:

```
r.Perform;
```

After that the r.Data array is filled with filtered k-space data seen below



Adding your own Parameter to MRecon

Sometimes it may be necessary to add an own parameter to the existing parameter groups in MRecon. Assume that we want to extend the example above by adding an own Recon Flag, checking if we already performed the Hamming filter or not. Own parameters can be added in the corresponding parameter files under *MRecon_rootdir*/userdata. In our example we would add our own Recon Flag to the file ReconFlagParsUsr.m as shown below:

```
classdef ReconFlagParsUsr < ReconFlagPars
```

```
        % User Defined Reconstruction Flags
        properties
            % Add your own parameter here
            ishammingfiltered = 0;
        end

        methods
            % Add your own functions here
        end

    end
```

We now have added our own Recon Flag which is visible under Parameter.ReconFlag after creating the reconstruction object:

```
clear classes;
r = My_Recon( 'raw_file_name' );
r.Parameter.ReconFlags

ans =

  ReconFlagParsUsr handle

  Properties:
    ishammingfiltered: 0
               isread: 0
             issorted: 0
            isgridded: 0
            isimspace: [0 0 0]
           iscombined: 0
       isoversampled: 0
     isreadparameter: 1
     israndphasecorr: 0
           ispdacorr: 0
      isdcoffsetcorr: 0
          isdepicorr: 0
     ismeasphasecorr: 0

  Methods, Events, Superclasses
```

As you can observe, our new parameter shows up on top of the existing ones. The « clear classes » command at the beginning is necessary to reinitialize all classes. Without it, our modifications would not have any effect.
The newly created Recon Flag can now be used in the HammingFilter function, for example:

```
% Definition of the Hamming-filter function
function HammingFilter( MR )
    % Prerequisites: Check if the data has the right form
    if ~MR.Parameter.ReconFlags.isread
        error( 'Please read the data first' );
    end
    if ~MR.Parameter.ReconFlags.issorted
        error( 'Please sort the data first' );
    end

    if MR.Parameter.ReconFlags.ishammingfiltered
        error( 'The data is already Hamming filtered' );
    end
```

```
        % MR.Data can be a cell or an array depending on the dataset.
        % To make our function work with all kinds of data, we convert
        % MR.Data to a cell in any case and then let the hamming filter
        % work on every cell element (using cellfun)
        MR.Data = MR.Convert2Cell( MR.Data );

        % Filter the Data by calling the filter function on every cell
        % element
        MR.Data = cellfun( @(x)MR.filter_image( x ), MR.Data, …
        'UniformOutput', 0 );

        MR.Parameter.ReconFlags.ishammingfiltered = 1;

        % Convert MR.Data back to oroginal state
        MR.Data = MR.UnconvertCell( MR.Data );
    end
```

Access to GoalC parameters and objects

New

When an MRecon object is created a selective set of parameters is directly stored in the object under r.Parameter. These parameters have been preselected as they are considered to be the most commonly used ones for the reconstruction. If however you need access to a parameter which is not already available in r.Parameter then you can get access to any GoalC parameter and object which was available during the acquisition of your scan (all parameters/objects which were visible in the Sequence Development Mode).

There is a set of functions available to obtain and display these parameters and objects:

Search for a parameter:
To search for a parameter you can use the Search function of MRecon:

```
r.Search('gamma');
```

You can also use it in combination with wildcards:

```
r.Search('EX_*gamma');
```

The search function searches the whole MRecon object and will display all parameters/object which match the search string.

Obtain the value of a parameter and object:
To obtain the value of a known parameter or object name you can use the following function:

```
val = r.Parameter.GetValue('UGN1_RNAV_beam_order');
```

The return value “val” is an array because the parameter is also an array. If for example we only want to obtain the first 2 elements we can use the second input argument:

```
val = r.Parameter.GetValue('UGN1_RNAV_beam_order', 1:2);
```

If the parameter type is an enumeration then val will be a string:

```
val = r.Parameter.GetValue('UGN1_RNAV_enable');
```


If we prefer a numeric value for the enumeration instead of the string we can use the third input argument:

```
val = r.Parameter.GetValue('UGN1_RNAV_enable', [], true);
```

We can also ask for values of an object attribute in the following way:

```
val = r.Parameter.GetValue('SQ`base:dur');
```

If the object is a composite object we should also specify the composite index (note that it starts at 1):

```
val = r.Parameter.GetValue('LCA`ima:[1]:prep_dir');
```

If we want to obtain the values for all composite indices we can also replace the index with the Matlab colon placeholder:

```
val = r.Parameter.GetValue('LCA`ima:[:]:prep_dir');
```

Get/Display all parameters in a GoalC group

The group information of GoalC is also preserved in MRecon. To display all parameters of a known group you can call:

```
r.Parameter.DisplayParameterInGroup('UGN1_RNAV_all_pars')
```

To obtain a cell array of all parameters in that group use:

```
P = r.Parameter.GetParameterInGroup('UGN1_RNAV_all_pars');
```

Display all objects names

To display all object names you can call:

```
r.Parameter.DisplayObjectNames();
```

To display all object names of a certain object class use:

```
r.Parameter.DisplayObjectNames('RF');
```

Display all attributes of an object

To display a whole object use:

```
r.Parameter.DisplayObject('SQ`base');
```

Please find a more complete description of all the parameter functions at the back of this document in the Parameter Methods section

Graphical representation of the used sequence

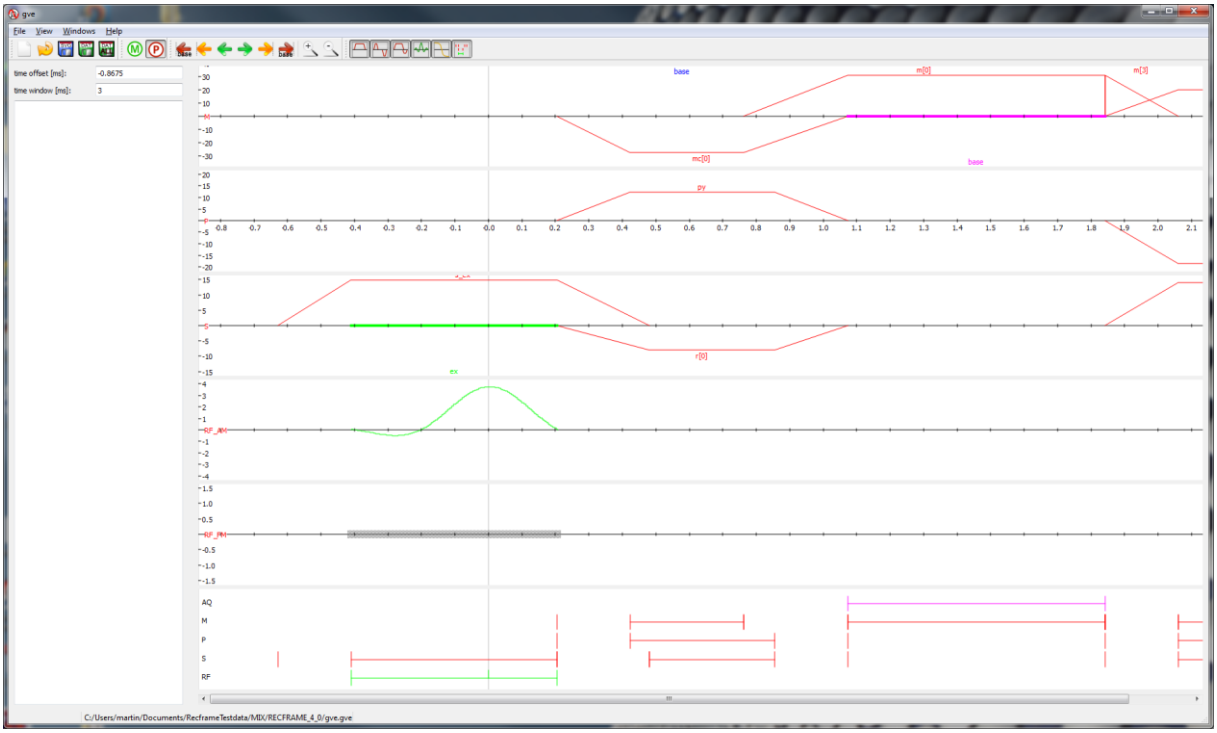
New

The raw file also contains all the information for a graphical representation of the MRI sequence. To display it do the following:

Extract the information and write it to a .gve file:

```
r.Parameter.ExtractPDFFile ('<filename.gve>');
```

Then open the .gve file with the Philips graphical viewer and display the PDF sequence:



Performing the Reconstruction on the Scanner

When you are finished developing your reconstruction offline, you can then perform it directly on the scanner console after acquiring the data. After the reconstruction is finished the images will be written back to the database, seemingly integrating ReconFrame into the workflow.
To perform your own reconstruction on the Scanner, the following steps have to be executed:

1. Modify ScannerReconstruction.m

After you are finished developing your reconstruction, open the file ScannerRecon.m which is located in: *MRecon installation path*\ScannerRecon\:

```
classdef ScannerRecon
    properties

    end

    methods
        function SR = ScannerRecon( lab_filename, raw_filename )

            % ---------- Place you own code here ----------

            r = MRecon(lab_filename, raw_filename);%
            r.Perform;
            r.Parameter.Recon.ExportRECImgTypes = {'M'};

            % ---------- Place you own code here ----------

            try
                r.WriteRec('G:\patch\pride\tempoutputseries\DBIEX.REC');
                r.WriteXMLPar('G:\patch\pride\tempoutputseries\DBIEX.XML');
             catch
                r.WriteRec('DBIEX.REC');
                r.WriteXMLPar('DBIEX.XML');
            end
        end
    end
end
```

Between the lines “Place your own code here”, place whatever code is necessary to reconstruct the final images from the raw data. In the example above an MRecon object is created and a standard reconstruction is performed. If for example you have written an own reconstruction object “My_Recon” with a “Perform” function then simply replace “MRecon” with “My_Recon”.

2. Compiling, Packing and Installation on the Scanner

In order to run our reconstruction on the scanner, the Matlab code has to be compiled. There are two ways to do that, manually or using the matlab’s DeployTool. Both methods are described below. *Please note that Matlab’s compiler toolbox has to be installed to execute the following steps. Also, for compatibility, we strongly recommend using the 32bit version of Matlab for compilation.*

Manual Installation

Compiling

In a first step the Matlab compiler has to be configured. To do so, please execute the following command in the Matlab command line:

```
>> mbuild –setup
Please choose your compiler for building standalone MATLAB
applications:

Would you like mbuild to locate installed compilers [y]/n? y
```

When asked to locate the installed compilers, say yes:

```
Select a compiler:
[1] Lcc-win32 C 2.4.1 in C:\PROGRA~2\MATLAB\R2010a\sys\lcc
[2] Microsoft Visual C++ 2005 SP1 in C:\Program Files
(x86)\Microsoft Visual Studio 8

[0] None

Compiler:
```

Afterwards a list of all installed compilers is displayed. *You can choose either one, but for compatibility we recommend to use the Matlab Lcc Compiler.*
This configuration has to be performed only once. For future compilations the settings will remain.

After configuration, we can start compiling out code with the command:

```
>> mcc -m ScannerRecon.m
```

This will produce an executable ScannerRecon.exe in the local directory after completion. *Please make sure that all the programs you need for your reconstruction to run are either in the Matlab path or in the local directory!*

Packing the required files for the scanner

When the compilation is performed manually we have to make sure to get all the files which are required to run our own reconstruction on the scanner. Therefore you should collect:

- The compiled executable ScannerRecon.exe
- The Matlab runtime environment “MCRInstaller.exe” which is located in:
  MATLAB_root\toolbox\compiler\deploy\win32 (32bit version)
  MATLAB_root\toolbox\compiler\deploy\win64 (64bit version)
- The license file “license.lic” which is located in: *MRecon installation path*\license\

When you have collected these files, copy them to a memory stick or transfer them to the scanner over the net.

Installation on the Scanner

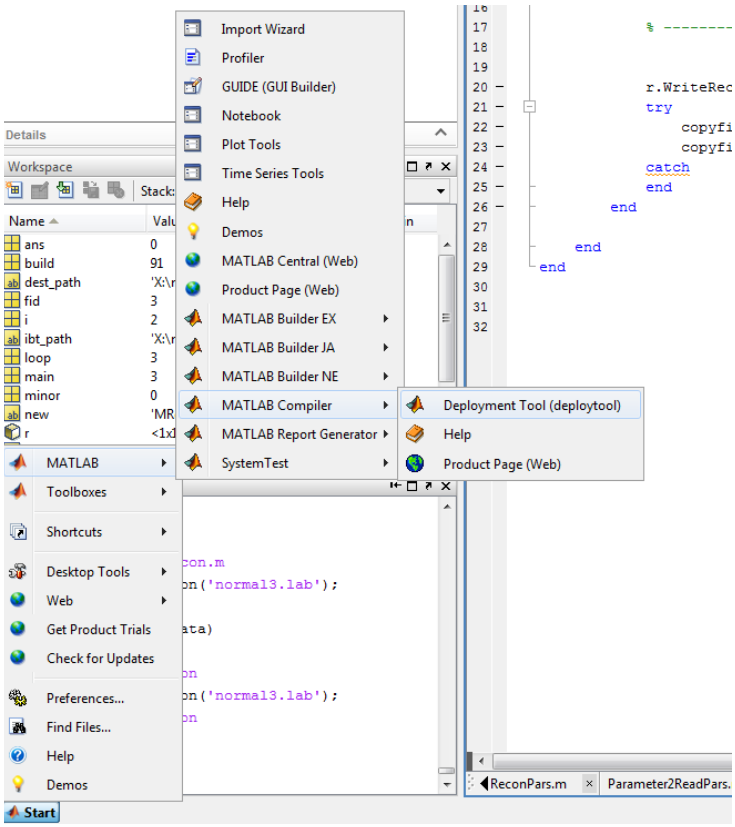
To install the reconstruction on the scanner, execute the following steps:

1. Install the Matlab runtime environment “MCRInstaller.exe”. *The runtime environment only has to be installed once on the scanner!*
2. ***Reboot the scanner console!***
3. Copy “ScannerRecon.exe” and “license.lic” to:
   G:\Site\AvailablePatches\ReconFrame\pride\ScannerRecon\

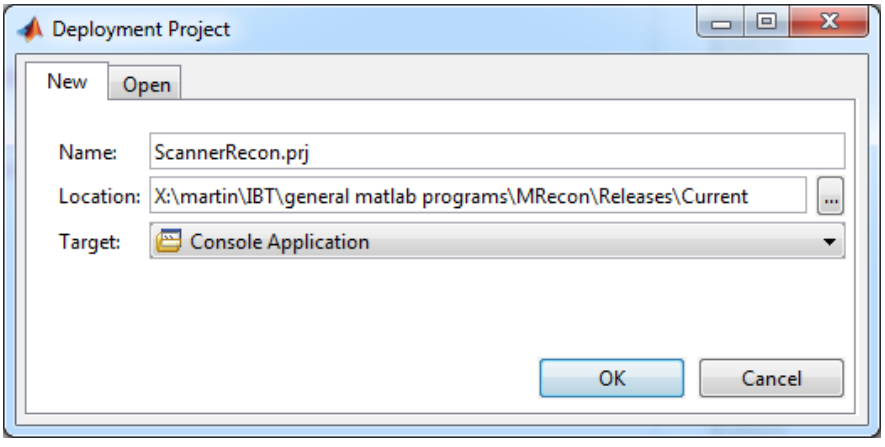
4. Start the ReconFrame Patch

Using the Matlab Deployment Tool

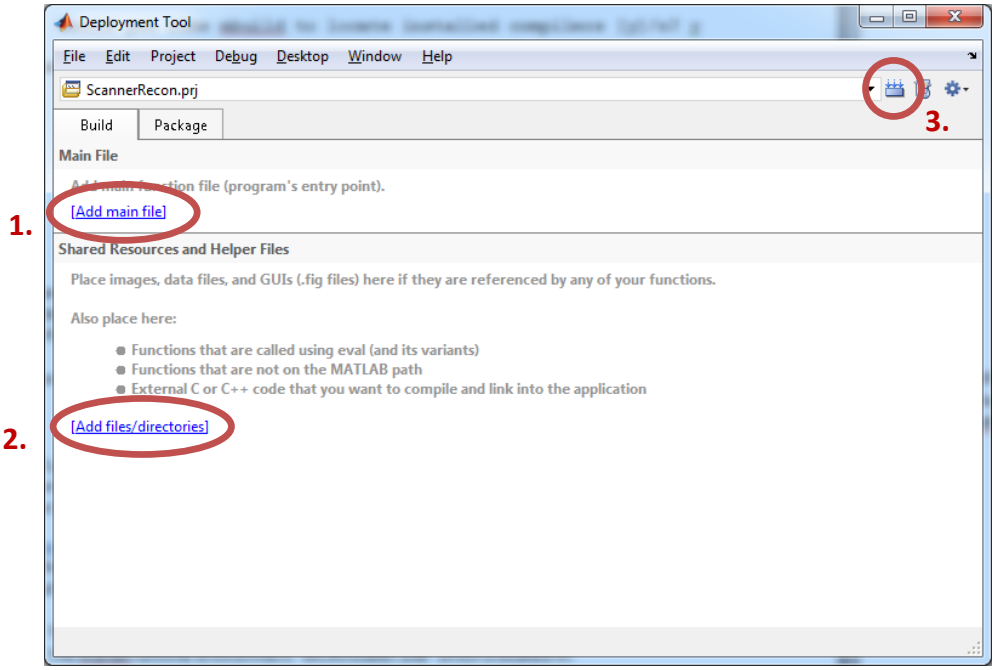
Another way to compile and pack the necessary files is to use the Matlab Deployment Tool. The Deploy Tool is a graphical user interface for compiling and packaging Matlab programs.
The tool can be started over the Matlab Start button:



After Starting it the project dialog appears. Choose a project name (e.g. ScannerRecon.prj) a location where to save your project (e.g. *MRecon installation path*\ScannerRecon\) and as Target choose "Console Application". Afterwards click OK

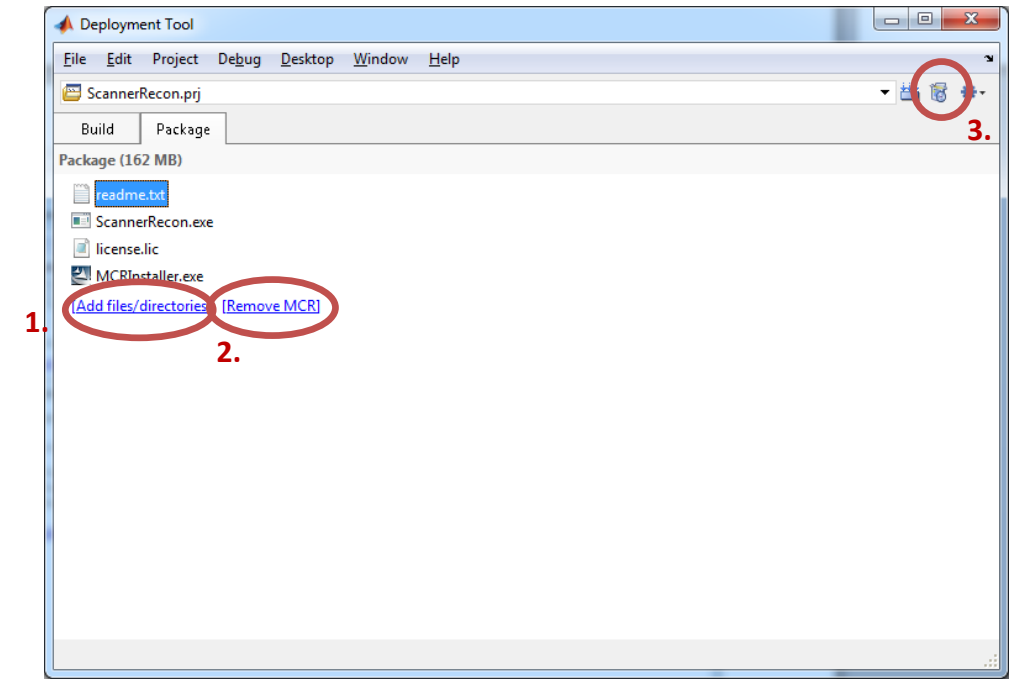


After the deployment tool has opened, execute the following steps:



1.) Click on “Add main file” and add the ScannerRecon.m, which is located in *MRecon installation path*\ScannerRecon\
2.) If any additional files are needed for the compilation (c-code, Matlab figures etc.) they can be added under “Add files/directories”. Usually nothing has to be added.
3.) When finished, click on “Build”

When the compilation is complete click on the “Package” tab



1.) Click on “Add files/directories and add the license file “license.lic” located in *MRecon installation path*\license\
2.) Click on “Add MCR”
3.) When finished click on “Package” and chose a location for the file to be saved.

After completion copy the resulting file *ProjectName*_pkg.exe (e.g. ScannerRecon_pkg.exe) in the selected output directory to a memory stick or copy it to the scanner over the net.

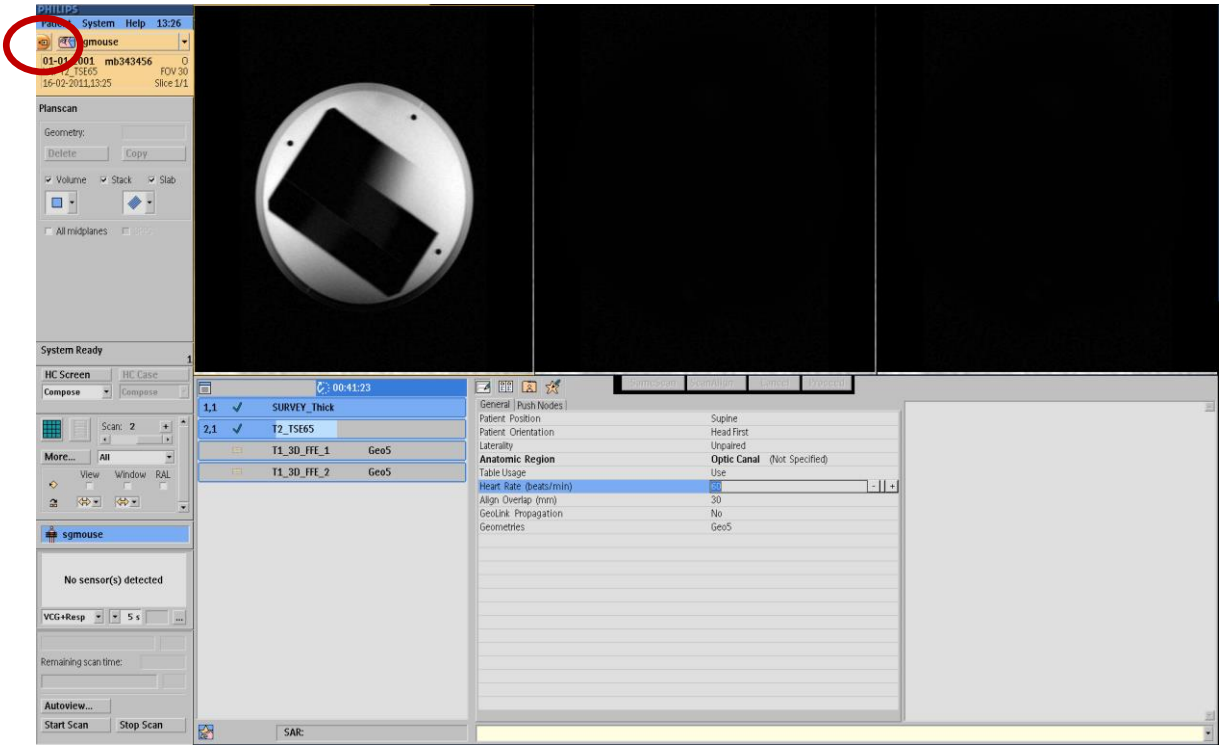
Installation on the Scanner

To install the reconstruction on the scanner, execute the following steps:

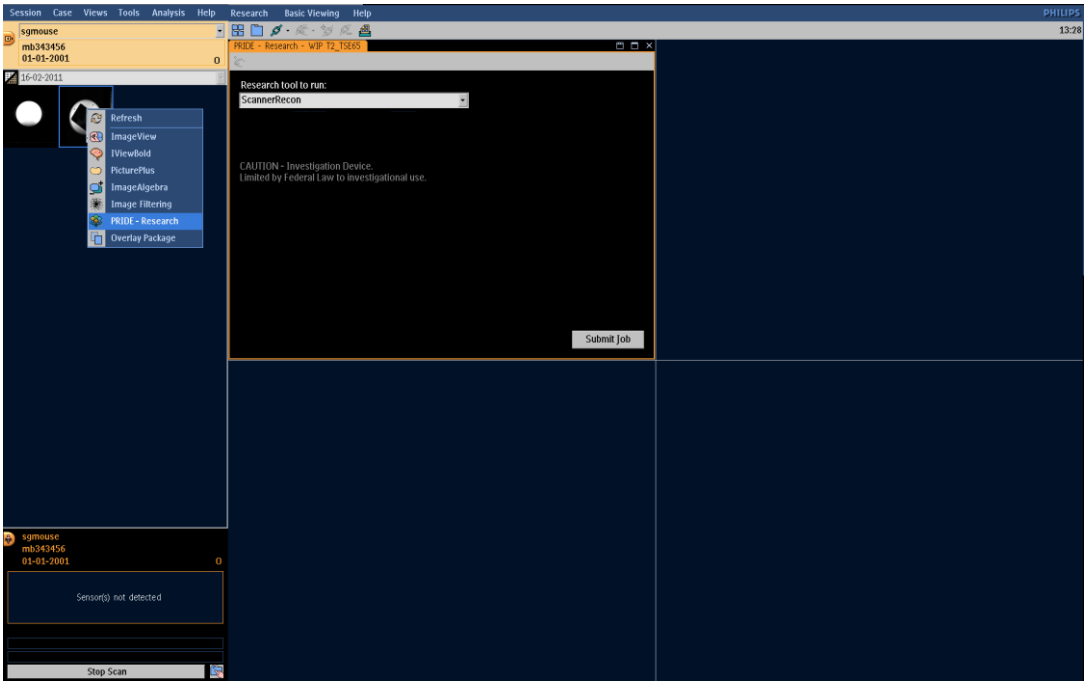
1. Plugin the Memory Stick and double click on *ProjectName*_pkg.exe (e.g. ScannerRecon_pkg.exe)
2. Wait until the files are unpacked and the runtime environment has been installed
3. ***Reboot the scanner console!***
4. Copy "ScannerRecon.exe" and "license.lic" to: G:\Site\AvailablePatches\ReconFrame\pride\ScannerRecon\
5. Start the ReconFrame Patch

2. Performing the Reconstruction on the Scanner

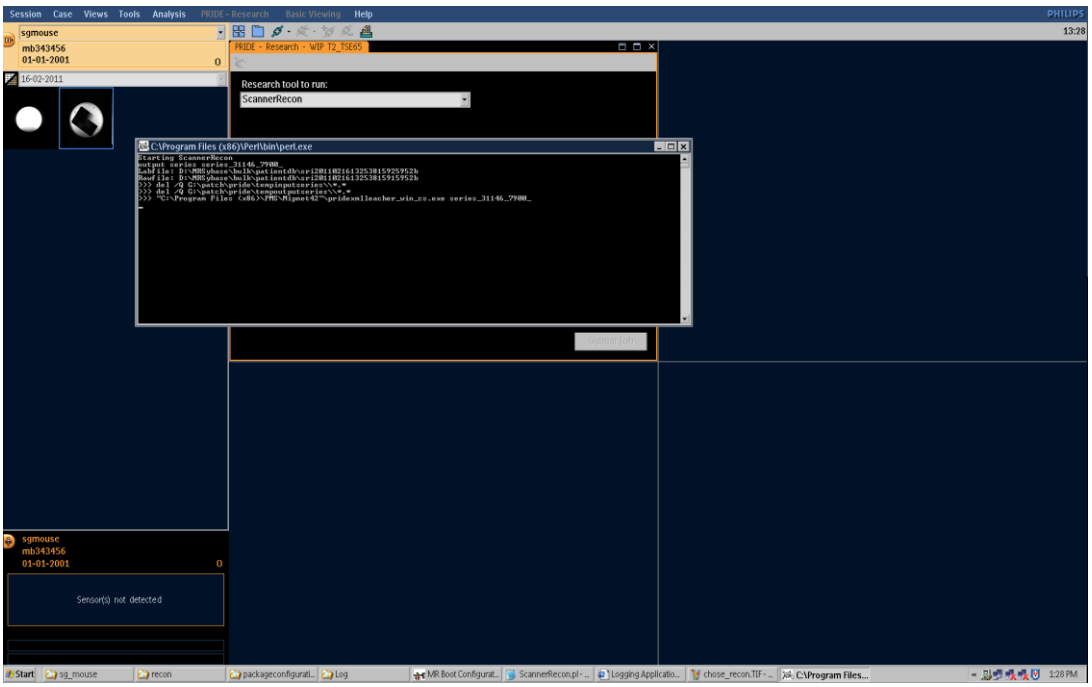
1. Acquire a scan where "Save raw data" is switched to "yes" in the "Postproc" tab
2. After completion of the scan switch to the view context



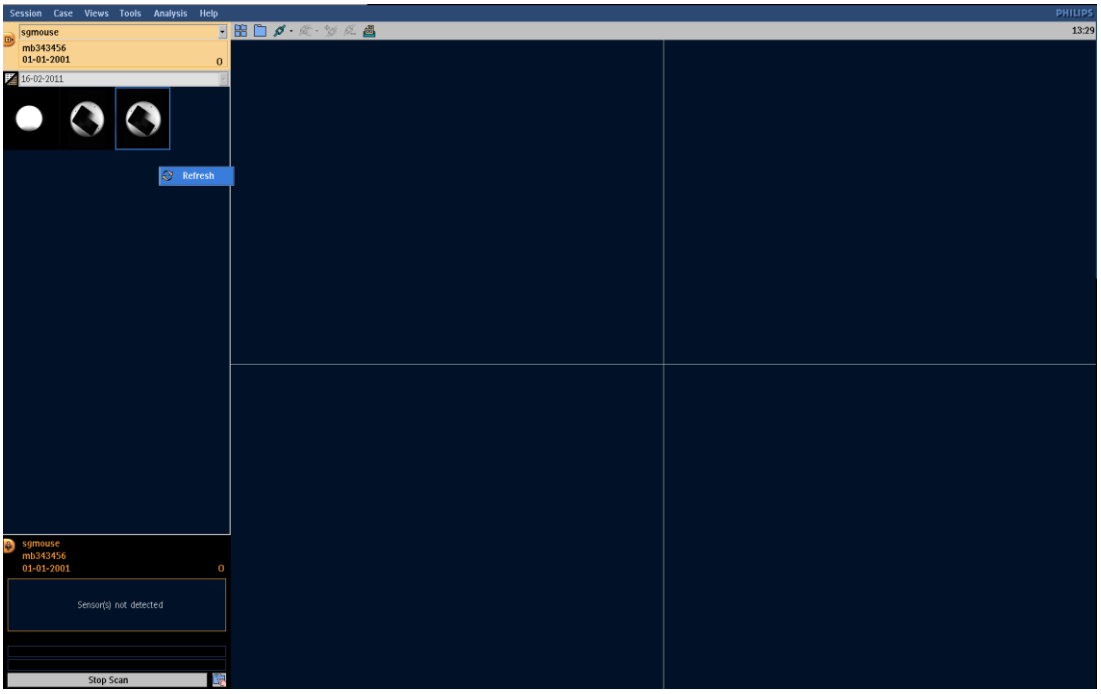
3. Right click on the acquired scan and chose "PRIDE Research"
4. In the window on the right chose "ScannerRecon" from the drop down menu and click on "Submit Job"



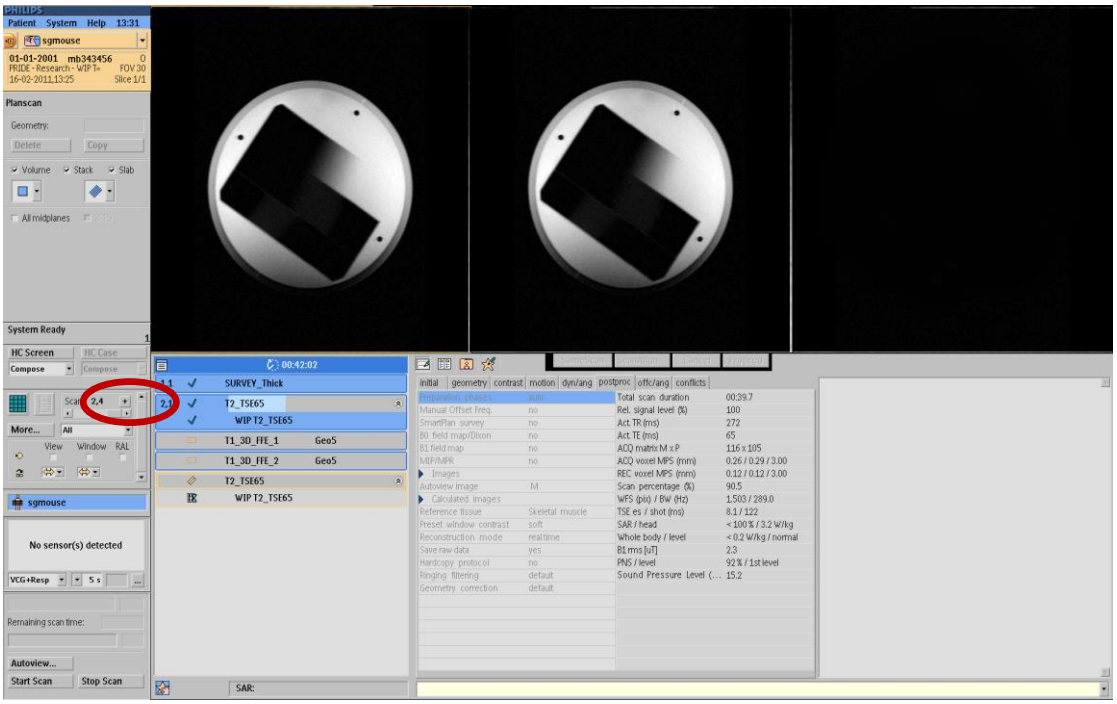
5. Your own reconstruction will start and a black command window appears. Wait until it has disappeared again



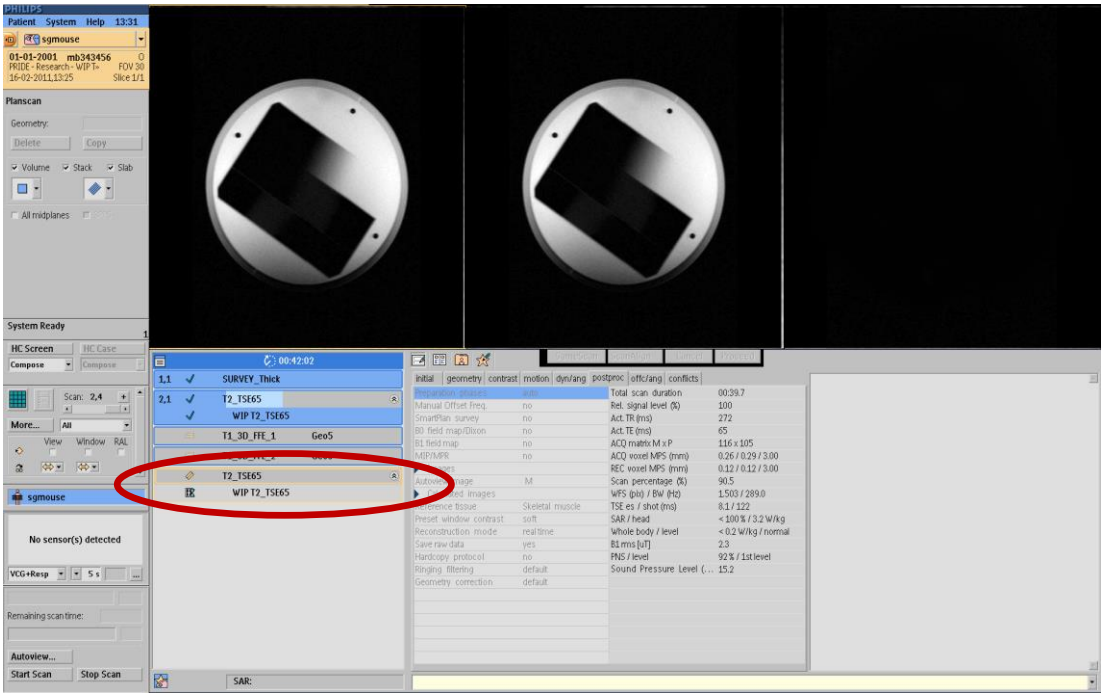
6. Wait a few seconds until the newly reconstructed scan appears or right click in the left window and chose “refresh”. The new scan is now reconstructed with your own recon.



7. If you switch back to the scan environment, you can load the newly reconstructed scan by clicking on the little “+” symbol



8. If you now copy the just reconstructed scan, your own reconstruction will automatically start after the scan and you don’t have to repeat the process above.


PHILIPS
Patient System Help 13:31
sgmouse
01-01-2001 mb343456
Planscan
Geometry:
Delete
Copy
Volume
Stack
Slab
All midplanes
System Ready
HC Screen
HC Case
Compose
Scan: 2,4
More...
All
View
Window
RAL
sgmouse
No sensor(s) detected
VCG+Resp
5 s
Remaining scan time:
Autoview...
Start Scan
Stop Scan
00:42:02
1,1 SURVEY_Thick
2,1 T2_TSE65
WIP T2_TSE65
T1_3D_FFE_1 Geo5
T2_TSE65
WIP T2_TSE65
SAR:
initial
geometry
contrast
motion
dyn/ang
postproc
offc/ang
conflicts
Total scan duration 00:39.7
Rel. signal level (%) 100
Act. TR (ms) 272
Act. TE (ms) 65
ACQ matrix M x P 116 x 105
ACQ voxel MPS (mm) 0.26 / 0.29 / 3.00
REC voxel MPS (mm) 0.12 / 0.12 / 3.00
Scan percentage (%) 90.5
WFS (pix) / BW (Hz) 1.503 / 289.0
TSE es / shot (ms) 8.1 / 122
SAR / head < 100 % / 3.2 W/kg
Whole body / level < 0.2 W/kg / normal
B1 rms [uT] 2.3
PNS / level 92 % / 1st level
Sound Pressure Level (... 15.2

MRecon Methods

Average

Averages the data in the 12th dimension of the data array.

Syntax: r.Average;

Parameters used:

- Parameter.Scan.Diffusion:
Used to check if it is a diffusion scan. Only the magnitude is averaged for diffusion scans since phace cancelations can occur otherwise.

Location: k-space | image-space

Formats: Raw | ExportedRaw

Description/Algorithm: This function averages the data in the 12th dimension of the data array (where the different averages are stored). This function only has an effect if immediate averaging is disabled (r.Parameter.Recon.ImmediateAveraging). If immediate averaging is enabled, then the data is averaged in the SortData function already. Some data, however should not be averaged in k-space (e.g. diffusion scans). That is why this function is called in image space to average the remaining averages which have not been processed yet.

Notes: For diffusion scans only the magnitude is averaged.

CombineCoils

Combines the individual coil images into one single image. Several combination methods have been implemented in MRecon (see below).

Syntax: r.CombineCoils

Parameters used:

- Parameter.Recon.CoilCombination: {'sos'} | 'pc'
When set on 'sos' a sum-of-squares combination is performed. Note that the image phase is lost with this method. 'pc' is used in phase contrast flow reconstructions where the magnitude is a sum-of-squares combination and the phase is a magnitude weighted combination of the individual phase images.

Recon Flag: Parameter.ReconFlags.iscombined

Location: Image-space.

Formats: Raw | ExportedRaw | Cpx | Bruker

Notes: - For a non-phase-contrast scan the image phase is lost after CombineCoils. If you want to preserve the phase please perform a SENSE/CLEAR reconstruction see the SENSEUnfold function.

- The CombineCoils function destroys the label lookup table (Parameter.LabelLookupTable) since the profiles cannot be related to the labels after combining the coils.
- The coils have to be in the 4th dimension of the Data array to be combined.

Compare

Compares to MRecon classes and displays the differences

Syntax: r.Compare(r_other);

Parameters used: All

Formats: All

Description/Algorithm: The compare function iterates through all the parameters values of the MRecon class and compares them with the values of another reconstruction object.

Examples: To compare the current reconstruction object (r) with another one (r1) call:

```
>> r.Compare(r1)
```

ConcomitantFieldCorrection

Performs a concomitant field correction on the current data.

Syntax: r.ConcomitantFieldCorrection;

Parameters used:

- **Parameter.Recon.ConcomitantFieldCorrection: {'yes'} | 'no'**
  Enables/disables the concomitant field correction.
- **Parameter.Labels.ConcomFactors:**
  holds the correction factors.
- **Geometry parameters:**
  The correction has to be performed in the scanner fixed xyz-coordinate-system. Therefore all necessary geometry parameters for the coordinate transformation (Transform function) have to be available.

Recon Flag: Parameter.ReconFlags.isconcomcorrected

Location: Image-space.

Formats: Raw

Description/Algorithm: Maxwell's equations imply that imaging gradients are accompanied by higher order spatially varying fields (concomitant fields) that can cause artifacts in MR imaging. The lowest order concomitant fields depend quadratically on the imaging gradient amplitude and inversely on the static field strength. On the scanner these extra magnetic fields are expressed along the directions of the physical gradients (scanner fixed xyz-coordinate system). Therefore the images have to be transformed into that system before the correction. The correction itself adds a quadratic phase to the images defined by the correction parameters.

Notes: - The concomitant field correction will not work on data measured without the ReconFrame patch.
- The concomitant field correction is only executed for phase contrast flow scans. Only then correction parameters are provided by the scanner.

DcOffsetCorr

Corrects for a signal offset that might occur in the acquired samples.

Syntax: r.DcOffsetCorrection;

Parameters used:

- Parameter.Recon.DcOffsetCorrection: {'yes'} | 'no'
Enables/disables the DC-offset correction.
- Parameter.DataFormat:
Used to check if raw data is passed to the function
- Parameter.LabelLookupTable:
Defines the profiles which are currently stored in the Data array.
- Parameter.Encoding.KxRange:
If no noise samples are present in the raw file, this parameter specifies where the outer k-space (noise like) samples are located.

Recon Flag: Parameter.ReconFlags.isdcoffsetcorr

Location: k-space (before the partial Fourier reconstruction)

Formats: Raw

Description/Algorithm: Acquired raw samples can sometimes exhibit a signal offset originating from hardware inaccuracies of the receiver. If not corrected this offset can cause an artifact which will appear as bright signal peak in the middle of the image. Since noise in the MR signal is normally distributed with a mean of zero, the offset can be calculated by calculating the mean of the noise samples. Therefore if the raw data contains noise samples (from a noise acquisition prior to the scan), these samples are read and the mean is calculated. If the data does not contain any noise, the offset is calculated from the noise like outer k-space positions. After calculating the offset it is subtracted from the data.

Notes: The offset is calculated separately for every channel.

EPIPhaseCorrection

Performes the EPI correction to correct for the FOV/2 ghost originating from eddy current effects.

Syntax: r.EPIPhaseCorrection;

Parameters used:

- Parameter.Recon.EPIPhaseCorrection: {'yes'} | 'no'
Enables/disables the EPI correction.
- Parameter.DataFormat:
Used to check if the right data format is passed to the function
- Parameter.EPICorrData:

The linear/nonlinear phase used in the EPI correction. If a linear correction is used (default) then the phase is described by a slope and offset value. For the nonlinear phase correction the full correction profile is given. If this parameter is empty these values are calculated in the function itself. The parameter can also be filled with user defined slopes and offsets prior to the correction. Please see the example below on how to do that.

- **Parameter.ReconFlags.iszerofilled, Parameter.ReconFlags.isoversampled and Parameter.ReconFlags.isgridded:**
  Used to check in what reconstruction state the data currently is. The correction profiles have to be brought into the same state to match the imaging data.
- **Parameter.Recon.EPICorrectionMethod: {linear} | nonlin**
  Enables/disables the non-linear EPI correction (see below).
- **Parameter.Recon.EPI2DCorr: yes | {no}**
  Enables/disables the image based EPI correction (see below).
- **Encoding Parameter:**
  Used to read and format the correction profiles approprately.

Recon Flag: Parameter.ReconFlags.isdepicorr

Location: Between FFT in readout and phase encoding direction.

Formats: Raw | ExportedRaw | Bruker

Description/Algorithm: In EPI imaging multiple k-space lines are acquired after a single excitation of the imaging volume. This is achieved by inverting the readout gradient after every acquired profile and thus sampling the k-space in a forward-backward fashion. Due to eddy current effects however, the positive readout gradients (forward sampling) is not exactly the same as the negativ gradient (backward sampling), which results in a temporal shift between the forward and backward sampled profiles. After fourier transformation this shift results in a ghost like artifact located at half of the FOV.

To correct for this effect EPI correction profiles are acquired prior to every scan. The correction profiles consist of a full EPI shot, but acquired without phase encoding blips, therefore sampling the central k-space line multiple times. Since these profiles should be identical (apart from relaxation effects), the temporal shift, caused by eddy current effects of the readout gradients, can be calculated. A temporal shift in k-space corresponds to a linear phase in image space. Therefore the correction profiles are first transferred into image space and the phase difference between two subsequent profiles is calculated. The phase difference is then linearly fitted and the linear phase is added to every odd (or even) profile, thus realigning them in k-space.

Optionally a nonlinear correction can be enabled via Parameter.Recon.EPICorrectionMethod. In the nonlinear correction the calculated linear phase is subtracted from the original phase difference. The remaining part is then smoothed and added to the linear phase.

Optionally an image based correction can be enabled via Parameter.Recon.EPI2DCorr. This correction tries to improve the images further by minimizing the amount of ghosts in the image. Thereby the offsets found by the regular EPI crrection are slighly modified and the amount of ghosts in the resulting images is quantified, to find the optimal correction value.

Notes: - Every coil has its own EPI correction parameters.
- The x-axis for the linear fit is given by:

```
x = -floor(n/2):ceil(n/2)-1;
```

where n is the number of samples in the correction profiles.
- The linear fit is weighted by the magnitude of the correction profiles (points with a high magnitude value are weighted higher in the fit)
- The EPI correction profiles are labeled as data type 3 and can be read by setting: r.Parameter.Parameter2Read.typ = 3, followed by r.ReadData.

Examples: The user has the possibility to specify own correction parameters and pass them to the function. To do that Parameter.EPICorrData must be filled:

Every coil has its own slope and offset. To identify the coil we must label them in the correction parameter:

```
r.Parameter.EPICorrData.coil = r.Parameter.Parameter2Read.chan;
```

Assign the same slope for all the coils:

```
r.Parameter.EPICorrData.slope = 0.1;
```

Assign a different offset for every coil (4 coils in this example):

```
r.Parameter.EPICorrData.offset = [0, 0.2, 0.4, 0.6];
```

Call the EPI correction:

```
r.EPIPhaseCorrection;
```

ExportLabels

Exports the imaging labels, which are stored in the .lab file to a textfile

Syntax: r.ExportLabels(filename);

Parameters used:

- Parameter.Labels.Index:
The labels to be exported.

Location: k-space, image-space.

Formats: Raw | ExportedRaw | Bruker

Description/Algorithm: This function exports the imaging labels, which were sent from the spectrometer to the reconstructor, to a textfile. Every row in the file corresponds to an measured k-space line in the order they have been acquired. Exporting the labels is useful to get an overview over the sampling order for example.

Notes: The format of the textfile is virtually identical to the one of the .list file of the exported raw data format.

GeometryCorrection

Corrects for gradient non-linearity's over large FOV's

Syntax: r.GeometryCorrection;

Parameters used:

- **Parameter.Recon.GeometryCorrection: {'yes'} | 'no'**
  Enables/disables the geometry correction.
- **Parameter.Labels.GeoCorrPars:**
  Holds the correction factors.
- **Geometry parameters:**
  The correction has to be performed in the scanner fixed xyz-coordinate-system. Therefore all necessary geometry parameters for the coordinate transformation (Transform function) have to be available.

Recon Flag: Parameter.ReconFlags.isgeocorrected

Location: Image-space.

Formats: Raw

Description/Algorithm: Imaging gradients are not perfectly linear over the whole possible FOV. The further away we are from the isocentre the stronger the non-linear effects become, which will result in a slight distortion of the resulting image. This effect is corrected with the correction parameters provided by the scanner.

Notes: The correction parameters are scanner specific and vary with the gradient system.

GridData

GridData: Grids k-space data on a predefined trajectory.

Syntax: r.GridData;

Parameters used:

- **Parameter.Recon.Gridding: {'yes'} | 'no'**
  Enables/disables the gridding.
- **Parameter.Gridder:**
  The gridder parameter.
- **Parameter.Labels.NusEncNrs:**
  The non-uniform sample position for EPI scans. In EPI acquisitions the samples in readout direction are not uniformly spread due to the sampling during gradient ramp up. Therefore EPI scans are usually gridded in readout direction using the k-space positions specified here.

Recon Flag: Parameter.ReconFlags.isgridded

Location: k-space after SortData.

Formats: Raw | ExportedRaw

Description/Algorithm: The fast fourier transform (FFT) requires the data to be on a regular cartesian grid. Therefore in order to use the FFT on non-cartesian data such as radial or spiral acquistions, we have to grid the k-space data from the nominal trajectory to a cartesian grid. There are several predefined trajectories already built-in into MRecon, including: 2D-radial,

3D-stack of radials, 3D-kooshball, 2D/3D spiral, EPI ramp sampling. On the other side, user defined trajectories can easily be defined by setting the Gridder Parameters. Please see the example below on how to do that. If the Gridder Parameters are not set manually MRecon automatically detects the type of the scan and decides if it should be gridded or not.

Notes: - For non-cartesian scans the zero-filling process in k-space and oversampling removal is automatically performed during gridding, by chosing the grid appropriately. See example 3)
- The Gridder parameter can also be filled using the GridderCalculateTrajectory function.

Examples: **1) Reconstruct a golden angle radial acquisition by changing the radial angles in the gridder parameters:**

Golden angle scans exhibit a radial angle increment of 111.246°, whilethe rest is exactly the same as a "normal" radial trajectory. Therefore all we have to do to reconstruct a golden angle scan is to change the radial angles in the gridder parameter:

Create a new reconstruction object with a golden angle scan:

```
r = MRecon('golden_angle_scan.raw');
```

Calculate the total number of phase encoding profiles:

```
nr_profs = r.Parameter.Scan.Samples(2) * ...
r.Parameter.Scan.Samples(3)
```

Set the radial angles to an increment of 111.246°

```
r.Parameter.Gridder.RadialAngles = 0:pi/180*111.246: ...
(nr_profs-1)*pi/180*111.246;
```

Perform the recon

```
r.Perform;
```

2) Define a completely new trajectory and weights.

In this example we mirror the final image in readout direction by gridding a cartesian scan to a inverted kx trajectory. To do that we define a completely new trajectory and pass it to the gridder parameters:

Create a new reconstruction object using a cartesian scan:

```
r = MRecon('cartesian.raw');
```

Obtain the number of samples in every direction:

```
no_kx_samples = r.Parameter.Scan.Samples(1);
no_ky_samples = r.Parameter.Scan.Samples(2);
no_kz_samples = r.Parameter.Scan.Samples(3);
```

Create a regular cartesian grid. These are the sampled k-space coordinates of a cartesian scan. By definition the k-space coordinates of a sampled trajectory are in the range of -floor(n/2):ceil(n/2)-1, where n is the number of samples:


```
 [kx,ky,kz] = ndgrid( -floor(no_kx_samples / 2):ceil(
no_kx_samples /2)-1, ...
-floor(no_ky_samples / 2):ceil( no_ky_samples /2)-1, ...
-floor(no_kz_samples / 2):ceil( no_kz_samples /2)-1 );
```

Modify the k-space grid by inverting the readout direction:

```
kx_inverted = -kx;
```

Create the modified k-space. These are the coordinates we grid on. By definition the coordinates must be of dimension: no_samples x no_y_profiles x no_z_profiles x 3 (See Gridder Parameters for more information):

```
k = cat( 4, kx_inverted, ky, kz );
```

Assign the new trajectory to the Gridder:

```
r.Parameter.Gridder.Kpos = k;
```

Assign gridder weights (sampling density). In this example the samples are uniformly distributed on the whoe k-space (all weights are set to 1):

```
r.Parameter.Gridder.Weights = ones(no_kx_samples, ...
no_ky_samples, no_kz_samples);
```

Perform the reconstruction:

```
r.Perform;
```

Show the data. Observe that the image is flipped in readout direction compared to the ungridded reconstruction (a flip in k-space corresponds to a flip in image-space):

```
r.ShowData;
```

3) Gridder and matrix sizes:

Oversampling:

During the gridding process we have the possibility to sample on an arbitrary grid. This means that we can for example sample on a finer or coarser grid than the acquired one, which corresponds to add or remove oversampling and therefore changing the FOV in the reconstructed image. The parameter which specifies the amount of oversampling introduced by the gridder is Parameter.Gridder.GridOvsFactor. For example if we set:

```
r.Parameter.Gridder.GridOvsFactor = 2;
```

then we grid to half the sampled k-space distance and therefore to twice the FOV. A value smaller than 1 corresponds to a coarser grid than acquired and therefore a smaller FOV.

In radial scans the acquired oversampling factor in readout direction is always 2 (hardcoded in the pulse programming environment). However If we check the oversampling factor in

r.Parameter.Encoding.KxOversampling we can usually find a value like 1.25. This is the oversampling factor AFTER gridder, which means that we should set:

```
r.Parameter.Gridder.GridOvsFactor =
r.Parameter.Encoding.KxOversampling / 2;
```

This is done automatically done for radial scans, and means that we grid to a coarser grid leaving an oversampling factor of 1.25.

Zero Filling:

Apart from gridding on a finer grid we also have the possibility to grid on a larger/smaller k-space and therefore changing the resolution of the scan. Gridding to a larger k-space means that we perform zero filling whlie gridding to a smaller one is reducing the resolution of the scan. The parameter which specifies the size of the k-space to be gridded on is: Parameter.Gridder.OutputMatrixSize.

The output size is the matrix size after gridding. Lets assume we have a radial scan with 256 acquired samples. If we set:

```
r.Parameter.Gridder.GridOvsFactor = 1;
r.Parameter.Gridder.OutputMatrixSize = [512,512,1];
```

The the output matrix has the size 512x512 which means that we have zero filled the 256x256 sampled k-space to that value. However if we set:

```
r.Parameter.Gridder.GridOvsFactor = 2;
r.Parameter.Gridder.OutputMatrixSize = [512,512,1];
```

Then we have gridded it to a grid twice as fine but NOT zero-filled. To zero-fill and oversample we have to set:

```
r.Parameter.Gridder.GridOvsFactor = 2;
r.Parameter.Gridder.OutputMatrixSize = [1024,1024,1];
```

Summary:

For radial scans the zero filling and oversampling removal usually takes place directly in the gridder. Below is an example from a real radial experiment:

```
Number of acquired samples: 464 (oversampling factor = 2)
r.Parameter.Encoding.KxOversampling = 1.25
r.Parameter.Encoding.XRes = 240
```

--> Matrix size after gridding = 240 * 1.25 = 300
--> Gridding oversampling = 1.25 / 2 = 0.625

--> Set gridder parameters to:

```
r.Parameter.Gridder.GridOvsFactor = 0.625;
r.Parameter.Gridder.OutputMatrixSize = [300,300,1];
```


GridderCalculateTrajectory

Calculated the gridder trajectory for the current scan and fills the gridder parameter

Syntax: r.GridderCalculateTrajectory;

Parameters used:

- **Parameter.Gridder:**
  The gridder parameter.
- **Parameter.Labels.NusEncNrs:**
  The non-uniform sample position for EPI scans. In EPI acquisitions the samples in readout direction are not uniformly spread due to the sampling during gradient ramp up. Therefore EPI scans are usually gridded in readout direction using the k-space positions specified here.

Location: k-space after SortData.

Formats: Raw | ExportedRaw

Description/Algorithm: When the gridder parameter are not defined by the user, they are calculated internally and will not be visible or changeable. If one wants to observe or change the gridding parameter, this function can be called which calculates the trajectory for the gridding process and fills the gridder parameter accordingly. The function can be called for any preset scan (radial, spiral, EPI), before calling the GridData function. This is useful if the user wants to modify a preset trajectory without calculating it by himself. To do so one would call this function, modify the gridder parameter followed by the call of GridData.

GridderNormalization

Corrects for the signal weighting in image-space originating from the convolution with a bessel kernel in the gridding process.

Syntax: r.GridderNormalization;

Parameters used:

- **Parameter.Gridder:**
  The gridder parameter.
- **Parameter.ReconFlags.isgridded:**
  Used to check if the data was gridded.
- **Parameter.Encoding.WorkEncoding.KxOversampling:**
  The resulting oversampling after gridding, needed in the normalization.

Location: Image-space.

Formats: Raw | ExportedRaw

Description/Algorithm: Gridding is essentially a convolution with a bessel kernel in k-space. In image-space this corresponds to a multiplication with the fourier transform of the bessel-kernel leaving a signal weighing on the reconstructed image. This function corrects for that by dividing the image by the fourier-transform of the bessel kernel.

I2K

Performs a fourier transformation from image-space to k-space

Syntax: r.I2K;

Parameters used:

- **Parameter.Encoding.FFTDims:**
  Specifies in which dimension the fourier transformation should be executed. This is a 3 elements vector specifying the dimension in matrix notation.
- **Parameter.ReconFlags.isimspace:**
  Used to check which dimensions are already in image space

Recon Flag: Parameter.ReconFlags.isimspace

Location: Image-Space

Formats: Raw | ExportedRaw | Cpx | Rec | Bruker

Description/Algorithm: I2K first checks which dimensions should be transformed, defined in Parameter.Encoding.FFTDims. Afterwards it checks which of these dimensions are in image-space and then only transforms these ones. The fourier transformation along one dimension is given by:

```
kspace = 1/sqrt(n)*fftshift(fft(ifftshift(img,dim),[],dim),dim);
```

where n are the number of samples along the specific dimension.

K2I

Performs a fourier transformation from k-space to image-space

Syntax: r.K2I;

Parameters used:

- **Parameter.Encoding.FFTDims:**
  Specifies in which dimension the fourier transformation should be executed. This is a 3 elements vector specifying the dimension in matrix notation.
- **Parameter.ReconFlags.isimspace:**
  Used to check which dimensions are still in k-space
- **Parameter.Encoding.FFTShift:**
  Specifies in which dimension the image should be shifted after the fourier transform. This is a 3 elements vector specifying the shift in matrix notation.
- **Parameter.Encoding.XRange, Parameter.Encoding.YRange, Parameter.Encoding.ZRange:**
  Define the shifts of the image after the fourier transform, for every dimension.

Recon Flag: Parameter.ReconFlags.isimspace

Location: k-space

Formats: Raw | ExportedRaw | Cpx | Rec | Bruker

Description/Algorithm: K2I first checks which dimensions should be transformed, defined in Parameter.Encoding.FFTDims. Afterwards it checks which of these dimensions are in k-space and then only transforms these ones. The fourier transformation along one dimension is given by:

```
img = sqrt(n.*fftshift(ifft(ifftshift(kspace,dim),[],dim),dim);
```

where n are the number of samples along the specific dimension.

After the fourier transform from k- to image-space, the images are shifted in image-space as defined in Parameter.Encoding.X/Y/ZRange. Thereby the images are shifted by the amount which leads to a symmetric range around 0. E.g. if the range is given as [-42, 21] then the image is shifted by 10 pixel to the right. ([-42, 21] + 10 = [-32, 31])

K2IM

Performs a fourier transformation from k-space to image-space in measurement direction. See K2I for more information.

K2IP

Performs a fourier transformation from k-space to image-space in phase encoding directions direction. See K2I for more information.

MeasPhaseCorrection

Corrects for an offset phase which might have been added to certain profiles

Syntax: r.MeasPhaseCorrection;

Parameters used:

- **Parameter.Recon.MeasPhaseCorrection: {'yes'} | 'no'**
  Enables/disables the measurement phase correction.
- **Parameter.DataFormat:**
  Used to check if raw data is passed to the function
- **Parameter.LabelLookupTable:**
  Defines the profiles which are currently stored in the Data array.
- **Parameter.Labels.Index.meas_phase:**
  Holds the measurement phases.

Recon Flag: Parameter.ReconFlags.ismeasphasecorr

Location: k-space

Formats: Raw

Description/Algorithm: Different profiles can sometimes exhibit a phase shift, which originates from either the RF pulse phase or the acquistion phase specified in the AQ object. Different averages for example are usually acquired with a phase difference of pi between them. If we wouldn't correct for this phase then the signal would effectively cancel out when summing

up the averages. MeasPhaseCorrection therefore corrects for this possible phase shift using the values in the labels, received by the scanner.

Let mi be the measurement phase of the i-th profile:

```
mi = Parameter.Labels.Index.meas_phase(i);
```

then the phase added to that profile is given by:

phase = -mi * pi/2

PDACorrection

Corrects the profile dependent amplification (PDA)

Syntax: r.PDACorrection;

Parameters used:

- **Parameter.Recon.PDACorrection: {'yes'} | 'no'**
  Enables/disables the PDA correction.
- **Parameter.DataFormat:**
  Used to check if raw data is passed to the function
- **Parameter.LabelLookupTable:**
  Defines the profiles which are currently stored in the Data array.
- **MR.Parameter.Labels.PDAFactors:**
  Holds the PDA correction factors.
- **MR.Parameter.Labels.Index.pda_index:**
  Specifies which correction factor should be taken for a certain profile.

Recon Flag: Parameter.ReconFlags.ispdacorr

Location: k-space

Formats: Raw

Description/Algorithm: In some cases the receiver weights certain k-space profiles differently than others. In 3D imaging for example the outer k-space lines are amplified compared to the central ones, to ensure a sufficient signal range. Of course the amplification has to be corrected in reconstruction, if not the edges in the image would be enhanced. The amplification factors are provided by the scanner and this function simply divides the profiles by them.

PartialFourier

Performs a partial fourier reconstruction (homodyne method).

Syntax: r.PartialFourier;

Parameters used:

- **Parameter.Recon.PartialFourier: {'yes'} | 'no'**
  Enables/disables the partial fourier reconstruction.
- **Parameter.DataFormat:**
  Used to check if the right data format is passed to the function

- **Parameter.ReconFlags.isimspace:**
  Used to check if the data is in k- or image-space.
- **Parameter.Encoding.KxRange, Parameter.Encoding.KyRange,Parameter.Encoding.KzRange:**
  Specifies the sampling ranges in k-space. These parameters are used to check if the scan is acquired using partial Fourier and define the fully sampled k-space center.
- **Parameter.Scan.SENSEFactor:**
  The SENSE factor used to define the fully sampled k-space center.

Recon Flag: Parameter.ReconFlags.ispartialfourier

Location: k-space | image-space

Formats: Raw | ExportedRaw | Bruker

Description/Algorithm: In partial fourier imaging only a fraction of k-space is acquired for improved scan- or echo-time. If the k-space is partly sampled in readout direction we speak of partial echo and halfscan for a partly sampled space in phase encoding direction. In the first case the echo time is reduced while in the second one the scantime is shortened. Performing a standard imaging reconstruction on the partly sampled k-space however leads to image bluring and reduced SNR. Therefore a partial fourier reconstruction is performed which utilizes the k-space symmetry to reconstruct images of better quality. In MRecon a so called homodyne reconstruction is implemented: Noll, D.C.; Nishimura, D.G.; Macovski, A.; "Homodyne detection in magnetic resonance imaging" Medical Imaging, IEEE Transactions on, vol.10, no.2, pp.154-163, Jun 1991.

Notes: In principle the partial fourier reconstruction can either be performed in k- or image space. However if executed in k-space it is not possible to perform a SENSE or CLEAR reconstruction afterwards since the phase information will be lost. Therefore we suggest to execute it in image space after SENSEUnfold which works in any case.

Perform

Performs a complete reconstruction for imaging. Only standard data (typ = 1) is read and reconstructed. Furthermore if not enough memory is available the data is reconstructed in chunks of: x-y-z-coils. The complete code for this function can be found earlier in this manual or in the file StandardRecon.m in the example directory.

RandomPhaseCorrection

Corrects for the random phase which is added to the measured profiles

Syntax: r.RandomPhaseCorrection;

Parameters used:

- **Parameter.Recon.RandomPhaseCorrection: {'yes'} | 'no'**
  Enables/disables the random phase correction.
- **Parameter.DataFormat:**
  Used to check if raw data is passed to the function
- **Parameter.LabelLookupTable:**

Defines the profiles which are currently stored in the Data array.

- **Parameter.Labels.Index.random_phase:**
  Holds the random phases which are subtracted from the profiles.

Recon Flag: Parameter.ReconFlags.israndphasecorr

Location: k-space

Formats: Raw

Description/Algorithm: The receiver adds a random phase to every acquired profile to reduce artifacts. This random phase has to be subtracted in reconstruction since the phase is used for spatial encoding. The added phase is stored in the profile labels and the phase which is subtracted is given by:

Let ri be the random phase label of the i-th profile:

```
ri = Parameter.Labels.Index.meas_phase(i);
```

then the phase subtracted to that profile is given by:

```
phase = 2*pi / double( intmax('uint16') ) * ri;
```

ReadData

Reads all supported data formats into Matlab.

Syntax: r.ReadData;

Parameters used:

All formats:

- **Parameter.DataFormat:**
  Used to check which data format is passed to the reader
- **Parameter.Parameter2Read:**
  Specifies what data is read from the file.
- **Parameter.Filename.ijk:**
  The filename of the data and parameter file to be read.
- **Parameter.Labels:**
  The data labels which hold the image attributes for every profile.

Raw and ExportedRaw data

- **Parameter.Encoding.KxRange:**
  The sampled k-space range in readout direction used to place the data correctly in k-space (e.g. in partial echo acquisitions). Please note that for data which has to be gridded (e.g. radial, spiral, EPI data), this parameter specifies the k-space range after gridding.
- **Parameter.Recon.ArrayCompression:**
  Enables the array compression, according to: Buehrer, M., Pruessmann, K. P. , Boesiger, P. and Kozerke, S. (2007), Array compression for MRI with large coil arrays. Magnetic Resonance in Medicine, 57: 1131–1139. The individual profiles are compressed to the userd defined number of virtual coils immediately after reading.
- **Parameter.Recon.ACNrVirtualChannels:**
  The number of virtual coils used in array compression.

- Parameter.Recon.ACMatrix:
The compression matrix used in array compression.

Recon Flag: Parameter.ReconFlags.isread

Location: Beginning of recon

Formats: Raw | ExportedRaw | Cpx | Rec | Bruker

Description/Algorithm: The ReadData function reads the MR data from the selected file and stores it into Matlab. The function supports selective reading, meaning that the user can specify what data should be read via r.Parameter.Parameter2Read. After calling the ReadData function, the data is stored in the order it was saved in the data file. This means that for raw data we obtain a matrix containing the individual k-space profiles in the order they have been measured. For Rec and Cpx data however we obtain (partly) reconstructed images in image-space. Generally the data is stored in a cell array where the different rows hold different data types (e.g. imaging data, correction data, noise data etc) and the columns different data sizes (e.g. kt-undersampled and training data). The number of rows in the cell array is predefined while the number of columns varies depending on the data in the file. Generally the cell array has the following form:

	Data size 1	Data size 2
1. Accepted imaging dat	{ STD },	{ STD }
2. Rejected imaging data	{ REJ },	{ REJ }
3. Phase correction data	{ PHX },	{ PHX }
4. Frequency correction data	{ FRX },	{ FRX }
5. Noise data	{ NOI },	{ NOI }

If a data type does not occur in the file the the selected cell element is left empty. If only one data type and size is read (cell array with one element) then a ordinary Matrix is returned instead of a cell.

Examples: Read only imaging data from the first coil and the third dynamic:

```
r = MRecon( 'rawfile.raw' );
r.Parameter.Parameter2Read.typ  = 1;   % Imaging data
r.Parameter.Parameter2Read.chan = 0;   % First coil
r.Parameter.Parameter2Read.dyn  = 2;   % Third dynamic
r.ReadData;
```

Read the noise samples from a raw file:

```
r = MRecon( 'rawfile.raw' );
r.Parameter.Parameter2Read.typ  = 5;   % Noise data
r.ReadData;
```

RemoveOversampling

Removes the oversampling from the image by cropping the data in image-space

Syntax: r.RemoveOversampling;

Parameters used:

- **Parameter.Recon.RemoveMOversampling: {'yes'} | 'no'**
  Enables/disables the oversampling removal in readout (measurement) direction
- **Parameter.Recon.RemovePOversampling: {'yes'} | 'no'**
  Enables/disables the oversampling removal in phase-encoding direction(s)
- **Parameters.ReconFlags.isimspace:**
  Used to check if the data is in k- or image-space.
- **Parameter.Encoding.KxOversampling, Parameter.Encoding.KyOversampling, Parameter.Encoding.KzOversampling:**
  The oversamping factors in x/y/z-directions. After calling the RemoveOversampling function the Data matrix is smaller by these factors.
- **Parameters.ReconFlags.issorted:**
  Used to check if the data is already sorted.

Recon Flag: Parameter.ReconFlags.isoversampled

Location: k-space | Image-space.

Formats: Raw | ExportedRaw | Cpx | Bruker

Description/Algorithm: Generally the receiver samples data denser in readout direction than specified in the user interface. The denser sampling produces a larger field-of view (FOV) and prevents fold-over artifacts in readout direction. Oversampling in readout direction does not increase the scan time as the receiver can sample at a much higher frequency than desired. In 3D acquisitions the data is usually oversampled in slice encoding direction as well to correct for improper slice excitation of the outer slices. Oversamlling in slice encoding direction does affect the scan time however. Since the denser sampling basically results in the acquisiton of a larger FOV we have to remove the oversampling by cropping the image in image-space and thereby preserving the signal-to-noise ratio (SNR). Removing the oversampling in k-space would not be beneficial as we would lose SNR. In MRecon the RemoveOversampling can be called at three different locations:

1) Before SortData: The oversampling is only removed in readout direction by fourier transforming the data in that direction and cropping it. This is usually done to reduce the amount of data and save memory.
2) After SortData: The oversampling is removed in all directions by fourier transforming the data in every oversampled direction and cropping it in image space.
3) In image-space: The oversampling is removed in all directions by cropping the data.

Notes: The oversampling is not removed for data which has to be gridded when called before SortData (radial, spiral, EPI).

RingingFilter

Removes ringing artifacts by applying a hamming filter on the k-space data

Syntax: r.RingingFilter;

Parameters used:

- **Parameter.Recon.RingingFilter: {'yes'} | 'no'**
  Enables/disables the ringing filter.
- **Parameter.Recon.RingingFilterStrength:**

A value between 0 and 1 which specifies the strength of the ringing filter. 0 means no filter is applied and 1 is the maximum filter strength. The filter strength can be specified in every direction (Measurement, Phase Encoding, Slice Encoding).

Location: k-space

Formats: Raw | ExportedRaw

Description/Algorithm: Sharp edges in k-space can result in ringing artifacts in image-space. Such edges are introduced by zero-filling or by sampling a low resultion image. The ringing filter tries the soften the edges by applying a hamming filter on the k-space data, lowering the values on the edge. The amount by which the edge-values are lowered is specified by the filter strength. A filter strength of 0.5 for example will bring down the edge value by 50%. A strength of brings the value down to 0 and therefore removing the edges completely. However a large filter strength might cause image blurring. Therefore it is important to find an optimal compromise between artifact removal and image blurring. The default filter strength is set to 0.25.

Notes: The ringing filter should be applied before k-space zero filling to work effectively.

RotateImage

Rotates the images such that they are oriented in the same way as on the scanner console.

Syntax: r.RotateImage;

Parameters used:

- **Parameter.Recon.RotateImage: {'yes'} | 'no'**
  Enables/disables the image rotation.
- **Parameter.Scan.ijk:**
  Specifies the coordinate system of the images before the rotation (current coordinate system).
- **Parameter.Scan.REC:**
  Specifies the coordinate system of the images after the rotation (coordinate system of the Rec images).

Recon Flag: Parameter.ReconFlags.isrotated

Location: Image-space

Formats: Raw

Description/Algorithm: After reading the raw data it is oriented in The MPS system, meaning that the rows in the Data matrix are aligned along the measurement direction and the columns along the phase encoding direction. The MPS system however heavily depends on the chosen scan parameters, such as orientation, fold-over direction, fat shift direction etc. The reconstructed images, displayed on the scanner console on the other hand are oriented in a well-defined fashion expressed in the patient coordinate system (right-left (RL), anterior-posterior (AP), feet-head (FH)). The RotateImage function transforms the Data from the current coordinate system (usually MPS) the final orientation displayed on the scanner console.

Examples: The Rotate Image basically performs a transformation between the system specified in r.Parameter.Scan.ijk and r.Parameter.Scan.REC. These coordinates systems are always expressed in the patient system (RL, AP, FH). Let's assume these values are given as:

```
r.Parameter.Scan.ijk = [RL, FH, AP]
r.Parameter.Scan.REC = [HF, RL, AP]
```

To transform the ijk system to the REC system, we have to switch the first and the second axis of the Data matrix (permute it):

```
[RL, FH, AP]  -->  [FH, RL, AP]
```

and then flip the first axis (flipdim):

[FH, RL, AP] --> [HF, RL, AP]

Search

Search for a parameter within the whole MRecon class and all GoalC parameters, objects and groups

Syntax: r.Search('search_string');

Parameters used: All

Formats: All

Description/Algorithm: The search function iterates through all the parameters objects and groups of the MRecon class and checks if it matches the search string. Wildcards are possible.

Examples: To search for the field-of-view (FOV) call:

```
>> r.Search('FOV')
```

which will display all MRecon fields, and GoalC parameter/objects which contain the string FOV. You can also use wildcards to limit or extend your search space:

Assume we want to find all ex parameters with "FOV" in its name:

```
>> r.Search('EX_*FOV');
```

Assume we want to find all control parameters related to diffusion:

```
>> r.Search('CSC_*diff');
```

SENSEUnfold

Performs a SENSE reconstruction using the coil sensitivity information provided by the user.

Syntax: r.SENSEUnfold;

Parameters used:

- **Parameter.Recon.SENSE: {'yes'} | 'no'**
  Enables/disables the SENSE unfolding.
- **Parameter.Recon.Sensitivities:**
  The sensitivity object (MRsense) used in the unfolding process. The sensitivity object holds the coil sensitivities, the noise covariance matrix as well as regularisation images.
- **Parameter.Recon.SENSERegStrength:**
  The regularization strength used during the SENSE unfolding. A higher regularizazion results in a stronger suppression of low signal values in the unfolded images.
- **Parameter.Scan.SENSEFactor:**
  The SENSE reduction factor.

Recon Flag: Parameter.ReconFlags.isunfolded

Location: image-space

Formats: Raw | ExportedRaw

Description/Algorithm: Please see the MRsense manual for more infomation on the implemented algorithms and examples.

ScaleData

Clips the data such that 98% of its histogram is preserved

Syntax: r.ScaleData;

Parameters used: None

Location: Image-space

Formats: Raw | ExportedRaw | Cpx | Rec | Bruker

Description/Algorithm: A histogram is calculated over all values in the Data array and the value is found which includes 98% of the histograms integral. The data is then clipped at that value. This function is usefull to remove unwanted signal peaks.

ShowData

Displays the Data array using MRecon's built-in 3D image viewer.

Syntax: r.ShowData;

Parameters used: None

Location: k-space | Image-space

Formats: Raw | ExportedRaw | Cpx | Rec | Bruker

Description/Algorithm: Opens the built-in 3D image viewer of MRecon and displays the Data array.

Notes: - The MRecon image viewer is a standalone tool and can be used to visualize any Matlab matrix. It can be called as:

```
image_slide( Matrix )
```

r.ShowData therefore is aequivalent to:

```
image_slide( r.Data )
```

SortData

Sorts the acquired profiles by their image attributes and generates proper k-space data.

Syntax: r.SortData;

Parameters used:

All formats:

- Parameter.Recon.ImmediateAveraging:
When enabled the averages are added immediately during SortData. If you want to reconstruct the individual averages seperately then disable this parameter.

- Parameter.Labels:
The data labels which hold the image attributes for every profile.

- Parameter.LabelLookupTable:
Links the profiles which are currently stored in the Data array with the data lables.

- Parameter.Encoding.KyRange, Parameter.Encoding.KzRange:
The sampled k-space ranges in phase encoding directions used to place the data correctly in k-space (e.g. in halfscan acquisitions).

- Parameter.Encoding.KyOversampling, Parameter.Encoding.KyOversampling:
The oversampling factors in phase encoding directions used to place thedata correctly in k-space.

Recon Flag: Parameter.ReconFlags.issorted

Location: After ReadData

Formats: Raw | ExportedRaw | ExportedCpx

Description/Algorithm: After reading the data into Matlab with ReadData, the acquired profiles are stored in temporal order (in the order they have been measured). The SortData function sorts the profiles by their image attributes phase encoding number, cardiac phase, dynamic scan number etc. After the sorting the data the data array holds proper k-space data which can be fourier transformed to generate images. Please note that averages are summed up immediately during sorting if immediate averaging is enabled (see above).

After SortData the data array has a predifined form with 12 dimensions, where every array dimension corresponds to an image attribute:

r.Data = [kx, ky, kz, coils, dynamics, cadiac phases, echoes, locations, mixes, extra1, extra2, averages]

Transform

Transforms coordinates from one scanner coordinate system to another

Syntax: [xyz_transformed, A] = r.Transform(xyz, from_system, to_system, stack_nr);

Or

A = r.Transform(from_system, to_system, stack_nr);

Parameters used:

- **Parameter.Scan.ijk, Parameter.Scan.MPS, Parameter.Scan.xyz, Parameter.Scan.REC**
  The different scanner coordinates system expressed in the patient coordinate system.
- **Parameter.Scan.Angulation, Parameter.Scan.Offcentre:**
  Angulation and offcentre of the scan.
- **Parameter.Scan.curFOV:**
  The current field of view of the scan. curFOV corresponds to the FOV of the current data and does change during the reconstruction process. Removing the oversampling for example reduces the FOV.
- **Parameter.Scan.SliceGap:**
  The slice gap of the scan.

- **Parameter.Encoding.XRes, Parameter.Encoding.YRes, Parameter.Encoding.ZRes Parameter.Encoding.KyOversampling, Parameter.Encoding.KyOversampling:**
  These parameters are only used if the Data matrix is empty. The default matrix size used in the Transform function, when no data is present, is the one which we would obtain after the SENSE unfolding (which has to be used for calculating the coil sensitivities). It is given by:

```
[XRes, YRes*KyOversampling, ZRes*KzOversampling]
```

Location: k-space | image-space

Formats: Raw | Rec

Description/Algorithm: Please see the transformation manual for more information on the different coordinate systems and examples.

To transform a coordinate from one coordinate system to another the following parameters have to be known:

- Patient Position (defines coordinate system)
- Patient Orientation (defines coordinate system)
- Image Orientation (defines coordinate system)
- Fold Over Direction (defines coordinate system)
- Fat Shift Direction (defines coordinate system)

- Angulation (fixed, used in Transform)
- Offcentre (fixed, used in Transform)
- Slice Gap (fixed, used in Transform)

- Resolution (variable, used in Transform: Resolution = curFOV / size(Data))
- Image Matrix (variable, used in Transform for ijk, REC systems)

The first 5 parameters are used to define the different coordinate systems (ijk, MPS, RAF, xyz, REC), which are stored in Parameter.Scan and used in the Transform function. Therefore the parameters itself are not needed in the Transform function anymore. Basically the parameters which are used in the Transform function can be divided into 2 groups: fixed and variable parameters. The fixed ones, such as the image angulation, do not change during the reconstruction process whereas the variable ones can change. The image resolution for example changes if we perform a k-space zero filling. In transform the current image resolution is calculated by dividing the current FOV by the current matrix size. Therefore if you are performing your own reconstruction and are not using the built-in MRecon functions (e.g. RemoveOversampling) always make sure that Parameter.Scan.curFOV corresponds to the current FOV of your data when calling the Transform function.

WriteExportedRaw

Exports the data to Philips exported raw format.

Syntax: [datafile, listfile] = r.WriteExportedRaw(Filename, Parameter2Write);

Location: k-space | image-space.

Formats: Raw | ExportedRaw

Description/Algorithm: Exports the images in the data array to Philips exported format (data/list file pair). The Parameter2Write which have to be given as input is usually the current Parameter2Read struct (r.Parameter.Parameter2Read).

WritePar

Exports the parameter to a Philips par file.

Syntax: r.WritePar(filename);

Parameters used:

- **Parameter.Scan:**
  Holds the general parameters (uper part of the par file)
- **Parameter.ImageInformation:**
  Holds the individual parameters for every image (lower part of the parfile).

Location: image-space.

Formats: Raw | ExportedRaw | Cpx | Rec

Description/Algorithm: Exports the parameters in to a Philips par file.

WriteRec

Exports the data to Philips rec format.

Syntax: r.WriteRec(filename);

***Parameters* used:**

- **Parameter.Recon.ExportRECImgTypes: {'M', 'P', 'R', 'I'}**
  Specifies the images which should be exported (Magnitude, Phase, Real part, Imaginary part).
- **Parameter.ImageInformation.RescaleIntercept,**
  Parameter.ImageInformation.RescaleSlope:
  Defines the scaling of the images in the rec file. If these values are empty (default) then the scaling is calculated automatically in the WriteRec function. For details see below.

Location: image-space.

Formats: Raw | ExportedRaw | Cpx | Rec

Description/Algorithm: Exports the images in the data array to Philips rec format. During the export the data is rescaled according to:

Magnitude data, Real part, Imagenary part:

```
Value_in_Rec File = round( (Value_in_r.Data - ...
       RescaleIntercept) / RescaleSlope )
```

Phase data:

```
Value_in_Rec_File = round( (1000 * Phase_in_r.Data - ...
       RescaleIntercept) / RescaleSlope )
```

Notes: When the exported rec data is reimported into the scanner database the values are rescaled such that they correspond the original values in r.Data before exporting.

WriteXMLPar

Exports the parameter to a Philips XML par file.

Syntax: r.WriteXMLPar(filename);

Parameters used:

- **Parameter.Scan:**
  Holds the general parameters (uper part of the par file)
- **Parameter.ImageInformation:**
  Holds the individual parameters for every image (lower part of the par file).

Location: image-space.

Formats: Raw | ExportedRaw | Cpx | Rec

Description/Algorithm: Exports the parameters in to a Philips XML par file.

ZeroFill

Zero pads the data either in k-space or image-space

Syntax: r.ZeroFill;

Parameters used:

- **Parameter.Recon.kSpaceZeroFill: {'yes'} | 'no'**
  Enables/disables the k-space zero filling.
- **Parameter.Recon.ImageSpaceZeroFill: {'yes'} | 'no'**
  Enables/disables the image-space zero filling.
- **Parameters.ReconFlags.isimspace:**
  Used to check if the data is in k- or image-space.
- **Parameters.ReconFlags.isoversampled:**
  Used to check if and in which directions the data is oversampled.
- **Parameter.Encoding.XRes, Parameter.Encoding.YRes, Parameter.Encoding.ZRes:**
  The data is zero filled to these dimension in k-space (times the remaining oversampling factor).
- **Parameter.Encoding.XReconRes, Parameter.Encoding.YReconRes, Parameter.Encoding.ZReconRes:**
  The data is zero filled to these dimension in image-space (times the remaining oversampling factor).
- **Parameter.Encoding.KxOversampling, Parameter.Encoding.KyOversampling, Parameter.Encoding.KzOversampling:**
  The oversamping factors in x/y/z-directions.
- **Parameters.ReconFlags.issorted:**
  Used to check if the data is already sorted.

Recon Flag: Parameter.ReconFlags.iszerofilled

Location: k-space | Image-space.

Formats: Raw | ExportedRaw | Cpx | Bruker

Description/Algorithm: The ZeroFill function zero-pads the current data to the dimensions specified in the Encoding parameters (see above). Thereby we have a different dimension depending whether we are in k-space or image-space. For example the dimension in y-direction after zero filling in k-space is given by:

```
yres = r.Parameter.Encoding.YRes * cur_ovs
```

where the current oversampling cur_ovs is given by:

```
cur_ovs = r.Parameter.Encoding.KyOversampling
```
(if oversampling is not removed)

Or

```
cur_ovs = 1
```
(if oversampling is removed)

Whether and in which directions the data is oversampled is defined by the oversampling ReconFlag (Parameters.ReconFlags.isoversampled).

Zero filling in k-space is used to correct for anisotropic voxel sizes and the artificially increase image resolution. Zero filling in k-space isused to correct for a rectangular field-of-view and to produce quadratic images.

Parameter Methods

DisplayAllGroups

Displays all GoalC parameter groups

Syntax: r.Parameter.DisplayAllGroups;

Formats: Raw

Description/Algorithm: Displays all parameter groups which are available in the pulsprogramming environment

DisplayGroupOfParameter

Displays the group(s) of the specified GoalC parameter

Syntax: r.Parameter.DisplayGroupOfParameter('<parameter name>');

Formats: Raw

Description/Algorithm: Displays all groups the specified parameter is a member of.

Examples: Assume we want to find the parameter group of **UGN4_RFE_bw_sel_echo**. To do so, execute the following command:

```
r.Parameter.DisplayGroupOfParameter('UGN4_RFE_bw_sel_echo');
```

DisplayObject

Displays all attributes of the specified object

Syntax: r.Parameter.DisplayObject('<object name>');

Formats: Raw

Description/Algorithm: Displays all attributes and their values of the specified GoalC object.

Examples: Assume we want to see the attribute of the base sequence object:

```
r.Parameter.DisplayObject('SQ`base');
```

DisplayObjectNames

Displays all objects of the specified object-class

Syntax: r.Parameter.DisplayObjectNames('<object class>');

Formats: Raw

Description/Algorithm: Displays all GoalC objects of the specified class.

Examples: Assume we want to display all RF objects

```
r.Parameter.DisplayObjectNames('RF');
```

DisplayParameterInGroup

Displays all parameter in the specified group

Syntax: r.Parameter.DisplayParameterInGroup('<group name>');

Formats: Raw

Description/Algorithm: Displays all GoalC parameter in the specified group

Examples: Assume we want to see all parameters in the respiratory navigator group:

```
r.Parameter.DisplayParameterInGroup('UGN1_RNAV_all_pars');
```

ExtractPDFFile

Creates a PDF file for the Philips graphical viewer

Syntax: r.Parameter.ExtractPDFFile ('<filename>');

Formats: Raw

Description/Algorithm: Creates a .gve file which can be opened with the Philips graphical viewer and contains the graphical representation of the PDF code. The input argument <filename> is optional. If it isn't specified the .gve file will have the same name as the current raw file.

GetGroup

Gets the specified parameter group.

Syntax: group = r.Parameter.GetGroup('<group name>');

Formats: Raw

Description/Algorithm: Gets the specified parameter group and returns a GoalcGroup object containing all the information.

Examples: Assume we want to get the respiratory navigator group:

```
G = r.Parameter.GetGroup('UGN1_RNAV_all_pars');
```

GetGroupOfParameter

Gets the group of the specified parameter.

Syntax: group = r.Parameter.GetGroupOfParameter('<parameter name>');

Formats: Raw

Description/Algorithm: Gets the parameter group of the specified parameter and returns a GoalcGroup object.

Examples: Assume we want to get the group which `UGN4_RFE_bw_sel_echo` is a member of:

```
G = r.Parameter.GetGroupOfParameter ('UGN4_RFE_bw_sel_echo');
```

GetObject

Gets the specified object.

Syntax: object = r.Parameter.GetObject('<object name>');

Formats: Raw

Description/Algorithm: Gets the specified object and returns a GoalcObject object containing all the information.

Examples: Assume we want to get the excitation pulse:

```
O = r.Parameter.GetObject('RF`ex');
```

GetObjectNames

Gets all the object names of the specified class

Syntax: object_names = r.Parameter.GetObjectNames('<object class>');

Formats: Raw

Description/Algorithm: Gets all the object names of the specified class and returns them as a cell array.

Examples: Assume we want to get all the RF object names:

```
N = r.Parameter.GetObjectNames('RF');
```

GetParameter

Gets the specified parameter

Syntax: par = r.Parameter.GetParameter('<parameter name>');

Formats: Raw

Description/Algorithm: Gets the specified parameter and returns a GoalcParameter object containing all the information

Examples: Assume we want to get the parameter attributes of `UGN4_RFE_bw_sel_echo`:

```
P = r.Parameter.GetParameter('UGN4_RFE_bw_sel_echo');
```

GetParameterInGroup

Gets all the parameter of the specified group.

Syntax: par_array = r.Parameter.GetParameterInGroup ('<group name>');

Formats: Raw

Description/Algorithm: Gets all the parameter of the specified group and returns an array of GoalCParameter objects.

Examples: Assume we want to get all the parameter in the respiratory navigator group:

```
P = r.Parameter.GetParameterInGroup('UGN1_RNAV_all_pars');
```

GetValue

Gets the value of a Goalc parameter or object.

Syntax: value = r.Parameter.GetValue ('<name>', array_index, numeric);

Formats: Raw

Description/Algorithm: Gets the value of a Goalc parameter or object.

The input arguments “array_index” and “numeric” are optional. “array_index” can be used to selectively return certain elements if the specified parameter is an array. “numeric” is only valid for parameters of type enumeration and if it is true will return a numeric value instead of a string.

The return value depends on the type of the parameter/object and its dimension and can either be a scalar, array or cell array.

Examples: Assume we want to get all value of UGN1_RNAV_beam_order:

```
val = r.Parameter.GetValue('UGN1_RNAV_beam_order');
```

Please note that val is an array because the parameter is also an array. If we only want the first 2 values we can use the second input argument:

```
val = r.Parameter.GetValue('UGN1_RNAV_beam_order', 1:2);
```

If the parameter type is a string or enumeration then val will also be a string:

```
val = r.Parameter.GetValue('UGN1_RNAV_enable');
```

If we prefer a numeric value for the enumeration instead of the string we can use the third input argument:

```
val = r.Parameter.GetValue('UGN1_RNAV_enable', [], true);
```

We can also ask for values of an object attribute in the following way:

```
val = r.Parameter.GetValue('SQ`base:dur');
```

If the object is a composite object we should also specify the composite index (note that it starts at 1):

```
val = r.Parameter.GetValue('LCA`ima:[1]:prep_dir');
```

If we want to obtain the values for all composite indices we can also replace the index with the Matlab colon placeholder:

```
val = r.Parameter.GetValue('LCA`ima:[:]:prep_dir');
```

IsParameter

Checks if a parameter exists

Syntax: is_par = r.Parameter.IsParameter ('<parameter name>');

Formats: Raw

Description/Algorithm: Checks if a GoalC Parameter is available in the MRecon object

Examples: `is_par = r.Parameter.IsParameter('UGN1_RNAV_all_pars');`

IsObject

Checks if an object exists

Syntax: is_obj = r.Parameter.IsObject ('<object name>');

Formats: Raw

Description/Algorithm: Checks if a GoalC object is available in the MRecon object

Examples: `is_obj = r.Parameter.IsObject('SQ`base');`

Search

See the Search function in the MRecon methods.

MRecon Properties

Parameter

Parameter2Read

Specifies what data to read at the next call of ReadData. Upon initialization the fields display all scan parameters which occur in the dataset. Please note that not all possible combination of the field values might exist in the dataset.

typ

Defines the image-types to be read. The values ranges from 1-5, where each number corresponds to a certain image type which might occur in the raw file***: 1 = Standard Data, 2 = Rejected Data, 3 = Phase Correction Data, 4 = Frequency Correction Data, 5 = Noise Data***

mix

Defines the mixes to be read.

dyn

Defines the dynamics to be read.

card

Defines the cardiac phases to be read.

echo

Defines the echoes to be read.

loca

Defines the locations to be read. Locations depict either different slices in M2D or MS scans, or different stacks in 3D acquisitions.

chan

Defines the coils (channels) to be read.

extr1

Defines the extr1 attribute to be read. extr1 depicts different types of data, depending on the scan. In flow imaging the different flow encoding directions (segments) are assigned to the extr1 label.

extr2

Defines the extr2 attribute to be read. Extr2 depicts different types of data, depending on the scan.

ky

Defines the y-phase encoding profiles to be read.

kz

Defines the z-phase encoding profiles to be read.

aver

Defines the averages to be read.

rtop

Defines the trigger delay range to be read. The trigger delay of a certain profile is defined as the time between the ECG-trigger and the acquisition of that profile.

ImageInformation

Provides information for every image in the Data property of the MRecon object, such as image angulations, offcentres, resolution etc. The ImageInformation property is a struct array with the same dimensions as the Data property. To access information about a specific image in the Data property use indexing like: Parameter.ImageInformation(5) or Parameter.ImageInformation(1,2,5,3,1). In the first case information about the 5-th image is provided and in the second case information from the first slice, second coil, 5th dynamic etc.

This information is also used for writing the parameter file in the WritePar and WriteXMLPar function. The individual fields in the ImageInformation struct can also be updated manually by the user if desired. This is useful when user defined information should be written to the .par file.

Slice
The slice number of the specified image.

Coil
The coil whereby the specified image was acquired.

Dynamic
The dynamic number of the specified image.

CardiacPhase
The cardiac phase of the specified image.

Echo
The echo number the specified image.

Location
The location (stack in 3D or slices in MS and M2D) of the specified image.

Mix
The mix number of the specified image.

Extr1
The extr1 number of the specified image.

Extr2
The extr2 number of the specified image.

Average
The average of the specified image.

Resolution
The resolution (matrix size) of the specified image.

RescaleIntercept
Used for scaling of the images on the scanner console. The displayed value on the console is given by:
displayed_value = value_in_rec_file * RescaleSlope + RescaleIntercept

RescaleSlope
Used for scaling of the images on the scanner console. The displayed value on the console is given by:
displayed_value = value_in_rec_file * RescaleSlope + RescaleIntercept

ScaleSlope
Used to restore the original floating point value of the complex data from the integer values in the rec file. Thus if both phase and magnitude images are stored in the recfile the original k-space can be restored. The floating point value is thereby given as:
floating_point_value = value_in_recfile / ScaleSlope

WindowCenter
Used for clipping of the images on the scanner console. All values of the image which are larger or smaller than WindowCenter ± WindowWidth / 2 are clipped when displayed on the console.

WindowWidth
Used for clipping of the images on the scanner console. All values of the image which are larger or smaller than WindowCenter ± WindowWidth / 2 are clipped when displayed on the console.

Angulation
The angulation of the specified image.

Offcentre
The offcentre of the specified image.

SliceThickness
The slice thickness of the specified image.

SliceGap
The slice gap of the specified image.

Orientation
The orientation of the specified image. The orientation is encoded in numbers, where 1 = Transversal, 2 = Sagital, 3 = Coronal

AcqVoxelSize
The acquired voxel size of the specified image.

RecVoxelSize
The reconstructed voxel size of the specified image.

EchoTime
The echo time of the specified image.

TriggerTime
The trigger time of the specified image.

FlipAngle
The flip angle of the specified image.

TurboFactor
The turbo factor of the specified image.

Chunk
To be memory efficient, the data can be read and reconstructed individually in portions (chunks). Every time a chunk is defined, MRecon automatically calculates how many of these data blocks occur in the raw-file.

Def
Defines the chunk by specifying which parameter of the raw file should be read and reconstructed at once. The chunk is defined via a cell array consisting of several strings defining the parameter. Hereby the notation is the same than in the Parameter2Read struct. To reconstruct blocks of x-y-z-coils for example set the Chunk.Def to {'ky', 'kz', 'chan' }. Possible values for Def are***: 'ky', 'kz', 'chan', 'dyn', 'card', 'loca', 'extr1', 'extr2', 'mix', 'aver', 'echo'***
Dependencies: Chunk.CurLoop, Chunk.NrLoops, Chunk.Loops, Parameter2Read are updated.

CurLoop
A number referring to the chunk which is currently reconstructed. CurLoop cannot be larger than Chunk.NrLoops. Every time CurLoop is changed the Parameter2Read struct is automatically updated such that the desired chunk is read the next time ReadData is called.
Dependencies: Parameter2Read is updated

NrLoops
Specifies how many chunks (defined in Def) occur in the raw data.

Loops
Is a similar structure as the Parameter2Read struct but contains only fields which are not in the chunk definition and have more than one entry. The sum over all entries in this struct corresponds to the number of loops (NrLoops)

Scan
Contains basic scan parameters which can be used in the reconstruction. *Please Note that the ReconFrame patch is need for these parameters to be filled out completely.*

ScanMode
Specifies the scan mode. Possible values are: ***2D, 3D, MS, M2D, SV, 1D***

FastImgMode
Specifies the fast imaging mode. Possible values are: ***None, TSE, TFE, EPI, GRASE, TFEEPI, TSI***

TFEfactor

The TFE factor which was used in the current scan.

AcqMode

Specifies the acquisition mode. Possible values are: ***Cartesian, Radial, Spiral, Propeller***
Dependencies: ky and kz are updated (for radial and spiral scans, ky/kz lables start from 0), Gridder Preset is updated.

UTE

Specifies if Ultra-Short-TE was enabled.

KooshBall

For 3D radial sampling this parameter specifies if the trajectory was a Kooshball.

TR

The repetition time

TE

The echo time

FlipAngle

The flip angle which was used in the current scan.

FOV

The field of view.

RecVoxelSize

The voxel size in the reconstructed images.

AcqVoxelSize

The acquired voxel size.

Offcentre

The offcentres which were used in the current scan (in mm). Please note that every "location" and every "mix" has its own offcentre value.

Angulation

The angulations which were used in the current scan (in degrees). Please note that every "location" and every "mix" has its own angulation value.

Orientation

The orientations used in the current scan. Possible values are: ***"TRA"*** (Transversal), ***"COR"*** (Coronal), ***"SAG"*** (Sagital). Please note that every "location" and every "mix" has its own orientation value.

FoldOverDir

The fold-over directions used in the current scan. Possible values are: ***"RL"*** (Right-Left), ***"AP"*** (Anterior-Posterior), ***"FH"*** (Feet-Head). Please note that every "location" and every "mix" has its own fold-over direction.

FatShiftDir

The water-fat shift directions used in the current scan. Possible values are: ***"R"*** (Right), ***"L"*** (Left), ***"A"*** (Anterior)***, "P"*** (Posterior) ***, "F"*** (Feet), ***"H"*** (Head). Please note that every "location" and every "mix" has its own water-fat shift direction.

SENSEFactor

The SENSE factors used in the current scan.

Rotate

Specifies the operation which has to be performed to orient the image in the same way as on the scanner screen.

NrDatabaseImages

Specifies how many images are expected from the database.

Encoding

Provides information about the encoding, such as matrix sizes, sampling ranges, oversampling factors etc. For a more detailed description about the matrices and sizes in the reconstruction process, refer to the previous section.

NrMixes

The number of mixes in the current dataset.

NrEchoes

The number of echoes in the current dataset

Mix

Specify the mixes that corresponds to the different rows in the following encoding parameters

Echo

Specify the echoes that corresponds to the different rows in the following encoding parameters

KxRange

Specifies the k-space sampling range in measurement direction. Usually the sampling is symmetric but can become asymmetric in case of partial echo for example. KxRange is an array with dimensions: NrMixes*NrEchoes x 2.

KyRange

Specifies the k-space sampling range in y-phase encoding direction. Usually the sampling is symmetric but can become asymmetric in case of half scan for example. KyRange is an array with dimensions: NrMixes*NrEchoes x 2.

Dependencies: ky labels are updated.

KzRange

Specifies the k-space sampling range in z-phase encoding direction. KzRange is an array with dimensions: NrMixes*NrEchoes x 2.

Dependencies: ky labels are updated.

XRange

Specifies the range in k-space/image-space along measurement direction after k-space zero filling and oversampling removal. Note that images in that stage already have isotropic voxel sizes in the y-x plane. An asymmetric range means that the image should be shifted after Fourier transformation. The amount of the shift should be chosen such that the range becomes symmetric around 0 after shifting.

YRange

Specifies the range in k-space/image-space along y-phase encoding direction after k-space zero filling and oversampling removal. Note that images in that stage already have isotropic voxel sizes in the y-x plane. An asymmetric range means that the image should be shifted after Fourier transformation. The amount of the shift should be chosen such that the range becomes symmetric around 0 after shifting.

ZRange

Specifies the range in k-space/image-space along y-phase encoding direction after k-space zero filling and oversampling removal. Note that images in that stage already have isotropic voxel sizes in the y-x plane. An asymmetric range means that the image should be shifted after Fourier transformation. The amount of the shift should be chosen such that the range becomes symmetric around 0 after shifting.

XRes

Specifies the number of pixels in k-space/image-space along measurement direction after k-space zero filling and oversampling removal. This number should correspond to XRange. Note that images in that stage already have isotropic voxel sizes in the y-x plane.

YRes

Specifies the number of pixels in k-space/image-space along y-phase encoding direction after k-space zero filling and oversampling removal. This number should correspond to YRange. Note that images in that stage already have isotropic voxel sizes in the y-x plane.

ZRes

Specifies the number of pixels in k-space/image-space along z-phase encoding direction after k-space zero filling and oversampling removal. This number should correspond to ZRange. Note that images in that stage already have isotropic voxel sizes in the y-x plane.

XReconRes
The final resolution of the reconstructed image in measurement direction.

YReconRes
The final resolution of the reconstructed image in y-phase encoding direction.

ZReconRes
The final resolution of the reconstructed image in z-phase encoding direction.

KxOversampling
The oversampling factor in measurement direction.

KyOversampling
The oversampling factor in y-phase encoding direction.

KzOversampling
The oversampling factor in z-phase encoding direction.

FFTShift
Specifies if the image should be shifted after the Fourier transformation or not. The 3 entries correspond to x-y-z direction. Please note that this enables/disables only the check if the image is shifted or not. If the image has a symmetric X-Y-ZRange than it is never shifted even when the FFTShift is enabed. Possible values for FFTShift are: ***0, 1***

FFTDims
Specifies in which directions the data is Fourier transformed. The 3 entries correspond to x-y-z direction. Possible values for FFTDims are: ***0, 1***

Recon
Defines various parameters which are used during the reconstruction process. These parameters are used when the Perform function of MRecon is called. Entire reconstruction steps can be switched on and off, and parameters for single steps can be set.

DcOffsetCorrection
Switches the dc offset correction on or off. Possible values are: ***yes, no***

PDACorrection
Switches the profile dependent amplification correction on or off. Possible values are: ***yes, no***

RandomPhaseCorrection
Switches the random phase correction on or off. Possible values are: ***yes, no***

MeasPhaseCorrection
Switches measurement phase correction on or off. Possible values are: ***yes, no***

PartialFourier
Switches the partial fourier reconstruction (partial echo, half scan) on or off. Possible values are: ***yes, no***

PartialFourierNrIterations
Specifies the number of iterations for the partial fourier reconstruction. A larger value results in a longer reconstruction. If the value is set to -1 then the reconstruction is time dependent and does not take longer than 20 seconds. Possible values are: ***-1, any integer number lager than 0***

kSpaceZeroFill
Switches the k-space zero filling on or off. Possible values are: ***yes, no***

EPIPhaseCorrection
Switches the EPI phase correction on or off. Possible values are: ***yes, no***

CoilCombination

Specifies the coil combination type. At the moment only a sum-of-squares reconstruction is implemented. Possible values are: ***sos*** (sum-of-squares)

ImageSpaceZeroFill

Switches the image space zero filling on or off. Possible values are: ***yes, no***

RotateImage

Switches the image rotation on or off. Possible values are: ***yes, no***

ArrayCompression

Enables Array Compression which is a method for k-space coil precombination to save memory and reconstruction time. A detailed description can be found here: *Buehrer, M., Pruessmann, K. P., Boesiger, P. and Kozerke, S. (2007), Array compression for MRI with large coil arrays. Magnetic Resonance in Medicine, 57: 1131–1139. doi: 10.1002/mrm.21237*. Possible values are: ***yes, no***

ACNrVirtualChannels

Specifies how many virtual coils are created after array compression. Possible values are: ***Any integer number between 1 and the number of physical coils***

ACMatrix

Specifies the compression Matrix for array compression. The matrix is automatically calculated when array compression is switched on or the number of virtual coils is changed. However it is possible to specify the matrix manually by overwriting it. Possible values are: ***any matrix with the size: nr_of_virtual_coils x nr_of_physical_coils***

ImmediateAveraging

Switches immediate averaging on or off. If set to off every average is reconstructed individiually. Possible values are: ***yes, no***

ExportRECImgTypes

Specifies what image types are written to the rec and par files when the WriteRec, WritePar or WriteXMLPar functions are called. Magnitude, Phase, Real or Imaginary images can be written to the rec file. Possible values are: ***M, P, R, I***

AutoUpdateInfoPars

Specifies if the ImageInformation parameters should be updated after every reconstruction step. While its desirable that these parameters are always up to date, reconstruction performance might suffer for large datasets with a lot of images when set to yes. For optimal performance switch this parameter to no at the beginning of your reconstruction and to yes at the end of it. Possible values are: ***yes, no***

SENSE

Switches the SENSE reconstruction on or off. Please note that the coil sensitivities have to be provided by the user. If the coil sensitivities are not present the SENSEUnfold function has no effect. Possible values are: ***yes, no***

Sensitivities

Specifies the coil sensitivities for the SENSEUnfold function. The sensitivities have to have the same geometry and orientation as the actual data. Furthermore the different coils have to be stored in the 4-th dimension of the matrix. If the matrix size of the sensitivities are different than the ones from the actual scan they are interpolated to the correct size.

SENSEPsi

Specifies the noise covariance matrix for the SENSE reconstruction. If the noise covariance is not given the identity matrix is used in the reconstruction. **The noise covariance has to be a matrix with dimension coil x coil**

Gridder

Defines the gridder parameters. There are certain presets (e.g. radial or spiral) which are predefined parameter sets for a certain application. Beside the presets the gridder can also be configured freely.

Preset
Specifies the Gridder preset. Possible values are: ***None, Radial, Spiral***
A Gridder presets, defines all necessary gridder parameter such as k-space sampling positions, weights, kernel size etc. automatically.

Kpos
The k-space sampling positions. Kpos has to be an array with dimensions: #sampling points x 3

RadialAngles
For radial scans a user defined radial angle can be set if desired (e.g. golden angle scans). When left empty then the standard angles are used which are predefined by the scanner.

Weights
The gridding weights, which describe the sampling density in k-space. Weights has to be a vector with the size: #nr of samples x 1

KernelWidth
The gridding kernel width

KernelBeta
The kernel beta

GridOvs
The gridding oversampling factor

Normalize
Specifies if the data is normalized after gridding (needs an additional Fourier transform)

Cardiac

Synchronization
Specifies the method of cardiac synchronization. Possible values are: ***None, Retro*** (retrospective gating)
In retrospective gated scans there are no predefined heart phases. Instead every profile is assigned the acquisition time in the RR-interval as well as the length of the RR-interval. When the Synchronization is set to Retro then a heart phase is assigned to every profile based on the timing information. Since the heart rate changes during the measurement there might still be missing profiles (holes) in k-space after the assignment. These holes are filled with the best matching profile we have acquired.
Dependencies: Cardiac phase labels, RetroPhases and PhaseWindow are updated. Furthermore additional labels are created which correspond to the holes in k-space.

RetroPhases
Specifies the number of heart phases which should be created. This number might be different from the number which was set in the scanner user interface or was effectively measured. However, note that the data is simply temporally interpolated if the number of heart phases is set to a larger number than the one which was actually measured.
Dependencies: Cardiac phase labels are updated. Furthermore additional labels are created which correspond to the holes in k-space.

PhaseWindow
Alternatively a single heart phase instead of several ones can be reconstructed. The PhaseWindow is a 2-dimensional vector which specifies the start and end of the window in ms after the cardiac trigger.
Dependencies: RetroPhases and Parameter2Read.rtop are updated.

RNAV
The respiratory navigator positions (if enabled), as calculated by the scanner. The positions are split between the navigator preparation phase and the acquisition phase.

DataFormat
Specifies the data format of the current dataset. Possible values are: ***Raw, ExportedRaw, Cpx, Rec***

Filename
Data
The filename of the data-file

Parameter
The filename of the parameter-file

ReconFlags
Provide informations about the current status of the reconstruction.

Isread
Specifies if the data has already been read

Issorted
Specifies if the data has already been sorted

Ispartialfourier
Specifies if the data has already been corrected for halfscan

Isgridded
Specifies if the data has already been gridded

Isimspace
Specifies if the data is in image-space

Iscombined
Specifies if the coils are already combines

Isoversampled
Specifies if the oversampling has already been removed

Isreadparameter
Specifies if the parameter file has already been read

Israndphasecorr
Specifies if the data has already been random phase corrected

Ispdacorr
Specifies if the data has already been PDA corrected

Isdcoffsetcorr
Specifies if the data has already been DC offset corrected

Isdepicorr
Specifies if the data has already been EPI phase corrected

Ismeasphasecorr
Specifies if the data has already been measurement phase corrected

Isunfolded
Specifies if the data has already been SENSE reconstructed

Labels
Contains all the information from the parameter file. The Labels struct can be different for the different file formats. Most of the parameter in the Labels struct are passed to other parameter groups such as the encoding or Scan parameters.
In raw-files Labels.Index contains the label information for every individual profile in the order they have been measured.

DATA

Contains the data at the current state of the reconstruction process. Every time a reconstruction function (method) is called the data is updated.
Data is a predefined 12-dimensional array where the single dimensions are defined as:
x – y – z – coils – dynamics - cardiac phases – echoes – locations – mixes – extr1 – extr2 – averages

