

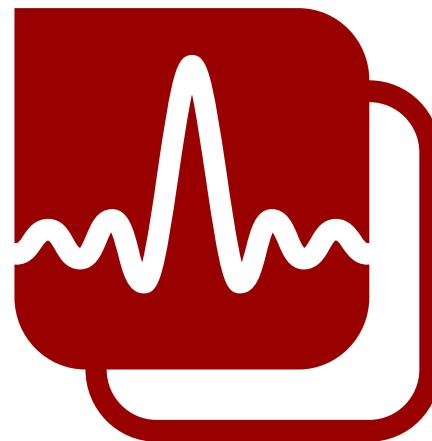
syngo MR E12U

IDEA Users Manual

IDEA

Integrated
Development
Environment for
Applications

syngo MR E12U



Erlangen, 2019

Manufacturer's note: Products that bear a CE mark fulfill the provisions of the Council Directive 93/42/EEC of 14 June 1993 regarding medical equipment.

The CE mark applies exclusively to medical equipment and products that are released under the relevant EU guidelines mentioned above.

Preface

Since the first delivery of a NUMARIS software version, it has been a SIEMENS tradition that external sequence programmers at universities and other research institutions have the possibility to write their own pulse sequences.

Before you start working with IDEA, you should be aware that the following is assumed and will not be covered in this manual:

- Proper use of NUMARIS/4 and its operating instructions.
- Basic knowledge of C/C++ is indispensable for simple sequence programming. Writing ICE programs or taking advantage of the more sophisticated sequence features requires a good understanding of C++. There are a number of books which cover C++; the most prominent one written by the originator of C++:
Bjarne Stroustrup: "The C++ Programming Language" (Addison-Wesley).
- Knowledge of basic MR principles, basic imaging concepts, Fourier transform concepts and fundamental applications. Here there are also a variety of textbooks; a good suggestion might be:
"Magnetic Resonance Imaging: Physical Principles and Sequence Design" by E. Mark Haacke, Robert W. Brown, Michael R. Thompson and Ramesh Venkatesan (John Wiley & Sons, 1999).

This IDEA manual is the official documentation, which covers both the operation of the tools and the generation of pulse sequences. It addresses the IDEA novice, but serves also as a reference for the experienced programmer. It consists of 5 parts:

Chapter A is the Installation guide. One important new feature of the NUMARIS/4 IDEA is that the complete development kit can also be installed on a standard office PC. This means that most of the sequence and ICE program development can be done in your office during regular working hours, and it is no longer necessary to spend the nights in front of the scanner with testing and debugging. You can prepare your application in the office, and then focus on image quality and scientific results in the usually rare time slots on the scanner.

Chapter B is a 'Getting Started' primer. It presents some fundamentals of working with IDEA that are demonstrated with a few examples. It is a good idea for a novice IDEA user to start with the first of these examples. As these examples are also used during the lab exercises of the IDEA course, the later ones are more complicated and should be skipped during first reading of this document until you are a bit more experienced.

Chapter C is the User's Guide which describes in detail all programs and tools provided by IDEA. On first reading, it is certainly not necessary to read all the details in the program descriptions, but get only a general idea about how everything works and what possibilities exist. Then refer to these sections later on if you have a special problem.

Chapter D is the Programming Manual. Here you can find all the information you need for a good understanding of how a pulse sequence works and how you can write your own sequences. This chapter deals both with the sequence itself and the syngoMR framework which provides the interface for the sequence. Later on, you find some more advanced topics, concerning both MR applications and C++ features.

If you are interested in the actual interfaces for each library class, you will not find this information in this documentation, but in the doxygen documentation, included as part of the IDEA installation. This file has been exported in HTML format and represents the ultimate reference for any library module. Since the doxygen documentation is generated out of the code it is always up to date - this means you get the complete information about the libraries.

If you have old sequences you would like to port to E12U please be aware of a new feature: starting with D13C support for different B1 shim modes is introduced. Whereas in previous versions it was sufficient to calculate and export the global energy transmitted by the sequence, more detailed knowledge on the specific absorption rate (SAR) is required when using the new B1 shim modes. Please refer to the description of the SeqIF::prepare method for more details on how to export the sequence energy in such cases.

Finally, there is also a topic that is not (yet) covered by this manual: in the current version, you can find all information about ICE (Image Calculation Environment) in an additional manual.

For your daily work with IDEA, there are some other points left to mention:

You are not alone with your problems. Other people who started earlier likely experienced the same pitfalls and now know how to avoid them. To communicate this information, there is a user group you can contact directly via the web: www.mr-idea.com. This web site is organized like a discussion board, and you are invited to take part. There you will also find the mail address for contacting the authors of this documentation; corrections or suggestions for improvements are always welcome. When accessing the IDEA information board in the Internet, we ask you to read and agree to follow the „Terms of Use“ as they are listed in this information board.

Up to now, the sequence developer has never had as much power and possibilities on a MAGNETOM scanner as with IDEA. But please be aware that it has never been so easy to do damage to your MAGNETOM scanner and to the NUMARIS software as with IDEA. Keep strictly to the rules indicated in this manual and do not try to alter NUMARIS. If you have any doubt about the operating condition of your system, you should not use it any longer, but call your service engineer.

Remember: **IDEA is a tool for investigational use.**

Because magnetic resonance image quality is fundamentally dependent on the timing and parameters of the acquisition sequence, you must ensure clinical utility for any and all new sequences developed for use on humans through sufficient testing.

In addition, it is possible to inadvertently create and incorrectly label images with sequences developed using IDEA. All labeling of images from investigational sequences must be verified by the developer on phantoms before use on humans. Examinations performed using sequences generated with this package should be for investigational purposes, and should not be used for diagnosis without corroborative examination using SIEMENS standard sequences.

Please follow any local laws and regulations governing patient scanning. Before commencement of the clinical investigation, it may be necessary to obtain the approval of the appropriate ethics committee of your research facility.

We recommend a detailed code reading and a complete unit test be performed with all new sequences.

Finally, some words of thanks. It is impossible to mention the names of all those people who designed, implemented, tested, improved and released NUMARIS/4 and IDEA - the list would just be far too long. But the contribution of each individual was essential for the total result and is gratefully acknowledged.

We hope you will appreciate IDEA as a valuable tool which enables you to put all your good ideas into practice and collect a cornucopia of exciting new results.

The best results start with the right IDEA!

Erlangen, 2019

General Safety Information

Proper, safe use of IDEA requires

- computational knowledge on the part of programming
- scientific knowledge about MR physics
- technical knowledge about the hardware and software user interface of a Magnetom MR scanner
- and a high degree of familiarity with the IDEA operating instructions.

Read these operating instructions carefully prior to first experiments with IDEA.



WARNING

IDEA is a tool limited to investigational use only.

Because magnetic resonance image quality is fundamentally dependent on the timing and parameters of the acquisition sequence, the user must assure sufficient testing of clinical utility for any and all new sequences developed for use on humans.

In addition, it is possible to inadvertently create and incorrectly label images with sequences developed using IDEA. All labeling of images from investigational sequences must be verified by the user on phantoms before use on humans. Examinations performed using sequences generated with this package should be for investigational purposes only, and should not be used for diagnosis without corroborative examination using SIEMENS standard sequences.



WARNING

Loud gradient noise might cause a hearing damage.

Most of the Siemens sequences are written in a way that keeps the gradient system away from the mechanical system resonance. This leads to a smaller noise exposure for the patients.

IDEA enables the programmer to make full use of the gradient system. During sequence operation, this noise might exceed the permitted levels and cause hearing damages to patients or operating personnel.

- Use hearing protection (headset or ear plugs) to protect patients against injury.
- Ensure that personnel in the examination room wear hearing protection during the examination.

**WARNING**

High vibration amplitudes can cause fatigue fractures to the heavy duty connectors to the coil.

Most of the Siemens sequences are written in a way that do not use echo-spacing close to the gradient coil resonance frequency.

IDEA enables the programmer to make full use of the gradient system. To avoid damages to the gradient system, the sequence developer needs to make sure he avoids parameters leading to echo-spacings corresponding to the critical frequency range Please see section E.

**WARNING**

Operating the MR scanner in resonance may lead to unexpectedly high helium boiloff.

Most of the Siemens sequences are written in a way that keeps the gradient system away from the mechanical resonances of the system.

IDEA enables the programmer to make full use of the gradient system. This allows sequences to be written using gradient waveforms that operate at these mechanical resonance frequencies. In this case an unexpectedly high helium boiloff may be observed.

Third Party Licenses

The seqence simulator tool delivered with IDEA contains the open source component CFolderDialog published on <http://www.codeproject.com>. The license terms for this component can be obtained from <http://www.codeproject.com/info/cpol10.aspx>.

A What's New	11
B Installation Guide	12
B.1 General Requirements	12
Hardware	12
Software.....	12
B.2 Required Software	14
B.3 IDEA installation	15
Additional software.....	15
Baseline.....	15
Installation modes.....	16
Update IDEA	16
B.4 Standalone Installation.....	16
Installation Procedure	17
B.5 Optional additional packages	17
B.6 Virtual Machine	17
User for MARS Linux connection	18
Virtual Environments	19
ExternalMars configuration.....	19
Mac OS X	20
Mixed Environment.....	23
C Hands On	28
C.1 General concepts	28
C.2 Hardened syngo MR software & Embedded Control.....	30
Deploy IDEA sequence or ICE Program with Embedded Control.....	30
C.3 IDEA Tools.....	33

change system	33
sequence unit test.....	34
view unit test result	34
sequence simulator.....	34
sequence visualizer.....	35
sequence unit test with own parameters	35
complete sequence unit test.....	35
unit test mask.....	36
change make configuration (and make)	36
toggle debug mode.....	37
change directory (to) ICE program / sequence	37
new current directory.....	37
list targets	37
exit.....	38
C.4 Protocol development tools	39
make protocol	39
Protocol offline editing tool (POET)	39
Import & export of .edx/.exar1	39
mephisto	39
Analyzing a sequence.....	40
Sequence test methods.....	40
Debugging methods.....	41
Debugging with Visual Studio.....	43
Tracing	44
Execution of a sequence	47
Starting points for sequence development	48
C.5 Hints	49
C.6 Exercise: UI and ICE	50
Introduction	50
Create a new sequence	50
C.7 Win64 support	58
Compile Win64 targets (ICE programs).....	58
Execute Win64 ICE simulation.....	58

D User's Guide.....59**D.1 General Overview59**

IDEA Directory Layout	59
Basic IDEA Program Tools	60
Summary of Program Flow	66
Binary Files.....	66

D.2 Development Tools.....68

Sequence Development Tools	68
SeqTestFrame.....	68
Protocol development tools	75

D.3 DSP Simulation 86

Sequence Simulator	86
--------------------------	----

D.4 Pulse Tool 110

Introduction	110
Software System Architecture.....	110
User Interface.....	111
RF Pulse files types.....	124

D.5 Working on a MAGNETOM 129

syngo MR Directory Layout	129
Sequence execution (MARS)	132
Additional Tools	133
Dump Protocols of the Exam DB	133
UTraceConfigUI	134
Raw Data	136

E Programming Manual 146**E.1 The Sequence in NUMARIS/4** 146

Critical frequency range	146
Sequences in NUMARIS/4	147
SeqIF::initialize in Detail.....	151

SeqIF::prepare in Detail	153
SeqIF::check in Detail	161
SeqIF::run in Detail.....	162
E.2 Real Time Events and MR Physics	167
The Real Time Event Block Concept.....	167
Real time imaging	169
Automatic real time execution of sequences	170
Real Time Feedback.....	170
SeqData communication from a sequence to ICE.....	173
k-Space Sampling in NUMARIS/4	176
Real Time Events	189
Defining the Coordinate System of the Current Event Block.....	192
RF Excitation	193
Data Acquisition	203
Phase Encoding	213
Traveling k-space using Gradient Moments	217
Practical Aspects of Implementation.....	220
E.3 Beyond the Timing Table	224
Reordering	224
LibKSpace - library for calculating the k-space trajectory.....	232
iPAT	250
The Sequence Unit Test.....	268
Using libSBB	273
E.4 UI Sequence Interface.....	280
Introduction	280
Standard Interface	280
UILink Framework	283
WIPParameterToolkit (WPT)	291
Soft Limits and Binary Search.....	293
Resource Identifiers	294
Description of UI handler functions.....	295
get-label-id	297
get-unit-id	298
Additional Methods of MrUILinkBase	308

List of search keys	309
For a complete list, please refer to \\n4\\pkg\\MrServers\\MrProtSrv\\MrProtocol\\UILink\\MrStdNameTags.h.....	327
Direct Protocol Access:	327
E.5 Advanced Sequence Programming.....	328
Arbitrary RF Pulses	328
E.6 Spectroscopy	335
Introduction	335
Common Concepts	335
Ice Program	345
Spectroscopy Output Files.....	348
X-Nucleus Operation	349
E.7 pTX	353
Introduction	353
pTX pulse interface to the scanner.....	364
Data interface from the scanner.....	366
pTX pulse sequence programming.....	369
F Miscellaneous	370
F.1 MR Coordinate Systems	370
Device Coordinate System (DCS)	370
Patient Coordinate System (PCS).....	371
Gradient Coordinate System (GCS) a.k.a. PRS Coordinate System.....	372
TableBoard Coordinate System (TBCS)	372
Table Coordinate System (TCS)	373
Series Block Coordinate System (SBCS)	375
Shim Coordinate System (ShimCS)	376
Pulse design coordinate system (LCS)	377
F.2 User Defined Diffusion Vector Sets	379
DVS File Format.....	380

A What's New

Please check the readme.hta file in the root folder of your IDEA DVD for information regarding latest changes in IDEA, e.g.:

- required installation setup and patch level, e.g. new Microsoft KBs starting with VE11A*
- updated or new IDEA commands, e.g. ExternalMars*

Since VE11C the syngo MR software is based on a security hardened base system. Please check section C.2 and your Operator Manual for further information and changes for IDEA users.

B Installation Guide

Before the installation of IDEA, please check the General Requirements.

B.1 General Requirements

Hardware

For the installation of IDEA, the following hardware requirements need to be fulfilled. The minimum hardware requirements for running IDEA are:

- 64 bit capable processor
- Processor must support hardware virtualization (Intel VT-x / AMD-v). Intel and AMD provide tools to look this up in your computer.
- 15 GB free hard disk space
- 2 GB RAM

Software

Microsoft Visual Studio 2008 Professional with SP1 (including VCRedist updates, see below) and Microsoft Windows SDK v6.1 must be installed prior to IDEA. It is absolutely mandatory to follow the installation instructions to get all path settings right. This will take about 20 Minutes.

If not available at your local retailer, Microsoft Visual Studio 2008 Professional can still be obtained with a MSDN subscription directly from Microsoft.

Windows

The minimum software requirements for running IDEA are:

- Windows 7 x64
- A HTML browser such as Mozilla Firefox or Internet Explorer (IE)
- Microsoft Visual Studio 2008 Professional with SP1
 - Including latest Microsoft (security) updates for VCRedist (see B.2 Required Software /Microsoft Visual Studio 2008, for latest information also check out the readme.htm on your IDEA DVD)
- Microsoft Windows SDK v6.1.

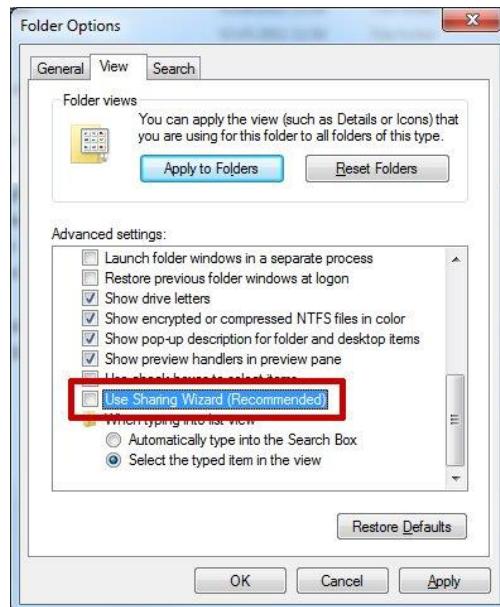
For the installation, administrative privileges are required on your system. Working with IDEA requires additional space; the amount depends on the number of projects under development and the simulations being run. If a number must be specified, 10.0 GB should be sufficient. These minimum system requirements should not be taken as a recommendation when considering buying a new workstation.

Network driver

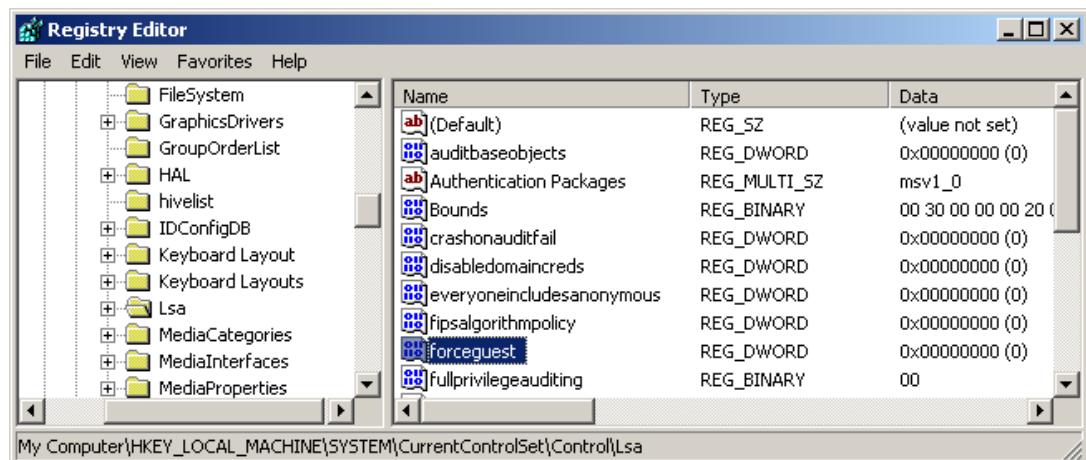
For internal communication between IDEA processes, AT command handling is used. Therefore it is necessary that a network driver is installed and the network has been configured. This is also valid if your system is not connected to a real network.

Additionally in Windows simple file sharing needs to be turned on . This can be done in two ways:

1. In Windows Explorer goto 'Tools -> Folder Options'. In the dialog window select the tab 'View' and scroll down the list and uncheck 'Use Sharing Wizard (Recommended)'.



2. Using the Registry with regedit.exe. Navigate to the registry key HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Lsa and set the value 'forceguest' to 0.



C++ compiler

Developing pulse sequences or applications within syngo-MR requires a C++ compiler. This compiler is

not included on the IDEA distribution DVD and must be purchased separately. IDEA is validated with the **Microsoft Visual C++ 9.0**.

On a standalone PC, the use of an edition or version of the MS C++ compiler other than this recommended one is at the customer's risk.

The **Microsoft Visual Studio 2008** must be installed before starting with the IDEA installation itself. It is recommended to install Microsoft Visual Studio 2008 before Microsoft Windows SDK. If you change the installation order the changes of the target directories described below will not be completely successful.

B.2 Required Software

Microsoft Visual Studio 2008

This section describes how to install Microsoft Visual Studio 2008 using the original Microsoft DVD.

- Login as a user with full administrator privileges, such as "administrator".*
- Insert CD-ROM and start installation by running DVD:\Setup.exe.*

Accept the default values, but change the following points:

Target directory: C:\MSVS2008

Note

Do not accept the default target directory proposed by Microsoft, but use C:\MSVS2008. Otherwise the target compilation will not work properly.

Installation type: Custom

Options:

- Select "Visual C++ Tools".*
- Select "Visual C++ Run-Time Libraries".*
- Select "Visual C++ Class & Template Libraries".*

Download and install SP1 for Visual Studio 2008 if installation did not contain SP1 already.

Also make sure the latest security patches for Visual Studio 2008 SP1 and VCRedist are installed.

Following patches are required: **KB2538243** and **KB2538241** (for latest information also check out the readme.hta on your IDEA DVD).

Microsoft Windows SDK

This section describes how to install Microsoft Windows SDK v6.1.

- Login as a user with full administrator privileges, such as "administrator".*
- You can download the installation from the official Microsoft website:
<http://www.microsoft.com/download/en/details.aspx?id=11310>*

Accept the default values, but change the following points:

Target directory: C:\WinSDK\Windows\v6.1

Note

Do not accept the default target directory proposed by Microsoft, but use C:\WinSDK\Windows\v6.1.

Otherwise the target compilation will not work properly.

Installation type: Custom

Options:

- Select "Developer Tools".

B.3 IDEA installation

Additional software

There is some additional software on the IDEA distribution DVD:

- Info-ZIP compression utility *NOTE: IDEA is packaged on this DVD using Info-ZIP's compression utility. The installation program uses UnZip to read zip files from the DVD. InfoZIP's software (Zip, UnZip and related utilities) is free and can be obtained as source code or executables from various anonymous-ftp sites, including <http://sourceforge.net/projects/infozip/files/>. The complete, unmodified package can be found on the IDEA DVD in the file \n4\pkg\MrApplications\MrIDEA\install\bin\zip+unzip.exe.*
- Cygnus GNU Unix tool. This software also is available from <http://sources.redhat.com/cygwin/>.

Note that this software is not part of the IDEA product and that it is not sold, but distributed for free.

Baseline

For every NUMARIS version, the matching IDEA version must be used. For example, for NUMARIS VA12A, use IDEA for VA12A; IDEA VA12B will not work. Never try to mix different versions!

Every NUMARIS package has a baseline label which identifies the build of this package, e.g. N4_VE12U_LATEST_20181030. It specifies both the software version and the date (in a yyymmdd format).

The baseline label of the IDEA package is printed on the distribution DVD. It is also stored in the file DVD:\baseline_snapshot.txt.

The baseline label of the NUMARIS software on the scanner can be found out by Help -> Info -> General.

Caveat: A NUMARIS version sold under a marketing name (such as syngo MR 2002B) can belong to different revisions (VA21A, VA21B, VA21C) and hence belong to different baselines (N4_VA21A_LATEST_20020801, N4_VA21B_LATEST_20021003 etc.). These revisions are different

baselines and must not be mixed.

Warning

The baseline labels of IDEA and the NUMARIS installation on the scanner must be identical! Different baselines must never be mixed.

If you have different baselines on your scanner and your stand-alone IDEA, never take an application from your PC to your scanner or vice versa.

Installation modes

There are two different installation modes for IDEA:

- Standalone Installation: IDEA can be installed on any standard PC or Laptop. This offers the possibility to develop applications without the need to sit in front of the NUMARIS console. This installation does not require syngo or NUMARIS and needs about 12GB of disk space.*
- Archive Installation: This mode is reserved for SIEMENS in-house developers who have access to the source code archive. This mode sets up only the registry but does not install any files.*

If the installation is aborted by the user, it can be restarted or repeated at any time without any problems.

Update IDEA

If there already exists an IDEA installation with another baseline on your standalone PC, it does not have to be uninstalled prior to the new installation, because the new IDEA.Net framework will take care of the different versions.

Unless the required version of the Microsoft Visual Studio or the Cygwin tools is also subject to change, there is no need for the deinstallation and reinstallation of this software package.

After an IDEA update (i.e. a change of the baseline), it is necessary to recompile all your sequences.

B.4 Standalone Installation

This chapter describes the installation of IDEA on a standalone standard Windows 7 x64 personal computer. No NUMARIS software is required for this installation. Sequences can be developed and the image reconstruction with raw data from the scanner can be simulated on the PC. The installation takes place in two parts:

- The installation of required software that is not distributed as part of the IDEA delivery (see A).*
- The installation of the Integrated Development Environment for Applications (IDEA) itself.*

Remarks:

Administrator privileges are required for all installations.

Installation Procedure

The installation of IDEA will take about 30 Minutes.

- Login under your usual login name (you must have administrator privileges).*
- Insert the IDEA DVD*
- Start a DOS command window and change to the DVD drive. Type "ideainstall.cmd" and follow the instructions in the window (to view additional informations to installation options, type "ideainstall.cmd -h").*

Remarks:

If there is not enough disk space on drive C: IDEA can be installed on any other drive (e.g., drive D:). You will be asked to specify the drive letter during the installation dialog.

IDEA requires a drive letter to be assigned to the IDEA root directory. The first unassigned drive letter is automatically selected from the alphabet starting from Z.

To check the results of the installation, go to Start -> IDEA -> IDEA.net. Select the desired baseline by entering its shortcut, e.g. CS0 (or its drive letter) and press enter, a new CMD Window should appear presenting the menu. During the first start of this baseline the command `initprod` is called, which sets up your virtual MAGNETOM configurations.

B.5 Optional additional packages

It is recommended to install some more software tools on your system. For legal reasons, they cannot be distributed on the IDEA DVD, but they can be downloaded from the web for free:

- DICOM Viewer. A list of free viewers can be obtained from e.g. <http://www.cabiatl.com/micro/>*
- Other viewers are e.g. Osiris, OsirisX, Sante DICOM Viewer.*

B.6 Virtual Machine

The virtualisation solution **Oracle® VirtualBox** is installed during IDEA's installation phase. This section covers additional information regarding troubleshooting and concurrent virtual environments.

Due to the fact, that the delivered virtual machine has a 64 bit operating system installed, it is required to have a 64bit capable CPU, which also supports virtualisation. The latter option might not be activated by the computer's vendor and can in general be activated in the computer's BIOS. See your computer's manual on how to activate it. If your IT department has restricted BIOS access, please consult your local IT administrator to enable hardware virtualisation.

The default installation setup for IDEA and the MARS Linux is:

Install IDEA on your host system. IDEA will install VirtualBox and a VirtualBox image of the MARS Linux. IDEA will then control the MARS Linux via VirtualBox commands.

If you do not want to use this standard setup (or cannot), please read the following sections regarding different IDEA and MARS Linux setups.

User for MARS Linux connection

By Default IDEA will use a locally created user called **`mars`** to connect the local view folder (shared as **`SnapshotView`**) into the MARS Linux. Initially the password for the user is created as random password.

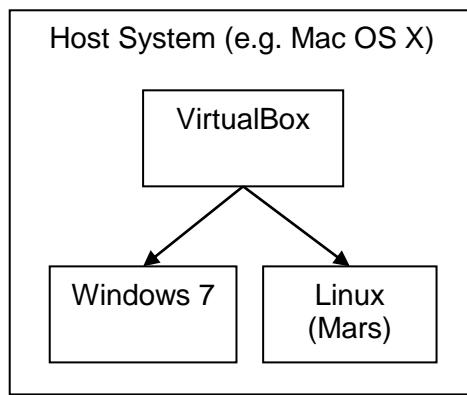
The password can always be recreated with **`vboxctl -inituser`**. This will use the configured **`ShareUser`** and **`ShareUserPasswd`** from the configuration file **`%MIDEA_HOME%\bin\syngo.MR.MrIDEA.idea.exe.config`** in Windows and on the MARS Linux.

If an interactive login or a domain login instead of a local login is required, one has multiple options:

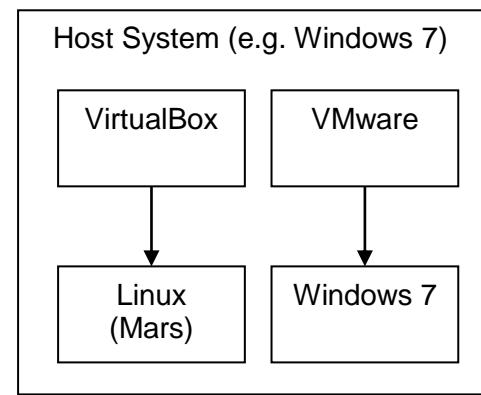
1. Modify the file **`~/cifs-credentials.txt`** on the MARS Linux (connect via **`ideassh`**):
 - a. set user and domain accordingly.
 - b. If no password is set, the mount command will ask for it interactively before each mount
2. Use the new option “**`-o`**” for the **`mntview.sh`** script:
 - a. Modify the configuration
`%MIDEA_HOME%\bin\syngo.MR.MrIDEA.idea.exe.config` and change the **`MountCommand`** element:
e.g. **`<MountCommand Command="%SSHCMD%" "chmod 755 /tmp/mntview.sh; /tmp/mntview.sh %MOUNTARGS% -o user=[username],dom=[domain or localhost],nounix,serverino,file_mode=0700; retValue=$?; echo Return: $retValue; exit $retValue"/>`**
 - b. By default IDEA will use following mount options:
`-o credentials=/root/cifs-credentials.txt,nounix,noserverino,file_mode=0700`

Virtual Environments

If you are running IDEA itself in a Virtual Machine already, you cannot run VirtualBox inside it. For this you have to have a special setup. In the section **Mac OS X** we assume, that IDEA is installed within Windows in a VirtualBox Environment. The following illustration briefly shows the system which is described exemplarily for Mac OS X:



Another setup may be, that you want to run IDEA inside a virtual Windows 7 using VMware. The instructions for this setup are described exemplarily for Windows 7 in subsection **Mixed Environment**. The next graph shows the assumed setup, from which this recipe was derived from:



If you are not using the default setup and your MARS Linux is not running inside the VirtualBox software on the IDEA system, then please also check our the section **ExternalMars configuration**.

ExternalMars configuration

The ExternalMars feature can be used if you are not running the MARS Linux on the same host as IDEA. Thus the MARS is running parallel to the IDEA host system and is connected and accessible from the IDEA host via network.

If enabled, the ExternalMars feature will disable the usage of the vboxctrl tool and prevent IDEA from assuming the MARS Linux is running inside the VirtualBox on your IDEA system. Instead IDEA will always try to connect to the configured ExternalMars IP (e.g. mc, sim). You can configure the IP upon first connection to an external MARS or with the “-ip” parameter of the ExternalMars IDEA command.

The ExternalMars feature can either be enabled during installation or later inside IDEA.

Enable ExternalMars during installation of IDEA:

- Answer the *ExternalMars question during installation* with Y
- Or specify the parameter “**-extm** or **-externalmars**” to the *ideainstall.cmd* script for installation

If you choose to enable the ExternalMars feature during installation, no VirtualBox and virtual mars

Linux image will be installed on your IDEA system and your installation will enable ExternalMars automatically inside IDEA.

Enable or disable ExternalMars feature inside already installed IDEA:

- Use the command `ExternalMars` inside IDEA to enable or disable the feature:

`ExternalMars true` will enable ExternalMars feature

If you enabled the ExternalMars feature during installation and want to use the default use-case for the MARS Linux, you have to install the VirtualBox software and MARS Linux virtual image first:

- Insert the IDEA DVD and run the VirtualBox installation from the /Downloads folder
- Start IDEA and disable ExternalMars: `ExternalMars false`
- Install MARS Linux virtual image: `vboxctrl -reinstallVM`

Then you should be able to use the MARS Linux image inside the VirtualBox on the same host system as IDEA.

Mac OS X

Please use VirtualBox in the recommended version. After the first start VirtualBox will create a storage folder in ~/Library/VirtualBox/, so start it once before proceeding.

If you already have VirtualBox installed and registered a Windows 7 virtual machine in it you need to reregister it in the delivered distribution. The easiest way to accomplish this is that you start VirtualBox and export your current Windows 7 as a Virtual Appliance. You can later import it into IDEA's settings again. If you want to avoid this, you need to create a new virtual machine in the delivered distribution and use your original virtual disk image (please see VirtualBox's handbook for details).

Extract the virtual Linux machine

Open up a terminal and unzip the file

`/Volumes/CD_ROM/n4/pkg/MrApplications/MrIDEA/VMConfig/vbox.zip` (whatever your CD/DVD-ROM is called) to the storage folder.

Warning: Before you run this command make a backup of the folder's contents, as it could overwrite your current configuration files!

Edit configuration files

Remarks:

This paragraph is only relevant if your host operating system is not Windows.

The directory `~/Library/VirtualBox/.VirtualBox/` is hidden in Finder. To make it visible execute this command: `defaults write com.apple.finder AppleShowAllFiles TRUE`. After this you need to kill all Finder processes: `killall Finder`. This will make all invisible files visible to Finder. To make them hidden again type `defaults write com.apple.finder AppleShowAllFiles FALSE` and kill all Finder processes again.

Next it is needed to modify the delivered configuration files in `~/Library/VirtualBox/.VirtualBox/`: Change to `~/Library/VirtualBox/.VirtualBox/` and edit the files `VirtualBox.xml` and `Machines/mars/mars.xml`:

`VirtualBox.xml`:

- change all Windows backward slashes to the common unix forward slashes*
- make sure the entries of the hard disks point to the right location.*

`Machines/mars/mars.xml`:

- remove the shared folder from the configuration, as it is not required and refers to a non existing path*

Add the virtual Windows machine

If you already have a Windows 7 registered in your VirtualBox installation you need to reregister it in the delivered distribution. The easiest way to accomplish this is that you start VirtualBox and export your current Windows 7 as a Virtual Appliance. You can later import it into IDEA's settings again. Start VirtualBox with the commands below, and import the afore exported Virtual Appliance:

```
export VBOX_USER_HOME=~/Library/VirtualBox/.VirtualBox/
```

```
virtualbox &
```

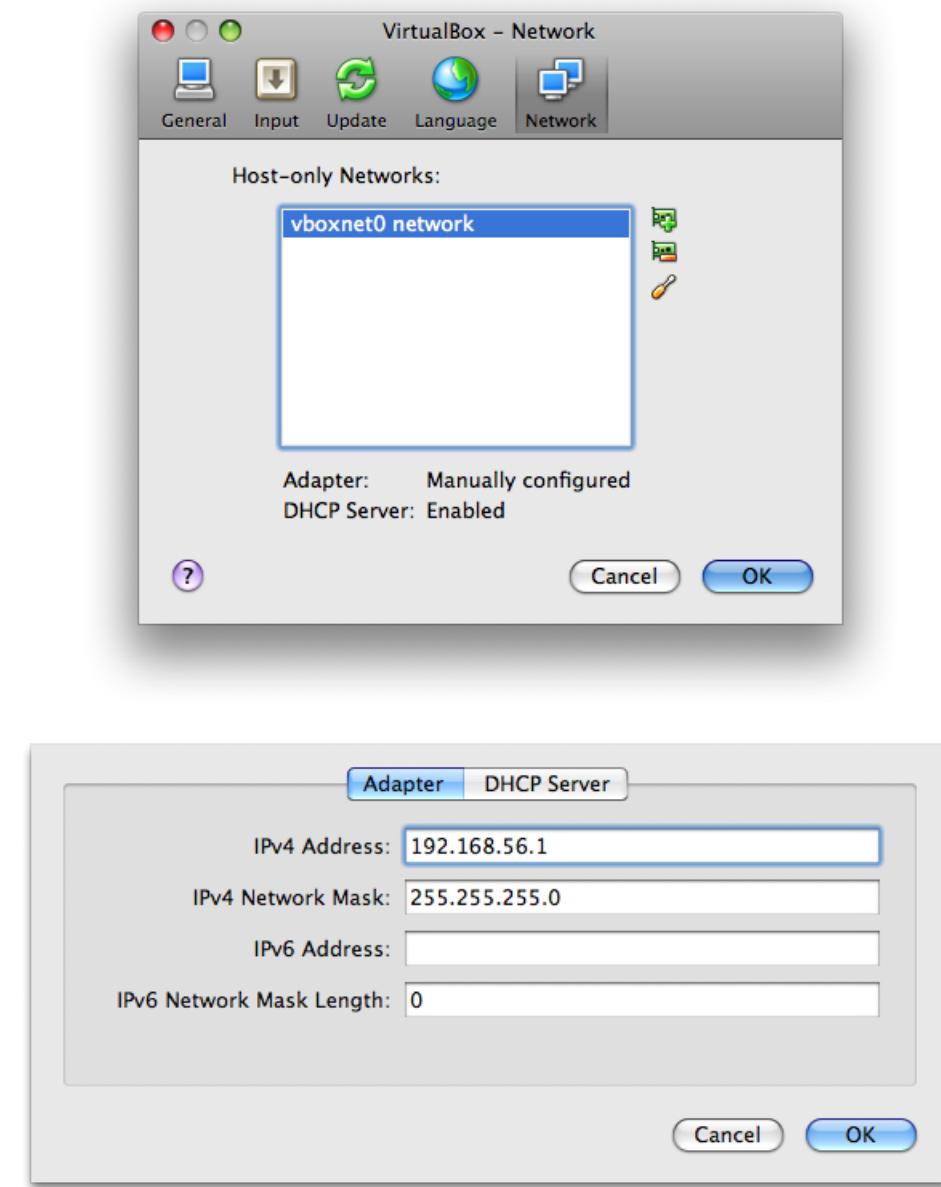
You could as well put them into a script, e.g. `StartIDEAVirtualBox.bash` and call this one instead.

Remarks:

IDEA uses the environment Variable `VBOX_USER_HOME` to tell VirtualBox to look in a different folder for its configuration files. IDEA uses this approach to separate itself from existing configurations.

Network Setup

Before you can start both virtual machines, you need to check network settings. Start VirtualBox with the upper commands and goto `VirtualBox -> Preferences select network` and set the appropriate settings for the network (see screenshots)



To make networking more reliable assign static IP addresses to both virtual machines:

Start the virtual Linux and log in as user root (no password required). Once logged in, open the file `/etc/networking/ipupdown` using vi. Change the adapter settings for eth0 to become a static assigned IP address:

```
iface eth0 inet static  
    address 192.168.56.3  
    netmask 255.255.255.0
```

Save and close vi using the command ':wq'. Now you need to either restart networking (`/etc/init.d/networking restart`) or do a reboot (`shutdown -r now`).

In the Windows VM you can do this via the Control Panel, but please keep in mind, that you have to assign a static IP v4 address to the Local Area Connection and it has to be in the same subnet. E.g. assign 192.168.56.4 as IP and 255.255.255.0 as netmask.

After the IDEA installation inside Windows has been finished, it is necessary to modify the script MIDEA_HOME%\bin\vboxctrl.cmd by replacing its content with:

```
cp -p %MIDEA_HOME%\bin\vboxctrl.cmd %MIDEA_HOME%\bin\vboxctrl.cmd.org
echo @ECHO OFF > %MIDEA_HOME%\bin\vboxctrl.cmd
echo echo virtual machine is not on this pc... >> %MIDEA_HOME%\bin\vboxctrl.cmd
echo echo make sure it is started >> %MIDEA_HOME%\bin\vboxctrl.cmd
```

Next open the file %MIDEA_HOME%\bin\syngo.MR.MrIDEA.IDEA.exe.config and edit the value for <VBoxIP Value="192.168.56.3"/> to match the IP of the Linux VM, which you have assigned before.

Mixed Environment

A mixed Environment is e.g. running IDEA inside a virtualised Windows 7 x64 using e.g. VMware and running the Linux part parallel inside VirtualBox. This section describes how this can be realised.

Configure VMware network

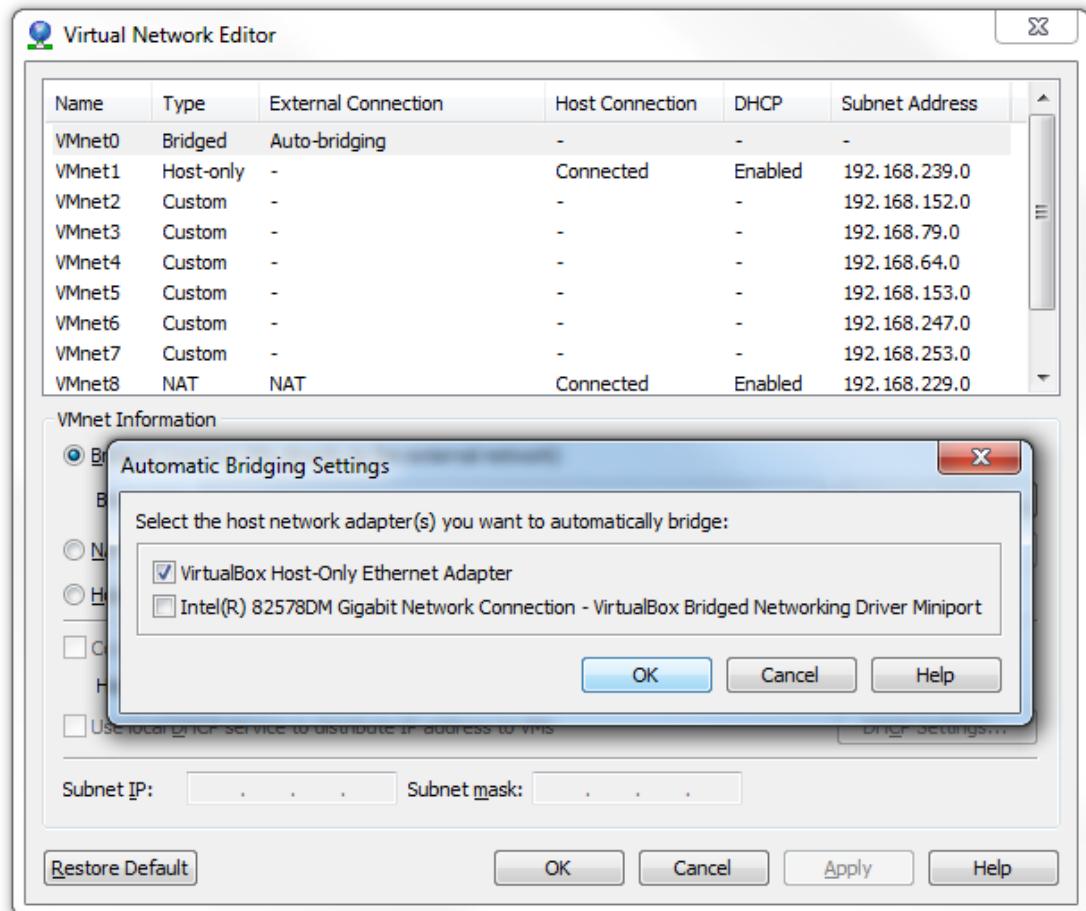
The nature of both virtualisation platforms, VMware and VirtualBox, in our case, does not allow communication between each other. To make this work, it is necessary to change VMware's networking configuration. VMware provides a tool called vmnetcfg.exe, which is a graphical network editor. This tool is not installed by the player's setup, so we have to extract it from the setup-file itself. VMware Server already includes the configuration utility.

In this guidance we use VMware player version 3.0.0 (VMware-player-3.0.0-203739.exe). To extract the archive use: VMware-player-3.0.0-203739.exe /e %TEMP%\extract3.0.0. Next switch to that temporary folder and open the file network.cab inside Windows Explorer. Copy the file vmnetcfg.exe to the player's installation directory.

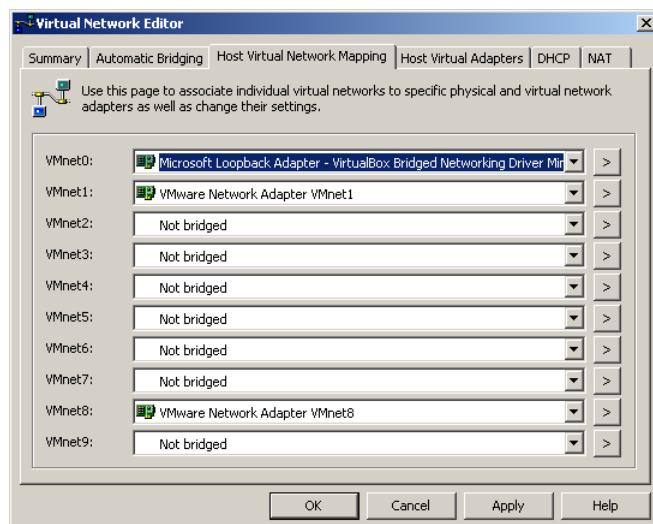
vmnetcfg.exe is typically located in %PROGRAMFILES%\VMware\VMware Player\vmnetcfg.exe, resp. %PROGRAMFILES(x86)%\VMware\VMware Server\vmnetcfg.exe on Windows 7 x64.

On Windows 7 x64 the vmnetcfg.exe must be copied to %PROGRAMFILES(x86)%\VMware\VMware Player\vmnetcfg.exe

Start vmnetcfg.exe and modify network settings for the virtual interface VMnet0:



On VWware Server the configuration dialog looks a bit different:



But please keep in mind to assign the bridged networking interface to the VMWare VM.

IDEA Installation

On the host create a directory, in which you want to extract the virtual Linux machine, e.g.:

C:\IDEA_VM. Insert the IDEA DVD and extract the file

[cddrive]:\n4\pkg\MrApplications\MrIDEA\VMConfig\vbox_[VXYYZ].zip to this directory:

```
[cddrive] :\n4\pkg\MrApplications\MrIDEA\install\bin\unzip.exe
[cddrive] :\n4\pkg\MrApplications\MrIDEA\VMConfig\vbox_[VXYYZ].zip -d C:\IDEA_VM
```

Now you can power up the virtualised Windows 7 and start installing IDEA, but do not forget to insert the DVD into the virtual machine (consult the handbook of your VMware product for further details). Then you can install the DVD the normal way. After the installation is finished, it is needed to modify the script %MIDEA_HOME%\bin\vboxctrl.cmd by replacing its content with:

```
cp -p %MIDEA_HOME%\bin\vboxctrl.cmd %MIDEA_HOME%\bin\vboxctrl.cmd.org
echo @ECHO OFF > %MIDEA_HOME%\bin\vboxctrl.cmd
echo echo virtual machine is not on this pc... >> %MIDEA_HOME%\bin\vboxctrl.cmd
echo echo make sure it is started >> %MIDEA_HOME%\bin\vboxctrl.cmd
```

Start IDEA for the first time

It is highly recommended to assign static IP addresses to both virtual machines. After having started the Linux VM, log in using the user root (no password required). Once logged in, open the file /etc/networking/ipupdown using vi. Change the adapter settings for eth0 to become a static assigned IP address:

```
iface eth0 inet static
    address 192.168.56.3
    netmask 255.255.255.0
```

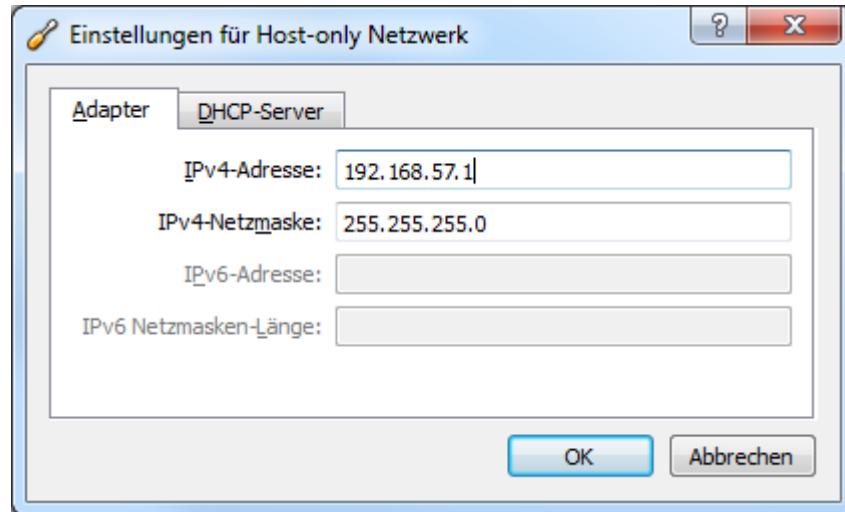
Save and close vi using the command ':wq'. Now you need to either restart networking (/etc/init.d/networking restart) or do a reboot (shutdown -r now).

In the Windows VM you can do this via the Control Panel, but please keep in mind, that you have to assign a static IP v4 address to the Local Area Connection and it has to be in the same subnet. E.g. assign 192.168.56.4 as IP and 255.255.255.0 as netmask.

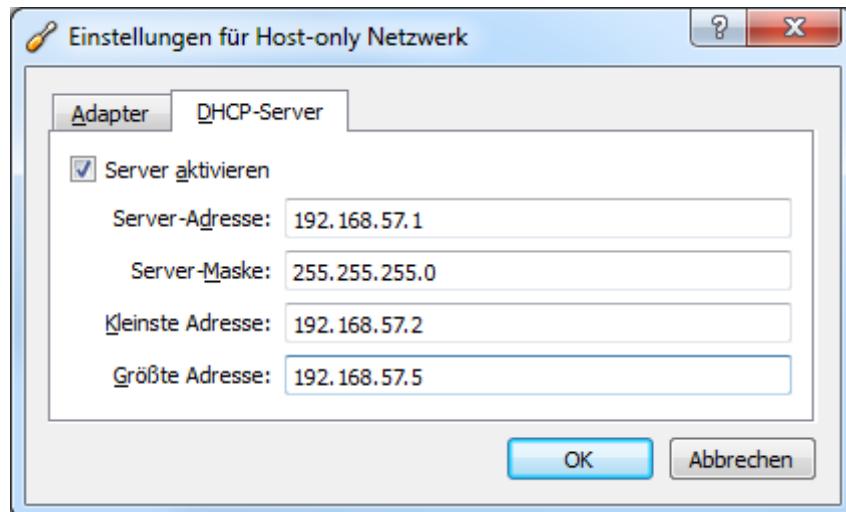
Next open the file %MIDEA_HOME%\bin\syngo.MR.MrIDEA.IDEA.exe.config and edit the value for <VBoxIP Value="192.168.56.3"/> to match the IP of the Linux VM, which you have assigned before.

Inside the Windows virtual machine please assign a different network to the VirtualBox Host-Only Network adapter, as it may interfere with the host's networking setup: start VirtualBox and go to File -> Preferences. In the upcoming dialog window click the screwdriver icon after having selected the Host-Only Adapter and change the IP-address of the adapter e.g. to 192.168.57.1. In the second tab

named DHCP Server you need to adapt the Server Address, and the bounds of the addresses that will be assigned to guests:



In the DHCP Server tab, enable the Server and adapt the addresses:



ipconfig should now write this to the console window:

```
C:\Documents and Settings\IDEA>ipconfig
```

```
Windows IP Configuration
```

```
Ethernet adapter Local Area Connection:
```

```
Connection-specific DNS Suffix . :
```

```
IP Address . . . . . : 192.168.56.4
```

```
Subnet Mask . . . . . : 255.255.255.0
```

```
Default Gateway . . . . . :
```

```
Ethernet adapter VirtualBox Host-Only Network:
```

```
Connection-specific DNS Suffix . :
```

```
IP Address. . . . . : 192.168.57.1
```

```
Subnet Mask . . . . . : 255.255.255.0
```

```
Default Gateway . . . . . :
```

Now you should be able to ping the Linux VM.

C Hands On

Introduction

This part of the IDEA manual contains a brief description of the processes for compilation and debugging of sequences and several exercises illustrating sequence development with IDEA.

C.1 General concepts

Targets

Sequences in NUMARIS/4 are dynamically linked libraries. Since two different operation systems are involved into the measurement, two different targets must be built: one for the host (Windows 7) and one for the MARS (Linux). On the host, the library files have the extension .dll, while the MARS files have the extension .so. The image reconstruction programs are .dll files for an x86 Intel processor, too.

Debug vs. Release

All targets and the test software exist as release and debug version (for step-by-step debugging). The debug targets are marked by an appended letter "d", e.g. se_15b130.dll and se_15b130d.dll.

Makefiles

A new makefile concept has been introduced with syngo MR D13. One type of makefile is used for all targets (i.e. debug / release, MARS sequences and ICE programs). This means you can build all flavors of sequence targets based on one single makefile.

The makefile may look like this:

```
##-----
## Copyright (C) Siemens AG 2010 All Rights Reserved. Confidential
##-----
##
## Project: NUMARIS/4
## File: \n4\pkg\MrServers\MrImaging\seq\FLASH\makefile.trs
## Version:
## Date: n.a.
##
## Descrip: Makefile for Sequence-DLL
##-----
##-----
```

NOSYNGO()

```
##-----
## local compiler flags
CPPFLAGS (-D_CONSOLE)
CPPFLAGS (-DBUILD_SEQU)
CPPFLAGS (-DSEQUENCE_CLASS_FLASH )
CPPFLAGS (-DSEQ_NAMESPACE=SEQ_FLASH)
CPPFLAGS (-DMRPROT_DEP_OPT)
```

```
##-----
## source files
CPPSOURCES (CompositeKernel)
CPPSOURCES (FISPKernel)
CPPSOURCES (FLASH)
CPPSOURCES (FLASH_UI)
```

```
##-----
## include paths
INCLPATHS (-I /n4_extsw/x86/extsw/MedCom/include)
```

```
##-----  
## libraries to be linked  
MSDEV_i586_WinNT400(LDLIBS (libUILink))  
  
LDLIBS (libRT)  
LDLIBS (libSSB)  
LDLIBS (libCSL)  
LDLIBS (libSL)  
LDLIBS (libSeqSysProp)  
LDLIBS (libSeqUT)  
LDLIBS (libSeqUtil)  
LDLIBS (libGSL)  
LDLIBS (libSSL)  
LDLIBS (libUICtrl)  
LDLIBS (MeasNucleiBase)  
LDLIBS (MeasNucleiIF)  
LDLIBS (MrGenericDC)  
LDLIBS (MrParc)  
LDLIBS (MrProt)  
LDLIBS (MrProtocolData)  
LDLIBS (MrTrace)  
LDLIBS (UTrace)  
LDLIBS (SeqBuffer)  
LDLIBS (Sequence)  
  
##-----  
## target name  
LIB (FLASH)
```

C.2 Hardened syngo MR software & Embedded Control

Starting with the syngo MR E11C version the operating system is hardened to provide best security for your system. This software includes the following changes in regards of security:

- Longer and more complex password rules possible, if used by customer (configuration: auto login off)
- Changes to Advanced User:
 - Advanced User has more rights than normal meduser account but is still running under the meduser account
 - Advanced User access is disabled by default. It can be enabled via Service UI
 - If you enable the Advanced User, you need to specify a site-specific non-standard password
- Virus scanner is replaced by the whitelisting software Embedded Control. Only registered software modules are permitted to be executed on the scanner



WARNING

With syngo MR E12U you can't execute scripts or programs not delivered with syngo MR E12U system itself while Embedded Control is in ENABLED mode (which is the system default).

Deploy IDEA sequence or ICE Program with Embedded Control

To be able to deploy new IDEA sequences, ICE programs, other IDEA artifacts to your hardened syngo MR E12U host or to execute own scripts and programs on your host, you have to switch the Embedded Control into UPDATE mode. After deployment of your files you have to switch back to the ENABLED mode to keep your system protected.



WARNING

Always make sure Embedded Control is ENABLED before starting an examination or scanning process with your system.

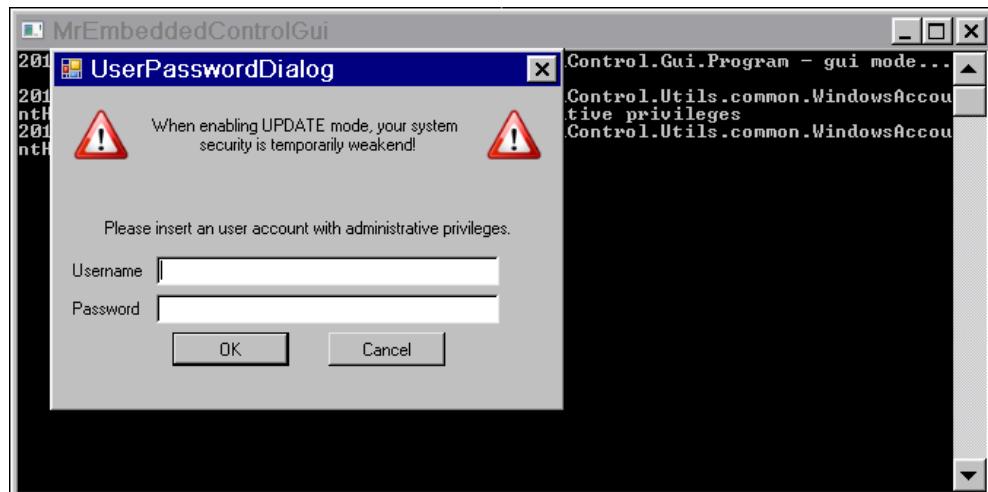
Your syngo MR E12U system is only CE-labeled if Embedded control is in ENABLED mode.

With the following steps you can switch the modes of the Embedded Control on your system:

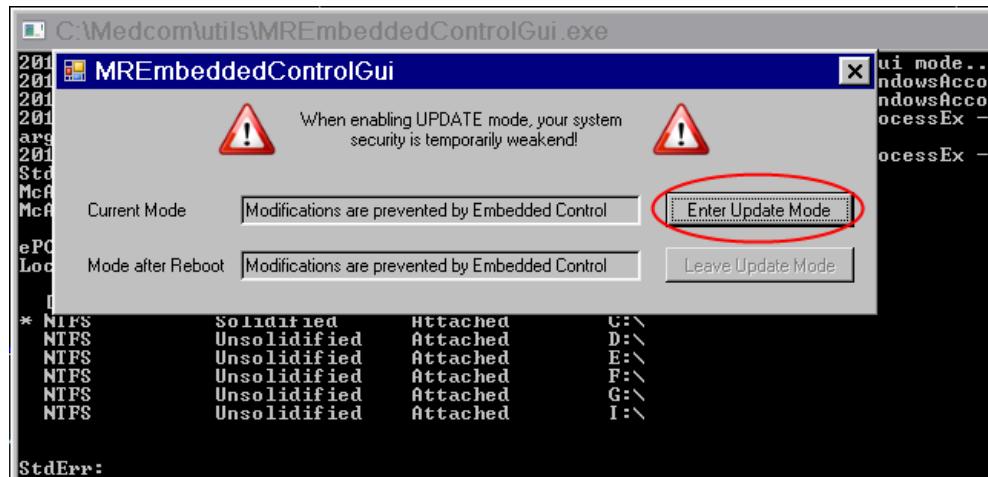
1. Switch into Advanced User mode (for the password, contact your IT-Admin, Lead Tech or Head of Radiology)
2. Press Ctrl+Esc to open the Startmenu
3. Start the MrEmbeddedControlGui tool (All Programs -> MrEmbeddedControl -> MrEmbeddedControlGui):



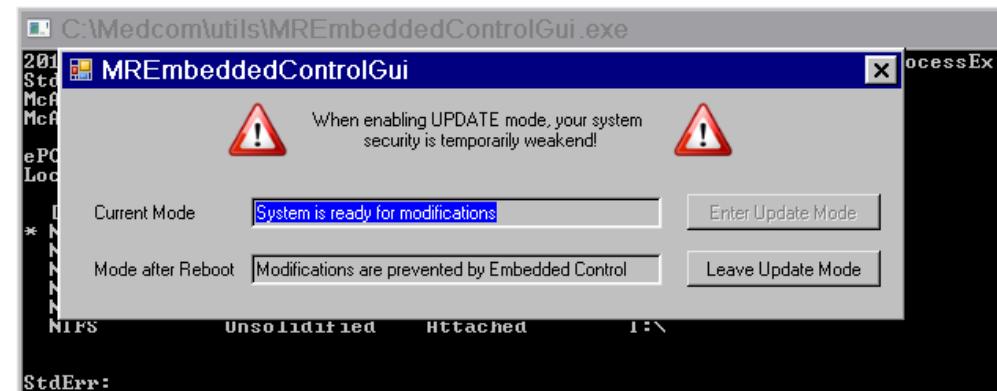
4. You will be prompted to give a special account to start the tool. You can use the SYADMIN account or any other system account inside the Administrators group. For the password, contact your IT-Admin, Lead Tech or Head of Radiology):



5. Switch the Embedded Control system into UPDATE mode by clicking the Update button:



6. The line "Current Mode" indicates that your system is now ready for your modifications. You can now deploy your IDEA files to the system or deploy own scripts and programs.



7. After finishing your modifications and to reenable security, press the button "Leave Update Mode". This reenables the Embedded Control security for your system.

Please make sure that the Embedded Control system is always running in ENABLED mode. The UPDATE mode is only allowed and required for copying your IDEA artifacts to the syngo MR E12U host.

C.3 IDEA Tools

Tools

For sequence development, there are several tools available, which are listed below with a short explanation and a link to their appropriate section:

1. Sequence Test Frame: All gradient instructions, ADC and RF events, and several additional parameters can be simulated, including gradient strengths and moments. See Page 60.
2. Sequence Visualizer: The above mentioned parameters can be graphically displayed. See page 94.
3. Protocol offline editing tool: For offline protocol development or testing UI changes. See page 76.
4. PulseTool: For examination of RF pulses, the Pulse Tool is available. See page 110.
5. DotCockpit: To run a sequence on a scanner, the DotCockpit of the User Interface is used. For more information on its use, see the MAGNETOM Operators manual.
6. Microsoft Visual Studio: For detailed debugging, one can use the Microsoft Visual Studio. See page 41 clause "Breakpoints in MSDEV".
7. Image Calculation Environment (ICE): All tools for image reconstruction programs are described in a separate ICE Manual.

IDEA and command help

On the next few pages, the most important IDEA commands are described. However, this list is not complete. Please call "`menu`" in the IDEA shell to obtain the full list. Furthermore, a detailed description of a command can be retrieved by calling the command with the option "`/?`" (e.g. "`cs /?`") or by calling help with the command as option (e.g. "`help cs`").

The main help page can be accessed with the "`help`" command inside IDEA.

sys

change system

The hardware specification and therefore also the sequence behavior depends on the MAGNETOM system (e.g. Aera-XJ). Therefore, the correct system to be used must be selected.

Enter "`sys`" at the IDEA command prompt. IDEA offers a list of available systems: choose one by typing its name on the command prompt.

Result: The system parameters are changed.

Note: The system "Measurement" is provided for comparison purposes and is usually not a system that is actually available for a given SW version.

ut

sequence unit test

The sequence unit test (page 60 paragraph SeqUT) is used to perform a comprehensive test of the sequence for the selected system and the adjusted protocol. The test checks various aspects like correct timing, image orientation, etc.

It is highly recommended to perform a sequence unit test on every new or modified sequence before it is used on the scanner!

If invoked without arguments, the sequence unit test uses the default protocol of the current sequence for the current system type.

The sequence unit test can be performed for the debug versions of the sequence .dlls, as well as for the release versions. While tests and simulations of release sequences are noticeably faster, the debug versions might give you more information due to debug assertions, etc.

Enter "ut" at the IDEA command prompt.

Result: The program SeqTestFramed is started and writes the test results to an html file. The location of this file is printed.

vut

view unit test result

The sequence unit test generates a HTML file that can be viewed with any HTML browser. If the text color is green, the sequence unit test completed successfully. Errors are displayed in red. The results can be quickly inspected with the view-unit-test-command.

Enter "vut" at the IDEA command prompt.

Result: A browser window appears showing the unit test result document.

sim

sequence simulator

The sequence simulator generates several .dsv files which contain the simulation results for the different hardware components (e.g. the gradient strengths as functions of the time). The simulation can be performed with debug and release .dll files. Note that it is not possible to simulate Siemens product sequences.

Enter "sim" at the IDEA command prompt (ensure that you are in the active sequence directory first).

Result: The unit test runs and the .dsv files are created in the current working directory (this may take some minutes).

vis

sequence visualizer

The sequence visualizer is used to have a graphical display of the output files created by the sequence simulator.

Enter "vis" at the IDEA command prompt.

Result: The visualizer window appears.

ut -p

sequence unit test with own parameters

If called without additional parameters, the sequence unit test will use the default protocol. It is also possible to run the sequence unit test with a modified protocol. However, modifying protocol files manually can lead to inconsistencies. Therefore, protocol files are typically modified by using the tool "mephisto" (see page 81). This program simulates the UI behavior on the basis of token files (extension: .tok). These files contain a list of parameter changes that shall be executed successively. The important feature is here, that parameter changes that are required for maintaining the consistency are performed implicitly. Finally, "mephisto" generates a protocol file (extension: .pro). Alternatively, the tool "poet" (see page 76) can be used on Standalone PCs.

The sequence unit test command switch "-p" accepts both .pro and .tok files and calls "mephisto" automatically if a .tok file is given.

Enter "notepad test.tok" and enter the following text: "slices = 5". Save and exit the editor.

Enter "ut -p test.tok" at the IDEA command prompt.

Result: From the input in the token file, a protocol file test.pro is created. The unit test is performed on this protocol file with 5 slices.

Enter "vut" to view the result.

NOTE: The location of the sequence .dll is contained in the protocol file.

ut -2

complete sequence unit test

Since a single (default) protocol will not necessarily detect all possible problems, an extended sequence unit test can be run which uses multiple standard token files on all available MAGNETOM systems. This test can be started by adding the option "-2".

Enter "ut -2" at the IDEA command prompt.(This takes some time, typically more than 5 minutes). Check the result with "vut".

Result: The sequence unit test result shows all tested protocols. Scroll through the document to see if all tests were successful (indicated by green font color).

mask**unit test mask**

For debugging purposes, it may be useful for the sequence to dump information into a log buffer. The type of information to be logged can be adjusted by setting a *debug mask*. Information will be dumped only if the corresponding bit in the *debug mask* is set. The "mask" command allows toggling these bits. If a hexadecimal number (format: 0x.....) is provided, the bits will not be toggled but set to that bit pattern.

- ❑ Enter "mask" at the IDEA command prompt and turn on bit 9 (prep) by entering "9" (bit 0 should be set by default). Start the Sequence Unit Test by entering "ut".

Result: Additional information generated inside of the preparation routine is dumped to the screen.

- ❑ Turn off all bits by entering "mask 0x0" and enter "ut".

Result: All bits of the mask are set to zero; no additional information is dumped.

mc**change make configuration (and make)**

The "mc" command is a convenient way to toggle between the different build modes. If called without arguments, it will allow you to toggle the different targets. If called with a number, the bits for the corresponding targets will be set.

After "mc", "ms" is called automatically, to compile the sequence or ICE program for the selected targets.

- ❑ Enter "mc 1" at the IDEA command prompt.

Result: The debug version is built.

- ❑ Then, enter "mc <enter> "1" <enter> "4" <enter> at the IDEA command prompt.

Result: The debug version (option 1) is deselected and the Linux target (option 4, extension: so) is built.

- ❑ Enter "mc 6" at the IDEA command prompt.

Result: The Linux version is deselected and the Windows release target (extension: dll) is built.

You can find the resulting files in:

- ❑ for .dll files (release and debug): \n4\x86\prod\bin\

- ❑ for .so files: \n4\linux\prod\lib\

After the .dll files are created on a standalone workstation, they may be copied to the appropriate directory (C:\Medcom\MriCustomer\seq) on the scanner.

Note that for convenience all build results are also copied to your %TEMP% folder; make sure not to mix up partial builds.

runmode

toggle debug mode

For testing the debug version of a sequence, the debug version of the test program must be executed. This can be ensured by calling the command "`runmode d`".

IDEA can also perform unit tests or other analyses using the release versions. The release mode can be turned on by calling "`runmode r`". This will not affect build settings, as they are configured by the command "`mc`".

- Enter "`runmode r`" at the IDEA command prompt and press <Enter>. Display the current state by calling "`runmode`" without any option.

Result: The command prompt will become e.g. [VE12U-CS-r].

- Return to debug mode by entering "`runmode d`".

ci / cs

change directory (to) ICE program / sequence

Change the current directory of an arbitrary target. If one wants to change to a sequence or ICE program the commands `ci` or `cs` can be used. If no parameter is specified both commands will display a list of all found programs and their source code directory. Specifying the corresponding number will change the current directory to the selected source code directory.

If a parameter is specified IDEA will match the parameter against all target names in the appropriate list (sequence or ICE). If only one matching target is found, the directory is changed without additional input. E.g. if you want to change to the source of the FLASH sequence, please enter `cs FLASH` and the current directory will become its source directory.

If more than one matching target is found, a list of all found targets will be displayed and the user can select one by entering the appropriate number. E.g. if you want to change to one of the "derive" ICE program demos, please enter `ci derive` and IDEA will display at least two ICE programs matching the name derive. Please select one by entering the appropriate number and the current directory will become the corresponding source directory.

ncd

new current directory

Change the current directory of an arbitrary target. If one wants to change to a program's directory not covered by the commands `ci` or `cs`, one can use this command to jump to its source location. E.g. if you want to change to the source of SeqTestFrame, please enter `ncd SeqTestFrame.exe` and the current directory will become its source directory.

lstarg

list targets

With this command one can search for targets registered in the used build system called MedMake. The search can be done using a substring of the target's name.

E.g: if one is searching for the target name of the sequence MINIFlash for the project n4 you enter:

```
lstarg -p n4 -t mini
```

This command can be used in conjunction with the command ncd.

exit

exit

IDEA exits on the command "exit".

C.4 Protocol development tools

For the development of protocols, there are the following tools available to facilitate this task. These are also called from within IDEA.

mp

make protocol

For testing purposes, it is often useful to use a protocol file in which all measurement parameters are stored. Every sequence has default parameters, which can be used to create a protocol file (page 80). This is done by "mp".

- Enter "mp" at the IDEA command prompt.

Result: The protocol se_15b130.pro is created.

poet

Protocol offline editing tool (POET)

In a standalone or archive environment, it is possible to modify protocols with the Poet tool (page 76). It uses a Protocol (.pro) file or ExaminationDataBase (.EDB) file as input and writes the modified protocol to the same file.

- Enter "poet se_15b130.pro" at the IDEA command prompt.

Result: A window appears that has the same functionality as the "OPEN protocol" tool on the MAGNETOM scanner. When closing it, you are prompted to save or discard the changes.

Import & export
of .edx/.exar1

Import & export of .edx/.exar1

Use the DotCockpit program to import .edx or .exar1 exam database exports into the USER tree of the exam database. This works only on MAGNETOM systems.

- In the DotCockpit, create a new region, new exam directory and a new program e.g.
"SequenceRegion\UserSequences\Test"
- Then use the Import button in DotCockpit to import an existing .edx or .exar1 into the currently selected program or use Export button to export current selected program to a .exar1 file .

For more information please refer to the MAGNETOM manual.

mephisto

mephisto

MEPHISTO (see page 81) is a tool to automatically create protocols that use (extreme) measurement parameters. It reads in a token file that contains commands that correspond to user interactions, like selecting a certain FOV, or number of slices etc.

- Create a file called meptest.tok in the current sequence directory by entering "notepad meptest.tok" at the command line. Enter the following lines:

```

 TR [All] = Max
 TE [All] = Max
 PhaseResolution =
 PhasePartialFourier = 75
 SliceThickness =
 SliceOrientation [0] = 6/8
 TE [All] = Min ?
 TR [All] = Min ?

```

The question mark means that the parameter will be set into a green region, i.e. without changing other parameters by solve handlers.

- Save and close the editor.
- Enter "mephistod %SiemensSeq%\se_15b130 meptest.tok" (on a scanner, enter "mephisto" instead of "mephistod").
Result: A new protocol file named meptest.pro is created.

NOTE: mephisto has its own help page (mephisto -h).

Analyzing a sequence

In most situations, one will modify a sequence .cpp file and then test the modified sequence. Here are some general strategies for testing and debugging sequences:

Sequence test methods

build sequence

Note that while sequences can be tested in the release version it is recommended to create the debug. This is done by the IDEA command "`mc 1`". After successful completion, the file `\n4\x86\prod\bin\sequencename.dll` exists.

default protocol

A sequence without a protocol is not particularly interesting. The measurement protocol provides the basic parameters necessary for sequence execution. For each sequence, there must be a default protocol that can be created and is consistent; that is, the default protocol is a set of parameters for the sequence that can be executed successfully. To create this protocol, the command "`mp`" exists (or `CrDefProt[d].exe`, see page 80). Once created, the protocol may be modified using POET: "`poet sequencename.pro`".

seqtestframe ut

The protocol can be tested using the IDEA command "`ut -p sequencename.pro`" (page 60 Section SeqUT). This will create an .html file that tells whether there are inconsistencies. To view the result, use the IDEA command `vut`.

simulator sim

The gradient and RF activity of a protocol can be simulated and visualized. It is recommended to reduce the number of lines/partitions (and TE/TR) to minimum values, since the simulation time is proportional to the real scan time. IDEA command "`sim -p sequencename.pro`".

visualizer vis

The results can be viewed using the visualizer: IDEA command "`vis`".

Debugging methods

Usually, there will be some programming problems that have to be resolved before the sequence will work. For this purpose, several tools and methods are explained here:

NLS_STATUS

Within NUMARIS/4, NLS_STATUS codes are used to signal the success or failure of a subroutine. Each error code represents a message that can be sent to the operator on the host console in a message window. NLS stands for Native Language Support, i.e., those messages are available in different languages.

Some examples:

- SEQU_NORMAL: Everything is ok.*
- SEQU_ERROR: An unspecified error occurred.*
- SEQU_NEGATIV_TEFILL: The specified timing cannot be realized due to a small TE.*

These codes are part of the standard message code set for all sequences and can be found in `\n4\x86\delivery\include\MrServers\MrMeasSrv\SeqIF\Sequence\sequmsg.h`.

These message codes are used within the Sequence preparation in the binary search mode. Depending on the code returned by `sequence prepare()`, the binary search will decide which parameter combination will be tested next. If an error code is returned outside the binary search mode, the preparation (and run) of the sequence is not possible, and an appropriate message will alert the user.

There are three user-specified NLS-error codes that can be used by the sequence programmer:

`SEQU_USER_ERROR_1, SEQU_USER_ERROR_2, SEQU_USER_ERROR_3.`

Use these to signal an error that is specific to your sequence modification.

SeqTestFrameSe qTestFrame options

Besides calling via the "ut" command, SeqTestFrame[d] (see page 68) can be invoked directly to gain additional information. Help on the SeqTestFrame[d] is available via the -h option; however, some options may be of special interest:

- "seqtestframed -p protocol.pro Event" generates an event dump of the given sequence protocol. Note that this is only possible with user-built sequences. Note also that the protocol contains the path to the sequence .dll.*
- "seqtestframed -s %customerseq%\flash TX" dumps all RF information of the sequence FLASHd.dll using the default protocol.*

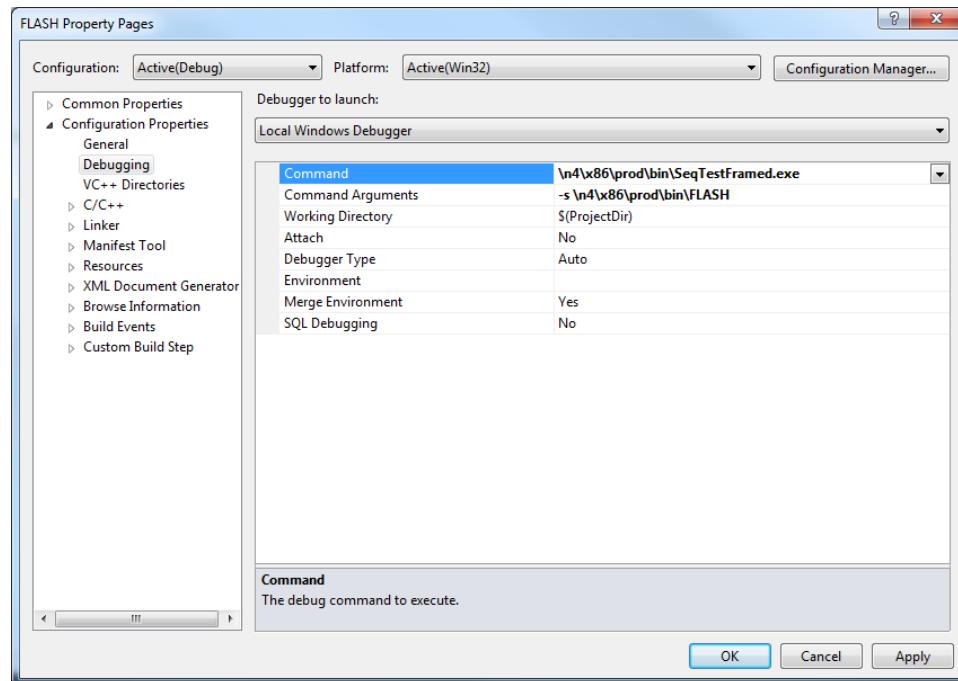
The above mentioned Debug Mask can be provided to SeqTestFrame[d] via the "-d" option.

Breakpoints in MSDEV

Sometimes it is necessary to go through a sequence step-by-step. To be able to do this with Microsoft Visual Studio, A breakpoint must be inserted into the seqtestframe.cpp file, since a copy of the sequence .dll is used. Simply follow these steps:

- Create a Visual C++ project file (*.vcproj).*

- Build the sequence within Microsoft Visual Studio.
- Have the following entries in the project settings:

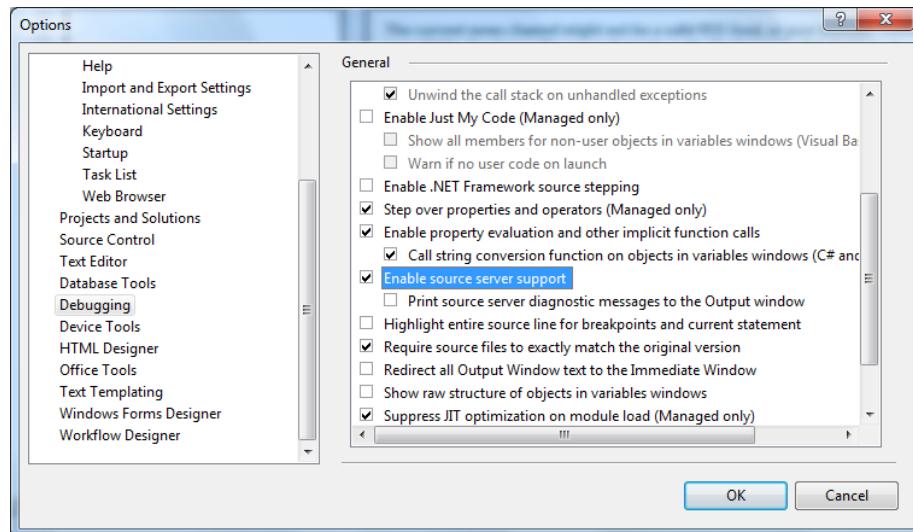


- Activate the sequence .cpp file and go to the point where the debugger should stop.
- Set a breakpoint
- Press "Go" (F5).
- The desired break point within the sequence .cpp file should be displayed.

Debugging with Visual Studio

To be able to debug source code with Visual Studio 2008 you need to activate the Source Server functionality of Visual Studio.

Therefore go to **Tools → Options** and enable “**Enable source server support**”.



Tracing

There are different methods for tracing. In sequences historically TRACE_PUT macros have been used, s. below. However, the underlying tracing mechanism ("UTrace") has become much more flexible to allow for finer selection of traces to be shown, depending on the module's **category**, a defined **marker** and the severity or **level** of the trace. Furthermore the target of the traces, so called **appenders** can be configured.

The configuration itself is performed via UTraceConfigUI (s. D.5) and becomes immediately effective after a modification. The log output is usually (depending on the appenders) redirected to the main log file UTraceSrv.utr (in %UTRACE_LOG_DIR% in IDEA, or C:\Medcom\log) and can be viewed using the tool **logviewer**.

UTRACE

UTRACE is the basic underlying macro for all tracing macros and methods; it offers highest flexibility and should therefore be used.

The macro itself and some derived variants are defined in
/MR/public/MrCommon/UTrace/Macros.h
and can simply be included via

```
#include "MrCommon/UTrace/Macros.h"
```

Parameters:

```
UTRACE ( level, marker, <streamed output>)
```

Example:

```
UTRACE ( Debug, 0, "Sequence prepare <" << rSeqLim.getLinkedSeqFilename() << "> started");
```

The trace **level** can be one of the following:

InOut, Debug, Info, Notice, Warn, Error, Crit, Alert, Emerg, Always, NotSet

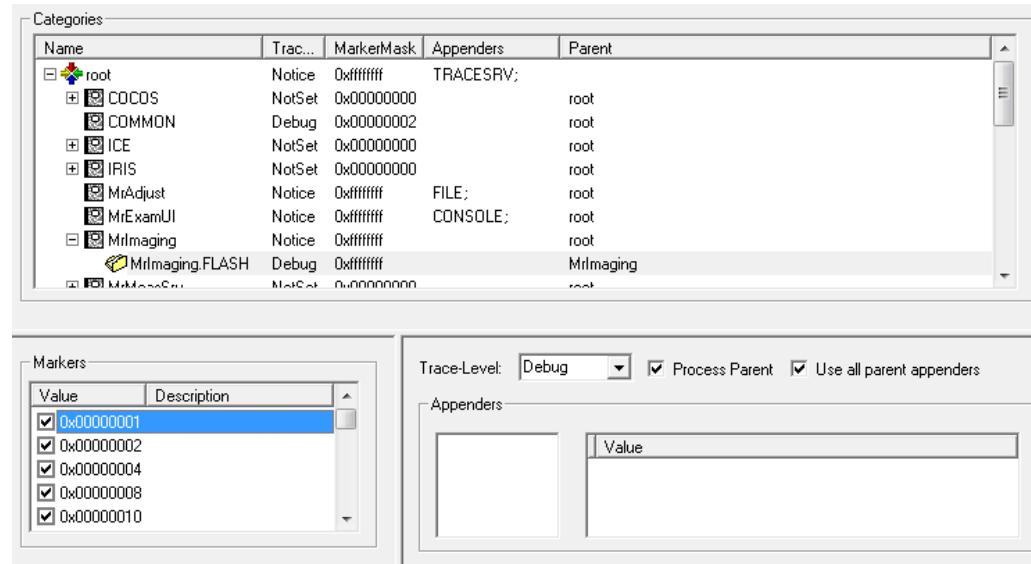
Note: A lower level (more to the left) includes all higher levels (e.g. when the "Alert" level is selected to be traced via UTraceConfigUI, all Emerg and Always level messages are traced as well).

The **marker** is a 32-bit mask identifying the trace message; the bit mask can be set in the trace configuration via UTraceConfigUI (s. D.5) in order to suppress or enable the trace. While any 32-bit value is supported for the marker, the message is traced as soon as any of the marker's bits are set in the trace configuration. A marker mask equal to 0 cannot be suppressed and the marker will always show up. With the bit wise assignment up to 32 markers can be distinguished per component; any combination of markers to be traced can be configured via the bit mask in UTraceConfigUI.

The **categories**, mentioned above, are determined automatically by the build chain and consist of the component and subcomponent name. The FLASH sequence, for example, is identified in UTraceConfigUI by "MrImaging.FLASH". If the category of a module is not found in the TraceConfig.xml, the settings of the next higher category become effective (e.g. MrImaging, and in a next step the root configuration).

A common reason why traces do not show up is that the level is not set correctly or the marker not selected. It is important that either an **appender** is specified specifically for a category, or that the appenders of the parent are processed; this has to be configured in UTraceConfigUI. Appenders are the targets for the traces and can be files, the console, memory, etc.

The following screenshot shows the required settings for the example UTRACE statement shown above, set at the beginning of the FLASH::prepare:



Note that MrImaging.FLASH does not have a specified appender; only if "Use all parent appenders" is selected, the TRACESRV appender specified in the root node becomes effective and the trace is written to the UTraceSrv log file.

TRACE_PUT

Note that the TRACE_PUT n macros are deprecated and should not be used anymore; they are described due to legacy support. The UTRACE macro described above should be used instead.

The TRACE_PUT n macros are defined in

```
\n4\pkg\MrCommon\MrNFramework\MrTrace\MPCUTrace\MPCUTrace.h.
```

They allow messages to be sent to the logging server during the execution of a sequence on the MARS, and to a log file on the host. The messages dumped by the MARS can be viewed on the host using the logviewer. Enter "logviewer" in a command prompt window on the host console, and you can view the messages as they appear. On the host, the output is written to the file c:\MedCom\log\UTraceSrv.utr. In all cases, date and time are dumped in front of the given message. The number n in the name indicates the number of variables in the trace; for example, TRACE_PUT1 accepts 1 variable and so on. It is a recommended convention to let ptModule be the name of the current routine.

Parameters:

```
TRACE_PUTn (class, facility, "format", var1, var2, var3);
```

Example:

```
TRACE_PUT2 (TC_INFO, TF_SEQ, "%s: current line: %d", ptModule, lLine);
```

Note that the TRACE_PUTn macros use the UTrace mechanism as described above in the section of the UTRACE macro; however, the class and facility parameters are not used anymore and will be ignored. The priority level is derived from the beginning of the message text (e.g. a text starting with "ERROR..." will result in an Error trace).

cout

Note that cout is not suited for the realtime context and should not be used at all.

Text written to the cout stream will also appear in the same file that is used by TRACE_PUT: C:\MedCom\log\UTraceSrv.utr. However, on the MARS, the cout stream is not redirected to a file.

mPrintTrace

Before the introduction of UTrace, in order to configure the debug output depending on an externally available setting, the mPrintTrace macros could be used. These macros check if some bits in the debug mask are set before printing debug info via TRACE_PUT. Just as with the TRACE_PUT macros, new code should avoid mPrintTrace and directly use UTRACE, since it is more flexible and can be configured easierly; furthermore any changes performed via UTraceConfigUI become immediately effective, unlike for debug masks used in the mPrintTrace macro, which require a restart of the process.

Parameters:

```
mPrintTrace3(severity, origin, "format", var1, var2, var3);
```

The appropriate codes for severity and origin can be found in

```
\n4\pkg\MrServers\MrImaging\seq\SeqDebug.h.
```

Some examples:**Severity codes:**

```
DEBUG_CALL, DEBUG_RETURN, DEBUG_INPUT, DEBUG_RESULT, DEBUG_LANDMARK, DEBUG_INTERNAL.
```

Origin codes:

```
DEBUG_INIT, DEBUG_PREP, DEBUG_CHECK, DEBUG_RUN, DEBUG_KERNEL, DEBUG_LIBSEQUIT.
```

Example:

```
mPrintTrace2 (DEBUG_RUN,
DEBUG_CALL, "(slice=%ld,line=%ld)", sSLC[1ChronologicSlice].getSliceIndex(),lLine);
```

This piece of code is usually used in fSeqRunKernel() to dump information only if the debug bits "DEBUG_RUN" and "DEBUG_CALL" are set.

mask

Now, how are these bits set? In IDEA, there is the "mask" command that toggles the appropriate bits of the debug mask. This debug mask is passed to SeqTestFrame[d] by IDEA's "ut" command and controls the debug output.

Execution of a sequence

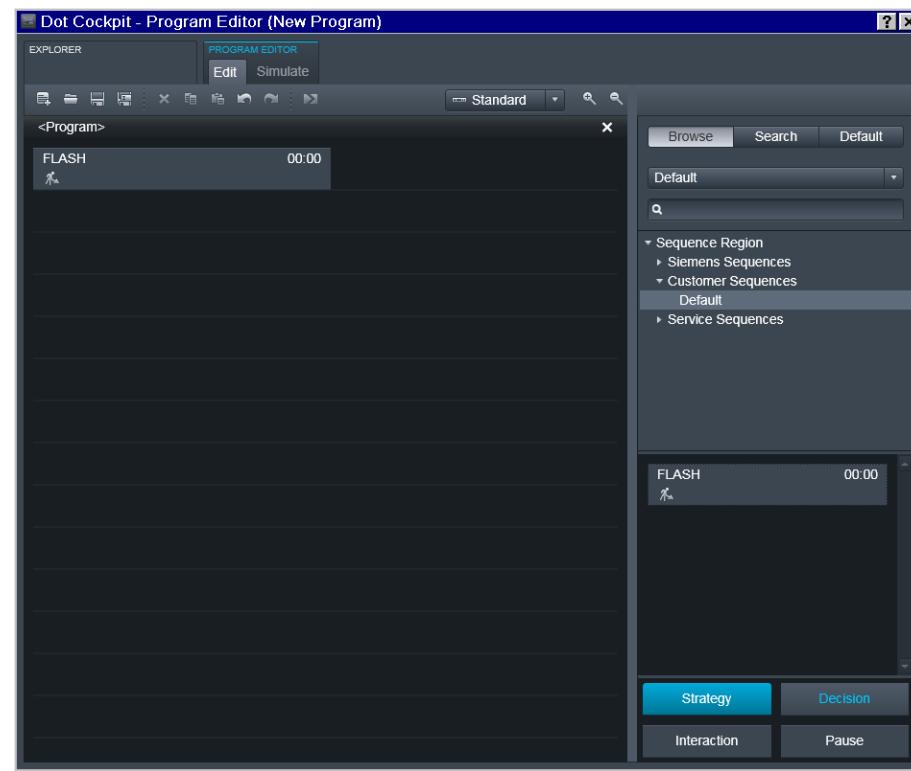
DotCockpit

Finally - if all tests and debug efforts were successful - to actually run the sequence...

First, ensure that the baseline versions for IDEA and syngoMR are identical. If not, DO NOT attempt to execute the sequence on the MAGNETOM, but recompile using the correct version of IDEA for the scanner. It will be necessary to compile the sequence .dll and the corresponding MARS .so using the appropriate IDEA commands (`mc`) (it is not possible to use Visual Studio since there is no cross-compiler for the MARS available). The resulting .dlls (and MARS .so) must be copied to `C:\Medcom\MriCustomer\Seq (%CustomerSeq% on a MAGNETOM)`. To actually run a sequence on a MAGNETOM, a default protocol has to be generated by the DotCockpit. Note that the DotCockpit is accessible via the Main menu item View->Dot Cockpit or via an icon next to the Parameter Card (the icon can be switched from "Program Card" to "Dot Cockpit").

In the USER tree, create a suitable region, e.g. TestRegion and a suitable exam (e.g. TestExam) by a right mouse click. In the Exam, click the "Edit" button or select "Edit..." from the right mouse click context menu of the exam. In the new Program Editor window select Browse on the right side, select Default in the Database selection and then select "Sequence Region/Customer Sequences/Default". On the lower right side of the Program Editor all your IDEA sequences which are available in `%CustomerSeq%` directory should be visible. You can add your sequence to the current program by drag&drop of the sequence to the left side of the Program Editor. Note that it is only possible to insert a sequence if a valid default protocol (page 39) can be created. You can try to create a default protocol yourself with Crdefprot or Mrdefprot programs. The logviewer may contain additional (error) informations.

If your IDEA sequence is not visible, please verify that the IDEA baseline used to build the sequence matches the MAGNETOM baseline. If your sequence is visible but cannot be inserted, please make sure all dependent libraries are located in `MedCom\bin` (Windows) and `MedCom\MCIR\Medllib` (Linux).



Starting points for sequence development

As a starting point for developing sequences, the [FLASH.cpp](#) sequence is provided, which is richly commented and does not use the SeqLoop class. Fewer comments are included in [a_se_b130_15000.cpp](#), which uses the SeqLoop class. A very short sequence is also included on the IDEA DVD, however mainly for demonstration purposes, the [MiniFLASH.cpp](#).

C.5 Hints

Useful hints for updating your sequences to a newer *NUMARIS* version can be found in the *IDEA Hints* documents in the IDEA installation (`\n4\pkg\MrApplications\MrIDEA\Manual\Hints\ldeaHints_VD11D.pdf`) as well as on the IDEA Discussion Board:

<https://www.mr-idea.com>

in the version specific folder under "Home > Documents > Siemens".

If you do not have login credentials to the Discussion Board please contact your responsible SIEMENS contact person or apply online for an account.

C.6 Exercise: UI and ICE

Introduction

In this exercise, the sequence se_15b130 shall use a new ICE program, which sets image pixels above or below a certain threshold to zero.

The exercise includes:

- adding a selection box (for the processing mode) and a long parameter (the threshold) box to the Sequence/Special card.
- handing the selected parameters to the ICE program (use both YAPS and Mdh as an example)
- creating the ICE program

Your first task:

Create a new sequence called se_15b130_ICE from the demo sequence se_15b130.

Note: This exercise includes references to some terms that will be explained in more detail later in this manual, e.g. UI handlers (E.4)

Create a new sequence

1. Start IDEA.
2. Make se_15b130 the active sequence (**cs se_15b130**)
(Bold type indicates user input).
3. Create new sequence directory and makefile with ns command
 - [se15b130d]> **ns**
 - Enter name of new sequence directory: **se_15b130_ICE**
 - Name of the new sequence DLL: **se_15b130_ICE(d).dll**, directory: ...
 - Enter name of template sequence directory [FLASH]: **se_15b130**

This command copies all required files to the new sequence directory.

Note that the names of the .cpp files are not changed.

4. Create a new sequence directory and edit the makefile (manually)
 - [se15b130d]> **md ..\se_15b130_ICE**
 - [se15b130d]> **copy * ..\se_15b130_ICE**
5. Adapt the new makefile:

[se15b130d]> notepad ..\se_15b130_ICE\makefile.trs

In the makefile.trs, change the line LIB(se_15b130) into

LIB(se_15b130_ICE)

Additional hints for adapting product sequences beyond the se_15b130:

- Please check also the list of CPPSOURCES in the makefile, containing the list of compiled cpp-files. If those contain (relative) path entries like `CPPSOURCES(..\a_se_b130_15000)`, you probably want to copy these files to the new sequence directory and remove the path part.
- The same applies, if there are entries like `CPPSOURCESFROM(SBBGREBase, ..\Kernels)` in `makefile.trs`. In that case, copy the file to the local directory and change the line to `CPPSOURCES(SBBGREBase)`.
- It is also a good idea to check for include statements for Kernels, which are shared by several sequences, like `#include "MrServers/MrImaging/seq/Kernels/SBBGREKernel.h"`. You might want to copy these .h-files also to the new sequence directory and adapt all include-statements accordingly.

6. Change to the new sequence directory: `cs se_15b130_ICE`.
7. register the new sequence with the IDEA command `regtarg` in the buildsystem. (You need to confirm for both the dll version and the .so-version with "y")
Hint: You can unregister the target using the IDEA command `unregtarg`).

compile

8. Test whether the new sequence will compile: `mc 1`.

unit test

9. Run a unit test on the sequence and look at the results

```
[se_15b130_ICEd]> ut
```

```
[se_15b130_ICEd]> vut
```

Now the exercise can begin. The first part will deal with the sequence only, and will illustrate how to add parameters to the Sequence/Special card and how to pass this information to the ICE program. Unfortunately, most of the code in the sequence is kind of standard code that one must use for the UI parameters, and will usually be copied from an already existing sequence; so if you want to skip this, advance to step (5).

The sequence / ICE program should do the following: Set all image pixels above or below a user defined threshold to zero. Therefore, we need two fields in the user interface:

- A box which contains long values and serves as the threshold
- A selection box which has three states: Do nothing / Cut above threshold / Cut below threshold

The ICE program must receive both settings and perform the appropriate calculations.

10. As a first task, you have to add the UI functions to `a_se_b130_15000_UI.cpp`. First, we want two new parameters on the UI (a selection box to choose the cutting method and a long type field to enter the threshold in), so we use an enum:

syngo MR E12U 2019	enum eSeqSpecialParameters {WIP_Selection_Box = 1, WIP_LongParameterBox};	© Siemens Healthcare GmbH confidential
-----------------------	--	---

11. For the user interface, several functions must be implemented (and finally registered); e.g., the labeling of the boxes. The variable lIndex is not really necessary in this example, but is needed if more than 1 selection box is used. Do not forget to enclose all code modifications for UI in `#ifdef WIN32 .. #endif`. Otherwise, the sequence will not compile on Linux, because these functions are not needed and not defined there:

```
unsigned _WIP_SELECTION_GetLabelId(LINK_SELECTION_TYPE*
    const _this, char* arg_list[], long lIndex)
{
    static const char pszLabelSelectionBox[] = "ICE cut moc";
    static const char pszInvalid[] = "Invalid";
    switch (lIndex)
    {
        case WIP_Selection_Box : arg_list[0] = (char*) pszLabelSelectionBox;
        break;
        default : arg_list[0] = (char*) pszInvalid ;
        break;
    }
    return MRI_STD_STRING;
}

// Now for the long parameter:
unsigned _WIP_LONG_GetLabelId(LINK_LONG_TYPE* const,
    char* arg_list[], long lIndex)
{
    static const char* const pszLabel0 = "Threshold";
    static const char* const pszLabelInvalid = "NotSupported";
    switch(lIndex)
    {
        case WIP_LongParameterBox : arg_list[0] =
            (char*)pszLabel0;
        break;
        default : arg_list[0] = (char*)pszLabelInvalid;
        break;
    }
    return MRI_STD_STRING;
}
```

- A selection box must contain some text, which is also given in a function:

```
const char* const pszFirstChoiceTextInSelectionBox =  
    "active";  
const char* const pszSecondChoiceTextInSelectionBox =  
    "below threshold";  
const char* const pszThirdChoiceTextInSelectionBox =  
    "above threshold";  
const char* const pszInvalid =  
    "Invalid";  
  
int _WIP_SELECTION_Format(LINK_SELECTION_TYPE* const pTh  
    unsigned nID, char* arg_list[], long lIndex)  
{  
    unsigned uVal = GET_MODIFIER(nID);  
    if (lIndex == WIP_Selection_Box)  
    {  
        switch(uVal)  
        {  
            case WIP_SelectionBoxNormal: arg_list[0] = (char*)  
                pszFirstChoiceTextInSelectionBox;  
                break;  
            case WIP_SelectionBoxCutLow: arg_list[0] = (char*)  
                pszSecondChoiceTextInSelectionBox;  
                break;  
            case WIP_SelectionBoxCutHigh: arg_list[0] = (char*)  
                pszThirdChoiceTextInSelectionBox;  
                break;  
        }  
    }  
    return 1;  
}
```

- The UI must be able to get and set the values (and to retrieve the possible options in case of the selection type box). For long and selection type boxes, there are separate set/get methods:

```

unsigned _WIP_SELECTION_GetValue(LINK_SELECTION_TYPE* const _this,
                                 long lIndex)
{
    unsigned nRet = MRI_STD_STRING;
    SET_MODIFIER(nRet,pThis->prot().wipMemBlock().alFree[lIndex]);
    return nRet;
}

unsigned _WIP_SELECTION_SetValue(LINK_SELECTION_TYPE* const _this,
                                 unsigned nNewVal, long lIndex)
{
    pThis->prot().wipMemBlock().alFree[lIndex]=GET_MODIFIER(nNewVal);
    return pThis->value(lIndex);
}

bool _WIP_SELECTION_GetOptions(LINK_SELECTION_TYPE* const _this,
                               std::vector<unsigned>& rOptionVector, unsigned long lIndex)
{
    if (lIndex == WIP_Selection_Box)
    {
        rulVerify = LINK_SELECTION_TYPE::VERIFY_ON;
        rOptionVector.resize(3);
        for (long lI=0; lI<3; lI++) {rOptionVector[lI] = MRI_STD_STRING;};
        SET_MODIFIER(rOptionVector[0],WIP_SelectionBoxNormal);
        SET_MODIFIER(rOptionVector[1],WIP_SelectionBoxCutLow);
        SET_MODIFIER(rOptionVector[2],WIP_SelectionBoxCutHigh);
        return true;
    } else return false;
}
// Also for the long parameter:

long _WIP_LONG_GetValue(LINK_LONG_TYPE* const _this, long lIndex)
{
    return _this->prot().wipMemBlock().alFree[lIndex];
}

long _WIP_LONG_SetValue(LINK_LONG_TYPE* const _this, long lIndex)
{
    return (_this->prot().wipMemBlock().alFree[lIndex] =
}
```

- For the long parameter box, the valid limits of the parameter must be given. The limits are given as a vector, which is filled with the desired steps:

```

bool _WIP_LONG_GetLimits(LINK_LONG_TYPE* const _this,
                         mrstd::vector<MrLimitLong>& rLimitVector, unsigned lo
                         rulVerify, long lIndex)
{
    long lMin, lMax, lInc;
    switch(lIndex)
    {
        case WIP_LongParameterBox lMin= 10 ; lMax= 4095; lInc=1; break;
        default : lMin= 0 ; lMax= 0; lInc=1; break;
    }

    rulVerify = LINK_LONG_TYPE::VERIFY_BINARY_SEARCH;
    rLimitVector.resize(1);
    rLimitVector[0].setEqualSpaced(lMin,lMax,lInc);
    return true;
}

if (LINK_SELECTION_TYPE* pSelection = _create< LINK_SELECTION_TYPE >(pSeqLim, MR_TAG_SEQ_WIP1, WIP_Selection_Box))
{
    pSelection->registerGetLabelIdHandler(_WIP_SELECTION_GetLabelId);
    pSelection->registerGetOptionsHandler(_WIP_SELECTION_GetOptions);
    pSelection->registerGetValueHandler(_WIP_SELECTION_GetValue);
    pSelection->registerSetValueHandler(_WIP_SELECTION_SetValue);
    pSelection->registerFormatHandler(_WIP_SELECTION_Format);
}

if (LINK_LONG_TYPE* pLong = _create< LINK_LONG_TYPE >(pSeqLim, MR_TAG_SEQ_WIP2, WIP_LongParameterBox))
{
    pLong->registerGetLabelIdHandler(_WIP_LONG_GetLabelId);
    pLong->registerGetUnitIdHandler(_WIP_LONG_GetUnitId);
    pLong->registerGetValueHandler(_WIP_LONG_GetValue);
    pLong->registerSetValueHandler(_WIP_LONG_SetValue);
    pLong->registerGetLimitsHandler(_WIP_LONG_GetLimits);
}

```

- Finally, all of these functions must be registered with the UI in `Se15b130UI::registerUI ()` in `a_se_b130_15000_UI.cpp`. This is done by the following piece of code
- The position of the boxes within the Sequence/Special card is given by the number in `MR_TAG_SEQ_WIP1`, so the position of the new UI switches can be configured.

12. After this “copy and paste - action”, we can now start to think about the functionality itself. But first try to compile the sequence and run a “poet” on it to verify that the two parameter boxes in the UI do appear.

13. If you have looked at the UI selection box, you will have noticed that there was no default value - which is unacceptable. Therefore, modify the protocol so that there is a default value. However, during the normal preparation, one cannot write into the protocol. Therefore, one has to force a prepare in the "ContextPrepForMrProtUpdate" mode; i.e., if the parameters are not initialized yet, return an error and hope for a successful preparation in the subsequent call of sequence

```

if (pSeqLim->isContextPrepForMrProtUpdate())
{
    if (!pMrProt->wipMemBlock().alFree[WIP_Selection_Box]) pMrProt->wipMemBlock().alFree[WIP_Selection_Box] = WIP_SelectionBoxNormal;
    if (!pMrProt->wipMemBlock().alFree[WIP_LongParameterBox]) pMrProt->wipMemBlock().alFree[WIP_LongParameterBox] = 400;
}
if ((!pMrProt->wipMemBlock().alFree[WIP_Selection_Box])
    || (!pMrProt->wipMemBlock().alFree[WIP_LongParameterBox]))
)
return SEQU_ERROR;

```

prepare(). In this context, the protocol can be adapted. The code looks like this:

14. Now we are ready to pass the information to the ICE program. There exist two possibilities: via the YAPS, for parameters that do not change during the acquisition, and via the Mdh, for parameters that change from line to line. Unfortunately, the example has only data of the first kind, but as an exercise, send the threshold value via YAPS and the method via Mdh. In `Se15b130::prepare`, `seqExpo` is filled, so here is the proper place for this parameter:

```
pSeqExpo->setICEProgramParam(6,pMrProt->wipMemBlock()
.alFree[WIP_LongParameterBox]);
```

There is an array of ICEProgramParameters available; use the 7th position, since the ICE program may already use the indices 0-5.

The array size is given by `ICE_PROGRAM_PARA_LEN` in `Measurement\Aqc-Defs.h`.

15. The Mdh is informed about the threshold method via a "FreeParameterByIndex" array. Here we choose the index 0:

```

switch (pMrProt->wipMemBlock().alFree[WIP_Selection_Box])
{
    case WIP_SelectionBoxNormal: sCutMode = 0; break;
    case WIP_SelectionBoxCutLow: sCutMode = 1; break;
    case WIP_SelectionBoxCutHigh: sCutMode = 2; break;
    default: TRACE_PUT1(TC_INFO, TF_SEQ, "%s: Unknown cut mode\n", ptModule); return SEQU_ERROR;
}
sADC01.Mdh.setFreeParameterByIndex(0,sCutMode);

```

-
16. Before we turn to the ICE program, we have to tell the sequence, which one to use: This is done by replacing the call of `SeqLoop.setIceProgram()` by this line:

```
pSeqLim->setICEProgramFilename (   
    "%CustomerIceProgs%\MyIceProgram");
```

17. Now we're done with the sequence.
18. Let's continue with the ICE program. Unfortunately, a step-by-step-solution could not be included in this manual, but will be given in the course only.

C.7 Win64 support

Compile Win64 targets (ICE programs)

To compile Win64 targets the following MedMake commands have to be used inside IDEA (mc, mt, ms, mi IDEA commands don't support Win64 targets).

Register Win64 Target

Keep in mind that all required targets need to be registered inside MedMake n4x64 configuration with the following command executed from inside the targets source directory:

```
medregister -p n4x64
```

Build Win64 Debug Target & evp

To build a Win64 debug target, execute the following commands:

1. MedMake -p n4x64 -mf -r -m make -bm DebugIDEA <targetName_with_d>.dll
2. MedMake -p n4x64 -mf -r -m make -bm DebugIDEA <targetName_with_d>.evp

e.g.

1. MedMake -p n4x64 -mf -r -m make -bm DebugIDEA IceInverterd.dll
2. MedMake -p n4x64 -mf -r -m make -bm DebugIDEA IceInverterd.evp

Build Win64 Release Target & evp

To build a Win64 release target, execute the following commands:

1. MedMake -p n4x64 -mf -r -m make -bm ReleaseIDEA <targetName>.dll
2. MedMake -p n4x64 -mf -r -m make -bm ReleaseIDEA <targetName>.evp

e.g.

1. MedMake -p n4x64 -mf -r -m make -bm ReleaseIDEA IceInverter.dll
2. MedMake -p n4x64 -mf -r -m make -bm ReleaseIDEA IceInverter.evp

Execute Win64 ICE simulation

The ICE simulation supports execution and simulation of Win64 targets.

Start ICE for Win64

Start the ICE simulation with the `-win64` parameter to enforce Win64 usage. Keep in mind that all used binaries and ICE programs need to be compiled for Win64 first.

```
IceStart -win64
```

Simulate ICE program in Win64

To simulate a `meas.dat` in Win64 environment ensure that the ICE environment is started with the `-win64` parameter and execute `IceSimu`:

```
IceSimu meas.dat
```

D User's Guide

D.1 General Overview

IDEA Directory Layout

When installed, all IDEA version related files reside in a directory e.g. 'C:\MIDEA\N4_VE12U_LATEST_20181030'. This directory can be accessed directly by the drive substitution done automatically by IDEA (e.g Z:). On a standalone installation, the directory is substituted. (for example: Z:\ = C:\MIDEA\N4_VE12U_LATEST_20181030). The reason for this is that accessing drive Z: offers the same view on the file structure as in the source code archive. This is necessary due to fixed path references in some files.

NOTE: The IDEA home directory can be changed during the installation (page 15).

In this section, the main directory structure is explained. Please note that the archive structure has been completely revised in VA25A/VB11A for a better disentanglement between the different components of the software:

\n4

This is the NUMARIS/4 main archive root directory.

\n4\pkg

This tree contains all software packages. Each package usually belongs to an application (which has a user interface), a server (which is running as a background process) or to a common component (i.e. some central source codes used by several packages). In this directory tree, there are only sources and documentation files, but no executable target files that belong to a NUMARIS/4 build.

\n4\pkg\MrServers\MrImaging

The component of the imaging sequences consists of pulse sequences and support tools to test these sequences.

\n4\pkg\MrServers\MrImaging\seq

In this directory reside the pulse sequences. Each sequence has its own subdirectory. If a sequence directory starts with the prefix 'a_', this denotes that this sequence can run on all MAGNETOM systems.

\n4\pkg\MrServers\MrProtSrv

The protocol server component directory contains files for the user interface and the protocol access.

\n4\pkg\MrApplications\MrIDEA\install

These files are mainly needed during the installation of IDEA.

\n4\pkg\MrApplications\MrIDEA\Manual

Here are the IDEA manuals in the form of .pdf files.

\n4\pkg\MrServers\MrVista

The files in this sub tree belong to the Image Calculation Environment (ICE).

\n4\pkg\MrServers\MrVista\Ice

Here are the source codes of the ICE programs.

\n4\pkg\MrServers\MrMeasSrv\SeqIF

This subdirectory tree contains all libraries that are necessary to control or support measurement and control of the scanner.

\n4\x86\delivery\bin

In this directory all NUMARIS/4 executables (.exe, .dll and .ocx files) for the Host computer (x86 intel platform) can be found. They have been built at SIEMENS.

\n4\x86\prod\bin

This directory will contain all sequence binaries built within IDEA for the x86 intel platform.

\n4\x86\extsw\MedCom

This directory contains all binaries which are part of the syngo software.

\n4\linux\prod\lib

This directory will contain all sequence binaries built within IDEA for the Linux based MARS platform.

\n4\opensource

This directory contains opensource software that is used by NUMARIS/4.

Basic IDEA Program Tools

For a better understanding of the relationships between different files, this section provides a brief overview of the most important programs a sequence developer will use. A detailed description can be found in the following chapters.

SeqUT

This is a wrapper script for the Sequence Unit Test that calls the other test tools described below, such as SeqUT2 or SeqTestFrame. For a full description, refer to page 68.

SeqTest frame

The sequence test frame is a tool which simulates the execution of a sequence. All functions of the sequence (init, prep, check, run) are executed with the same underlying libraries as on the scanner. The difference is that the events will not be sent to the DSPs, but are either discarded or dumped to stdout. For a full description, refer to page 68.

libSeqUT

The Sequence Unit Test library cannot be executed directly, but it can be attached to the sequence .dll. It controls the correct calculation of the gradients and the timing of the sequence. The test results will be written into an HTML file.

imprint

This tool displays useful information about your scanner (e.g. the gradient properties) and your sequence. For a full description, refer to page 73.

SeqUT2

This script runs the Sequence Unit Test. The sequence is executed in the SeqTestFrame under supervision of libSeqUT and subject to several demanding protocols. A test certificate will be generated. Passing this test is mandatory prior to using this sequence for patient examinations. For a full description, refer to page 68.

Simulator

The sequence simulator consists of two programs:

1. *MriSeqSimDSPClient* executes the sequence .dll file and creates an intermediate graphic information file (.dsv) for each event (e.g., Y gradient). Derived quantities (such as duty cycles or patient stimulation) are calculated and also dumped in *.dsv files. Within IDEA the shortcut "sim" is available to start the simulation of the current sequence.
2. *MriSeqSimViewer* is a visualization tool which displays the .dsv files and offers various evaluation functions. Note that the Viewer is started automatically after the simulation, when "sim" is used.

Pulsetool

This graphical tool is used to display or modify both external RF pulse shape and gradient shape libraries.

Poet

The protocol offline editing tool offers the possibility to edit a protocol or to test the solve handlers. The UI and the libraries are the same as on the scanner, but a GSP (Graphical Slice Positioning) is not available. For a full description, refer to page 76.

Mephisto

This tool can create protocols in batch-driven mode. Instead of using the graphical UI and clicking around with the mouse, you can write your desired actions into a file which will be interpreted by this program. For a full description, refer to page 81.

**Crdefprot /
Mrdefprot**

This tool creates a default protocol for the specified sequence .dll file and saves it as a .pro file. For a full description, refer to page 80.

File Extensions used in IDEA

Working with IDEA, you will encounter files of the following file types:

.cpp / .h

These are C++ source and header files, respectively, that are typically used in a sequence. They are ASCII files and may be viewed with any text editor. The sequence source codes usually reside in the sequence working directory:

e.g.: z:\n4\pkg\MrServers\MrImaging\seq\seq\seq\seq_15b130\seq\seq_15000.cpp.

.obj

The sequence .cpp file will be compiled to a binary object file for the x86 platform. All created .obj files can be found in the directory \n4\x86\prod\lib.

.dll

One or (usually) more .obj files will be linked together to create a .dll file (Dynamically Linked Library). This sequence .dll is the executable form of a sequence. It can be tested in the SeqTestFrame or installed on the scanner. Building libraries other than sequence libraries is not supported outside the ClearCase archive environment. All created .dll files can be found in the directory \n4\x86\prod\bin directory.

.o

The sequence .cpp file can be compiled to a binary object file for the MARS with an x86 processor working on Linux by calling the delivered compiler. All .o files created in IDEA are in the sequence working directory.

.so

The .so file is the executable file that runs on the MARS. All created .so files can be found in the directory \n4\linux\prod\lib.

.vcproj

This is a Microsoft Visual Studio project file. It contains project settings such as linked files, debugging settings, compiler and link options.

.sln

This is a Microsoft Visual Studio Solution file. It can contain several projects and contains general settings.

.tok

This is a token file. It contains UI control commands in ASCII text format. It is read in by the mephisto program (page 81) and controls the generation of a protocol. Usually it is generated manually by the sequence developer.

.pro

This is a plain protocol file. It has ASCII text format. It looks like this:

```
### ASCCONV BEGIN ###
ulVersion          = 0xbee331
tSequenceFileName = "%SiemensSeq%\ep2d_diff"
tProtocolName     = "Initialized by sequence"
sProtConsistencyInfo.flNominalB0 = 1.494
sProtConsistencyInfo.flGMax      = 26
sProtConsistencyInfo.flRiseTime   = 5
sGRADSPEC.ucMode       = 0x1
sTXSPEC.asNucleusInfo[0].tNucleus = "1H"<...>
sAdjWatSupSpec.ulMode    = 0x1
```

```

alTR[0] = 500000
alTI[0] = 2500000
lContrasts = 1
alTE[0] = 200000
lCombinedEchoes = 1
sSliceArray.asSlice[0].sNormal.dTra = 1
sSliceArray.asSlice[0].dThickness = 3
sSliceArray.asSlice[0].dPhaseFOV = 283
sSliceArray.asSlice[0].dReadoutFOV = 283
### ASCCONV END ###

```

Within the ASCCONV-BEGIN-END-Block all protocol parameters not equal to zero will be dumped. The parameter names starting with "sProtConsistencyInfo" contain information about the Magnetom system for which the protocol has been created. For protocol parameter with enumerated values (such as Water Suppression), only the enumeration index is specified. There is no easy way to resolve these indices.

The .pro files are created by the program crdefprot or mephisto.

On a MAGNETOM host protocol files can also be created by Drag&Drop from the DotCockpit to the Windows Explorer for a selected program.



WARNING

Never change any protocol file manually. Most protocols will become inconsistent and cannot be loaded anymore. Use only appropriate tools such as poet or mephisto for any protocol modifications.

.exdb1

This is the new format (starting with VE11A) of the examination data base. There are now two data base files on a MAGNETOM system one containing the complete SIEMENS tree (Siemens.exdb1) and one containing all User trees (Customer.exdb1). The files are SQLite based databases with SQLite format.

.exar1

This is the new export format (starting with VE11A) of an examination data base export. It may contain one or more Programs (protocols), Exams and Regions. It has a SQLite format and consists of all required database information for the exported data. The file can be created with the DotCockpit.

.edb

This is the legacy format (pre VE11A) containing a protocol stored in the examination data base. It has ASCII text format and consists of a database information header plus the verbatim contents of a .pro file as described above. A .edb file is created by the syngo MR Exam Explorer. EDB files are used by the examination database to store protocols, but their format is also recognized by most IDEA tools such as poet, seqtestframe or the sequence simulator.



WARNING

Never change any edb file manually. Most protocols will become inconsistent and cannot be loaded anymore. Use only appropriate tools such as poet or mephisto for any protocol modifications.

.edx

This is an legacy (pre VE11A) examination database container file. It is a binary format and contains a complete subtree structure of the examination database. It is used for importing or exporting protocol sets. Usually these files are created and read in by the Exam Explorer.

.html

If such a HTML (Hypertext Mark-up Language) file resides in a sequence working directory, it usually contains the full output of the sequence unit test, i.e. the test results and the protocol used for this test. These files are created by libSeqUT or the SeqTestFrame (see page 68).

.txt

If such a text file resides in a sequence working directory, it usually contains the condensed output of a sequence unit test run over several different test protocols. These files are created by the SeqUT2 program (see page 68).

.bmp

In the prod directory, there may be bitmap files. These bitmaps are created by libSeqUT and visualize k space trajectories.

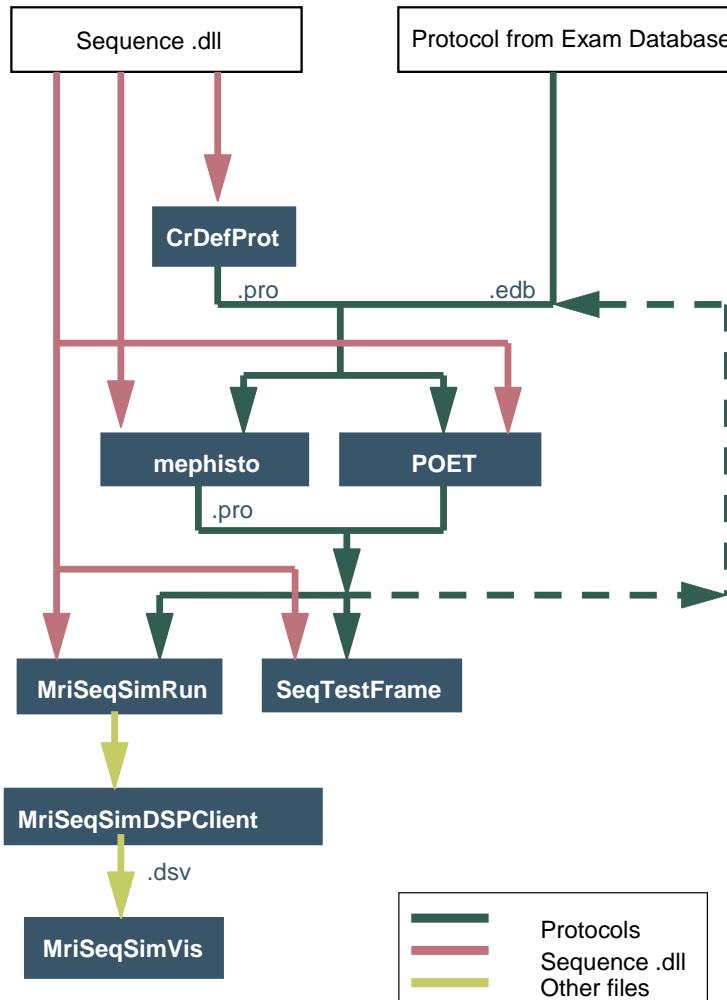
.dsv

Each of these files contain a diagram for the sequence visualization tool. They are in ASCII format albeit not intended for human reading. They are created by the *MriSeqSimDSPClient* tool. Details about the internal file format can be found on page 109.

makefile.trs

This is a platform independent makefile, which is used by the NUMARIS/4 build system in order to generate the platform specific (windows, linux) makefiles. It contains the sources to be built, its dependencies and the target .dll name.

Summary of Program Flow



Binary Files

Writing sequences or ICE programs means creating a dynamically linked library (.dll) which is linked into the measurement or image reconstruction process during run-time.

Unfortunately, there is not only one library for a sequence; things are a bit more complicated.

During the execution of a sequence, there are several hardware types involved, since a MAGNETOM system has two different computer systems: the Intel x86 host and the MARS (Measurement and Reconstruction) system. Therefore, several targets of a sequence need to be built from the same .cpp file.

Furthermore, there exist debug and release versions. The latter are usually installed on the scanner, the former are necessary to debug your code. A debug version contains full symbolic debugging

information, has a modified layout with guard bytes, modified function calls, is more safe against uninitialized pointers, and abandons any code optimizations. Therefore, debug versions execute substantially slower than release versions.

**WARNING**

When working with NUMARIS, never mix release and debug .dll versions. In particular, only use a release version of a sequence .dll on a MAGNETOM system.

The existence of these two versions applies not only for libraries, but also for executables.

How to find out if your system is working with release or debug versions? In IDEA you can use the command "runmode". Without any additional parameter it will show your current active mode in the IDEA shell. Alternatively you can check your environment for the variable MEDCOM_DEBUG. If it does not exist, you are working with release versions, if its value is 'd' you are working with debug versions (no other value is possible).

Here is a list of the possible target types:

Extension used for the library	Hardware the lib runs on	Explanation
.dll	HOST	The standard .dll that is used on the Main Console, i.e. Win7-x64 and the IDEA Development Environment.
d.dll	HOST	The .dll that is used for debugging and testing on Win7-x64 machines
.so	MARS	This ICE .so runs on the Image Reconstruction computer MARS

As a general rule, debug versions have always a 'd' as the last letter of their base file name.

D.2 Development Tools

Sequence Development Tools

This chapter describes additional tools that are useful for testing and debugging sequences. In particular, they provide the framework to perform the sequence unit test:

- SqTestFrame* (page 68)
- Imprint* (page 73)

SeqTestFrame

SYNOPSIS:

```
SeqTestFrame[d] -p <protocol_file> | -s <sequence_file> | -f <batch_file>.

[-w <working directory>] <DebugMask[hex]> [UTResultFile <filename>]
[dumpfile <dfile>] [-c <coil_context>] [ModifyCoilConfig] [Init] [Repair]
[PrepForBinarySearch] [PrepForScanTime] [PrepNoMeasurement] [Prep]
[Check] [Run] [Event] [GC] [RX] [TX] [FP] [RFCEL] [SeqBuffer]
[ProtConv] [Periphery] [-n <repetition>]
[-simlevel <n>] simulation level <n>:
    0: simulation off
    1: disable instruction processing
    2: disable libEQ event processing
    3: disable libRT event processing
```

DESCRIPTION:

A simulated run (i.e. the events will not be played out to the DSP hardware) of a sequence or a protocol can be executed using the 'SeqTestFrame' program. This program uses the same libraries as the real measurement system and has the same behavior concerning sequence calculation and parameter handling.

SeqTestFrame(d) is also invoked by SeqUT and SeqUT2 and generates the sequence unit test results in .html format.

PARAMETERS:
-p <protocol_file>

Specifies the name of the protocol to be executed. The full path and suffix must be specified. Both .pro and .edb files are recognized.

This option is very useful to check the consistency of a protocol.

-s <sequence_file>

Specifies the name of the sequence .dll file to be executed. The full path must be specified. If working with debug versions, the trailing 'd' of the sequence .dll file must be supplied. The suffix '.dll' must not be specified. The sequence will be loaded and its default protocol will be created for sequence execution.

-f <batch_file>

Specifies the name of a batch file, which contains one or several -p / -s commands. An example of such a batch file is:

```
-p c:\Protocol11.pro  
-p c:\Protocol12.pro  
-p c:\Protocol13.pro
```

This mode is useful to test the execution of several sequences / protocols taken from the same sequence cache.

OPTIONS: **[-w <working directory>]**

<DebugMask>

This hex number is passed to the SeqLim buffer attribute 'DebugMask' and controls the output of debug messages. Default is 0.

[UTResultFile <filename>]**[dumpfile <dfile>]****[-c CoilContext]**

From VA21A on, it is possible to work in a "virtual coil environment". This means, that you are not longer forced to actually plug in coils on a system to develop protocols etc. (see also page 76) The information, which coils are "virtually" plugged in, are stored in CoilContext files. You can specify with the -c option, which coil context to use; if it does not exist, it is created automatically.

Example: **seqtestframed -p flash.pro -c myTest** will create (and/or use) %MEASDAT%\MeasCoil_MyTest.dat.

[ModifyCoilConfig]

From VA21A on, the protocol is also checked, if a valid coil combination is plugged in. Otherwise, the preparation will return this error: Sequence parameter error: Invalid coil configuration (tx-element does not support the nucleus) Check coil configuration and nucleus. In that case, SeqTestFrame offers to correct the coil configuration. Simply specify the option ModifyCoilConfig, and the right coils are plugged in.

[Init]

Run only fSeqInit function of sequence.

[Repair]

Run fSeqInit and fSeqPrep functions of sequence; fSeqPrep is executed in 'PrepForUpdateProtocol' mode.

[PrepForBinarySearch]

Run fSeqInit and fSeqPrep functions of sequence; fSeqPrep is executed in 'PrepforBinarySearch' mode.

[PrepForScanTime]

Run fSeqInit and fSeqPrep functions of sequence; fSeqPrep is executed in 'PrepforScanTime' mode.

[PrepNoMeasurement]**[Prep]**

Run fSeqInit and fSeqPrep functions of sequence; fSeqPrep is executed in 'Prep for Measurement' mode (standard mode).

[Check]

Run fSeqInit, fSeqPrep, and fSeqCheck functions of sequence.

[Run]

Run sequence completely (default value).

[Event]

Display event dump (i.e. list event blocks sent to the DSPs). This option works only for user-compiled sequences.

[GC]

Dump gradient control information.

[RX]

Dump receiver DSP commands.

[TX]

Dump transmitter DSP commands.

[FP]**[RFCEL]****[SeqBuffer]**

After sequence execution, the contents of the internal buffers 'SeqExpo' and 'SysExpo' will be dumped to stdout.

[ProtConv]**[Periphery]****[-n <repetition>]**

[-simlevel <n>]

Change the simulation level to one of the available levels:

- 0: simulation off
- 1: disable instruction processing
- 2: disable libEQ event processing
- 3: disable libRT event processing

EXAMPLE:

```
[VE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\ep2d_diff> SeqTestFrame.exe -s
%SiemensSeq%\ep2d_diff
Fri Nov 04 17:20:22.377 fSequenceTest: Loading sequence
'\n4\x86\delivery\bin\ep2d_diff'...
2011/11/04-17:20:22.334087 3012 SeqTestFramed.exe 5308 Thread-5308 -1 Info 0
MrPMUSrv:PMUserIF PMUserIF.cpp:120 PMUserIF::PMUserIF PMUserIF::PMUserIF
Interface Testmode is ON
Fri Nov 04 17:20:22.746 Adding bool parameter SetExternalTrigger | 0 |
External_Control | SetExternalTrigger Adding bool parameterSetExternalTrigger | 0 |
External_Control | SetExternalTrigger
Fri Nov 04 17:20:22.758 Try to open file: d:\tmp\ep2d_diffusion.ini
Fri Nov 04 17:20:22.762 The file d:\tmp\ep2d_diffusion.ini does not exist !!!
Fri Nov 04 17:20:22.766 *----- Reading parameters -----
Fri Nov 04 17:20:22.770 *-----
Fri Nov 04 17:20:22.772 * reading bool parameters *
Fri Nov 04 17:20:22.775 *-----
Fri Nov 04 17:20:22.779 IniParameter::readValue: :: INIAccess::fGetValueP: no data
loaded [External_Control] SetExternalTrigger
Fri Nov 04 17:20:22.786 * -> no int parameters to read <- *
Fri Nov 04 17:20:22.788 * -> no double parameters to read <- *
Fri Nov 04 17:20:22.791 * -> no string parameters to read <- *
Fri Nov 04 17:20:22.795 *----- END Reading parameters -----
Fri Nov 04 17:20:22.798 ****
Fri Nov 04 17:20:22.803 * parameter set read from ini file *
Fri Nov 04 17:20:22.806 ****
Fri Nov 04 17:20:22.809 *-----
Fri Nov 04 17:20:22.812 * bool parameters *
Fri Nov 04 17:20:22.814 *-----
Fri Nov 04 17:20:22.817 ||| from ini file : 0 ||| value : 0.000 | category :
External_Control | Name : SetExternalTrigger || int. name is SetExternalTrigger
Fri Nov 04 17:20:22.824 * -> no int parameters <- *
Fri Nov 04 17:20:22.827 * -> no double parameters <- *
Fri Nov 04 17:20:22.829 * -> no string parameters <- *
Fri Nov 04 17:20:22.831 ****
Fri Nov 04 17:20:22.834 * end of Paramter set *
Fri Nov 04 17:20:22.836 ****
```

```
Fri Nov 04 17:20:22.839 fSequenceTest: Switching debugging mode for sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Dumping of Siemens sequences is disabled
Fri Nov 04 17:20:22.849 fSequenceTest: Initializing sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Fri Nov 04 17:20:23.039 fSequenceTest: Creating a protocol for sequence
'\n4\x86\delivery\bin\ep2d_diff'...
2011-11-04 17:20:24.285 AdjServerIFImpl::AdjServerIFImpl(): Adjustment server
TCP/IP configuration is 192.168.2.1:3101,3102,3103
2011-11-04 17:20:24.334 AdjDicoDataAdapter::readFromSpecAndSystem(): Get conversion
factors for nucleus 1H, channel number 0 !
2011-11-04 17:20:24.364 AdjDicoDataAdapter::readFromSpecAndSystem(): Get conversion
factors for nucleus 1H, channel number 0 !
2011-11-04 17:20:24.380 AdjCoilSelect::get(): No extension of coil select needed
for SystemBC or LcTx
Fri Nov 04 17:20:24.438 fSequenceTest: calculatePTX sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Fri Nov 04 17:20:24.461 fSequenceTest: Preparing sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Fri Nov 04 17:20:24.506 fSequenceTest: Checking sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Fri Nov 04 17:20:24.517 DiffusionDirections::dump: 1 vectors specified in PRS
coordinates
Fri Nov 04 17:20:24.521 DiffusionDirections::dump: Vector[0] = ( 0.000, 1.000,
0.000 )      Norm = 1.000
Fri Nov 04 17:20:24.548 fSequenceTest: Running sequence
'\n4\x86\delivery\bin\ep2d_diff'...
Fri Nov 04 17:20:24.558 Ep2d::run: running
\n4\x86\delivery\bin\ep2d_diff
Fri Nov 04 17:20:24.606 Ep2d::run: finished
\n4\x86\delivery\bin\ep2d_diff
=
```

Imprint

SYNOPSIS: imprint [-h] [-v] [-c] [<SeqPath>]

DESCRIPTION: This program displays selected parameters of the MAGNETOM system and the SeqLimits buffer (if a sequence .dll file is specified).

In detail, these parameters are taken from the system and adjustment parameters, RFPA proxy, RFCI proxy, RX4 proxy, GPA proxy, GC proxy as well as from the fSeqInit part of the sequence.

For GradMaxAmpl, GradMinRiseTime and GradMaxSlewRate, three or four values are listed; they refer to the ultrafast (listed if available), fast, normal and whisper gradient mode.

PARAMETERS: [<SeqPath>]

Is the (full) path of the sequence binary. The suffix '.dll' and the 'd' in case of a debug version must not be specified. This parameter is optional. If it is not specified, only the system properties will be displayed.

OPTIONS: -h display this manual page.

-v verbose mode (for debugging purposes).

-c Use own coil context 'imprint' and use the standard coil configuration therein

EXAMPLE: [VE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\se_15b130> **imprintd**
%SiemensSeq%\se_15b130

System Frequency 1H [MHz] = 63.6

Larmor Constant 1H [MHz/T] = 42.5756

GSDW stimulation monitor = on

MSU NominalB0 [T] = 1.494

RFPA[0] MaxPower for BC path 1H [W] = 20000

RFPA[0] MaxPower for BC path 1H [W] = 20000

RFPA[0] ChargeBalanceDutyCycle for BC path 1H [W] = 0.138

RFPA[0] MaxPower for LC path 1H [W] = 7200

RFPA[0] ChargeBalanceDutyCycle for LC path 1H [W] = 0.21

RFCI CoilCtrlLead time [us] = 100

RFCI CoilCtrlHold time [us] = 8

GPA Type = K2309_2000V_651A

GPA GradMaxAmplAbsolute [mT/m] = 33

GPA GradMaxAmplNominal [mT/m] = 17

```

GPA GradMaxAmp1           [mT/m] = 17 / 17 / 17
GPA GradMinRiseTimeAbsolute [us/ (mT/m) ] = 8
GPA GradMinRiseTime        [us/ (mT/m) ] = 10 / 20 / 40
GPA GradClipRiseTime       [us/ (mT/m) ] = 1
GPA GradMaxSlewRateAbsolute [mT/ (m*ms) ] = 125
GPA GradMaxSlewRate         [mT/ (m*ms) ] = 100 / 50 / 25
GPA GradClipSlewRate        [mT/ (m*ms) ] = 1000

```

This is a Aera-XJ-Gradient gradient system

```

GC Type                  = GC98Q
GC FoVMax                [mm] = 500
GC AcousticResonanceFrequency[0] [Hz] = 710
GC AcousticResonanceBandwidth[0] [Hz] = 100
GC AcousticResonanceFrequency[1] [Hz] = 1200
GC AcousticResonanceBandwidth[1] [Hz] = 220
GC AcousticResonanceFrequency[2] [Hz] = 0
GC AcousticResonanceBandwidth[2] [Hz] = 0
GC AcousticResonanceFrequency[3] [Hz] = 0
GC AcousticResonanceBandwidth[3] [Hz] = 0
GC AcousticResonanceFrequency[4] [Hz] = 0
GC AcousticResonanceBandwidth[4] [Hz] = 0

```

```

2011/11/04-17:40:16.528378 8852 imprintd.exe 9500 Thread-9500 -1 Info 0
MrPMUSrv:PMUSerIF PMUSerIF.cpp:120 PMUServerIF::PMUServerIF PMUSerIF::PMUServerIF
Interface Testmode is ON
LinkedSeqFilename          = \n4\x86\delivery\bin\se_15b130
SequenceOwner               = SIEMENS
SequenceCompileDate         = Oct 29 2011 23:53:05
SDEVersion                 = N4V1.0
MyOrigFilename              =
\n4\pkg\MrServers\MrImaging\seq\se_15b130\se_b130_15000.cpp
MinAllowedFrequency         [Hz] = 8000000
MaxAllowedFrequency         [Hz] = 500000000
RequiredGradAmp1            [mT/m] = 10
RequiredSlewRate             [mT/ (m*ms) ] = 10
NotSupportedSystemTypes      = <NONE>
Dimension_2D                 = 1
Dimension_3D                 = 0
TI_available                  = 1
EPIFactor_available           = 0
TurboFactor_available          = 0
DiffusionMode_available        = 0
MDS_available                  = 0
SequenceHintText              =
Application: All-purpose sequence
for T1-weighted imaging (TR < T1).

```

```
Basics: 2D spin echo; single echo;  
multi-slice; symmetric echo sampling;  
  
no gradient moment nulling.
```

Build: Oct 29 2011 20:13:28

```
2011-11-04 17:40:19.307 AdjServerIFImpl::AdjServerIFImpl(): Adjustment server  
TCP/IP configuration is 192.168.2.1:3101,3102,3103  
2011-11-04 17:40:19.357 AdjDicoDataAdapter::readFromSpecAndSystem(): Get conversion  
factors for nucleus 1H, channel number 0 !  
2011-11-04 17:40:19.391 AdjDicoDataAdapter::readFromSpecAndSystem(): Get conversion  
factors for nucleus 1H, channel number 0 !  
Fri Nov 04 17:40:19.841 Shutting down 1
```

ENVIRONMENT: MEASDAT

must contain the path to a valid MAGNETOM configuration.

EXIT STATUS: 0 Either no errors detected or all errors were corrected.

1 Sequence loading failed.

9 Program aborted during evaluation of command line arguments.

Protocol development tools

This part of the documentation contains the reference descriptions of the tools provided for protocol manipulations:

- poet* (page 76)
- crdefprot* (page 80)
- mephisto* (page 81)

POET

SYNOPSIS: poet [-help] [/online] [/d] <protocol | sequence>

AVAILABILITY: POET is available with the NUMARIS/4 IDEA software for a standalone installation (but not on the MAGNETOM system itself).

DESCRIPTION: POET (Protocol offline editing tool) offers a GUI tool to edit protocols offline without a scanner or exam database. It also allows testing and simulation of the underlying sequence on a stand-alone PC. Using poet, protocol parameters can be modified and the behavior of the sequence and the graphical user interface (GUI) can be investigated, especially concerning solve handlers as well as hard and soft parameter limits. The only limitation is that graphical slice positioning support is not available.

Protocol modification is achieved by means of parameter cards and their underlying UI libraries. These parameter cards and their behavior correspond exactly to the Exam Task Card of the scanner.

For testing the sequence with the actual protocol, POET offers dialogs to call either SeqTestFrame or the sequence simulator directly from the GUI.

PARAMETERS: <protocol | sequence>

specifies the name of the protocol to be executed. The full path and the suffix must be specified. Both .pro and .edb files are recognized.

Another possibility is to specify a sequence .dll (including the suffix [d].dll !). In this case, the default protocol will be used.

OPTIONS: /online

Run POET with /online option.

/d

Run POET without opening a protocol.

NOTE:

POET requires a running protocol access service component. In case no protocol access service is running on the current machine, POET automatically starts an instance in the background. The output of this instance is redirected to an edit field within the tabcards on the bottom of window. When shutting down POET, this instance will be terminated as well.

POET tries also to start a SpuSer Service component. In case you already have a running instance, POET will not do additional tasks. If there is no running occurrence and POET is unable to start the SpuSer Service it will give in error message. Anyway, POET will get started in both cases.

ENVIRONMENT: MEASDAT

must contain the path to a valid MAGNETOM configuration. "Measurement" configuration is not supported by POET. Switch to a valid configuration with IDEA sys command.

MEASCONST

The directory that contains files with installation information.

PATH

The search path for binary programs. If a path is included without a drive specification, it is searched on the current drive.

SCONFPATH

The search path for the config files of PoetProtAccessService.conf.

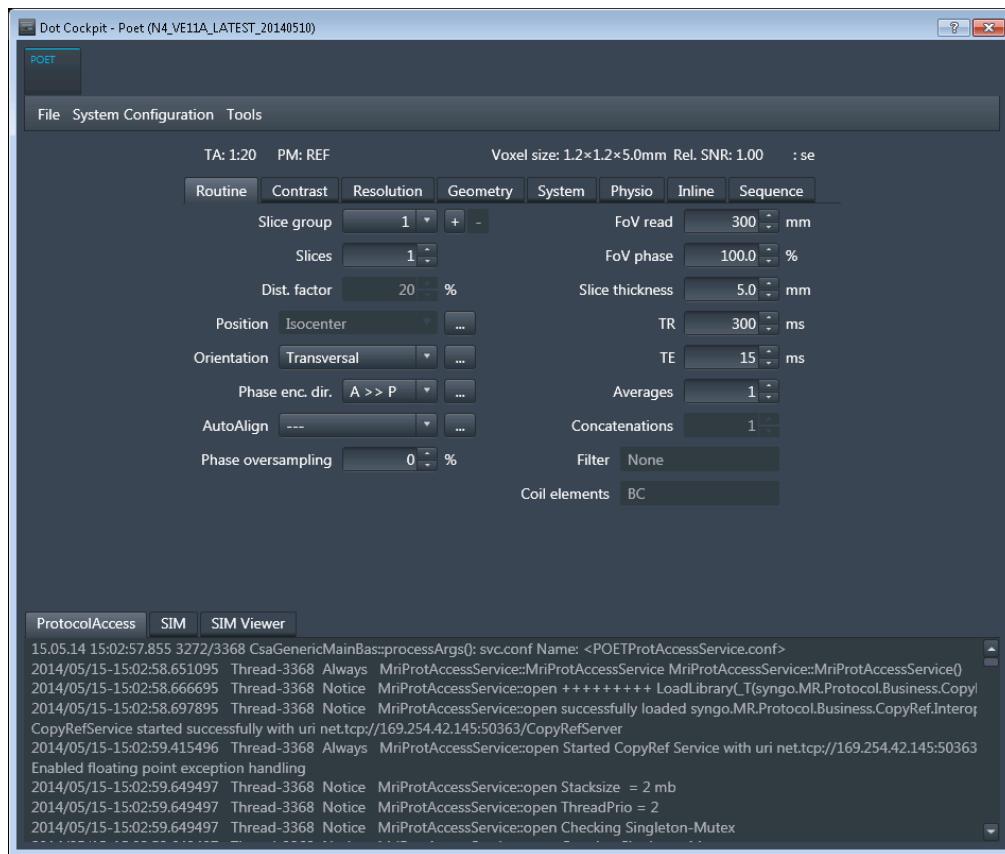
REQUIREMENTS:

The executable MrVirtCoils(d).exe has to be available in the environment.

EXAMPLE:

```
[VE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\{a_se_15b130> poetd test.pro
```

The following window will pop up, if test.pro contains a valid protocol:

**USAGE:****FILE menu:**

OPEN: Enables you to open existing protocol files or to create a default protocol for a sequence .dll file. The program offers the usual windows "File Open" dialog. Valid files are Protocol files (*.pro, *.edb) and sequence .dll files (*.dll). Instead of using the "File Open" dialog, files may be opened by using drag-and-drop from the Windows Explorer to the POET dialog.

SAVE: The current protocol is saved to a file. If POET has created a default protocol for a sequence and this protocol has not been saved yet, the "Save As" Dialog appears to allow you to specify the file

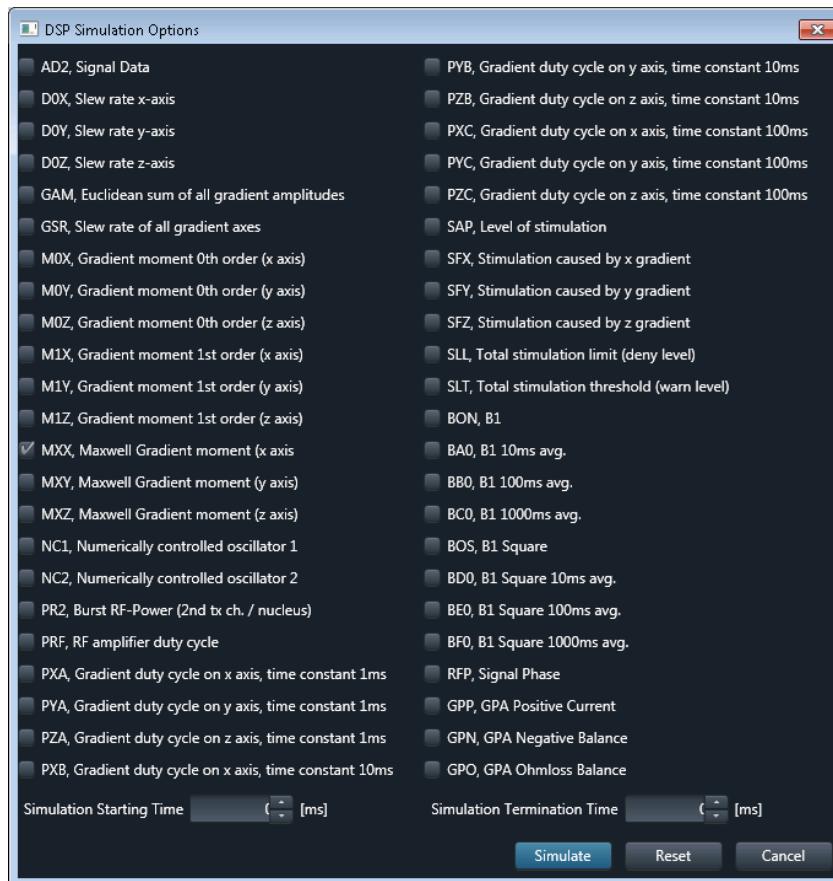
name. As the default file name, the name of the sequence with the extension *.pro is suggested.

SAVE AS: the "Save As" Dialog opens.

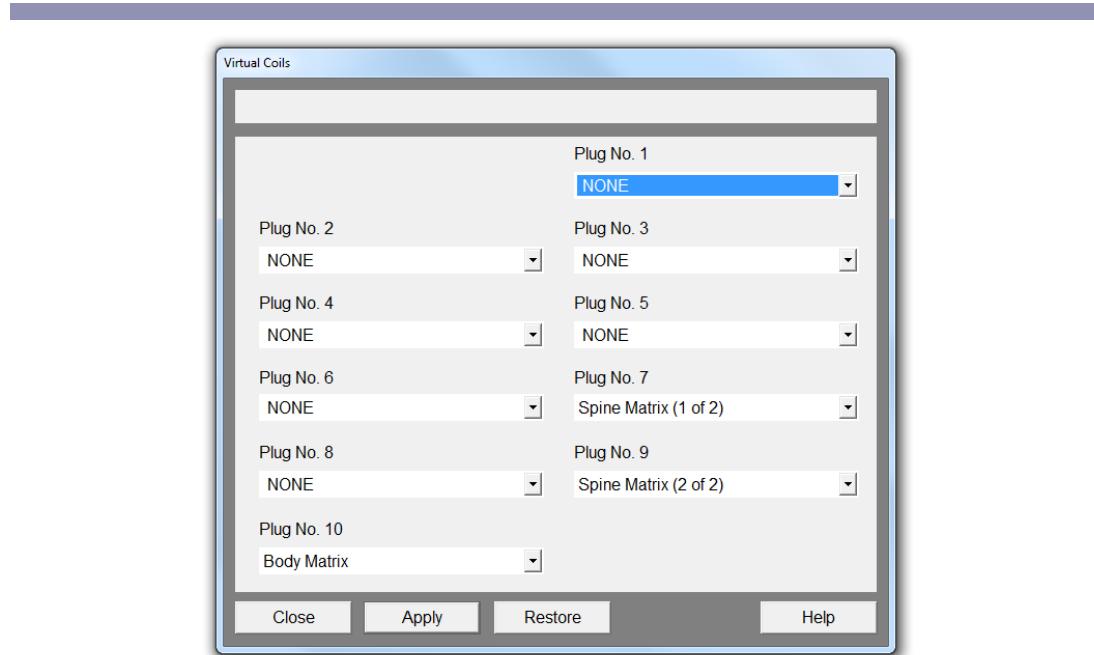
EXIT: Terminates the program. This item is disabled, when a tool is currently being executed. If the protocol was not saved yet, the program asks whether to save the changes now. In case POET has automatically started the protocol access service component, this will be terminated too.

TOOLS menu:

SIM: This item is disabled, when a tool is currently being executed. A dialog displaying the simulator options is presented. Clicking execute will launch the following routines sequentially: MriSeqSimDSPClient and MriSeqSimViewer. Here the output of the predecessor routine is used as input of the successive routine. As a last step, the visualize tool MriSeqSimViewer is just started by POET; it is left to the user to close it. The output is redirected to a file as well as displayed in an edit field within the tabcards on the bottom of the POET dialog. A message window allows the user to cancel the process during execution.



Set Virtual Coils: The virtual coils dialog enables the user to set the coils used on the system card. It presents you a list of coils for each existing plug on the machine. In case a wrong selection was made, the dialog returns an error message.



When entering the virtual coils dialog window, you might be prompted to register the ItfOCXd.ocx. This interface ocx is only required if you work with several processes accessing the SpuSer at the same time.

Please note that the 'Virtual Coils' menu simulates only plugging in the coil connector. In a next step you must select the coil on the 'System' tab.

Output windows:

PoetProtAccess: Every output generated by the protocol access service component and the sequence while working with POET is redirected into this output window. The window remains empty in case POET did not automatically start the protocol access service component, when already another instance of a protocol access service component was running. The output is also written into a log file in either %TRACE_LOG_PATH_WINDOWS% or %MEDHOME%\log with the name POET-<gradient id>.log.

SIM: Displays the log file of SIM frame to a certain size. The output is written by default to the file ***_ERR.dsv.

SIM Viewer: Displays the log file of SIM Viewer to a certain size.

crdefprot

SYNOPSIS: crdefprot <seqname> <protocol>

DESCRIPTION: CRDEFPROT creates the default protocol of a sequence and writes it to the disk in the current directory. It is saved as a .pro file (i.e. a plain ASCCONV-BEGIN-END format).

PARAMETERS: <seqname> is the path of the sequence .dll file (the suffix must not be specified).

<protocol> is the output filename of the created protocol.

EXAMPLE:
[VE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\se_15b130>
crdefprot ^%SiemensSeq%\se_15b130 default.pro
Created protocol file: default.pro

NOTE: Please note the following detail. If the created protocol is to be used not only for simulation purposes, but also on the MAGNETOM host for scanning: To specify a sequence .dll file in the protocol, do not specify an explicit path, but use either the environment variable %CustomerSeq% or %SiemensSeq%. To ensure that the '%' delimiters are actually written into the protocol, they must be further delimited by a '^' sign as shown in the example above.

Explanation: There are different directory structures on the host and the MARS. To use the same protocol on both architectures, these two environment variables are set to different paths on both machines.

mephisto

SYNOPSIS: mephisto [-h] | [-lv] [-o <outfile>] [-a <analysisfile>] <seqname> <tokenfile>.

DESCRIPTION: MEPHISTO generates or modifies protocols and writes the modified protocol to an output file. The protocol modification is not performed interactively by the user, but the protocol parameters to be modified are taken from a so-called token file. This file contains the same actions the user would perform using the UI. MEPHISTO uses exactly the same libraries as are used by the UI. Therefore, the requested changes are automatically checked for consistency and the modifications conform to the limits of the specified sequence. The green and red limit areas as well as the actual parameter settings will be written to 'stdout' (cf. option -l).

The performance of most sequence methods (such as initialize(), prepare(), etc.) can be measured while mephisto is running. To activate this measurement, a target file for the analysis output must be specified (use option -a <analysisfile>). The output is XML-compatible.

Before the performance measure is started, mephisto modifies the markers in category RUTA of current UTrace configuration by adding the TIMESTAMP_SEQ marker. This marker is automatically removed after performance measuring is done.

The modified protocol is written to a separate file.

PARAMETERS: <seqname>

Is the path of the sequence .dll file (the suffix must not be specified) or the protocol (the suffix must be specified). The distinction is made automatically. If a sequence .dll file is specified, mephisto will generate a default protocol as a starting point for the modifications.

<tokenfile>

Is the ASCII input token file. The format is the same as the NUMARIS protocol format. Additionally, comment lines starting with a '#' are allowed. It is also possible to include another batch file (for example, to avoid recursive looping). Parameters may be specified using small or capital letters. For specifying all parameters of a timing array, '[all]' can be used as an index (this is not allowed for slice groups). The maximum/minimum value of a parameter can be obtained by specifying 'max'/min' as its value, respectively. For slice positions, a vector x/y/z may be specified as well as a single number for a vector x/x/x. For slice orientations, cor, sag, or tra must be specified. If a numerical value is out of range, the parameter will be set to the nearest possible value. The timing arrays (TE, TR, etc.) are specified in milliseconds. An example of this is shown below.

OPTIONS: -h Display this manual page.

-l Display limits (i.e. red and green areas) of parameters.

-o <filename> the output filename of the modified protocol. The default is the input file name with a suffix '.tok' (and the old suffix '.pro' being deleted if applicable).

-a <analysisfile> is the name for the analysis file. If no file is specified, the performance will not be measured.

-v verbose mode (for debugging purposes). Increments debug level.

USAGE: The following keywords are supported. Please note that these keywords are usually substrings of the full expression, mapped to lower case.

The recognized keys can be looked up by entering "mephisto(d) -h".

By default, the values that are specified will be set in the red areas by calling a solve handler, if applicable. If the line terminates with a question mark (?), the soft limits will be respected. This works only for long, double, and simple variable selections. It is also possible to take the parameter names directly from \n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\MrStdNameTags.h.

The WIP parameters can be accessed by the tokens

seq_wip1, seq_wip2, ..., seq_wip14, etc.

The reserved special parameters can be accessed by the tokens

seq_res1, seq_res2, ..., seq_res8, etc.

Subelements of parameter arrays can be accessed by

param_array.0.sub_param, param_array.1.sub_param, ..., etc.

The values of the parameter type 'selection' are exactly those displayed in the pull-down menus displayed in the UI. There is no translation by this program, but they are selected in a generic way.

Note that mephisto is set to run with US English language settings; i.e. the language of the selection labels is fixed to US English.

EXAMPLE: Create a file called meptest.tok in the current sequence directory by entering "notepad meptest.tok" at the command line. Enter the following lines:

```
TR [A11] = Max
TE [A11] = Max
PhaseResolution = 75
PhasePartialFourier = 6/8
SliceThickness = Min
SliceOrientation [0] = 1 / 6 / -1
TE [A11] = Min ?
TR [A11] = Min ?
```

The question mark means that the parameter will be set into a green region, i.e. without changing other parameters by solve handlers.

Save and close the editor.

Start mephisto.

```
[VE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\se_15b130> mephistod -v
^%SiemensSeq%^%\se_15b130 meptest.tok
mephisto: Running in debug mode (level 1).
mephisto OutputFile: meptest.pro
=====
mephisto
=====
104 tokens in internal database
mephisto: Loading sequence file %SiemensSeq%\se_15b130
2010/04/13-16:16:19.347360 4680 mephistod.exe 3908 Thread-3908 -1 Info 0
MrPMUSrv:PMUserIF PMUserIF.cpp:120 - PMUserIF::PMUserServerIF Interface Testmode is ON
MrParcThreadPrioImpl singleton created without singleton lock
InterpreteProtocolFile (File=<meptest.tok>) <<<
ModifyProtocol -----
ModifyProtocol command = <tr [all] = max>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: ArrayDouble Param <tr> has 1 Elements.

Here you find information about the type of the protocol parameter.

ModifyProtocol: value before change: tr = <300>
ModifyProtocol: Limits of parameter <tr> [0,0] = {x|x=28+j*1;x<=999}<g> *
ModifyProtocol: Limits of parameter <tr> [0,1] = {x|x=1000+j*10;x<=5000}<g>
Here you find the hard and soft limit of the protocol parameter. The first line can be read as all TRs are allowed between 28 and 999 with an increment of 1. Green parameter areas are tagged by a <g>, red areas by a <r>. In this example, although there appears only one green bar in the UI, there are two different, but adjacent areas with different increments.

ModifyProtocol: value after change: tr = <5000> (request was pt<max> = d<5000>)
ModifyProtocol -----
ModifyProtocol command = <te [all] = max>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: ArrayDouble Param <te> has 1 Elements.

ModifyProtocol: value before change: te = <15>
ModifyProtocol: Limits of parameter <te> [0] = {x|x=15+j*1;x<=100}<g>
ModifyProtocol: value after change: te = <100> (request was pt<max> = d<100>)
ModifyProtocol -----
ModifyProtocol command = <phaseresolution = 75>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: LimitedDouble value before change: ph_res = <100>
ModifyProtocol: Limits of parameter <ph_res> [0] = {x|x=12}<g>
ModifyProtocol: Limits of parameter <ph_res> [1] = {x|x=13}<g>
ModifyProtocol: Limits of parameter <ph_res> [2] = {x|x=14}<g>
...
ModifyProtocol: Limits of parameter <ph_res> [86] = {x|x=98}<g>
ModifyProtocol: Limits of parameter <ph_res> [87] = {x|x=99}<g>
ModifyProtocol: value after change: ph_res = <75> (request was <75>)
ModifyProtocol -----
```

```

ModifyProtocol command = <phasepartialfourier =           6/8>
ModifyProtocol: Trying to override soft limits..
ModifyProtocol: Selection 'ppff' has 5 option(s):
ModifyProtocol: | 4/8 | 5/8 | 6/8 | 7/8 | (*) Off |
If unsure which options are allowed for a selection, use this list. The selection currently active is
marked by an asterisk (*).

ModifyProtocol: Selection value after change: | 6/8 |
ModifyProtocol -----
ModifyProtocol command = <slicethickness =           min>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: LimitedDouble value before change: sl_thick = <5>
ModifyProtocol: Limits of parameter <sl_thick> [0] = {x|x=2+j*0.5;x<=10}<g>
ModifyProtocol: value after change: sl_thick = <2> (request was <2>)
ModifyProtocol -----
ModifyProtocol command = <sliceorientation [0] =      1 / 6 / -1>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: PrmtrVectorPat value before change: sg.0.norm = <(0, 0, 1)>
ModifyProtocol: value after change: sg.0.norm = <(0.162221, 0.973329, -0.162221)>
(request was <1 ,...>)
ModifyProtocol -----
ModifyProtocol command = <te [all] = min ?>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: ArrayDouble Param <te> has 1 Elements.
ModifyProtocol: value before change: te = <100>
ModifyProtocol: Limits of parameter <te> [0,0] = {x|x=15+j*1;x<=100}<g>
ModifyProtocol: value after change: te = <15> (request was pt<min> = d<15>)
ModifyProtocol -----
ModifyProtocol command = <tr [all] = min ?>
ModifyProtocol: Trying to override soft limits...
ModifyProtocol: ArrayDouble Param <tr> has 1 Elements.
ModifyProtocol: value before change: tr = <5000>
ModifyProtocol: Limits of parameter <tr> [0,0] = {x|x=28+j*1;x<=999}<g>
ModifyProtocol: Limits of parameter <tr> [0,1] = {x|x=1000+j*10;x<=5000}<g>
ModifyProtocol: value after change: tr = <28> (request was pt<min> = d<28>)
ModifyProtocol -----
ModifyProtocol command = <ScanTime = ?>

This scantime parameter is read-only. It is called automatically by mephisto to update the required
scan time in the protocol.

ModifyProtocol: Trying to override soft limits...
ModifyProtocol Read-only PrmtrString value: <7.5 [s]>
SaveProtocolToFile: Writing protocol file meptest.pro
mephisto successfully finished

```

A new protocol file named meptest.pro is created.

NOTE:

Please note the following detail if the created protocol is to be used not only for simulation purposes, but also on the MAGNETOM host for scanning: To specify a sequence .dll file in the protocol, do not specify an explicit path, but use either the environment variable %CustomerSeq% or %SiemensSeq%. To ensure that the '%' delimiters are actually written into the protocol, they must be further delimited by a '^' sign as shown in the example above.

Explanation: There are different directory structures on the host and the MARS. To use the same protocol on both architectures, these two environment variables are set to different paths on both machines.

EXIT STATUS:

- 0 Either no errors detected or all errors were corrected.
- 1 Protocol access failed.
- 3 Token file not found.
- 9 Program aborted during evaluation of command line arguments.

REQUIREMENTS: The syngo and NUMARIS .dll files are required.**DIAGNOSTICS:**

"Parameter <...> exists but is not available."

In general, this parameter is a valid parameter, but for the current sequence, this parameter is not defined.

"ERROR: Parameter <...> has unexpected type."

This is a valid protocol parameter, but this mephisto version cannot handle it. Sorry.

D.3 DSP Simulation

Sequence Simulator

The **DSP simulation** program can be started via the command `MriSeqSimSimulatorClient` from the IDEA shell. It generates the signal plots from the digital signal processor (DSP) data and stores the diagram and protocol information in buffer files. During start-up, the specific sequence data to be calculated can be selected; among others the following options are available:

SEQUENCE DATA

Protocol file (summary)

This text file contains the simulation protocol (sequence being simulated and summary of measurement parameters) and can be displayed by the visualizer. Additional numerical values that accrue during data calculation (e.g. maximum hardware requirements of the sequence) can be added to this text file. Using this text file, a quick overview of the simulated sequence can be obtained.

Physical gradient data

Nominal output of the gradient DSPs for x, y, and z axis in [mT/m].

Please note that the rotation matrix is only applied if the simulator is not running in the logical coordinate system (i.e. the /LOG option has not been specified).

Magnitude of RF pulse

The signal is scaled with transmitter voltage (from TX-DSP).

ADC on/off

Has a value of 1 if the signal is received, and a value of 0 otherwise (from RX-DSP).

NCO time evolution of phase

The phase of the numeric controlled oscillator (NCO) can be adjusted and a frequency offset can be set through the NCO events. The phase of the NCO develops as a function of time until the frequency offset is reset (from TX-DSP and RX-DSP).

ADC measurement data header

There is a measurement data header associated with each line of raw data that is measured.

This contains primarily the loop indices of the SequenceRun function (text output).

SYNC events

This has to do with synchronization events (e.g., OSC0, OSC1, OSC2, HALT, WAKEUP).

Event block data

The current event block may be displayed as text by means of the visualizer program.

Zero-order gradient moment

Development over time of the zero-order gradient moment (time integral over G) for all gradient axes.

$$m_0(t) = \frac{1}{\gamma} k(t) = \int_{t_0}^t G(t') dt'$$

First-order gradient moment

Development over time of the first-order gradient moment (time integral over $t \times G$) for all gradient axes.

$$m_1(t) = \int_{t_0}^t (t' - t_0) G(t') dt'$$

Maxwell integral calculation

Actual Maxwell values for each gradient axis according to the following formula:

$$M_x(t) = \int_{t_0}^t G_x^2(t') dt'$$

Remarks with respect to the moment integrals:

1. t_0 : Integration starts at the center of the excitation pulse; the calculation is performed for all excitation pulses of the sequence.
2. Integration ends at the beginning of the next excitation pulse, i.e., with each excitation pulse, the moments are set to zero.
3. The current sign of the moment is inverted for a refocusing pulse, except for the Maxwell integral.

It should be noted that in addition to the output of the gradients, DSP information about the type (excitation pulse or refocusing pulse) and symmetry of the RF pulses between integration start and integration end is required. RF pulses that are neither excitation nor refocusing pulses are ignored.

Euclidean sum for gradient amplitude

Time average of the Euclidean sum over all three axes for gradient amplitude:

$$\mathbf{G} = \sqrt{G_x^2 + G_y^2 + G_z^2}$$

Slew rate

Derivative of the gradient signal with respect to time (dG/dt) for each gradient axis.

Euclidean sum for gradient slew rate

Time average of the Euclidean sum over all three axes for gradient slew rate:

$$\frac{d\mathbf{G}}{dt} = \sqrt{\left(\frac{dG_x}{dt}\right)^2 + \left(\frac{dG_y}{dt}\right)^2 + \left(\frac{dG_z}{dt}\right)^2}$$

Stimulation calculation

An algorithm is used to compute a time-dependent quantity from the output of the gradient DSPs to estimate whether or not the patient should experience nerve stimulation. If this quantity exceeds the stimulation threshold value (system user receives a pop-up warning window stating that stimulation will occur), the simulator generates a 1 in the graphic display; if the stimulation limit value is exceeded (no measurement is permitted when this value is reached), the simulator generates a 2. If neither of the two values is generated, the result is 0. In this manner, the critical and hypercritical areas for the sequence with respect to magnetic stimulation can be identified.

Gradient Duty Cycle

The duty cycle of the gradient power amplifier on one axis is specified in percent (%) and defined (!) as follows:

$$DC(t) = \frac{\int_{t_1}^{t_2} G(t)^2 dt}{G_{\text{nom}}^2 (t_2 - t_1)} \times 100$$

G_{nom} denotes the nominal maximum gradient amplitude.

Obviously the duty cycle depends on the integration time (t_2-t_1). Therefore the simulator calculates nine different duty cycles, combining the three gradient axes with three different integration times. The resulting output files are named "P<direction><time>.dsv", where "<direction>" is the gradient axis (X, Y or Z) and <time> is the integration time:

A	1 ms
B	10 ms
C	100 ms

Please note that the duty cycle on one axis may exceed 100%, as gradient amplitudes as high as G_{abs} (absolute maximum gradient amplitude) can be applied. Furthermore these duty cycle calculations cannot be used as a precise criterion for a GPA switch-off, but give you merely a feeling of what's going on in your sequence.

RF Duty Cycle

The time-dependent duty cycle of the RF amplifier is calculated from the outputs of the TX-DSP. The magnitude time integral characteristic for the RF pulse enters into this calculation.

Real value of RF pulse

Signal is scaled with transmitter voltage (from TX-DSP).

-
- Imaginary value of RF pulse*
Signal is scaled with transmitter voltage (from TX-DSP).
 - Phase of RF pulse (from TX-DSP).*

FILE IDENTIFIERS

The diagram files to be generated during the simulation run must be specified as an option at the start-up of the simulator program. The table below lists all diagrams that can be generated along with the corresponding ID codes (file identifiers).

File identifier	Contents	Creation
ADC	ADC (0=off, 1=on)	always
AD2	ADC Signal data 2 nd nucleus	optional
DOX	Slew rate of X axis	optional
DOY	Slew rate of Y axis	optional
DOZ	Slew rate of Z axis	optional
ERR	Error messages and information	always
GAM	Euclidean sum of all gradient amplitudes	optional
GPN	GPA negative balance	optional
GPO	GPA ohmloss balance	optional
GPP	GPA positive current	optional
GRX	X gradient	always
GRY	Y gradient	always
GRZ	Z gradient	always
GSR	Slew rate of all gradient axes	optional
INF	Measurement data headers and information blocks	always
MOX	Gradient moment 0 order - X axis	optional
MOY	Gradient moment 0 order - Y axis	optional
MOZ	Gradient moment 0 order - Z axis	optional
M1X	Gradient moment 1st order - X axis	optional
M1Y	Gradient moment 1st order - Y axis	optional
M1Z	Gradient moment 1st order - Z axis	optional
MXX	Maxwell integral - X axis	optional
MXY	Maxwell integral - Y axis	optional
MXZ	Maxwell integral - Z axis	optional
NC1	Numeric controlled oscillator 1	always
NC2	Numeric controlled oscillator 2	optional

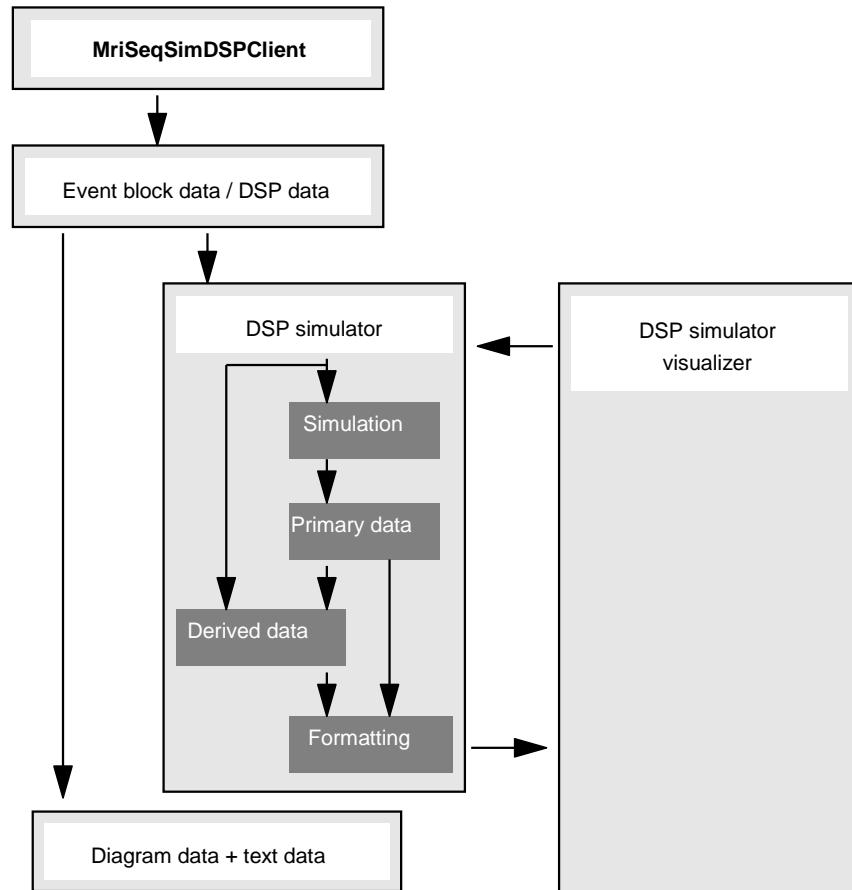
PXA	GPA Power duty cycle: X axis, time constant 1ms	optional
PYA	GPA Power duty cycle: Y axis, time constant 1ms	optional
PZA	GPA Power duty cycle: Z axis, time constant 1ms	optional
PXB	GPA Power duty cycle: X axis, time constant 10ms	optional
PYB	GPA Power duty cycle: Y axis, time constant 10ms	optional
PZB	GPA Power duty cycle: Z axis, time constant 10ms	optional
PXC	GPA Power duty cycle: X axis, time constant 100ms	optional
PYC	GPA Power duty cycle: Y axis, time constant 100ms	optional
PZC	GPA Power duty cycle: Z axis, time constant 100ms	optional
PR2	Burst RF-Power (2 nd tx ch. / nucleus)	optional
PRF	RF amplifier duty cycle	optional
RFD	RF signal data	always
RFP	RF signal phase	optional
SAP	Level of stimulation (0=ok / 1=1st level / 2=abort)	optional
SFX	X axis stimulation	optional
SFY	Y axis stimulation	optional
SFZ	Z axis stimulation	optional
SLL	Total stimulation Limit (i.e. if exceeds 100%, sequence aborts)	optional
SLT	Total stimulation Threshold (i.e. if exceeds 100%, switch to first level will be suggested)	optional
BON	B1	optional
BA0	B1 10ms average	optional
BBO	B1 100ms average	optional
BC0	B1 1000ms average	optional
BOS	B1 square	optional
BDO	B1 square 10ms average	optional
BEO	B1 square 100ms average	optional
BFO	B1 square 1000ms average	optional

SOFTWARE SYSTEM ARCHITECTURE

The block diagram below illustrates the dependencies of the **DSP simulation** program and **sequence data visualizer** applications:

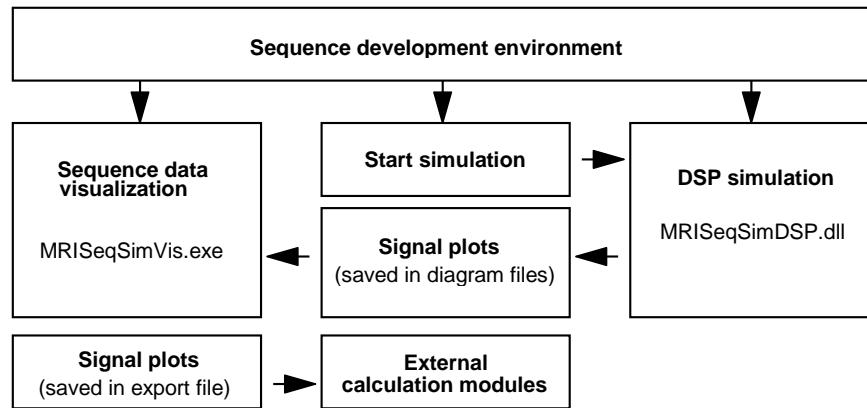
- ❑ The **DSP simulation** program generates primary diagram data from the sequence output data that can be saved in buffer files in a specific format (see page 109 for details). In addition, derived diagram data can be generated using additional input information and also saved in buffer files.
- ❑ Global information valid for all diagrams (e.g. trigger periods, event block data) is saved in text format in an information file.
- ❑ Using information from the information file, the **sequence data visualizer** generates and displays the respective signal plots from the diagram data.

DEPENDENCIES BETWEEN DSP SIMULATION PROGRAM AND VISUALIZER



Sequence Visualizer

The **Sequence data visualization program** provides the option of simultaneously displaying any number of diagrams generated by the **DSP simulation** program, as well as rearranging and scaling them independently from one another. There are also a number of mechanisms available that facilitate graph evaluation such as automatic diagram throughput, precise indication of signal values at specific intervals, and a grid that can be toggled.

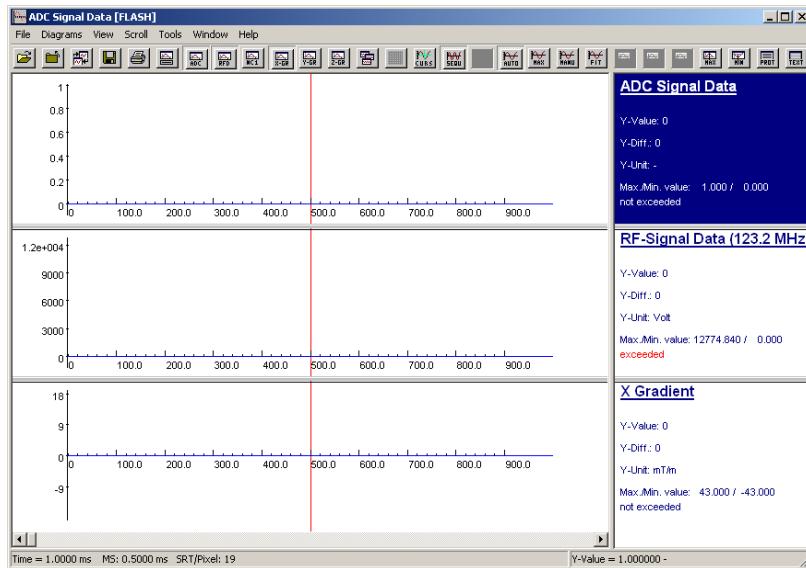


This program requires the input of diagram files generated by the simulator or files that were created elsewhere; the files must be, however, in the same format. In addition, a so-called information file containing information applicable to all diagrams of a test case is mandatory.

User interface

MAIN WINDOW

The main window consists of two parts. One part consists of a diagram field in which the individual graphs are displayed; the other part is an information field for entering diagram-specific information. An example of the program interface is shown in the figure below.



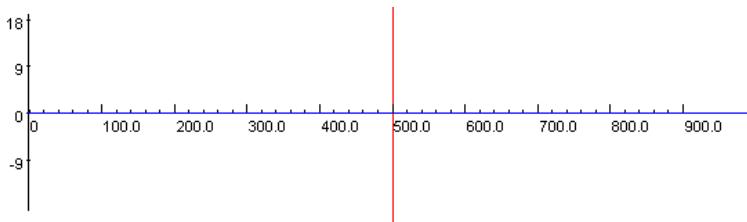
The main window has the following characteristics:

- Both the height and the width of the window can be increased or reduced with the mouse.
- Any number of diagrams can be displayed in the main window. The size of these diagrams is automatically adjusted to the size of the main window, i.e. the window is always completely filled.
- Moving the dividing line between the diagram field and the information field changes the display width of the information field.
- Moving the dividing line between two diagram fields changes the proportions of both diagrams.
- The diagrams may be scrolled to the left or right using a horizontal scroll bar. The entire time interval can be displayed, or scrolling can be performed either continuously or in increments (5% of the displayed time).
- A portion of the diagram may be selected by using the mouse and clicking in the diagram or information field (e.g. to determine Y-scaling). The diagram actually selected will be indicated by a colored diagram ID (displayed to the right of each diagram).
- Since several instances of the application can be called up, the respective sequence name is indicated in the header bar.

**CONTENT OF
WINDOW IN
DIAGRAM FIELD**

All of the diagram windows displayed in the main window have the following characteristics:

- The horizontal axis scale is the same for all diagrams and can be set using the View Mode dialog box.*
- The vertical axis scale can be individually configured for each graph.*
- The axes are labeled automatically according to the dimensions of the respective diagram window in order to prevent overlapping.*
- A vertical marker extends over the entire diagram field of the main window. Using this marker, a point in time can be marked whose numerical value is displayed in the View mode dialog window.*
- The trigger events are shown as thin, colored lines running vertically across the diagram field.*



**CONTENT OF
WINDOW IN
INFORMATION
FIELD**

The information window of the main window contains the following information for each of the diagrams indicated:

- Diagram name (e.g., X-gradient).*
- Current Y-value at the current marker location.*
- The difference in Y-values defined by the positions of the left and right marker.*
- Units of measure for Y-axis.*
- Indicates whether the signal exceeded or fell below the acceptable maximum or minimum at least once.*

<u>RF-Signal Data (123.2 MHz)</u>	
Y-Value:	0
Y-Diff.:	0
Y-Unit:	Volt
Max./Min. value:	12774.840 / 0.000 exceeded

In addition, this field has the following characteristics:

- A portion of the graph may be selected by clicking on the diagram field or the associated information field. This will be indicated by a shaded block of text.*

The individual diagrams can be rearranged together with their information fields. To do this, left-click in the diagram or info window, hold the left button, and drag the selected window below or above another window. Release the mouse button to reposition the diagram. The other diagrams are moved accordingly.

MENU BAR

The program functions may be called directly or via inserted dialog boxes with the help of main and submenus. For each main menu, a table is shown below which describes the individual submenu items.

The submenu **File** contains functions for loading diagrams from the relevant files, saving the settings, and the complete printing functionality.

Main menu	Submenu	Description
File	Open...	Calls up the Open dialog box. In the dialog box, a diagram file of a test run can be called up, and the graph can be loaded in the main window. The test name and the file directory are applied as the default setting for subsequent actions.
		In addition to a diagram file, a configuration file may be defined so that the diagrams displayed in the last session will be automatically reloaded and displayed in the same manner. Graphs that have been opened previously will be closed.
	Delete	Removes a marked diagram from the diagram window.
	Reload All...	Updates all displayed diagrams.
	Save	Saves type and mode of display for each loaded diagram in a common configuration file in the default directory. The same graphs will be automatically reloaded at the next program start-up if they still exist.
	Save Data As...	Calls up the Save As dialog box to determine an ASCII file in which the data points of all displayed diagrams are saved. In doing this, only the marked time period is saved.
	Print...	Activates the Print dialog box which is used to print all diagrams displayed in the main window.
	Print Setup	Activated the print setup dialog window
	Print Preview	Shows the print preview dialog window.

A series of diagrams can be loaded directly via the **Diagrams** menu item using the default test name and directory.

Main menu	Submenu	Description
syngo MR E12U 2019		- 97 -

Diagrams	ADC	Loads ADC on/off diagram in the main window.
	RFD	Loads RF diagram in the main window.
	NC1	Loads NC1 diagram in the main window.
	X-gradient	Loads X-gradient diagram in the main window.
	Y-gradient	Loads Y-gradient diagram in the main window.
	Z-gradient	Loads Z-gradient diagram in the main window.
	Additional...	Calls up the Additional Diagrams dialog box where selection of additional diagram types can be made.

The submenu **View** allows control of the signal display and the scaling of the Y-axes.

Main menu	Submenu	Description
View	Sequential	Displays signals sequentially.
	Overlapped	Displays signals overlapped according to their trigger events.
	Y-Scaling...	Activates Scaling Y-axis dialog box. This allows scaling of the Y-axes for each marked diagram.
	Manual Scaling	Allows to set own scaling values
Cross Hair		Switches cross hair cursor on or off.

The **Scroll** submenu enables you to continuously scroll forward or backward through all diagrams and jump to the maximum and minimum value of the marked diagram. The **View Mode** dialog box must be opened for the scroll function.

Main menu	Submenu	Description
Scroll	Forward	Synchronously scrolls through all diagrams at a constant velocity in the direction of the positive time axis.
	Backward	Synchronously scrolls through all diagrams at a constant velocity in the direction of the negative time axis.
	Stop	Stops the automatic scrolling through the diagrams.
	Maximum	Moves diagrams until the marker is located at the maximum value of the selected diagram.
	Minimum	Moves diagrams until the marker is located at the minimum value of the selected diagram.

The simulator can be called up directly from the **Tools** menu, a text file (e.g. protocol) can be viewed, and the grid can be toggled. The default scaling values can also be set.

Main menu	Submenu	Description
Tools	Grid	Toggles display of the grid.
	Grid Settings...	Calls up the Grid Settings dialog box to determine the grid characteristics.
	Protocol	Shows the protocol file in the default directory using a notepad that is activated automatically.
	Text File...	Activates the Open dialog box. Use this to select any text file and display it in the notepad. The notepad is started automatically after the file is selected.
	Options...	Calls up the Default Scaling dialog box with the option for setting the default scaling.
	Validity for all diagrams	Set validity for scaling and deletion to allo open diagrams.

The **Window** menu enables the configuration of the main window and the display of the **View Mode** dialog box.

Main menu	Submenu	Description
Window	Toolbar	Toggles display of the toolbar.
	Status Bar	Toggles display of the status bar in/out.
	View Mode...	Activates the View Mode dialog box. This is used for determining the time window, triggering and scroll options.

The Help table of contents and the Info dialog box are accessed under the **Help** menu.

Main menu	Submenu	Description
Help	Contents	Displays a table of contents for the Help topics.
	About dspvis...	Calls up the About dialog box.

TOOLBAR

The majority of functions that can be called up via the menu bar are also available using the buttons in the toolbar. If functions are on/off switches, the buttons appearance indicates the current status.

If the mouse pointer is positioned over one of these buttons, brief help texts appear beside the button (Tool Tips) along with a slightly more detailed description in the status bar.

The elements of the toolbar are described briefly below.



Loads a graph from a diagram file of a test run.



Removes the marked diagram from the diagram window.



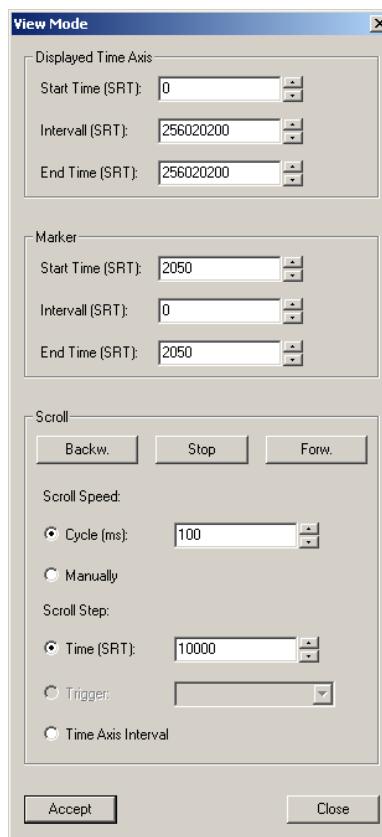
- Updates all displayed diagrams.
- Saves information regarding loaded diagrams for the next start-up.
- Prints all diagrams displayed in the main window.
- Toggles display of the **View Mode** dialog box.
- Loads the **ADC on/off** diagram in the main window.
- Loads the **RF** diagram in the main window.
- Loads the **NC1** diagram in the main window.
- Loads the **X-gradient** diagram in the main window.
- Loads the **Y-gradient** diagram in the main window.
- Loads the **Z-gradient** diagram in the main window.
- Activates the **Additional Diagrams** dialog box.
- Toggles the grid display on/off.
- Toggles the cross hair cursor on/off.
- Signals are sequentially displayed in the diagrams.
- Signals are displayed overlaid in the diagrams.
- Scales the Y-axis of the selected diagram to maximum permissible values.
- Scales the Y-axis of the selected diagram to largest signal value.
- Activates the **Manual Scaling** dialog box.
- Scales the Y-axis of the selected diagram to current extreme values.
- Automatically scrolls through all diagrams in the direction of the negative time axis.
- Stops the automatic scrolling.
- Automatically scrolls through all diagrams along the positive time axis.



- Moves diagrams until the marker is located on the maximum value of the signal.
- Moves diagrams until the marker is located on the minimum value of the signal.
- Shows dialog box for selection of an input file for the simulator.
- Displays the protocol files with the notepad.
- Shows the dialog box for opening the desired text files.
- Displays the **About** dialog box.
- Creates pointer for using Help table of contents.

**DIALOG
WINDOW FOR
VIEW MODE**

The **View Mode** window is a non-modal dialog box, i.e., it can remain on the desktop and be moved wherever necessary during execution of the visualization program.



This dialog box is used to control X-scaling and triggering, to read the marker values, and to operate the **Player** for the automatic diagram run. This dialog box contains the following elements:

DISPLAYED TIME**AXIS****Start Time**

Determines the earliest time that can be displayed in the diagram window. If a change is made, the end time is automatically corrected.

End Time

Determines the latest time that can be displayed in the diagram window. If a change is made, the interval is automatically corrected.

Interval

Determines the interval displayed in the diagram window (difference between the start time and end time). If a change is made, the end time is automatically corrected. The TR interval is the default value.

MARKER**Start Time**

Shows the time designated by the marker in microseconds. By changing this time, the marker can be repositioned between **Start Time** and **End Time**. If a change is made, the interval is automatically adjusted.

End Time

As soon as the marker is in 'time interval' mode, the time at the marker button on the right is displayed in this field; alternately the field also contains the **Start Time**. If the interval entry is changed, this value is updated.

Interval

This field displays the time span between the two markers. Otherwise, zero is displayed. If a change is made, the appropriate area is indicated by the marker.

SCROLL**Forward**

By activating this button, automatic scrolling in the direction of positive time is started.

Backward

By activating this button, automatic scrolling in the direction of negative time axis is started.

Stop

Stops automatic scrolling through the diagrams.

Scroll Speed

This section controls whether scrolling should be performed automatically or manually.

Cycle - If selected, the program automatically advances Forward/Backward by the increment entered in milliseconds.

Manually - If selected, the program advances Forward/Backward by dragging the mouse.

Scroll Step

The step width for scrolling can be set with this option field.

Time - Any desired increment can be entered here in microseconds.

Trigger - All available types of synchronization are listed in the box. If a type is selected, the program will skip each time to the preceding or following trigger event.

Time Axis Interval - The step width here corresponds exactly to the displayed interval of the time axis.

NOTE: Scrolling, whether manually or automatically, will not change the position of the marker or marked time period in the diagram window. However, both the time field in the **View Mode** dialog box and the data in the information field of the main window are updated. The marker must be repositioned to deselect the marked time period.

MISCELLANEOUS **Accept**

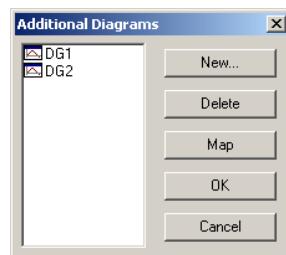
Applies the settings. Each new entry must be acknowledged with either the Return key or the **Accept** key.

 Close

Closes the **View Mode** dialog box.

**DIALOG
WINDOW FOR
ADDITIONAL
DIAGRAMS**

In order to load a diagram directly from the associated file when there is no button on the toolbar or corresponding menu entry present, select an icon from the list in the **Additional Diagrams** dialog box.



As shown in the figure above, the list field contains for allocating the respective file an icon for each additional type of diagram along with the ID. All file IDs that were amended for additional diagrams are saved in the initialization file so that they will be available again at the next program start-up. In addition, the following dialog box elements are present.

 New...

The **File Identifier** dialog box is displayed with this button. The character combination for identifying additional diagrams must be supplied.

 Delete

Removes a previously selected element from the list field and deletes the file ID from the initialization file.

Cancel

Closes the **Additional Diagrams** dialog box. Previously made changes are lost.

**DIALOG
WINDOW FOR
FILE ID**

The dialog box shown here is called up via the **New...** button in the **Additional Diagrams** dialog box and is used for entering a new file ID.



The following elements are available:

New File Identifier

The combination of letters and numbers that represent the corresponding diagram files are entered into this edit field.

OK

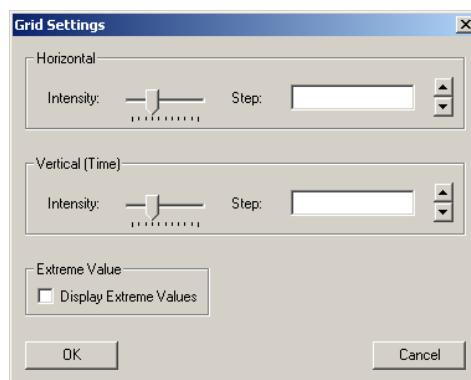
The new ID is saved by pressing this button and an additional icon is added to the list field of the **Additional Diagrams** dialog box. The new file ID is saved in the initialization file.

Cancel

This button closes the dialog box and ignores the entry performed in the edit field.

**DIALOG
WINDOW FOR
GRID SETTINGS**

To enable a more accurate reading of signal values, a grid can be superimposed over the graphs. This grid can be configured for each marked diagram using the **Grid Settings** dialog box.



The figure above shows an example window where the following dialog elements are available for use:

Horizontal/Intensity

Using this slide, the brightness of the horizontal grid lines between 0 and 100% (default value is 50%) can be adjusted.

□Vertical/Intensity

Using this slide, the brightness of the vertical grid lines between 0 and 100% (default value is 50%) can be adjusted.

□Horizontal/Step

With this edit field, the distance between the horizontal grid lines can be specified in microseconds.

□Vertical/Step

With this edit field, the distance between vertical grid lines can be specified in the appropriate units for the diagram.

□Extreme Value

By marking the **Display Extreme Values** check box, four additional colored horizontal lines are plotted for each diagram. Two of the lines represent the minimum and maximum permissible signal values; two represent the current extreme signal values.

□OK

Closes the window and saves all settings that have already been carried out. The new values are accepted immediately if the grid has been enabled.

□Cancel

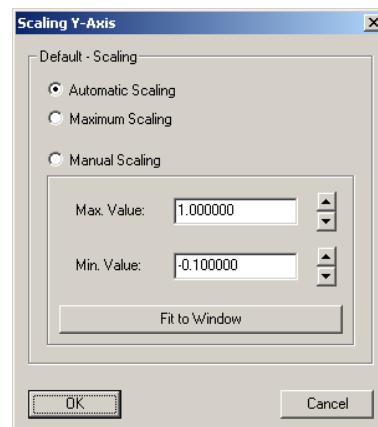
Closes the window without saving the settings previously performed.

**DIALOG
WINDOW
FOR Y-AXIS
SCALING**

The Y-axes can be scaled separately for each of the displayed diagrams. The three dialog boxes described below have been provided for this.

SCALING Y-AXIS

The **Scaling Y-axis** dialog box shows the configuration of the vertical axis for the selected diagram:



The window displays the following settings:

Automatic Scaling

The Y-axis of the corresponding graph is scaled so that the largest scaling value corresponds to the maximum or minimum of the total signal.

 Maximum Scaling

The Y-axis of the corresponding graph is scaled so that the largest scaling value is determined by the extreme value that cannot be exceeded by the signal.

 Manual Scaling

The Y-axis of the corresponding graph is scaled to user-specified limits.

 Fit to Window

Applies the maximum and minimum values from the current view as extreme values.

 OK

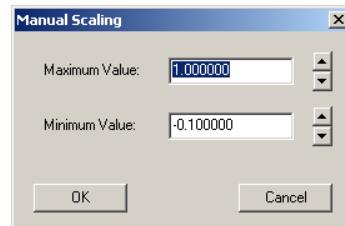
Closes the dialog box, applies the selected setting and adjusts the graph accordingly.

 Cancel

Closes the dialog box and retains the previous settings.

MANUAL SCALING

The **Manual Scaling** dialog box illustrated in the figure below has the same functionality as the corresponding edit field of the **Scaling Y-axis** dialog box. However, this window is evoked as soon as the corresponding button is activated on the toolbar.



The dialog window contains the following elements:

 Maximum Value

The dimensions entered in this field define the highest displayable values of the vertical axis for the selected diagram.

 Minimum Value

The dimensions entered in this field define the lowest displayable values of the vertical axis for the selected diagram.

 OK

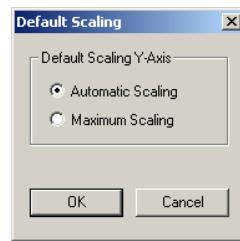
Closes the dialog box, applies the settings and adjusts the graph accordingly.

 Cancel

Closes the dialog box and retains the previous settings.

DEFAULT SCALING

The default setting for Y-scaling is set using the dialog box pictured below.



□Default Scaling Y-axis

Either Automatic Scaling or Maximum Scaling can be selected.

□OK

Closes the dialog box and saves the settings.

□Cancel

Closes the dialog box without saving.

CONTEXT MENU

To make the menu prompt more convenient to use, a context menu is available with a right-click of the mouse. The cursor is located in the dialog field of the main window. The position of the cursor determines which menu items are available for use.



The following entries are provided:

□Zoom in

Using this function, the signal segment covered by the marker is enlarged to the point that it completely fills the time period displayed. If no period is marked, the graph is doubled in size.

□Undo zoom In

Undo the last zoom step done with the menu entry 'Zoom In' described above.

□Zoom out

Zoom in is used to make the displayed diagrams smaller. The time period displayed doubles.

□Display All

This function undoes all Zoom and displays the complete diagrams.

❑ Measurement Header

Measurement data headers are available as additional information assigned to each time period. The measurement data header pertaining to this point in time is displayed in a window according to the position of the marker. When you scroll through the diagram window, this window displays the respective current measurement data header. The user will be notified if there is no information available for the respective point in time (ADC off).

❑ Event Block

Information about the current Event block can be called up with the context menu. The appropriate menu item, however, is specific for every diagram. This information is also time-dependent and is made available in a separate window.

❑ Y-scaling

In addition, the "Scaling Y-axis" dialog box can be called up from within the context menu. This shows the type of scaling for the diagram that was indicated by the mouse pointer at the time of the call-up.

❑ Delete

The selected diagram will be removed from the main window.

Error handling

The buffer files generated by the **DSP simulation** program form the only interface between the visualization program and other software components. These files are being checked for accuracy before loading. This check includes the following:

- ❑ If no corresponding buffer files with default test names exist in the default directory for the diagrams to be loaded via buttons or menu items, the user will be notified with a message window.
- ❑ If not all expected signal information is present in the buffer files in the required format, this will also be indicated by means of a message window.
- ❑ As long as no respective information file is present in the test file directory, no diagrams can be loaded either because the time information for the trigger points is missing. The user will also be notified.
- ❑ When the program first starts up, it searches for a configuration file in the default directory. If a configuration file is found, then only those diagrams that are specified in this file and still exist are loaded.
- ❑ The user receives an error message if no initialization file is present when the application starts.

DSV file format

Each signal being displayed in its own subwindow is saved in its own file. The filename follows the convention

<SequenceName>_<SignalType>.dsv

Where <SignalType> denotes a three character abbreviation such as "ADC" or "GRX".

Each dsv file has the following sections and keywords:

Keyword	Description	Example
[FILETYPE]		
FORMAT	This entry specifies the version of the Visualization tool which is required to open this dsv file.	DSV_V0100
[DEFINITIONS]		
TITLE	The title specifies the diagram type and is displayed in the right-hand side information window.	ADC Signal
HORIUNITNAME	Unit of the x axis (used for labelling the diagram and in the "view mode" window)	μsec
HORIDELTA	Distance of the sample points on the time axis	1.0
VERUNITNAME	Unit of the y axis (used for labelling the diagram and in the "view mode" window)	Volt
VERTFACTOR	All values in the [VALUES] section are to be divided by this factor.	1000
SAMPLES	Total number of samples listed in the [VALUES] section (uncompressed).	12345678
MAXLIMIT	Maximum which may not be exceeded by the signal values	1
MINLIMIT	Minimum below which the signal values must not fall	-1
[FRAME]		
STARTTIME	Simulation starts at the specified time point (in ms)	100
ENDTIME	The simulation was stopped at the specified time (in ms)	12000
[VALUES]		
<n>\n	The values are signed 32-bit integers. The real value will be obtained after division by 'VERTFACTOR'. If the same sample value is repeated several times, the value is only listed twice, and the third value specifies the number of repetitions. Only the first sample is stored as an absolute value. All other values are deltas to the preceding value.	The sample sequence "1,1,1,1,5,5,8,8,8,1,0,11" will be encoded as "1,1,2,5,5,0,8,8,1,1

(This way it is possible to compress linear slopes.) 0,11".

D.4 Pulse Tool

Introduction

In a pulse sequence, gradient and RF pulse data are obtained from two sources: calculated during fSeqPrep or supplied from an external data file. The **Pulse Tool** program is a standalone program for analysis and modification of pulses stored in an external library file. The pulse envelopes are displayed in different windows that can be positioned and sized as required. The program also includes functions for modifying, copying, moving, and deleting pulses. It also calculates the additional diagram data required for display.

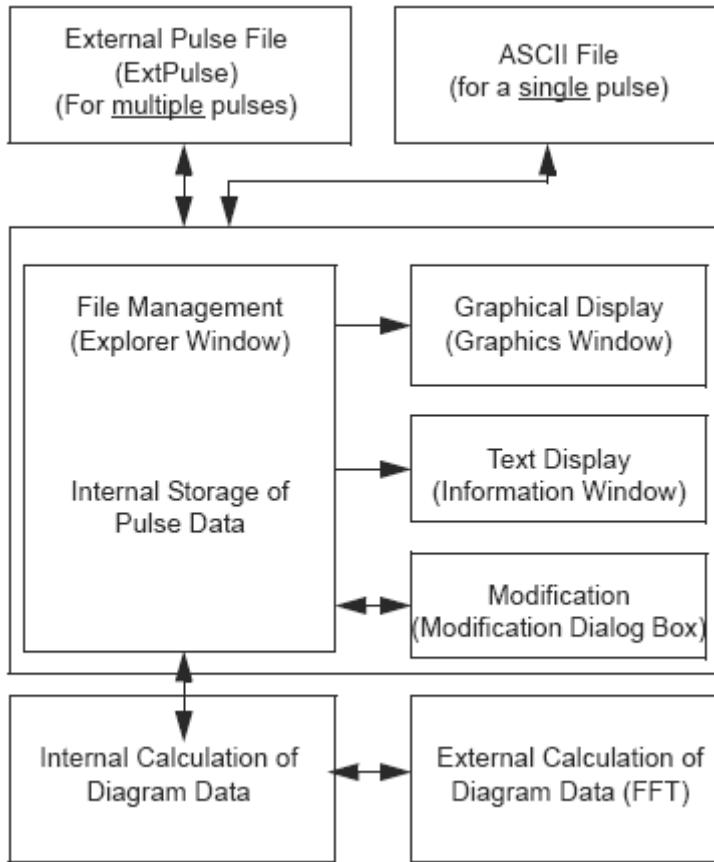
The calculation of additional diagram data may use an algorithm that is not a part of the program. An appropriate algorithm must be made available as a .dll file. In this document, the program functions are described using a Fourier transform as example.

NOTE: After having modified external RF pulses, it is necessary to either restart syngo or reboot the host in order for the RF database to be re-read. The RF database is read once when it is first accessed and thereafter kept in memory. (This is only true on the scanner -- on a standalone IDEA RF database changes take effect immediately.)

Software System Architecture

The following block diagram illustrates the dependencies between the various program sections and the external software components:

- ❑ The **Pulse Tool** application receives pulse information either from pulse files generated by the **PulseTool** (ExtPulse) or from manually generated ASCII files.
- ❑ Before pulse data can be displayed (in the graphics or information windows) or modified (using the Modification dialog box), the data must first be stored in the program.
- ❑ Changes made to the data records are applied when the records are saved to the pulse file.
- ❑ The calculation of the envelopes for the pulse types not contained in the pulse file (for RF pulses) is performed either within the program (an internal algorithm) or in an external .dll file (for example, Fourier Transform) serving as an interchangeable module.



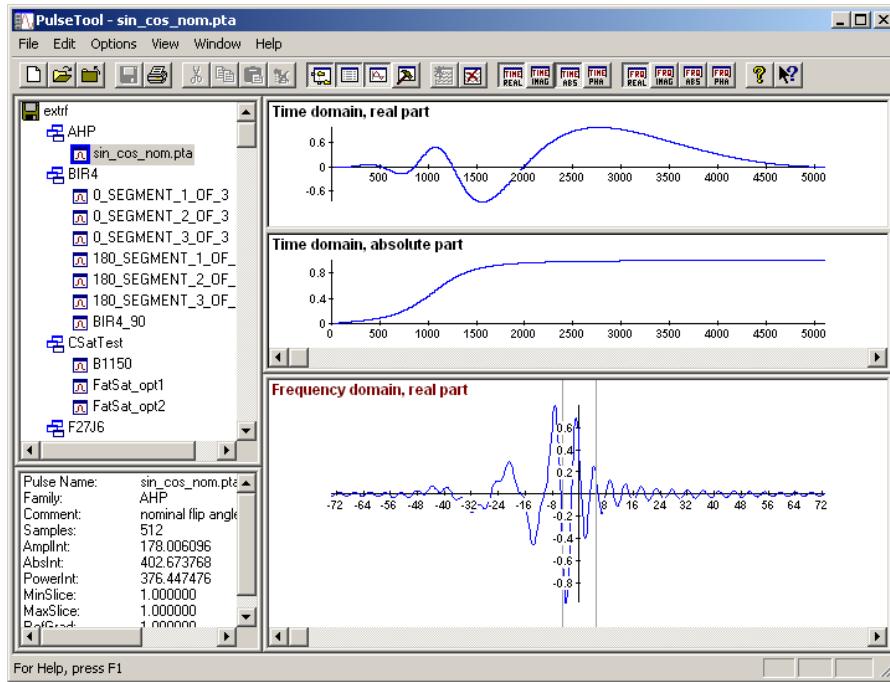
User Interface

The **PulseTool** application is an MDI Windows program comprising three separate windows and several dialog boxes. The software texts and output are in English. There is no NLS support. The program is not intended for use with black-and-white monitors, because it is difficult to differentiate the pulse envelopes on these monitors. The figures in this chapter are examples intended to show the graphical format of the user interface.

Upon opening a pulse file, the Pulse Tool copies the contents into a workspace. All operations are performed on this copy. Any changes are not stored in the original file until the Save command is selected.

Main Window

The main window comprises three sections. First is the Explorer window (upper left), listing the individual pulses contained in the pulse files. Second is the Information window (lower left), where pulse-specific data is summarized. Finally, there is the Graphics window (right), which displays the pulse envelopes in graphical format. The following figure is an example of the program interface.



The main window has the following characteristics:

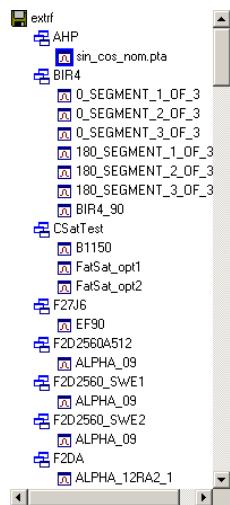
- The display of the three sub-windows can be adjusted as desired. If only the Explorer and Graphics windows are displayed, they fill the entire area of the main window. If only the Graphics window is displayed, it fills the entire area of the main window.
- Moving the dividing line between the Explorer, Information, and Graphics windows changes the proportions of these sub-windows.
- Moving the dividing line between two diagrams in the Graphics window changes the proportions of the diagrams.
- The arrangement (sequence) of signal diagrams can be changed by clicking and holding the left mouse button and dragging the diagram inside its window to a new position.
- Scrolling through the diagrams to the left or right is achieved using the horizontal scroll bar. Scrolling can be continuous (by moving the scroll button), step-by-step at 5 % of the displayed time interval (by clicking the arrows), or by the entire time interval (by clicking the scrolling region not covered by the scroll button).

CHARACTERISTICS OF THE EXPLORER WINDOW

The Explorer window displays the hierarchical organization of the pulses within a pulse file in a tree structure. The levels shown in the following table:

Level	Symbol	Description
syngo MR E12U 2019	- 112 -	© Siemens Healthcare GmbH confidential

File		The File level is the highest level to be displayed. A pulse file can contain multiple pulses. The file name is displayed next to the symbol.
Family (RF only)		RF pulses are grouped in families. This level is displayed only when the program is running in RF mode. The family name is determined from the first part of the pulse name.
Pulse		The lowest level contains the pulse data records. The pulse name is specified in the corresponding section of the pulse file.



The following actions can be performed within the Explorer window shown here:

- Left click using the mouse on a pulse to select it. To select multiple pulses, hold down the **Ctrl** key and single click on each pulse to be selected.*
- Double-click on a pulse to display it in the graphics window. Any pulse currently being displayed in the graphics window is replaced in the window.*
- A second click on a selected pulse (a slow double-click) will open the editing field, allowing renaming of the pulse within the Explorer window. The key combination **CTRL-R** allows doing the same. If multiple pulses have been selected, this second click selects the pulse to be modified.*
- Using Drag and Drop, a pulse can be moved from one file to another. By holding down the **Ctrl** key during Drag and Drop, the pulse will be copied from one file to another.*
- A pulse can be displayed by selecting it with the left mouse and dragging it into the Graphics window. Previously displayed pulse diagrams are deleted from the window.*
- Using the **Delete** key, the **Delete** menu item or the corresponding icon on the toolbar, pulses or families will be removed from the files.*

NOTE: All modifications performed on the pulse files in the Explorer window are not applied to external files until the files are updated using the **Save** command.

CHARACTERISTICS OF THE INFORMATION WINDOW

The Information window, as shown here, displays pulse-specific data describing the properties of the pulse. The following information is listed:

- Pulse name.
- Comment - a text string.
- Number of values (support points).
- Reference Gradient (RF pulses only).

Reference gradient = Amplitude of a slice selection gradient required to excite a 10 mm thick slice.

- Power integral (RF pulses only):

$$\text{Powerintegral} = \sum_{\text{allvalues}} [(\text{timesignal}_{\text{Real}})^2 + (\text{timesignal}_{\text{Imag}})^2]$$

The power integral is used for the SAR calculation and is scaled by the square of the reference amplitude.

- Amplitude integral (RF pulses only):

$$\text{Amplitudeintegral} = \sqrt{\left(\sum_{\text{allvalues}} \text{timesignal}_{\text{Real}} \right)^2 + \left(\sum_{\text{allvalues}} \text{timesignal}_{\text{Imag}} \right)^2}$$

The amplitude integral should be a value that will be multiplied times the reference amplitude that will produce the desired rotation. For amplitude modulated pulses, this is the sum of the support points. For phase-modulated pulses like SLR or other adiabatic pulses, you must decide what number to use so that the flip angle specified for the pulse gives the desired rotation.

- Magnitude Integral or Absolute Integral (RF pulses only):

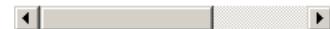
$$\text{Magnitudeintegral} = \sqrt{(\text{Timesignal}_{\text{Real}})^2 + (\text{Timesignal}_{\text{Imag}})^2} = \sum_{\text{allvalues}} \text{timesignal}_{\text{Abs}}$$

- Minimum slice thickness (RF pulses only).
- Maximum slice thickness (RF pulses only).

The Information window also has the following characteristics:

- Horizontal and vertical scroll bars are displayed if the complete set of information cannot be displayed in the window.
- If the displayed pulse is modified or a new pulse is displayed, the information in the window updates automatically.

Pulse Name:	sin_cos_nom_pta
Family:	AHP
Comment:	nominal flip angle = 4
Samples:	512
AmplInt:	178.006096
AbsInt:	402.673768
PowerInt:	376.447476
MinSlice:	1.000000
MaxSlice:	1.000000
RefGrad:	1.000000



- If multiple pulses are displayed on the Graphics window, data regarding the pulses is not displayed. The Information window remains blank.

CHARACTERISTICS OF THE GRAPHICS WINDOW

The Graphics window provides a visual display of the pulse data. The nature of the display depends on the nature of the selected pulse:

For gradient pulses, a single window is used, as only the pulse amplitude values are displayed.

If RF pulses are displayed, a so-called Split Window is used. The split window can be divided into up to eight different sections. The following pulse envelopes are displayed in these sections:

In stepped line display mode:

- Real part of the time signal
- Imaginary part of the time signal
- Magnitude value of the time signal
- Phase of the time signal

In interpolated display mode:

- Real part of the transformed signal
- Imaginary part of the transformed signal
- Magnitude value of the transformed signal
- Phase of the transformed signal

Choice of the eight windows for display is made using the button on the toolbar. The Graphic window region is always filled, regardless of the number of signal windows displayed.

NOTE: Time diagrams and their corresponding frequency diagrams will always be displayed together. Scroll and zoom functions are available for both diagram types, but they operate only on the selected diagram.

The Graphics displayed in the graphics window have the following characteristics:

- The x scale is the same for all displayed time diagrams and all displayed frequency diagrams (relevant only to RF pulses).
- Separate scroll bars are available for the time and frequency diagrams. This enables scrolling of the different diagram types independent of one another.
- When a pulse is displayed for the first time, the complete pulse (amplitude and signal duration) is displayed in the windows. At this point, the scroll bar is not active.

- A zoom function is available for both axes. Double-clicking the left mouse button in the diagram region displays horizontal and vertical lines that intersect at the marked point.

Zoom function:

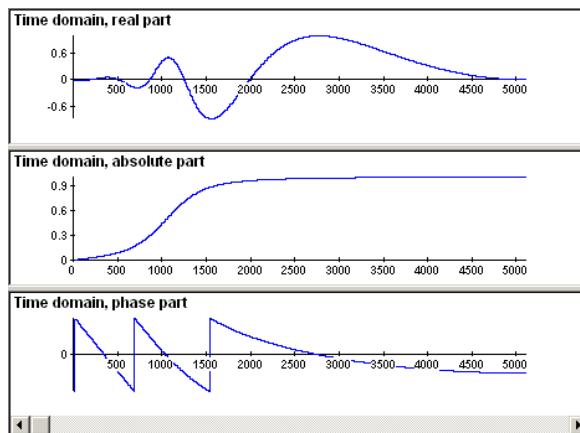
By dragging one of these lines and releasing the mouse button, the x or y section established is magnified accordingly.

Zoom out function:

By activating the context menu using the right mouse button, the signal display can be increased in either the horizontal or vertical direction. Starting at the line intersection, the graph is reduced by a factor of 2.

Zooming affects all graphs of the same diagram type (time or frequency).

- Axis labeling corresponds to the diagram size to prevent overlapping of numerical values.
- If multiple pulses are displayed at the same time, the individual pulse envelopes are displayed in various colors to differentiate them (corresponding colors identify the pulses in the Explorer window).
- The excitation bandwidth is marked by two vertical lines (not in the case of uniform display of multiple pulses).



MENU BAR

Using the main and submenus, the program functions can be activated either directly or through a series of dialog boxes. The next section of this document presents a table for each main menu item, and describes each of the submenu items.

The **File** menu contains the functions for loading and storing external pulse files, as well as all print functions.

Main Menu	Submenu	Description
-----------	---------	-------------

File	New...	Creates an empty file structure for pulses that can be added through copying and moving. The file is not actually created until Save or Save as is selected.
	Open...	Activates the Open dialog box. The selected external pulse file is opened and loaded into the program. Multiple files can be loaded at the same time, but they must be of the same pulse type. The first file loaded determines the mode (RF or Gradient) the program runs in until all files are closed.
	Close	Closes the pulse file. If the file has been modified, a prompt is displayed allowing saving of the file.
	Save	Saves the content of the selected pulse file. SIEMENS files are password protected.
	Save As...	Saves the content of the selected pulse file under a different file name.
	Print...	Activates the Print dialog box. All diagrams displayed in the graphics window can be printed using the entire page.
	Print Preview	Provides a preview of the page to be printed.
	Print Setup...	Activates the Print Setup dialog box, in which print options can be set.
	Recent Files	Lists the five most recently selected pulse files.
	Exit	Ends the PulseTool program. If the file has been modified, a prompt is displayed allowing saving of the file.

The **Edit** menu allows selection of a number of functions for managing and displaying pulses.

Main Menu	Submenu	Description
Edit	Show	Displays the pulse envelopes for all selected pulses in the graphics window. The diagram types to be displayed can be chosen using menu item View > Diagrams or the corresponding button on the toolbar.
	Clear	Clears all pulse envelopes from the Graphics window.
	Rename	Places the name of the selected pulse in an editing field in the Explorer window allowing renaming (not available if multiple pulses have been selected).
	Cut	Removes a pulse from a pulse file in the Explorer window. The pulse can be added to another (previously selected) file using the Paste command.
	Copy	Copies one or more pulses from a pulse file in the Explorer window. The pulse or pulses can be added to another (previously selected) file using the Paste command.

Paste	Pastes a cut or copied pulse or pulses into a previously selected pulse family or file.
Modify...	Activates the Gradient Pulse Modification or RF Pulse Modification dialog box, as appropriate. Use the dialog box to modify the characteristics of the selected pulse (not available if multiple pulses have been selected).
Delete	Deletes the selected pulse from a pulse file.
Import...	Activates the Open dialog box. An ASCII pulse file can be selected and loaded into a previously selected pulse file or family. If necessary, the sampling points of additional signals and the integral values are calculated. In addition, the program checks the type and compares the calculated values to the integral values provided in the file.
Export...	Stores the selected pulse in an ASCII pulse file. The file is selected using the Save As dialog box.

The **Options** menu allows specification of time information needed for the graphical display. In addition, the name of the .dll file containing the transform algorithm to be used can be specified.

Main Menu	Submenu	Description
Options	Grad Time Step...	Activates the Grad Time Step dialog box. Use this dialog box to set the time step between the values displayed in the gradient diagram.
	RF Pulse Duration ...	Activates the RF Pulse Duration dialog box. Use this dialog box to set the RF pulse duration required for the display of the frequency signal.
	Transformation DLL...	Activates the Open dialog box. The path to the .dll file containing the transform algorithm can be specified. (for example <code>\n4\x86\delivery\bin\PulseToolFFT(d).dll</code>)

The **View** menu allows specification of the main window elements to display, such as the toolbar, status bar and signal window.

Main Menu	Submenu	Description
View	Toolbar	Toggles display of the toolbar.
	Status Bar	Toggles display of the status bar.
	Point View	Displays each sampling point as a single point (no connecting line between the values).
	Unwrapped	Shows the phase in the frequency domain without the 2π phase wrap.
	Diagrams ...	Lists and allows selection of the diagram types that can be displayed in the Graphics window.

The **Window** menu establishes which windows to display in the client area of the main window.

Main Menu	Submenu	Description
Window	Explorer	Toggles display of the Explorer window.
	Information	Toggles display of the Information window.
	Graphic	Toggles display of the Graphics window.

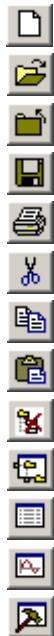
The **Help** menu takes you to the help table of contents and the **Info** dialog box.

Main Menu	Submenu	Description
Help	Contents	Toggles display of the help window.
	About	Activates the About dialog box.
	Pulse Tool...	

TOOLBAR

Many of the functions that can be selected from the menus can also be selected using the buttons on the toolbar. They are designed as on/off buttons, and their current status is indicated by their appearance on the toolbar. By moving the mouse pointer over one of these buttons, a brief help text known as a tool tip is displayed, along with a more complete description in the status bar.

The following describes the various buttons on the toolbar:



- Creates a blank file structure for accepting pulses.
- Loads a pulse file in the program and displays it in the Explorer window.
- Removes the selected pulse file from the Explorer window.
- Saves the content of the selected pulse file.
- Prints all diagrams displayed in the Graphics window.
- Cut a pulse from a file.
- Copies a pulse from a file.
- Pastes a previously copied or cut pulse to a file.
- Removes the selected pulse from a pulse file.
- Toggles display of the Explorer window.
- Toggles display of the Information window.
- Toggles display of the graphics window.
- Toggles display of the **Pulse Modification** dialog box.

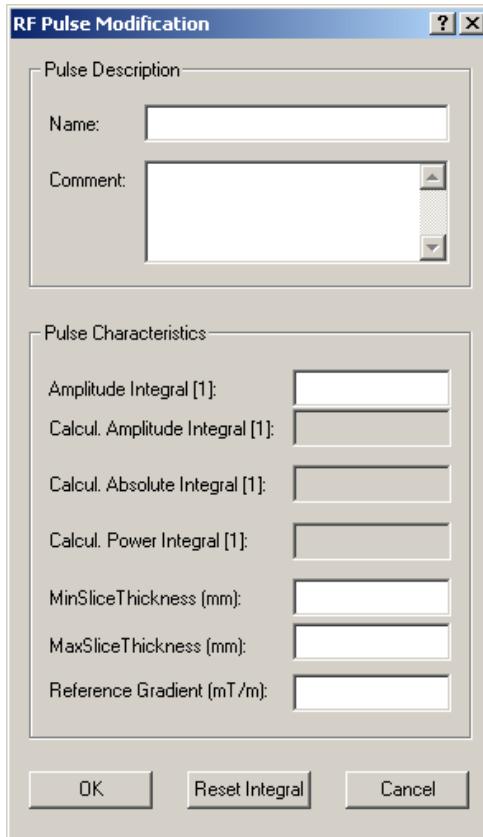


- Displays the pulse envelopes for all selected pulses in the graphics window.
- Deletes the displayed pulse envelopes from the graphics window.
- Displays the real part of the time signal in the graphic window.
- Displays the imaginary part of the time signal in the graphic window.
- Displays the magnitude value of the time signal in the graphic window.
- Displays the phase of the time signal in the graphic window.
- Displays the real part of the frequency signal in the graphic window.
- Displays the imaginary part of the frequency signal in the graphic window.
- Displays the magnitude value of the frequency signal in the graphic window.
- Displays the phase of the frequency signal in the graphic window.
- Displays the help table of contents.
- Provides a mouse pointer for context-sensitive help.

**MODIFICATION
DIALOG BOX
FOR RF PULSES**

Modification Dialog Boxes

The characteristics of RF pulses can be modified using a range of parameters, which are provided in the dialog box shown below. In addition, the pulse name can be changed and comments added as desired.



**PULSE
DESCRIPTION**

Name

Enter a new name for the pulse in this field.

Comment

Descriptive comments can be entered in this field.

**PULSE
CHARACTER-
ISTICS**

Amplitude Integral

Allows the entry of a new value for the amplitude integral (can be set back to the calculated value using the RESET button).

Calcul. Amplitude Integral

Displays the calculated value for the amplitude integral.

Calcul. Absolute Integral

Displays the calculated value for the magnitude integral.

Calcul. Power Integral

Displays the calculated value for the power integral.

 MinSliceThickness

Allows entry of a new value for the minimum slice thickness (must be less than or equal to the maximum slice thickness and greater than 0).

 MaxSliceThickness

Allows entry of a new value for the maximum slice thickness (must be greater than or equal to the minimum slice thickness and greater than 0).

 Reference Gradient

Allows entry of a new value for the reference gradient.

 OK

Accepts the changes made and closes the **RF Pulse Modification** dialog box.

 Reset Integral

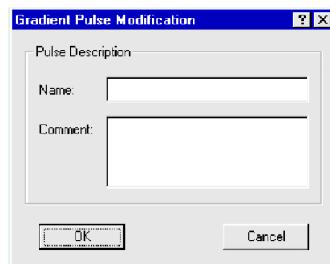
Resets the modified amplitude integral value back to the calculated value.

 Cancel

Closes the **RF Pulse Modification** dialog box without making any changes.

**MODIFICATION
DIALOG BOX
FOR GRADIENT
PULSES**

For gradient pulses, only the pulse name and comments can be changed. The modification window is displayed below.



**PULSE
DESCRIPTION**

 Name

Enter a new name for the pulse in this field.

 Comment

Descriptive comments can be entered in this field.

 Accept

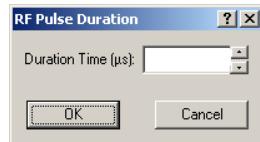
Accepts changes and close the **Gradient Pulse Modification** dialog box.

 Cancel

Closes the **Gradient Pulse Modification** dialog box without making any changes.

**DIALOG BOX
FOR SETTING
THE RF PULSE
DURATION**

In order to display a frequency diagram with the correct frequency scale, the RF pulse duration must be specified. This information can be entered in the **RF Pulse Duration** dialog box (refer to the figure below). This value is required for graphical display only.



Duration Time

In this field, the pulse duration can be modified either directly or using the scroll function.

OK

Accepts the changes made and closes the dialog box.

Cancel

Closes the RF Pulse Duration dialog box without making any changes.

**DIALOG BOX
FOR SETTING
THE GRADIENT
TIME STEP**

The time step between the displayed values (sampling points) in the gradient diagram can be specified. Use the dialog box shown below.



Time Step

In this field, the time step can be modified either directly or using the scroll function.

OK

Accepts the changes made and closes the dialog box.

Cancel

Closes the Gradient Time Step dialog box without making any changes.

CONTEXT MENU

The signal display range can be decreased (zoom in) using the mouse.

See page 115.

To increase the signal display range (zoom out) double click the diagram and right click to display the context menu for the zoom function. The following functions are available:

X-Axis Zoom Out (x 2)

This function reduces the graph in X-direction by a factor of 2 (doubling the X-Axis sector).

The displayed line indicates the center of the new sector.

Y-Axis Zoom Out (x 2)

This function reduces the graph in Y-direction by a factor of 2 (doubling the Y-Axis sector).

The displayed line indicates the center of the new sector.

X-Axis Zoom Out (complete)

Using this function, the graph is displayed in full in X-direction (display of the complete X-Axis).

Y-Axis Zoom Out (complete)

Using this function, the graph is displayed in full in Y-direction (display of the complete Y-Axis).

NOTE: Time and frequency displays for RF pulses can be controlled independently. All graphs of the same type (time or frequency) will be scaled equally.

RF Pulse files types

External Pulse Files

These files can contain one or more pulse data records; however, they must all be of the same type (RF or gradient). Depending on the pulse type, the data records contain the following information:

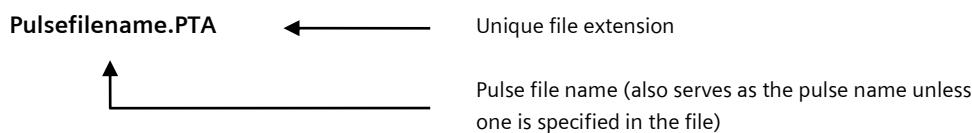
GRADIENT DATA RECORD	<input type="checkbox"/> <i>Pulse name</i> <input type="checkbox"/> <i>Number of amplitude values</i> <input type="checkbox"/> <i>Table of amplitude values</i>
-----------------------------	---

RF DATA RECORD	<input type="checkbox"/> <i>Pulse and family name</i> <input type="checkbox"/> <i>Integral values, slice thickness and reference gradient</i> <input type="checkbox"/> <i>Number of magnitude (and phase) values (support points)</i> <input type="checkbox"/> <i>Table of magnitude values, followed by phase values</i>
-----------------------	--

NOTE: External pulse files from NUMARIS/4 are not compatible with previous NUMARIS versions.

ASCII Pulse Files

ASCII pulse files always contain a single pulse data record. In contrast to external pulse files used for sequence execution, pulse data is stored in ASCII format instead of binary format. ASCII files have a unique file extension to differentiate them from external pulse files.



**PULSE-SPECIFIC
PARAMETERS**

The pulse-specific parameters describe the pulse and provide information regarding the integral and threshold values. Specifying these parameters is optional. The following table presents all elements that can be contained in a file, depending on whether it is for a Gradient or RF pulse.

If a pulse name is not specified, the file name becomes the pulse name as default. If the pulse file contains no integral data, the values are calculated after the data record is opened in the program.

The keywords listed below must be entered in the exact sequence specified. They must be written in upper case (case sensitive). Every entry following a semicolon is interpreted as comments. If you do not specify integral values, they are automatically calculated by the program.

LIST OF PULSE-SPECIFIC PARAMETERS IN THE ASCII FILE

Parameter	Description	RF	Grad
PULSENAME	Pulse name	Yes	Yes
COMMENT	Pulse description (optional)	Yes	Yes
REFGRAD	Value for the logical reference gradient of the pulse	Yes	No
MINSLICE	Minimum slice thickness for this pulse	Yes	No
MAXSLICE	Maximum slice thickness for this pulse	Yes	No
AMPINT	Amplitude integral value for the pulse	Yes	No
POWERINT	Power integral value for the pulse	Yes	No
ABSINT	Magnitude integral value for the pulse	Yes	No

NOTE: When storing a RF pulse in an ASCII file, the file name is created from the family and pulse name as default (e.g., family.pulse). For this reason, do not use periods in the family or pulse names.

VALUES

The pulse envelope is transferred as a floating decimal number. The sampling points are stored in two (magnitude and phase) columns, separated by blank spaces or tabs. When the phase is read, the magnitude value is scaled to 1.

When storing a pulse in an ASCII file, the sampling points are numbered in sequence; the index is written in each line behind the value as a comment.

SAMPLING POINT ENTRIES IN THE ASCII FILE

Entries	Description	Example
Floating point decimal values as ASCII character strings	The values are 32-bit floating point decimal numbers. This enables you to display a range between 3.4e-38 to 3.4e+38 .	0.000000 0.00430200 0.00706700 etc.

NOTE: In the RF data records, the magnitude and phase values are listed in alternating series (2 columns).

EXAMPLE PULSE (FOR AN RF FILE)	#PULSENAMESPACE:	E2560A90.SE90_10A2_1	Pulse name
	#COMMENT:	This is a test file	Comment
	#REFERENCEGRAD:	1.000000000	Logical reference Gradient
	#MAXSLICETHICKNESS:	1.000000000	Maximum slice thickness
	#MINSLICETHICKNESS:	1.000000000	Minimum slice thickness
	#AMPLITUDEINTEGRAL:	142.817993164	Amplitude integral
	#POWERINTEGRAL:	121.844820000	Power integral
	#ABSOLUTEINTEGRAL:	142.817993164	Magnitude integral
	0.000000000	3.141592979	1 st sampling point (magnitude value and phase)
	0.004302000	3.141592979	2 nd sampling point (magnitude value and phase)
	0.007067000	3.141592979	3 rd sampling point (magnitude value and phase)
	0.009828000	0.000000	4 th sampling point (magnitude value and phase)
	0.01258200	0.000000	5 th sampling point (magnitude value and phase)
	0.01532700	0.000000	6 th sampling point (magnitude value and phase)
	Etc.		

DLL Interface

A transform algorithm is required to display the frequency diagrams for RF pulses. To provide flexibility in selecting the required algorithm, the algorithm has been implemented in a dynamic link library (.dll) file. Using the Open dialog box, the path to the desired .dll file is specified. By default, this is \n4\x86\delivery\bin\PulseToolFFT(d).dll.

The FFT-DLL is implemented explicitly; this means you may select another .dll file at program run time and thereby select a different algorithm.

The following function is provided by the run time library.

CALCULATING THE FFT SIGNAL DATA

```
void CalcFTTSamples ( double* fRealPart, double* fImagPart, LONG lSamples )
```

The function is activated from the CPULSE class and DLL management is transferred to the document class of the application.

INTERACTION BETWEEN PULSETOOL AND THE FFT-DLL

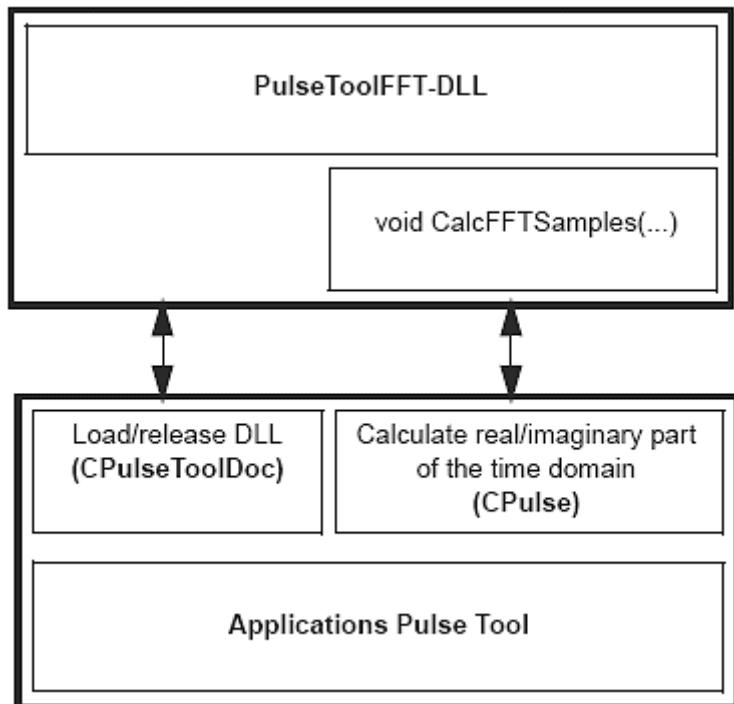
As shown in the following figure, a range of actions is performed using the classes CpulseToolDoc and CPulse. These actions are described as follows:

Load and release FFT-DLL

*Loads the simulator DLL in the address space of the visualizer and determines the address of the exported function **CalcFTTSamples**. When the program is ended, the FFT-DLL is released and deleted from the address space.*

Determine real and imaginary parts of the frequency domain (CalcFTTSamples).

Using this method, the real and imaginary parts of the frequency domain are calculated by means of a Fourier transform. In addition, the real and imaginary parts of the time diagram are provided.



D.5 Working on a MAGNETOM

Since VB13A, operation of IDEA on the MAGNETOM host is not supported anymore.

- The following features are available on the MAGNETOM:*
- Use the LogViewer (page 133) to see all traces written on the host and on the MARS.*
- The raw-data handling package (RHP) (page 134).*
- Dump of image header information (page 144).*

Please bear in mind the following:



WARNING

Please work very carefully on the MAGNETOM host. It is quite easy to corrupt the NUMARIS software by accidentally deleting some files, by modifying the registry file, by installing additional software, by modifying system files, etc. You are fully responsible for the consequences resulting from such unauthorized actions. Such steps will void CE certification and the scanner must not be used for clinical examinations prior to a reinstallation of NUMARIS.

syngo MR Directory Layout

Here you can find a short description of the directory tree of your MAGNETOM system.

On the host computer, there are the following drives:

Drive	Contents
C	Windows, NUMARIS
D	Data base files with header information
E	Data base files with image pixel information
I	Image Cache
R	Local CD ROM
S	Local CD writer
T	Local user directory
U	Local user directory

The following list of directories on drive C:\ contains files that may be of interest to sequence developers:

\MedCom	This is the main syngo MR directory where both the more general <i>syngo</i> and the MR-specific <i>Numaris</i> parts of the <i>syngo</i> MR software are located.
\MedCom\bin	This subdirectory contains the <i>syngo</i> MR executable files (*.exe and *.dll).
\MedCom\config	This subdirectory contains the <i>syngo</i> MR configuration files. These files should not be modified.
\MedCom\log	This subdirectory contains the log files of <i>syngo</i> MR. Each process has its own log file. When the system is rebooted, some log files named 'filename.log' will be copied to a backup file named 'filename.1.log' which might have been copied to a backup-backup file named 'filename.2.log'.

For the IDEA user, the following log files might be of special interest:

File name	Information about
MrUIBackends_container.log	The MRI protocol access service. It also contains the sequence output during prepFor BinarySearch in the UI context and protocol dumps in case a parameter change is not possible.
MrMeas_container.log	Measurement processes on the MARS. Sequence output as processed on the MARS.
MrPerser_container.log	Periphery server output, i.e. communication between host and the other units.
UTraceSrv.utr	The main default log file; opened by default by logviewer.

\MedCom\	This subdirectory contains files which have been created or modified by the user.
MriCustomer	

\MedCom\	Starting with VE11A this file contains the examination database with all regions, examinations, programs and protocols for all User trees. These protocols can be selected on the Program Card of the UI. They can be viewed or modified in the 'USER' section of the DotCockpit.
MriCustomer\	
Customer.exedb1	
\MedCom\	In pre VE11A versions this subdirectory contains the root of the examination database with all regions, examinations, programs and protocols. These protocols can be selected on the Program Card of the UI. They can be viewed or modified in the 'USER' section of the Exam Explorer.
MriCustomer\	
examdb	



WARNING

Do not access any subdirectory of
 \MedCom\MriCustomer\examdb or
 \Medcom\MriProduct\examdb with MRease (Program card or

Exam Explorer) and Windows Explorer simultaneously.

This will cause the protocol server to hang.

This warning is also true if the Windows Explorer "only" displays such a directory and is idle or iconified.

\MedCom
MriCustomer
seq

This subdirectory contains the sequence binary files written by the IDEA user. This directory contains both the x86 and MARS version(s) for each sequence.

\MedCom
MriProduct

This subdirectory contains files related to the MR NUMARIS system as delivered by SIEMENS.

\MedCom
MriProduct
AdjustSeq

This subdirectory contains the sequence binary files used for automated system adjustments.

\MedCom
MriProduct\Coils

This subdirectory contains the coil files, which define the interface between the coil hardware and the software.

\MedCom
MriProduct
Siemens.exedb1

Starting with VE11A this file contains the full Siemens reference examination database with all regions, examinations, programs and protocols. The contained protocols can only be selected in the 'SIEMENS' section of the DotCockpit.

\MedCom
MriProduct
examdb

In pre VE11A versions this subdirectory contains the root of the Siemens reference examination database with all regions, examinations, programs and protocols. These protocols can only be selected in the 'SIEMENS' section of the Exam Explorer. When installing the MRease software, they are copied to the USER section in their entirety (cf. directory \MedCom\MriCustomer\examdb).

\MedCom
MriProduct
MriServiceRawData

This subdirectory is the data directory of the raw data handling package (RHP, page 134) where the exported raw data files are saved.

\MedCom
MriProduct
MriServiceSeq

This subdirectory contains the service sequence binary files used by service for quality assurance.

\MedCom
MriProduct\seq

This subdirectory contains the released SIEMENS imaging sequence binary files. This directory contains both the x86 and MARS version(s) for each sequence.

\MedCom
MriSiteData

This subdirectory contains site-specific information about the configuration of the local system.

This subdirectory contains the so-called meas sections, i.e. a set of memory-mapped files which describe the hardware configuration and setup of the scanner. Never copy files to or from this directory; they are highly site-dependent and should never be exchanged!

Sequence execution (MARS)

As mentioned in C.1 , sequences have to be compiled for two systems: Windows (on the Host) and Linux (on the MARS): The version on the Host is executed during UI interactions, while MARS is responsible for actually playing out the sequence for measurements.

The real time capabilities required for the sequence execution under Linux are provided by the Xenomai framework; this has the consequence that in general two execution domains exist:

- Linux (non RT) domain: scheduled by Linux scheduler
- Xenomai (RT) domain: scheduled by Xenomai realtime scheduler

A thread is automatically transferred between the domains under the following conditions:

- Calling a Xenomai API transfers thread from Linux to Xenomai domain.
- Calling a non RT compatible API transfers the thread into the Linux domain. This triggers a so called SigXCPU signal.

A Linux thread can only be scheduled when there is no Xenomai thread ready to start.

Note that the sequence run() method is executed as a Xenomai RT thread in the Xenomai domain. While the sequence is running this thread is automatically monitored for SigXCPU signals. When the sequence executes a HALT (e.g. breath hold, etc.), monitoring is suspended, when it is (re)started, the monitoring is resumed.

When a SigXCPU is detected, a warning trace is issued, which includes a stack backtrace to support analysis. This itself is not an error, but most probably errors will result, because the real time performance cannot be ensured once the sequence run thread is in the Linux domain; it might get transferred back to the RT domain in time with the next Xenomai call – or it might not.

Applications should/must avoid calling non-RT compatible APIs during RT execution. There is no official list, but dynamic memory allocation and file I/O are non-RT compatible. Especially STL and std::string must be avoided!!

Additional Tools

LogViewer

SYNOPSIS: LogViewer

DESCRIPTION: All trace output created by the User Interface or the Measurement system is collected in a single file (UTraceSrv.utr), which can be viewed with the LogViewer tool. This is a graphical tool where you can sort the traces by time, process ID, text, etc. Its content is updated continuously.

For more usage information see Logviewer Help -> About Logviewer... (F1)

MeasID	Date/Time	Level	Imp	Component	File/Line	Function	ProcName	Text
2611 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RFCELK2250_D\~n4\~pk\~MrSe				SeqTestFrame	e RFCELK2250_DSP_F	
2612 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RFCELK2250_D\~n4\~pk\~MrSe				SeqTestFrame	e RFCELK2250_DSP_F	
2613 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CPANK2303_FL	\~n4\~pk\~MrSe			SeqTestFrame	e CPANK2303_FL_HEI	
2614 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CPANK2303_FL	\~n4\~pk\~MrSe			SeqTestFrame	e CPANK2303_FL_HEI	
2615 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CPANK2303_FR	\~n4\~pk\~MrSe			SeqTestFrame	e CPANK2303_FR_HEI	
2616 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CPANK2303_FR	\~n4\~pk\~MrSe			SeqTestFrame	e CPANK2303_FR_HEI	
2617 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice MPSK2276	\~n4\~pk\~MrSe			SeqTestFrame	e MPSK2276_HEI_c1	
2618 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice MPSK2276	\~n4\~pk\~MrSe			SeqTestFrame	e MPSK2276_HEI_tex	
2619 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CAN24K2304_1\~n4\~pk\~MrSe				SeqTestFrame	e CAN24K2304_HEI	
2620 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice CAN24K2304_1\~n4\~pk\~MrSe				SeqTestFrame	e CAN24K2304_HEI_t	
2621 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RxK2304_1	\~n4\~pk\~MrSe			SeqTestFrame	e RxK2304_1_HEI_c	
2622 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RxK2304_1	\~n4\~pk\~MrSe			SeqTestFrame	e RxK2304_1_HEI_te	
2623 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RxK2304_2	\~n4\~pk\~MrSe			SeqTestFrame	e RxK2304_2_HEI_c	
2624 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RxK2304_2	\~n4\~pk\~MrSe			SeqTestFrame	e RxK2304_2_HEI_te	
2625 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice TxK2304	\~n4\~pk\~MrSe			SeqTestFrame	e TxK2304_HEI_clic	
2626 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice TxK2304	\~n4\~pk\~MrSe			SeqTestFrame	e TxK2304_HEI_ters	
2627 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RFFAK2304	\~n4\~pk\~MrSe			SeqTestFrame	e RFFAK2304_HEI_c	
2628 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice RFFAK2304	\~n4\~pk\~MrSe			SeqTestFrame	e RFFAK2304_HEI_te	
2629 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice GFAK2309	\~n4\~pk\~MrSe			SeqTestFrame	e GFAK2309_HEI_c1	
2630 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice GFAK2309	\~n4\~pk\~MrSe			SeqTestFrame	e GFAK2309_HEI_ters	
2631 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice SHMK2259	\~n4\~pk\~MrSe			SeqTestFrame	e SHMK2259_HEI_c	
2632 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice SHMK2259	\~n4\~pk\~MrSe			SeqTestFrame	e SHMK2259_HEI_te	
2633 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice PMUK2301	\~n4\~pk\~MrSe			SeqTestFrame	e PMUK2301_HEI_c1	
2634 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice PMUK2301	\~n4\~pk\~MrSe			SeqTestFrame	e PMUK2301_HEI_ters	
2635 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice ACSK2300	\~n4\~pk\~MrSe			SeqTestFrame	e ACSK2300_HEI_c1	
2636 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice ACSK2300	\~n4\~pk\~MrSe			SeqTestFrame	e ACSK2300_HEI_ters	
2637 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice IFFK2300	\~n4\~pk\~MrSe			SeqTestFrame	e IFFK2300_HEI_c1	
2638 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice IFFK2300	\~n4\~pk\~MrSe			SeqTestFrame	e IFFK2300_HEI_ters	
2639 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerDevice PTABK2303	\~n4\~pk\~MrSe			SeqTestFrame	e Stop_PTAB_Switch	
2640 -1	2010-04-13 10:4' ALWAYS	EH<MrPerCon PerComIterator_0	\~n4\~pk\~MrSe			SeqTestFrame	e 000 closing Per	
2641 -1	2010-04-13 10:4' ALWAYS	EH<MrMeasSrv MeasUtil Always	\~n4\~pk\~MrSe	MeasComponent		SeqTestFrame	e shutdown of Meas	
2642 -1	2010-04-13 10:4' ALWAYS	EH<MrMeasSrv MeasUtil Always	\~n4\~pk\~MrSe	MrActorImpl		SeqTestFrame	e ERROR: ACE_Thres	
2643 -1	2010-04-13 10:4' ALWAYS	EH<MrMeasSrv MeasSections Always	\~n4\~pk\~MrSe	AMCstopCtrlIn		SeqTestFrame	e AMCStopCtrlImpl	
2644 -1	2010-04-13 10:4' ALWAYS	EH<MrMeasSrv MeasSections Always	\~n4\~pk\~MrSe	AccMeasPerm		SeqTestFrame	e Unmap MeasPerm!	

Dump Protocols of the Exam DB

Since VA25A, the Exam Explorer offers the functionality to dump protocols into a pdf file. With VE11A the DotCockpit fully replaced that functionality (Right mouse click context menu for a sequence contains "print"). Therefore the 'protlist' tool, which had been provided in earlier IDEA versions, has been discontinued since.

UTraceConfigUI

SYNOPSIS:

UTraceConfigUI [-ice]<pathToTraceConfigXml>

DESCRIPTION:

The tool UTraceConfigUI allows the user to create his own tracing configuration file, in order to specify, which traces should be shown (e.g. debug information for a certain component should only be displayed when required). One can edit tracing settings as well as setting the tracing level of each component (for details on level and component specification, please refer to the tracing section in C.4 , or see below). It is important to mention, that all changes in the tracing become effective immediately, e.g. intensive debugging information can be activated when the critical scenario is reached.

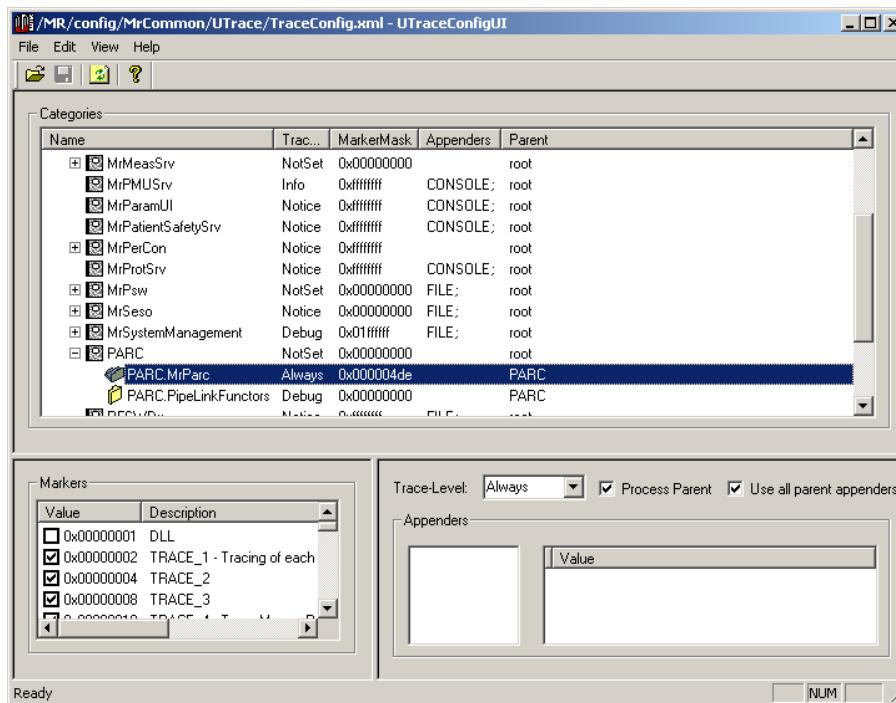
There are two configuration files that can be edited; default is the sequence trace configuration. To change the ICE specific trace configuration use option -ice.

OPTIONS:

No options	will open standard tracing configuration file (SEQ)
-ice	will open sequence tracing configuration file

EXAMPLE:

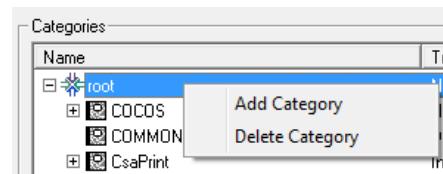
UTraceConfigUI -ice



In general, traces are identified by a category, consisting of a component and subcomponent, e.g.

MrImaging.FLASH for the FLASH sequence (for details please refer to the description on tracing in C.4). If there is no specific entry in the trace configuration, the settings of the next higher level are taken, as a last resort the root node configuration. In the following a description is given, how to create a component specific entry and adapt to the needs.

For adding a new entry for a category, please right click on the root node and select "Add Category" from the context menu:



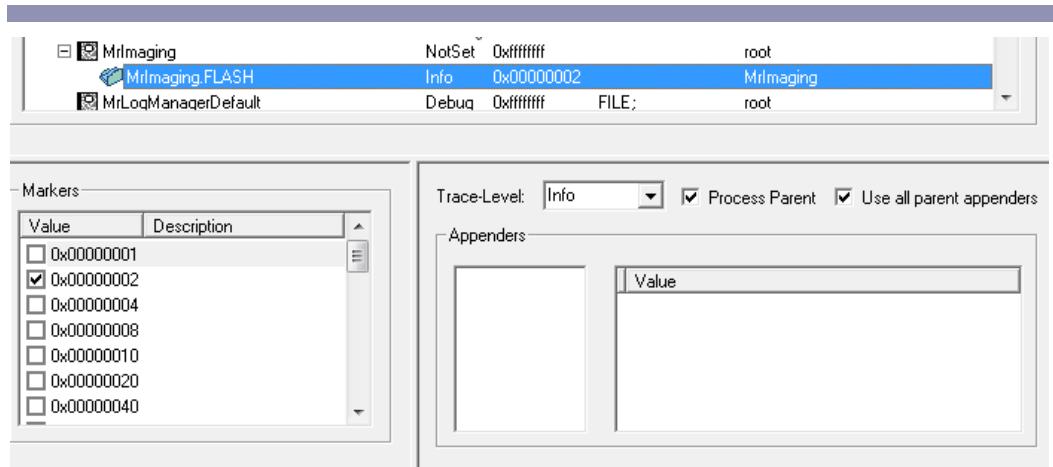
An entry with default name is created; for changing the name, please select the freshly created line and click once more, after a short pause:



Proceed similarly for adding the subcomponent name, e.g. for FLASH:

Name	Trac...	MarkerMask	Appenders	Parent
MrImaging	NotSet	0x00000000		root
MrImaging.FLASH	NotSet	0x00000000		MrImaging

A trace is only effective, if in the matching configuration the associated marker is set as active, along with the trace level. Additionally a so-called appender has to be specified, which processes the trace (e.g. writes it to a log file, etc.); in general it is recommended to use the appenders defined in the parent node, by selecting "Use all parent appenders". For configuring these details, respective sections are given in the lower pane of the UTraceConfigUI window (the example shows how to activate the marker with the second bit, trace level "Info" and above, using all appenders, defined by the parent):



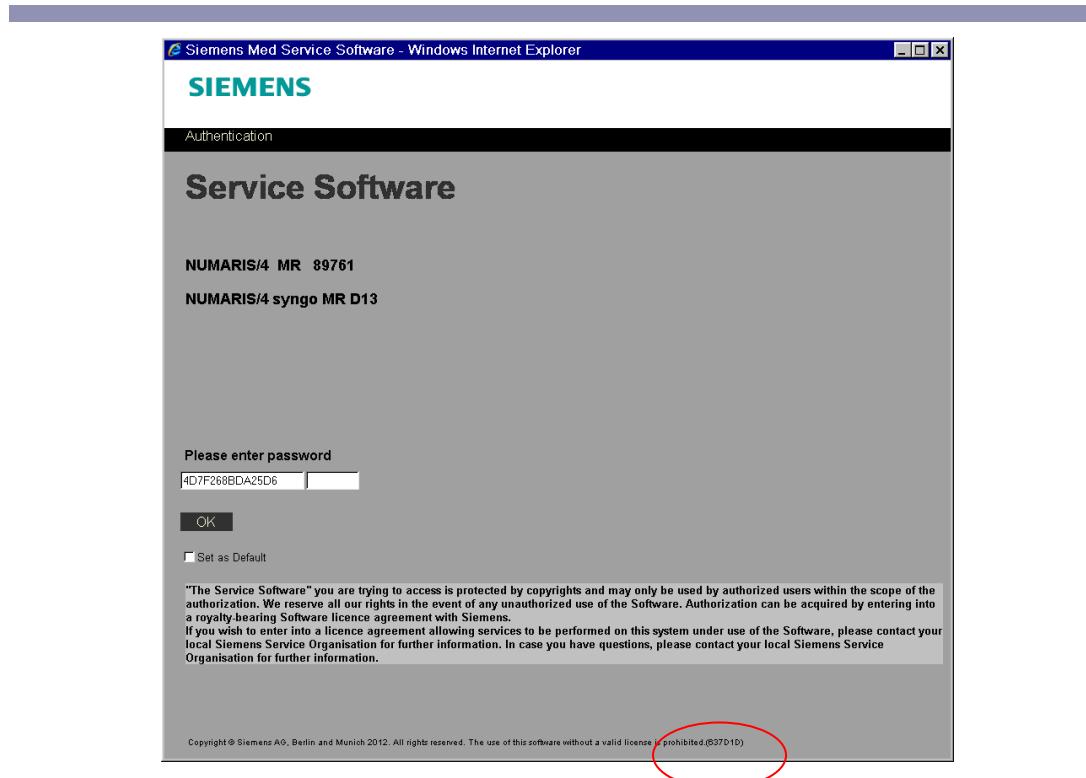
Raw Data

Raw data for imaging scans (non-spectroscopy) is stored on the MARS prior to image reconstruction. It is available for viewing or analysis only if it is copied from the MARS to the Host computer. This can be accomplished by setting the RawDataOn flag within the raw data Handling Package (RHP).

NOTE: The extraction of raw data with RHP will in general work correctly only, if

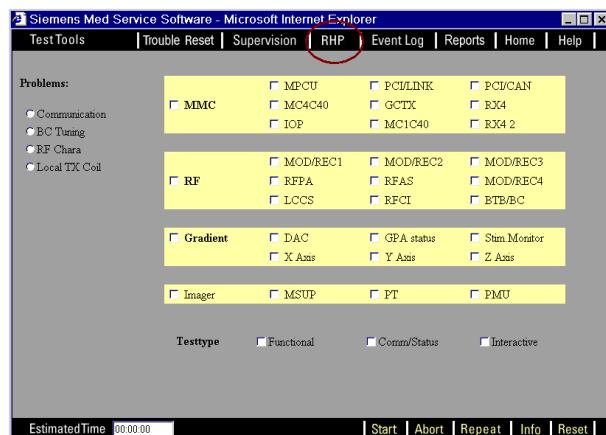
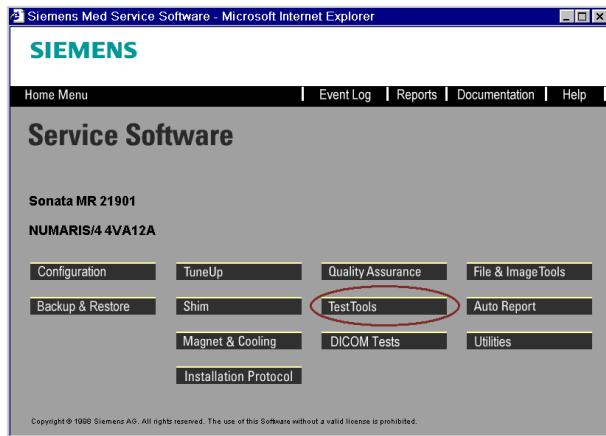
- an Office-Program is used (usually, this is the case, if a cross appears in front of the scan time display in the protocol editor).*
- Furthermore, in the measured protocol, you should switch off interpolation and filters (like raw data filter).*

The RHP (Raw data Handling Package) can be found on the service platform. To open it, select "Options" -> "Service" -> "Local Service" from the MRease main menu bar. The service tool login platform will come up:



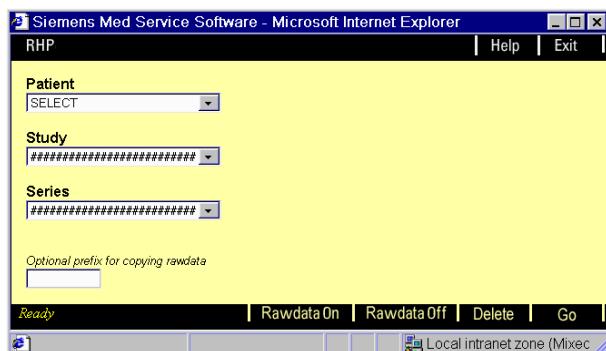
To log in, enter the password, which you can find in the line at the bottom on the window and click the **OK** button. The password is a 6 character alphanumerical code. This will enter the service software under the privileges of service level 1, which is sufficient for using the RHP.

Now the main service platform should come up:



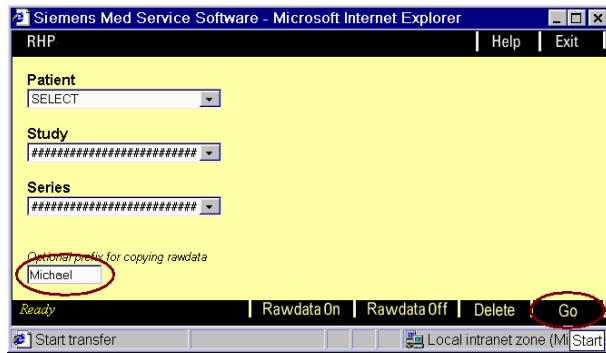
Click on **Test Tools**, and the Test Tool Platform will come up:

❖ Click now on **RHP** to open the RHP platform:



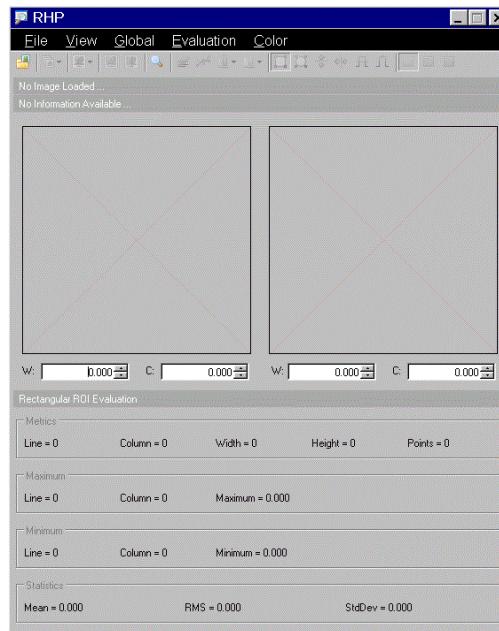
At this menu, select **Rawdata On** to tell the system to save the raw data. Now begin the measurement(s). When scanning has finished, switch off rawdata generation by clicking on **Rawdata Off** (unless more rawdata evaluation scans are desired). The next step is to retrieve the raw data and

store it in the RHP database. It is not necessary to select a patient, study and series; the default is to use the latest measurement. To keep track of specific data sets in the RHP database, it is possible to mark files by a prefix, e.g. your name or the sequence name. Enter this prefix in the field "Optional prefix for copying rawdata" if desired:

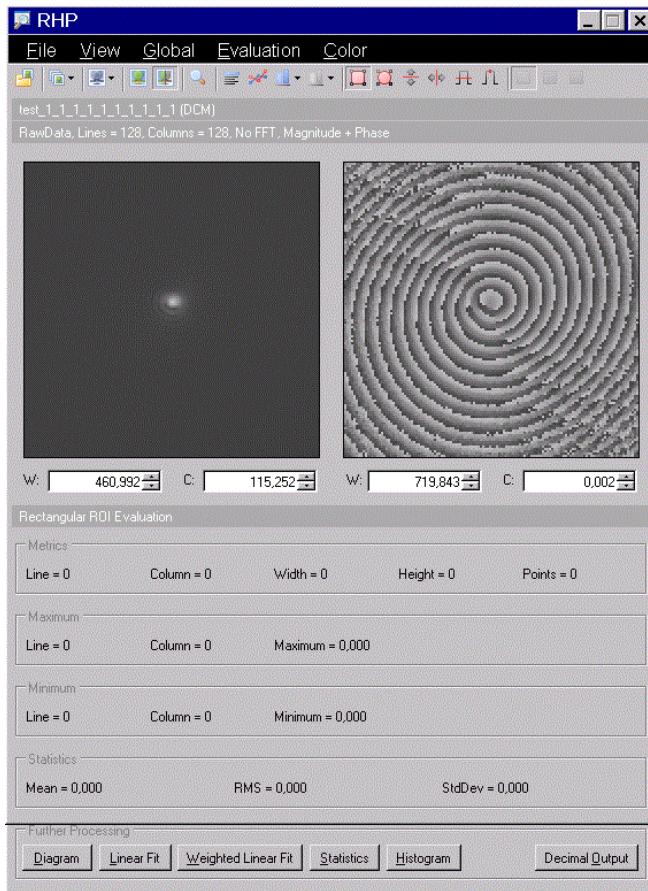


- ✧ Now click **Go** to start the RHP evaluation tool

Alternative: execute C:\MedCom\bin\MrSesRHP.exe from command prompt:



- ✧ Since several images may have been created during your measurement, select the rawdata for the specific image file. To do this, click **File, Open** to open the file explorer. This will list the directory contents of all extracted raw data. If a prefix was specified when extracting the data, it is appended to the default filename.
- ✧ Select one file to return to the main tool window:

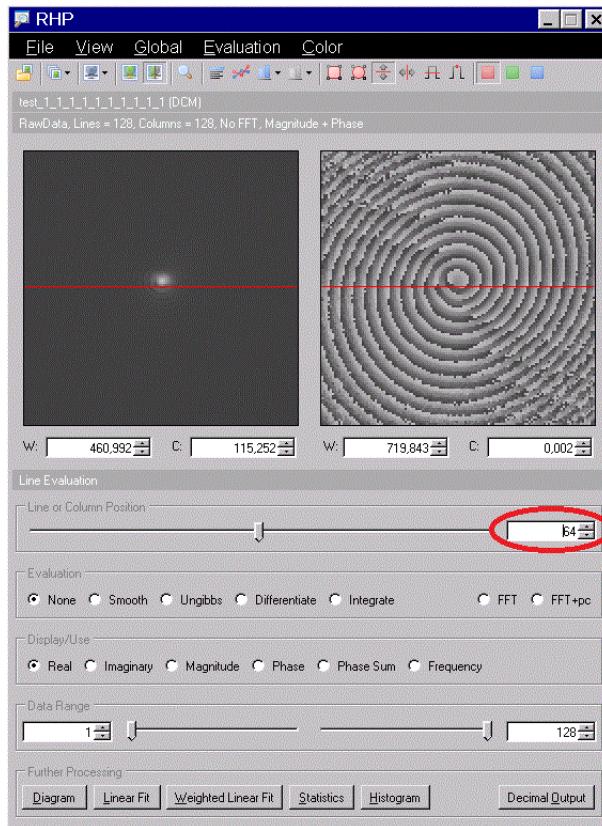


Contrast and brightness of the images can be controlled by the middle mouse button as usual. At the bottom of this window, there are some important toggles:

- Zoom +/-** will enlarge the window by a factor of 2.
- Re&Im / Magn&Phase** decides if the real and imaginary parts of the image are displayed or the magnitude and the phase. In many cases, the user wants to see the magnitude image; therefore he has to click on this button once.
- ROI y/n** enables the user to draw a ROI in the image. The geometry of the ROI is controlled by the button **ROI Rect/Oval**. The size of the ROI can be controlled by the mouse. When releasing the mouse button, statistical data of the ROI will be displayed.
- FFT y/n** switches the FFT on or off. This provides a quick impression of how the image of the transformed raw data will appear. In some cases, the FFT will refuse to work because of the image size.
- Cancel** is used to close this window.

In the Main Evaluation applet, there are more buttons that provide some statistical information as well as direct pixel information for the **global** image, a specific **line** or **column**, and the **max** or mean **values**.

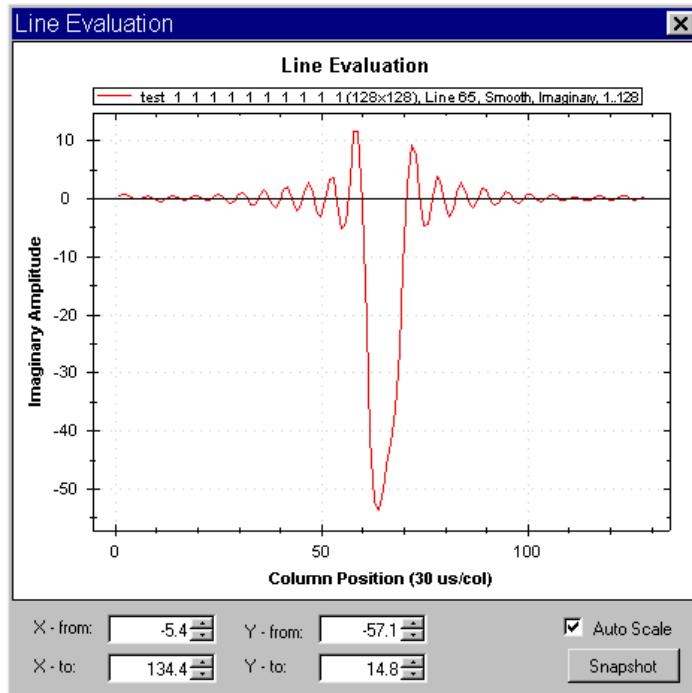
❖ As an example, clicking on the **Evaluation, Line** menu entry will open the following window:



Below the images, the particular line can be chosen for analysis.

At the bottom sub-window are some calculated statistical data. The calculations are started by the four buttons: **Linear Regression**, **Weighted Linear Fit**, **Statistics** and **Decimal Output**.

Clicking on the **Display** button at the bottom will open another window with a line profile:



- ❖ When finished, close the RHP tool by the X button or the **File, Exit** menu entry.
- ❖ Be sure to switch off saving raw data on the service RHP platform: Click "Rawdata Off".
- ❖ On the RHP platform, click the **Delete** button if you want to delete the raw data corresponding to the selected series. Then click **Exit** to close the window.

NOTE: There is an easy way to see if saving raw data is on or off. The icon in the program scheduler will change from the head to an oscilloscope if raw data is being saved to disk:



Image Header Information

Protocol in image

In general, the full protocol information is contained in every image and can be extracted easily:

- Select the image(s) in the viewer
- Choose "Transfer/Export to Off-line"
- After export to the desired directory, open the image (with extension .IMA) with Wordpad.exe
- Search for "ASCCONV", and you have the beginning of the protocol.

You can even copy the text between "ASCCONV BEGIN" and "ASCCONV END" to a new file and feed this file to POET or SeqTestFrame - it is a valid protocol.

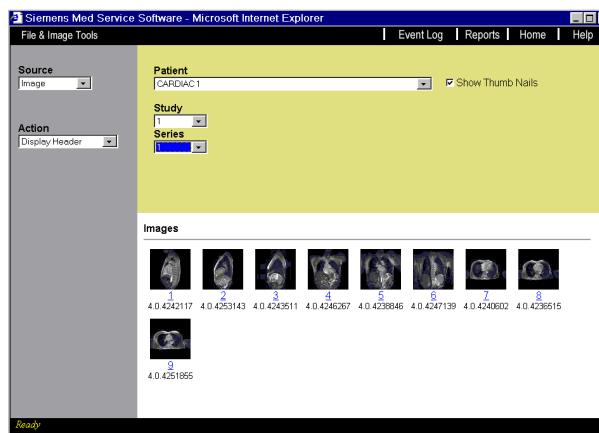
The "Phoenix"-feature allows you to easily re-run measured images (even if they have been imported to the database from a different scanner). Just drag & drop the image from the Patient browser to the measurement queue.

Hint: If the image comes from a system with different gradient properties, hold the "shift"-key while dragging the image. This will initiate a protocol conversion and adapt all necessary protocol parameters.

DICOM header

Alternatively, one can dump the DICOM header information stored in an image using this procedure, but requires the password for Service Level 5. The information is similar to the "Directory Extended" option in NUMARIS/3.

- Open Local Service Menu (as on page 134)
- Select "File and Image Tools"
- Select "Image"
- Select "Display Header"

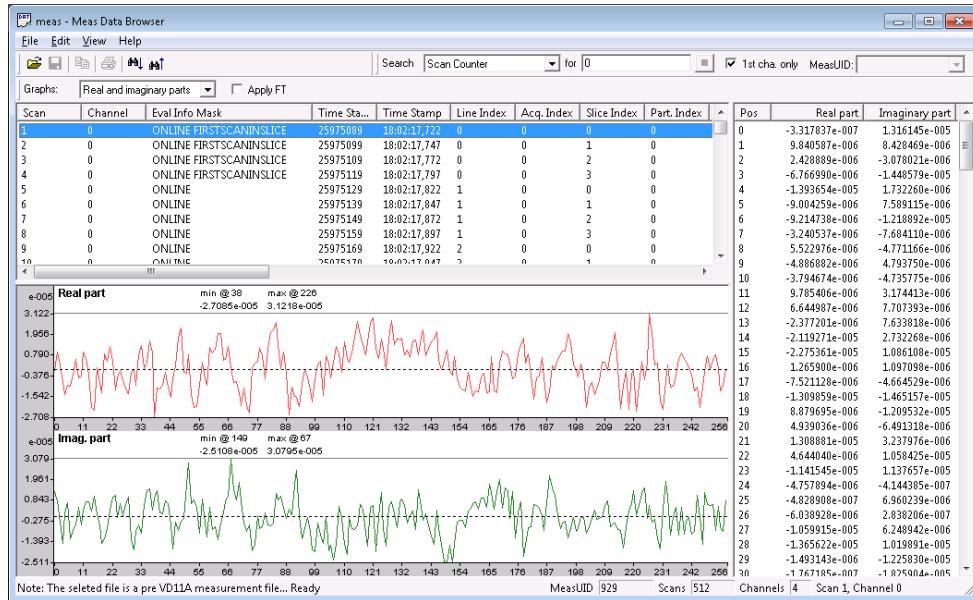


- Choose the desired Patient, Study and series
- Click on the number of the desired image
- A list with the header information appears.

Measurement Data Browser (MDB)

With the measurement data browser MDB, it is possible to look at the measured data that was stored in a meas.dat file (For information, how to create it, see either the ICE users guide).

Start it by entering "**mdb meas.dat**" in a command prompt.

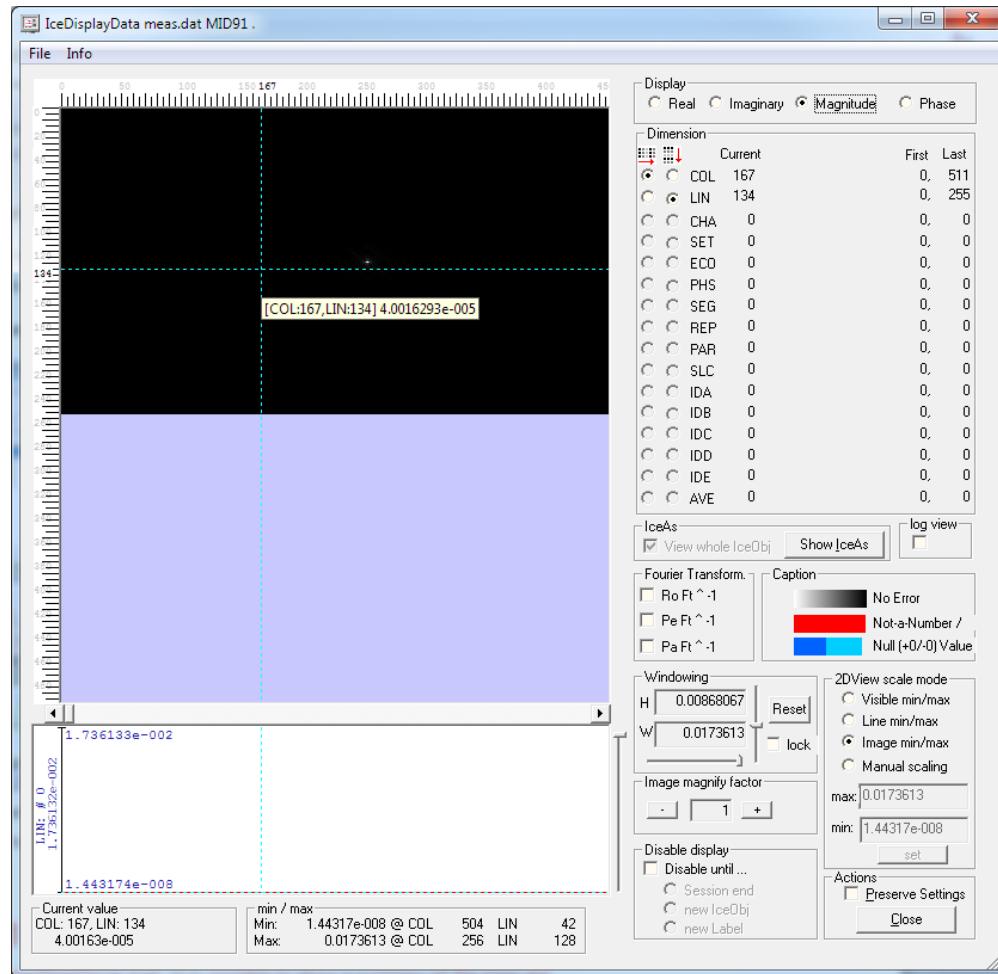


Within the platform, every received data is shown, sorted by the scan counter. You can step from one scan to the next and look, which MDH flags were attached to each ADC, which data is received by which channel, and so forth.

Meas Data Viewer

With the meas data viewer (MeasDataViewer.exe), it is possible to show a summary of the meas.dat contents and in the GUI to navigate the data.

Start it by entering "**measdataviewer meas.dat**" in a command prompt.



E Programming Manual

E.1 The Sequence in NUMARIS/4

Critical frequency range

Gradient activities are often cyclic and create a specific acoustic spectrum. If this spectrum falls into a resonance frequency of the specific system type, it can cause severe damage to the scanner's gradient system hardware.

The critical parameter is an echo-spacing where the frequency of the excited vibrations is close to the gradient coil resonance frequency. Under such circumstances, high vibration amplitudes can cause fatigue fractures to the heavy duty connectors of the coil.

Standard Siemens sequences do not use echo-spacing in this frequency range. However, using the IDEA licence it is possible to develop sequences that may accidentally hit a critical frequency range.

In order to prevent damage to the gradient system it is strongly recommended to include a check in SeqIF::prepare to make sure to avoid parameters leading to echo-spacings corresponding to a critical frequency range.

I.e. if one of the critical frequency ranges for the given system is **1120Hz +/- 110Hz** (i.e. frequency +/- 0.5*bandwidth), the critical ranges of echo-spacing values would be between 0,40-0,50ms for bipolar readout schemes (like EPI) and 0,81-0,99ms for monopolar readout schemes (like trufi).

To avoid these problems a solution is to check for EchoSpacing and/or TRvalue, etc. in SeqIF::prepare - depending on the sequence type - and if they lie in the critical range, return SEQ_ERROR.

There are two options: either do the check with every call of fSEQPrep, or only if the statement pSeqLim->isContextNormal() is true. The first option may result in complex UI behaviour; the second will give you the error message only after hitting "Apply".

Note that there can be more than one critical frequency range and that the ranges are system type specific; it is your responsibility to check all ranges relevant for the system type the sequence is running on.

You can get the acoustic resonancbands using the libSeqSysProp interface (the critical frequency range can be calculated as frequency +/- 0.5*bandwidth):

```
// range for index = 0...AcousticResonanceEntryNumber-1
int32_t getAcousticResonanceEntryNumber() const;

// Acoustic resonance frequency to be avoided
float getAcousticResonanceFrequency(int index) const;

// Bandwidth of acoustic resonance to be avoided
```

```
float getAcousticResonanceBandwidth (int index) const;  
  
// Which sequences are affected by this acoustic resonance  
bool isAcousticResonanceRelevant (int32_t lIndex, SEQ::SequenceType eSeqType);
```

Sequences in NUMARIS/4

This chapter gives a short overview how the sequence interacts with the MR system and which functionality must be provided to the MR system with your sequence. Subsequent chapters will treat the single topics in more detail.

A sequence is a shared library (DLL) that is executed on the host computer used by the operator and on the computer that controls the real time measurement (MARS). Since two different operating systems are involved, two targets have to be compiled.

In order to guarantee a certain set of entry points, sequences in NUMARIS/4 have to implement an interface that is declared in the class *SeqIF*. This class contains four pure virtual functions (*initialize*, *prepare*, *check* and *run*) that are mandatory to implement and a set of virtual function that may be overloaded on demand.

Note that a derived class, *StdSeqIF* is defined in *libSBB*, which add some functionality and should be used instead of *SeqIF* directly.

Mandatory Entry Points of the Sequence

The following chapter gives a short description of the mandatory function:

```
SeqIF::initialize virtual NLS_STATUS SeqIF::initialize (SeqLim *pSeqLim) = 0;
```

Sequence initialization:

- Specify the set of UI parameters used by the sequence and their hard limits (minimum and maximum values).
- Specify the default values of the UI parameters that are used for the default protocol of the sequence.
- Specify the MR system for which the sequence is intended for use.
- Register functions that overload UI functionality for certain parameters.

```
SeqIF::prepare virtual NLS_STATUS SeqIF::prepare (
    MrProtocolData::MrProtData *pMrProt,
    SeqLim *pSeqLim,
    MrProtocolData::SeqExpo *pSeqExpo ) = 0;
```

The *prepare* function of the sequence is used to

- prepare all components of the sequence, such as gradients, rf pulses, reordering tables and loop structures ...
- determine if the current protocol is consistent, i.e. if the measurement can be executed with the current protocol. It returns an error code (NLS_STATUS) that indicates whether the protocol is consistent or not.

```
SeqIF::check virtual NLS_STATUS SeqIF::check (
    MrProtocolData::MrProtData *pMrProt,
    SeqLim *pSeqLim,
    MrProtocolData::SeqExpo *pSeqExpo,
    SEQCheckMode *pCheckMode ) = 0;
```

The *check* function is called on the host computer after the measurement has been started. It is used to

- support the gradient overflow check

- and to check whether there will be a peripheral nerve stimulation of the patient (GSWD look ahead calculation).

```
SeqIF::run    virtual NLS_STATUS SeqIF::run (
    MrProtocolData::MrProtData *pMrProt,
    SeqLim             *pSeqLim,
    MrProtocolData::SeqExpo *pSeqExpo ) = 0;
```

The *run* function is used to

- execute the measurement, i.e. deliver the sequence timing
- and to invoke the sequence unit test.

Optional entry points of the sequence library

If the sequence library is able to exchange data with processes other than the standard ones (UI and measurement system libraries), the sequence library must provide one or two additional entry points. In general, the sequence libraries do not supply those entry points if they are not required.

```
SeqIF::receive virtual NLSStatus SeqIF::receive (
    SeqLim *           pSeqLim,
    MrProtocolData::SeqExpo * pSeqExpo,
    SEQData &          rSEQData );
```

SeqIF::receive is an optional entry point of the sequence code (in contrast to the mandatory functions *SeqIF::prepare*, *SeqIF::run*, etc.). It is called by the sequence framework in order to retrieve data that were sent from another software component, e.g. the ICE program.

```
SeqIF::deliver virtual NLSStatus SeqIF::deliver (
    SeqLim *           pSeqLim,
    MrProtocolData::SeqExpo * pSeqExpo,
    SEQData &          rSEQData );
```

SeqIF::deliver is an optional entry point that is called by the framework in order to retrieve data from the sequence to another software component.

```
SeqIF::convProt

    virtual NLSStatus SeqIF::convProt(
        const MrProtocolData::MrProtData& rMrProtSrc,
        MrProtocolData::MrProtData&           rMrProtDst
    );

```

Is used to adapt a protocol setting from one software version to another during protocol update (after import of older protocols).

NLS_STATUS Note that functions have the return type *NLS_STATUS*. This status indicates whether the function has successfully been executed or not. There is a set of predefined status values; arbitrary codes are not allowed. Successful execution of a sequence function is indicated by returning *SEQU_NORMAL*. Errors can be indicated using an NLS status such as *SEQU_ERROR* or *SEQU_NEGATIV_TEFILL*.

Example

To implement the entry point as C++ member functions one has to define a sequence class, here called *MySeq*, that is derived from the interface class *SqIF*. *MySeq* has to define the pure virtual functions of *SqIF*.

SEQIF_DEFINE **IMPORTANT:** Please note that the macro *SEQIF_DEFINE*(*MySeq*) has to be defined within the sequence. This necessary to create an instance of the sequence, i.e it will be impossible to use the sequence if this macro is missing.

```
// Include of interface class definition
#include "MrServers/MrMeasSrv/SqIF/Sequence/SqIF.h"

SEQIF_DEFINE (MySeq)

// Example of a C++ sequence declaration
class MySeq : public SqIF
{
    // mandatory entry points
    virtual NLSStatus initialize(SeqLim*) ;

    virtual NLSStatus prepare   (
        MrProtocolData::MrProtData*,
        SeqLim*,
        MrProtocolData::SeqExpo *);

    virtual NLSStatus check   (
        MrProtocolData::MrProtData*,
        SeqLim*,
        MrProtocolData::SeqExpo *,

```

```

SEQCheckMode*) ;

virtual NLSStatus run (
    MrProtocolData::MrProtData*,
    SeqLim*,
    MrProtocolData::SeqExpo *) ;

// optional entry points
virtual NLSStatus receive (
    SeqLim*,
    MrProtocolData::SeqExpo *,
    SEQData&) ;

virtual NLSStatus deliver (
    SeqLim*,
    MrProtocolData::SeqExpo *,
    SEQData &) ;

virtual NLSStatus convProt (
    const MrProt&,
    MrProt&) ;
};

```

SeqIF::initialize in Detail

Hard Limits

The primary purpose of `SeqIF::initialize` is to specify the hard limits for all parameters the sequence will offer to the user. The user will not be able to select a value outside the range specified by the hard limits. Therefore, the hard limits should not be too restrictive. Often, these are based on hardware limitations.

Note that the sequence typically does not run with all combinations of parameter settings within the hard limits. The task of the UI software is to offer the user only those parameter settings with which the sequence can really be executed. The sequence supports this task of the UI within the `SeqIF::prepare` function.

SeqLim

Hard limits are defined using the set methods of the `SeqLim` class. The set methods also are used to define the default protocol. There are four different types of parameters:

❑ numeric values (long or double):

`pSeqLim->setXXX (Minimum, Maximum, Increment, Default);`

e.g.

```

pSeqLim->setRSats ( 0, 6, 1, 0);
pSeqLim->setFlipAngle ( 90.000, 180.000, 1.000, 180.000);

```

❑ arrays of numeric values:

```
pSeqLim->setXXX (Index, Minimum, Maximum, Increment, Default);
```

e.g.

```
pSeqLim->setTE ( 0, 1000, 100000, 100, 10000);  
pSeqLim->setTE ( 1, 1000, 100000, 100, 30000);
```

❑ selection types (boolean or combo box):

```
pSeqLim->setXXX (SEQ::AAA, SEQ:BBB, SEQ:CCC, ...);
```

the first value is the default value

e.g.

```
pSeqLim->setFatSuppression (SEQ::FAT_SUPPRESSION_OFF,  
SEQ::FAT_SATURATION,  
SEQ::WATER_EXCITATION);  
pSeqLim->set2DInterpolation (SEQ::OFF, SEQ::ON);
```

❑ arrays of selection types:

```
pSeqLim->setXXX(Index,SEQ::AAA, SEQ:BBB, SEQ:CCC, ...);
```

the first value after the index is the default value

e.g.

```
pSeqLim->setFlowCompensation (0, // index  
SEQ::FLOW_COMPENSATION_NO,  
SEQ::FLOW_COMPENSATION_READOUT_ONLY,  
SEQ::FLOW_COMPENSATION_SLICESEL_ONLY);
```

In general, parameters for which no limits are set explicitly in fSEQInit are not shown in the UI.

Hardware Restrictions

The sequence must specify the hardware platform for execution. If the hardware specifications are outside the specified range, the sequence cannot be loaded.

```
pSeqLim->setAllowedFrequency (5000000,500000000); // [Hz]  
pSeqLim->setRequiredGradAmpl (20.00); // [mT/m]  
pSeqLim->setRequiredGradSlewRate (10.00); // [mT/m/ms]
```

These methods allow the sequence to be restricted to specific systems.

Most NUMARIS/4 product sequences adapt their timing to the given system properties so that they are able to run on every system. Sequences which have a fixed timing and therefore have minimum requirements for the gradient system or which work only at specific magnetic field strengths must specify their requirements here more specifically.

Default Protocol

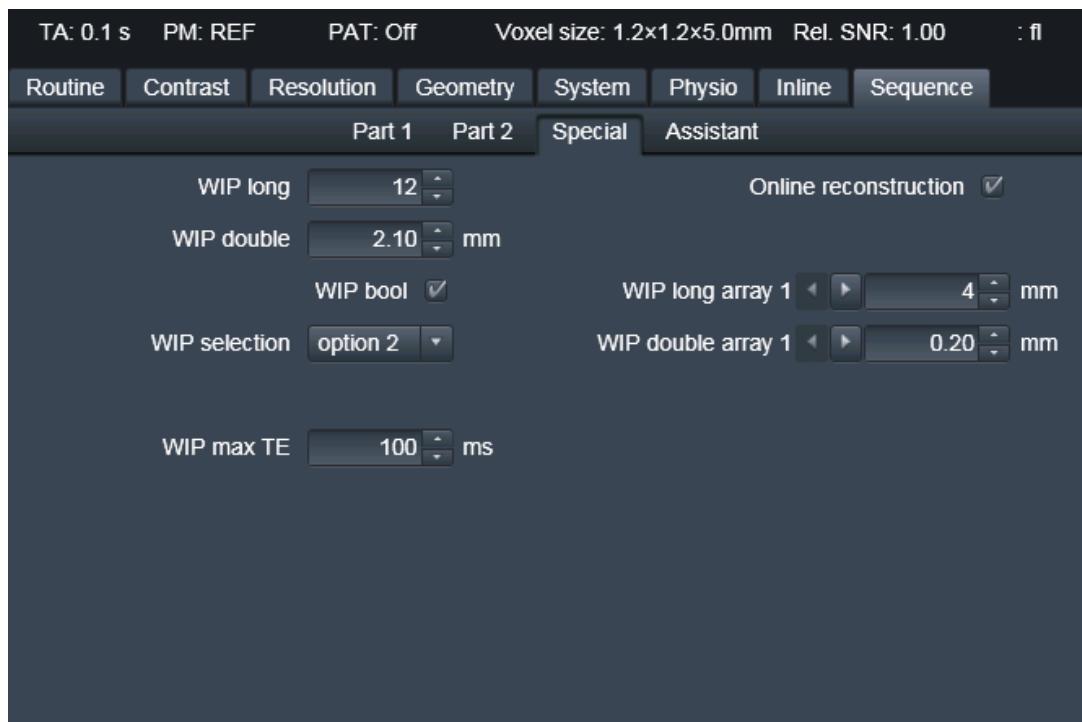
The default values specified with the set methods of SeqLim are used to generate a default protocol for the sequence. There is an important requirement to this default protocol: the sequence must be able to be executed with the default protocol on every system where it can be loaded!

It has been found - especially for the flexible sequences - that this requirement in some cases cannot be fulfilled without knowledge about the sequence timing. Since in *NUMARIS/4*, *SeqIF::initialize* has no information about the sequence the sequence must support the *prepAndUpdateProt* methods of the Sequence interface class within *SeqIF::prepare*. The *prepAndUpdateProt* method is described on page 265.

Register UI Functionality

The sequence can modify UI functionality by registration of new UI handlers. This may be necessary if the standard implementation of these handler functions does not fit the requirements of the sequence. Registration of new UI handlers has to be done in the *initialize* function of the sequence.

It is even possible to create new (WIP) parameters which will appear on the *Sequence Special* card in the UI. The UI handler functions implementing these WIP parameters have also to be registered in the *initialize* function of the sequence.



You will delve into the possibilities of UI-programming in chapter (E.4).

SeqIF::prepare in Detail

On the host, *SeqIF::prepare* is the essential function of the sequence library. It is called very frequently after the sequence is loaded into a protocol editor in order to determine whether a set of protocol parameters is valid or not (binary search). Before the measurement can be executed with a specific protocol, *SeqIF::prepare* is called at least one time on the host and exactly once on the MARS in order to prepare all components of the sequence.

Context Usage

Since *SeqIF::prepare* is called for different purposes. It is necessary to find out in which context it is called. This information can be retrieved from *SeqLim*:

❑ pSeqLim->isContextNormal()

A normal preparation of the sequence is required. This context is used when the sequence is actually being prepared for the measurement. This happens on the host after the "Apply" button is pressed and on the MARS directly before the measurement is started. Also, the gradient overflow check, the SAR look ahead and the GSWD look ahead calculations require a normal preparation.

The remainders of the context options for calling *prepare* are used in conjunction with the UI. These calls typically are used to check the consistency of a protocol with a particular set of measurement parameters. For these calls, a subset of the full calculations can be specified. These limited scope calls to *prepare* are made in three contexts:

❑ pSeqLim->isContextPrepForBinarySearch()

This context tests the consistency of the specified set of protocol parameters. It is used in the binary search to determine the soft limits in the UI. Therefore, *prepare* should be as fast as possible in this context. Since not all calculations need to be performed to decide whether or not the measurement can be executed, unnecessary steps can be bypassed to improve performance. Note that also some other processes may set this context.

❑ pSeqLim->isContextPrepForScanTimeCalculation()

In this context, *pSeqLim->isContextPrepForBinarySearch()* is also active. However, the UI tells the sequence that it should additionally perform all calculations necessary to determine the correct measurement time. *Prepare* itself should know if and how to react in this special context.

❑ pSeqLim->isContextPrepForMrProtUpdate()

In this context, *pSeqLim->isContextPrepForBinarySearch()* is also active. As mentioned above, there is an exception to the rule that the sequence must not change the protocol. When this context is set, *prepare* is explicitly asked to modify the protocol. This context is very important to help the system to solve certain problems, e.g. repair an inconsistent default protocol created by *initialize*.

Prepare Objects

Running in the context "normal" (i.e., *prepare* for execution), *prepare* should prepare all objects and calculate all variables which are needed and are constant during the measurement. As the sequence is executed in real time, the computations on the MARS must be faster than the execution of the measurement on the hardware.

Consistent Protocol

In the context "prepare for binary search", *prepare* is used to decide whether the protocol is consistent. This means that a measurement can be performed with the current protocol.

Note that none of the processes dealing with sequences will load an inconsistent protocol!

It is important that an *NLS_STATUS* code is returned indicating success if the protocol is consistent or

indicating an error if it is inconsistent. Of course, *prepare* must also return with an error code if problems occur in another context.

Once a protocol is loaded successfully into a protocol editor, the user interface must keep the protocol in a consistent state. In other words, if the protocol is modified within the UI, it should be possible at any time to start a measurement with the displayed protocol.

There are some additional checks performed directly before the sequence is executed, e.g. the gradient overflow check, the SAR look ahead and the GSWD look ahead. These checks depend on data of the current patient in the magnet and the current slice orientations. These checks may fail even if the protocol is consistent.

SAR Look Ahead

The SAR look ahead calculation needs two types of information to determine if the sequence will exceed SAR limits: the total measurement time and the total energy applied during the measurement (including all repetitions). This information must be calculated within the *prepare* function and exported using SeqExpo.

For IDEA versions before D13C the energy export is performed via

`❑rSeqExpo.setRFEnergyInSequence_Ws(dEnergy)
with dEnergy of type double`

With IDEA version D13C and later support for three B1 shimming modes is introduced:

`❑rSeqLim.setB1ShimMode (SEQ::TX_B1_SHIM_TRUEFORM, SEQ::TX_B1_SHIM_PAT_SPEC,
SEQ::TX_B1_SHIM_VOL_SEL)`

Here ‘TrueForm’ refers to the standard setting with CP/EP mode known from previous versions; ‘Patient-specific’ (PAT_SPEC) calculates the optimal complex transmit scale factors based on the full FOV; ‘Volume-selective’ calculates the optimal complex transmit scale factors for a volume specified by the user.

For a meaningful SAR prediction it is not sufficient for sequences using the SEQ::TX_B1_SHIM_PAT_SPEC and SEQ::TX_B1_SHIM_VOL_SEL modes to only provide a single global energy value. To address this issue a new class holding the global energy and local SAR information was introduced in D13C: *MrProtocolData::SeqExpoRFInfo*, defined in *MrServers/MrProtSrv/MrProt/SeqIF/SeqExpoRFBlockInfo.h*. For sequences using either of the two new pTX modes it is therefore mandatory to export the energy information via

`❑rSeqExpo.setRFInfo (rfInfo)
with rfInfo of type MrProtocolData::SeqExpoRFInfo`

Note that starting with D13C all product sequences perform the energy calculations with the new class. Functions in libSBB, etc. dealing with energy calculations have been renamed to reflect this change. Depending on the use of SBBs, etc. of your sequence porting the sequence to D13C or later will require more or less changes. The following list gives examples on how to port typical cases:

old	<code>double dEnergyAllSBBs = 0.0;</code>
new	<code>MrProtocolData::SeqExpoRFInfo rfInfoAllSBBs;</code>
old	<code>if(!m_mySBBList.prepSBBAll(rMrProt, rSeqLim, rSeqExpo, &dEnergyAllSBBs))</code>
new	<code>if(!m_mySBBList.prepSBBAll(rMrProt, rSeqLim, rSeqExpo, &rfInfoAllSBBs))</code>
old	<code>lStatus = checkChargeBalanceOfRFPA (rMrProt, rSeqLim, rSeqExpo, &dEnergyAllSBBs);</code>
new	<code>lStatus = checkChargeBalanceOfRFPA (rMrProt, rSeqLim, rSeqExpo, &rfInfoAllSBBs);</code>
old	<code>dEnergyAllSBBs *= (double) (m_REPOInfo.getNumberOfEchoTrains() * rMrProt.measurements () +3) / (double) ((m_REPOInfo.getNumberOfEchoTrains () +3) * rMrProt.measurements());</code>
new	<code>rfInfoAllSBBs *= (double) (m_REPOInfo.getNumberOfEchoTrains() * rMrProt.measurements () +3) / (double) ((m_REPOInfo.getNumberOfEchoTrains () +3) * rMrProt.measurements());</code>
old	<code>UTRACE(Info, 0, ptModule << ": dEnergyAllSBBs: " << dEnergyAllSBBs);</code>
new	<code>UTRACE(Info, 0, ptModule << """: rfInfoAllSBBs.getPulseEnergyWs(): " << rfInfoAllSBBs.getPulseEnergyWs());</code>
old	<code>rSeqExpo.setRFEnergyInSequence_Ws (dEnergyAllSBBs);</code>
new	<code>rSeqExpo.setRFInfo (rfInfoAllSBBs);</code>

From a model which takes also patient weight and reflected as well as absorbed RF power into account, the actually deposited RF power is calculated, based on the measurement time and the applied energy as exported by the sequence. This is compared to numbers that comply with legal requirements and the percentage compared to that value is displayed. If it exceeds 100%, one can either switch to SAR 1st level, where the threshold values are higher. Or you have to adapt other parameters which reduce the power deposition. The SAR safety watchdog automatically calculates parameter proposals to reduce the SAR below the limit. These values are shown in the SAR dialog and can be confirmed by the user. After the measurement is finished the values measured for SAR and those precalculated by the sequence are compared. If a deviation of more than 10% is observed an error message is displayed. This is due to the fact, that the system then assumes a hardware error and needs to be recalibrated.

During the measurement the PALI (Power Absorption Limiter) monitors the actually deposited RF power and stops the measurement if these values exceed a certain limit.

GSWD Look Ahead

If the GSWD look ahead calculation (called after the protocol has been prepared in the normal context) predicts peripheral nerve stimulations during the measurement, it uses *prepare* to find a

parameter setting that allows the execution of the sequence without stimulation. The sequence has two possible ways to support the GSWD look ahead:

- Allow measurements with larger FoV or thicker slices without changing the sequence timing (i.e. decreasing gradient amplitudes).
- Adapt timing according to a minimum gradient rise time provided by the protocol, adapt the sequence timing and - if necessary - repair the protocol because of increased TE/TR values.

During the measurement an online GSWD watchdog always compares the actual level of peripheral nerve stimulation to the limits. If the level exceeds the limits it stops the sequence.

Repair Protocol

As mentioned above, *prepare* may be invoked in order to repair an inconsistent protocol. For example this may be necessary if the default protocol provided by *initialize* is inconsistent. Then *prepare* will be called within the context "prepare for protocol update" to ask the sequence to modify the protocol in order to make it consistent. If *prepare* succeeds with repairing the protocol it must return an *NLS_STATUS* indicating success.

The complete process of repairing an inconsistent protocol is described in a separate chapter. For a detailed description, please refer to page 265.

Soft Limits

The protocol has to stay in a consistent state, which means it has to be avoided that the user can select parameter combinations that make the protocol inconsistent. It is not necessary that the sequence can be executed for every value of a parameter within the hard limit range independent of the current values of other parameters. The limits for a parameter depending on the current protocol are called "soft limits". The soft limits are always within the hard limit range.

Binary Search

The UI finds the soft limits for double and long parameters using a binary search algorithm. For selection types, all possible selections are tested.

The starting point for the binary search is a consistent protocol. Then this parameter is changed, while all other parameters remain unchanged during the search. The UI creates virtual copies of the protocol, modifies the specific parameter and checks if the protocol is still consistent. This way, it finds the maximum range the parameter can be modified between the current value and the hard limits. The binary search is performed in two directions: between the current value and the hard limit minimum and between the current value and the hard limit maximum.

Convex Parameter

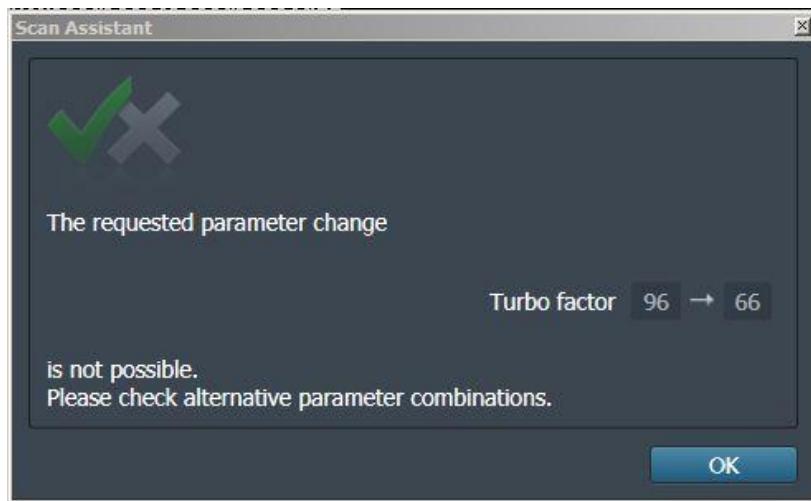
But the binary search algorithm only works for so-called "convex parameters". That means, if it is possible to execute the measurement with the values A and B for a certain parameter, it is also possible to execute the measurement for ALL values between A and B. If this is not the case for a

specific parameter, one has to use a different strategy to find all valid values.

One strategy might be to test all possible values of a parameter between the hard limit maximum and the minimum. This would work for all parameters not only for convex ones. But it would cost a lot of computing time and the performance of the UI would decrease dramatically. For some parameters, the UI indeed uses this technique to determine the soft limits, e.g. the base resolution or the turbo factor.

Internal UI Error

It is easy to recognize that there are problems with the parameter range and the binary search algorithm if a popup window like this appears while editing a protocol in the UI:



This indicates that something went wrong when the UI really tried to apply the parameter change to the protocol. The protocol became inconsistent due to the parameter change and the UI only could help itself by undoing the parameter change, because the protocol must be kept consistent.

Another reason for such a popup may be that the sequence has problems with static variables and objects and their behavior being dependent on the order of user actions during protocol editing. One reason which occurs very often in this context is that in the *prepare* a parameter is changed only in an 'if' statement and not resetted in an 'else' statement. Or the 'else' statement is completely missing.

UI Link handlers

Up to now, it was possible for the sequence to support the UI only by returning with error from *prepare*. The UI was able to support the user in creating valid measurement protocols for the sequence.

With *NUMARIS/4* a flexible sequence design was introduced for nearly all sequences. As a result complex dependencies between parameters emerged and the UI with its binary search algorithm was not able to handle the resulting problems alone.

The sequence itself now can implement UI functionality and can change the functionality of existing

parameters with so called 'handlers'.

A handler is a function with a predefined interface implemented in the sequence. The pointer to the handler must be passed to the UI software when the sequence is initialized calling *initialize*. The handler replaces the default functionality of the UI for the parameter for which it was registered by the functionality implemented in the handler.

Solve Handler

A special UILink handler is the "solve handler". It enables the user to select values for a certain parameter which require an adaptation of another parameter. Some solve handlers for common problems are already implemented in the UI software itself. For more complex parameters, *prepare* must inform the solve handler about the required parameter changes and the sequence must implement the functionality of the solve handler.

Green Zone

The UI shows areas for which no other parameter change is necessary as green. If another parameter must be changed for this specific value, this value is shown red. So 'red zones' and 'green zones' are displayed for parameters. Red zones are the result of the interaction of the UI with the solve handler.

Red Zone

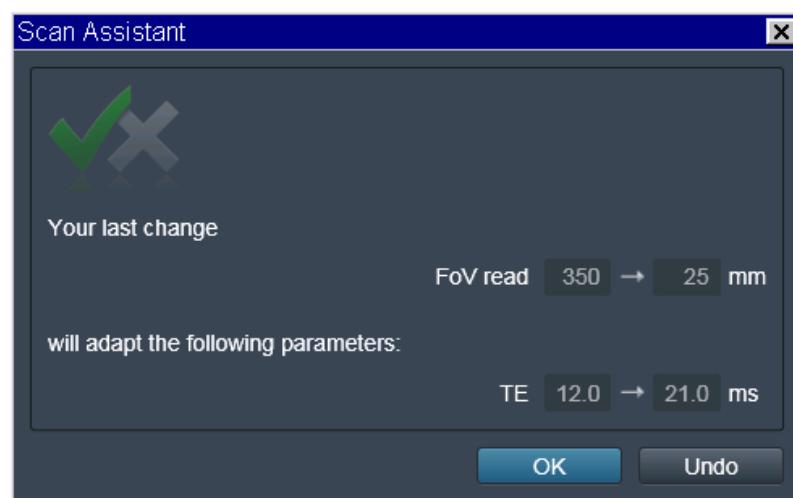
Note that - due to the screen resolution - the UI displays pixels which represent values partly contained in green and red zones in yellow.

Yellow Zone

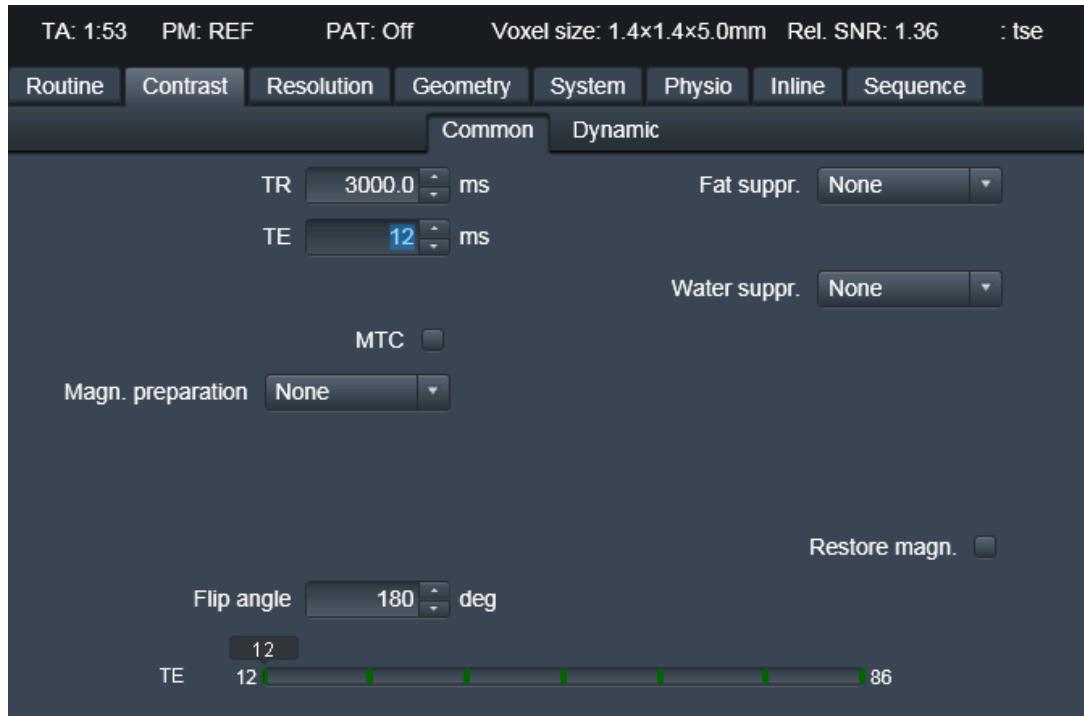
Consider the following example just to give an impression of what a red zone and a solve handler look like: Decreasing the FoV at a constant receiver bandwidth requires increasing the amplitude of the readout gradient. Allow the sequence to adapt the length of the ramps of this gradient automatically to the current amplitude. At a certain FoV value, an increase of TE and TR may be required, because the readout gradient does not fit into the current timing. The result of a solve handler for this problem is to allow selection of the smaller FoV although TE and TR are too small. (indicated as a red zone).



Select a small FoV now and the following popup appears.



An existing TI-get-limit-handler, which correctly handles the non-convex behavior of TI due to concatenated scanning, serves as example for a more colorful limit range.

**"scan all"**

One approach to handle non-convex parameter ranges is to force the UI to scan all values instead of performing a binary search. Overload the `getLimitsHandler` of this parameter and set the variable `rulVerify`, which is passed to the `getLimitsHandler`, to `LINK_XXX_TYPE::VERY_SCAN_ALL`. Then call the original `getLimitsHandler`. XXX is the type of the parameter (LONG, DOUBLE, SELECTION).

You will find more about the general parameter limit calculation concept of the UI, about solve handlers and about advanced UI programming techniques in chapter (E.4).

SeqIF::check in Detail

`SeqIF::check` is called on the host only. It is called directly after the protocol has been prepared for the measurement (in context normal).

`SeqIF::check` must execute a representative subset of the sequence timing, i.e. the part of the sequence with the highest expected gradient amplitudes. Typically, `check` executes the kernels which measure the corners of k-space in the PE/3D-plane for every slice.

In principle, `check` works like `run`. The entire sequence timing should not be used within `check` because this will significantly decrease the performance of the measurement queue while preparing sequences. Within the GSWD look ahead, `SeqIF::check` may even be called within a binary search.

The gradient waveforms are examined by the gradient overflow check and the GSWD look ahead using the coordinate system of the magnet (physical system) but not the coordinate system in which

the sequence programmer develops the sequences (logical system).

SeqIF::run in Detail

On an MR system, *SeqIF::run* is called only on the MARS during the measurement.

SeqIF::run deals with the following keywords:

libRT

The interface between the sequence and the hardware is the library libRT. It provides all necessary elements and functions to create the sequence timing.

RTE

Gradient pulses, rf pulses, ADCs, etc. are represented by so-called real-time events (RTE). All RTEs are classes realized within libRT. Each RTE that is needed during the measurement must be inserted into the sequence timing at a certain time.

RTEB

The smallest unit of timing of a real time sequence is the real time event block (RTEB). The sequence opens an event block, inserts all the necessary elements into the event block and closes the event block. The event block is processed by the MARS, which sends the data to the hardware, and the sequence can build up the next event block.

Logical Coordinate System

The RTEBs are programmed in the logical coordinate system. This system is defined by the three directions "phase encode", "read out" and "slice select". These three directions are related according to the following equation:

$$\vec{G}_{\text{PhaseEncode}} \times \vec{G}_{\text{ReadOut}} = \vec{G}_{\text{SliceSelect}}$$

Physical Coordinate System

The fixed coordinate system of the scanner hardware is the physical coordinate system. Standing in front of the scanner and looking into the magnet, the

- x-axis points from left to right;
- y-axis points from down to up;
- z-axis points from rear to front (towards you).

The gradients in the logical coordinate system are transformed into the physical coordinate system using a rotation matrix depending on the slice orientation. This rotation matrix R is defined as follows:

$$\begin{bmatrix} G_x \\ G_y \\ G_z \end{bmatrix} = R \cdot \begin{bmatrix} G_{\text{PhaseEncode}} \\ G_{\text{ReadOut}} \\ G_{\text{SliceSelect}} \end{bmatrix}$$

Gradient Overflow

Note that this transformation is one of the reasons for the gradient overflow check and why the sequence library must provide a function *check*. The maximum gradient amplitude of the gradient power amplifier may be exceeded and this will lead to an abort of the sequence execution on the hardware (e.g. "overcurrent on gradient axis x"). So the system tries to avoid such aborts by trying to prevent protocols which will cause them.

Patient Coordinate System

The user planning examinations on the UI deals with the patient coordinate system. This coordinate system depends on the positioning of the patient and is determined during patient registration. It is fixed within the patient with the

- sagittal axis points from right to left;*
- coronal axis points from anterior to posterior;*
- transverse axis points from feet to head.*

Note that with *NUMARIS/4*, the directions of polarity for the coronal and transverse axes have changed compared to former software versions (DICOM standard changed).

Real Time Sequence

In general, the sequence calculation is performed by the MARS as fast as possible while the hardware executes the sequence. However, it is possible to synchronize the calculation of RTEBs on the MARS with their execution on the hardware. This allows the sequence to react on certain events occurring during the measurement and to modify the timing calculation depending on those events.

In any case, the time for processing the sequence on the MARS must be shorter than the real measurement time. Otherwise the hardware would run out of data during the measurement ("fifo underflow"). Therefore it is necessary to reduce calculations within *run* to a minimum.

The RTE and RTEB concept is described in more detail on 189.

Sequence Unit Test

On the host it is necessary during the development phase of a sequence to debug the sequence and to perform an important test of the sequence, the so-called sequence unit test (SeqUT).

mSEQTest

The entire sequence unit test (see also page 268) is based on the event block concept. Therefore the sequence programmer must invoke the test by calling a macro *mSeqTest* inside the event block. Each event block in the sequence must have a call to this macro after the event block is opened and before it is closed.

SBB

In most sequences the kernel also contains a set of preparation pulses, e.g. spatially or frequency selective saturation pulses. As those pulses are more or less the same for every sequence they are implemented as sequence building blocks (SBB). They contain the standardized timing and are implemented as classes. In principle, the sequence only needs to create, configure, prepare and run them.

Loop Structure

As long as the ICE program can handle it, the loop structure can be completely arbitrary. It can be programmed in pure C++ and the sequence programmer can do what he/she likes. The only thing that needs to be ensured is that the ICE program receives all the data necessary to reconstruct the resulting images or spectra.

SeqLoop

All *NUMARIS/4* product sequences use a class called SeqLoop to handle the loop structure creation and other important tasks common to imaging sequences. The sequence itself only implements the innermost part of the sequence timing, the kernel.

CompositeSeqLoop (CSL)

A newer version of the SeqLoop, based on the Composite Design Pattern: In the sequence prepare a tree is created consisting of leaves (mainly loops) and nodes (mainly SBBs or kernels); this tree can be used for easy energy and timing calculations.

libSBB

The library *libSBB* provides a collection of common and useful sequence building blocks (SBBs). It also contains the SeqLoop class.

**Sequence
Controls
Everything**

With the sequence kernel and the loop structure the sequence has control over ALL events and the order of all events pulsed out by the DSPs during the whole measurement. Everything that happens during the measurement is initiated by the sequence run function.

Therefore the sequence must define and specify the properties of ALL elements of a measurement itself. But for many tasks, standard methods are available.

Mdh

With *NUMARIS/4*, the loop structure no longer has to be duplicated within the ICE program. The complete information about the loop indices of a scan are stored in a data structure that is passed to the image calculation just before the measured data. This data structure is called measurement data header (Mdh). Now the sequence must fill the Mdh with loop indices to tell the ICE-program where the current raw data belongs to.

The advantage of this technique is that the sequence programmer can now work with all possible reordering schemes without changing the ICE program. For example, the 2D standard ICE program of *NUMARIS/4* works for a simple spin echo sequence as well as for single shot echo planar imaging, turbo spin echo or turbo gradient spin echo (including postprocessing steps like regridding and phase correction). For more information about Mdh, refer to page 212.

Feedback

It is also possible for the sequence to request feedback and receive data from the ICE program. The sequence can react in real time on the data it receives. This is an advanced topic and by far beyond the scope of this general chapter.

SEQRunKernel

Sequence Kernel

In general, the MR sequence is built up using a timing table that defines the procedure to measure one or more lines of raw data for an MR image. This is often called the "sequence kernel". The sequence kernel is the part of a sequence that primarily distinguishes one sequence from another; for example, a gradient echo sequence from a turbo spin echo sequence.

The sequence interface `SeqIF` can be extended by a *run* kernel function using the class `SeqRunKernel`. Hence the sequence has to be derived from `SeqIF` (or rather `StdSeqIF`) and `SeqRunKernel`.

```
// Include of interface class definition
#include "MrServers/MrMeasSrv/libSBB/StdSeqIF.h"
#include "MrServers/MrImaging/libSBB/SeqRunKernel.h"

// Example of a C++ sequence declaration
class MySeq : public StdSeqIF, public SeqRunKernel
{
    virtual NLSStatus initialize(SeqLim*);
    virtual NLSStatus prepare  (MrProtocolData::MrProtData*,
                               SeqLim*,
                               MrProtocolData::SeqExpo *);
    virtual NLSStatus check   (MrProtocolData::MrProtData*,
                               SeqLim*,
                               MrProtocolData::SeqExpo *,
                               SEQCheckMode*);
    virtual NLSStatus run     (MrProtocolData::MrProtData*,
                               SeqLim*,
                               MrProtocolData::SeqExpo *);

    virtual NLS_STATUS runKernel(MrProtocolData::MrProtData*,
                                 SeqLim*,
                                 MrProtocolData::SeqExpo *,
                                 long, // kernel mode
                                 long, // slice index
                                 long, // line index
                                 long); // partition index
};


```

Loop Structure

Surrounding the sequence kernel, there may be a more or less complicated loop structure. A very simple 2d sequence would contain a line loop. A 3d sequence with simple reordering contains a line and a partition loop. An acquisition loop could be added directly around the kernel within the line loop. Other loops can be added in a similar fashion.

This separation of tasks - the sequence kernel on the one hand and the loop structure on the other hand - has led to a separation of the two tasks into two functions. `SeqIF::run` fulfills the task of running the loop structure of the sequence. In the innermost loop `runKernel` is called which executes the sequence kernel for a particular line of raw data. There is no limitation about when to declare and

define this function within the sequence.

E.2 Real Time Events and MR Physics

The Real Time Event Block Concept

Before explaining how to implement the timing in a pulse sequence, we have to introduce the fundamental concept of ‘real time events’ and ‘real time event blocks’.

Simply spoken, **real time events** are the basic sequence elements: rf pulses, gradient pulses, ADC readouts, etc. In terms of sequence software, real time events are derived objects of the real time event base class, provided by the real time library *libRT*. Details are given in the next section.

Real Time Event Blocks

Real time event blocks form the basic timing units of a sequence. They are processed at run time by the MARS computer, immediately before being played out on the hardware. Consider a real time event block as a package of real time events, which will usually constitute just a fraction of the whole sequence timing. During run time, the MARS will sequentially process block by block and send them into a FIFO buffer (FIFO = ‘first in, first out’), from where the sequence is finally played out in real time. Obviously, the event block processing on the MARS must be performed faster than real time execution; otherwise, a FIFO underflow error will stop the sequence execution. Typically, the entire measurement is processed by *libRT* long before the measurement has completed.

The following diagram demonstrates the sequential event block processing from the code level to the real time execution (t_p denotes the processing time for an event block, t_r denotes its duration in real time).

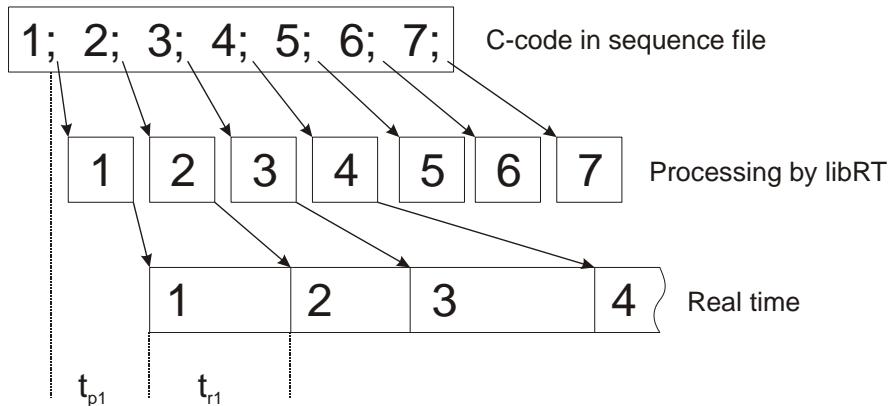


Figure D.2.1: Sequential event block processing from the code level to the real time execution

A common example might be an event block that contains the kernel of an imaging sequence, i.e. the excitation and acquisition scheme of one line in k-space. The event block could be implemented inside an acquisition loop, setting the phase encoding gradient amplitude for each line.

Note that real time event blocks are directly appended one after the other in a strictly sequential fashion. It is not possible to overlap them in time or insert pauses between blocks (the latter can be achieved by inserting ‘empty’ blocks).

The timing is defined with the help of three functions from *libRT* (real time library):

- fRTEBInit()** : Real-time event-block initialization
- fRTEI()** : Real-time event insertion, defining the initiation of events
- fRTEBFinish()** : Real-time event-block finish

fRTEBInit()
fRTEBFinish()

fRTEBInit() will begin a new event block. Thereafter, the real time events are inserted. Their timing is always specified relative to the event block start time. However, their order and position between *fRTEBInit()* and *fRTEBFinish()* is irrelevant, i.e. we are not restricted to a rigid timing table. This allows for very flexible sequence design, e.g. it is possible to call functions which insert further complex event timings.

Inside the real time event blocks, gradients are defined using logical axes (slice, read, phase). The rotation matrix, which transforms the logical axes into the physical coordinate system, is specified for each event block and cannot be changed within it.

Calling of *fRTEBFinish()* closes the event block and triggers the processing and execution on the MARS.

Note: *fRTEI()* only passes a pointer of the inserted event to *libRT*. Real time events are processed as soon as the *fRTEI()*-function or *.run* - method is called. Modifications of the events after insertions **have no effect!** A common error is to send the READOUT event, and only afterwards modify the Mdh. Since the READOUT has already been processed at that point in time these modifications on the Mdh will effect the next READOUT event.

Code example

We finish this section with an example of a real time event block, producing a simple gradient echo kernel. The details on how to implement the real time events of this sequence are covered in the sections below.

```

fRTEBInit(&asSLC[lChronologicSlice].m_sROT_MATRIX);
//***** S E Q U E N C E   T I M I N G *****
// * Start Time | NCO | SRF | ADC | Gradient Events | Sync
// * (usec) | Event | Event| Event| phase | read | slice |Event
//*****
fRTEI(lStart_GS, 0, 0, 0, 0, 0, &sGS, 0);
fRTEI(lStart_RF, &sRFNCOSet, &sRF, 0, 0, 0, 0, 0);
fRTEI(lEnd_RF, &sRFNCONeg, 0, 0, 0, 0, 0, 0);
fRTEI(lStart_GSRefoc, 0, 0, 0, 0, 0, &sGSRefoc, 0);
fRTEI(lStart_GRPrephase, 0, 0, 0, 0, &sGRPrephase, 0, 0);
fRTEI(lStart_GRO, 0, 0, 0, 0, &sGRO, 0, 0);
fRTEI(lStart_ADC, &sADCNCOSet, 0, &sADC, 0, 0, 0, 0);
fRTEI(lEnd_ADC, &sADCNCONeg, 0, 0, 0, 0, 0, 0);
fRTEI(lStart_GPE, 0, 0, 0, &sGPE, 0, 0, 0, 0);
fRTEI(lEnd_RTEB, 0, 0, 0, 0, 0, 0, 0, 0);

NLS_Status = fRTEBFinish();
// -> check NLS_Status

```

Real time imaging

The measurement instructions are calculated by the MARS and then sent to the Hardware, where they are executed. Usually, the MARS calculates as much data as possible in advance. The event block concept, however, offers a certain flexibility, for synchronizing the sequence with the measurement: The sequence can suspend the event block execution on the MARS (calling the function `fRTWaitForWakeups`, 'sleep' in the following diagram). The processing of the next event block is only continued when a 'Wakeup' signal (synchronization event) is received. The Wakeup signal has to be placed in the event block timing so that the time between Wakeup and the event block end is sufficient to process the following event block. Thus, a real time synchronization of the sequence is achieved.

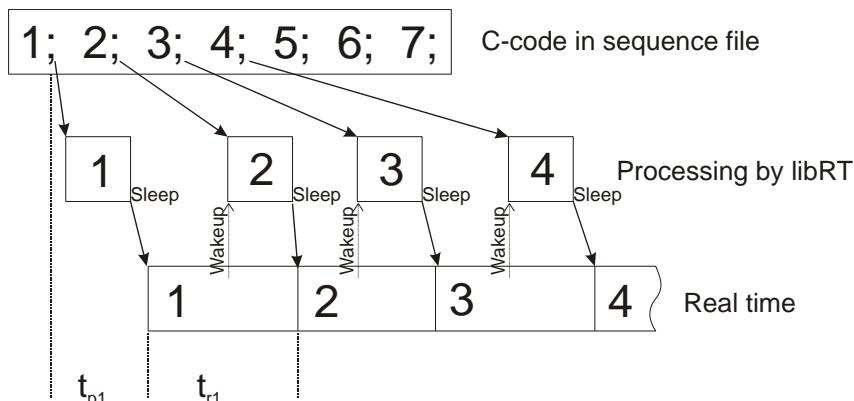


Figure: Sequential event block processing from the code level to the real time execution, when Sleep/Wakeup mechanism is used.

For example, in a sequence that is updated every slice and has the lines loop inside the slices loop, this could be implemented like this:

```
sSYNC_WAKEUP sWakeUp;
NLS_STATUS fSEQRunKernel ( ... )
{
    if (lLine == long(0.8*lLineToMeasure))
        sWakeUp.run();
    if( ADC.Mdh.isLastScanInSlice() )
        RTWaitForWakeups();
}
```

In this example, the feeding would halt at the end of each slice (`ADC.Mdh.isLastScanInSlice()`), and re-start the feeding once 80% of the lines of the last slice have been played out by the MARS (`lLine == long(0.8*lLineToMeasure)`).

In case of real time feedback, the signal may come from a number of sources. The Cine TrueFISP sequence continuously sends the sequence kernel to keep the magnetization in steady state, and uses a physiological trigger signal (e.g., ECG) to incorporate the phase encoding and readout events.

The real time feedback may also come from the ICE reconstruction process, such as for navigator

applications (s. sections below for an example). Within the sequence code, the programmer is free to implement the desired functionality.

Automatic real time execution of sequences

With VB13A a feature was introduced to facilitate running sequences in real time. For this purpose the RTController in the libRT provides a set of new methods:

```
// Enable or disable libRT realtime processing mode
virtual void setRealtimeProcessing(bool bEnable);

// Is realtime processing mode enabled?
virtual bool isRealtimeProcessingEnabled() const;

// Set realtime processing margin to dTime [s]
virtual void setRealtimeProcessingMargin(double dTime);

// Returns realtime processing margin [s]
virtual double getRealtimeProcessingMargin() const;
```

By using these methods the return of `fRTEBFinish()` can be delayed until the sequence execution has reached a sequence time of

$$t_{\text{End of EB}} - \Delta t_{\text{realtime margin}}$$

When the call returns, the sequence has a time interval of $\Delta t_{\text{realtime margin}}$ left to submit a new event block. This includes the time needed by the framework to process, translate and transfer the data to the hardware.

Care must be taken, that $t_{\text{realtime margin}}$ is not selected too low. The default value for the delay is $t_{\text{realtime margin}} = 2 \text{ ms}$.

The methods are accessible from the sequence code e.g. via

```
RTController::getInstance().setRealtimeProcessing();
etc.
```

Real Time Feedback

Many sequences use a relatively constant set of event blocks during execution (with the exception of the PE/3D gradient amplitudes). The real time event block concept allows easy incorporation of variability within a sequence during execution. One example of this is known as Real Time Feedback, where the executing sequence receives and responds to information from e.g. the ICE program. There are two basic components involved:

`SeqIF::receive`

`SeqIF::receive`

SeqIF::receive is an optional entry point of the sequence code (in contrast to the mandatory functions SeqIF::prep, SeqIF::run, etc.). It is called by the sequence framework in order to retrieve data that were sent from another software component, e.g. the ICE program.

SqData**SeqData**

The SeqData class contains the data being send from another software component to the sequence, an ID identifying the sender and the length of the data package.

An implementation might work as follows (also see code example below):

In *run*, data is acquired that should be evaluated by the ICE program (e.g. a navigator scan). Additional time is explicitly added to the event block (30 ms in the example) in order to ensure that the feedback to the sequence can be processed in time. Unfortunately, this time has to be determined empirically. An alternative approach might be to perform a loop of empty event blocks, until feedback is received.

In order to synchronize with real time execution, an empty event block is sent containing a 'WaitForWakeUp' sync event (realized in *fRunPause (dDuration_us)* in the example below). *fRTWaitForWakeUp* pauses, until this event is received.

The ICE program is informed to perform the real time feedback by setting the Mdh flag *MDH_RTFEEDBACK*. In the example, the ICE program sends a feedback data structure *rSEQData* to the sequence, which might contain e.g. some double values in the first elements. This data is received by the sequence via *receive(..., SEQData& rSEQData)*.

Within *receive*, the data are transferred to a *SeqFeedbackBuffer* object, e.g. by *sFBBuffer.init (rSEQData)*. Now the data can be accessed using *(const double *) sFBBuffer[0]*, which will return a pointer to the first data element (the correct type casting is up to the programmer). Before leaving *receive*, the flag *bFeedbackReceived* is set. This way, *run* will know that feedback has occurred.

```
// Wakeup bit to keep synchronous with event block processing
sSYNC_WAKEUP sWakeUp;

// Flag to signal, if feedback has been received static bool bFeedbackReceived;
// -----
// run()
// -----
NLS_STATUS Myseq::run (
    MrProtocolData::MrProtData * pProt,
    SeqLim*                  pSeqLim,
    SeqExpo*                 pSeqExpo);
{
    // Note: this code is not complete.
    // We just show some real time feedback functionality

    // Reset the feedback flag
    bFeedbackReceived = false;

    // Require real time feedback from ICE program
    sADC.Mdh.addToEvalInfoMask (MDH_RTFEEDBACK);
```

```

// Perform measurement and pause (e.g. 30ms) until feedback
// is received
fRTEBInit(&asSLC[1Slice].m_sROT_MATRIX);
// * -----
// * Start Time | NCO | SRF | ADC | Gradient Events | Sync
// * (usec)      |Event|Event|Event|phase|read|slice | Event
// * -----
fRTEI(...); // (arbitrary experiment, e.g. navigator)
fRTEI(lADCStart ,&sADCSet,0, &sADC,     0,     0,     0,    0);
fRTEI(lADCEnd   , sADCNeg,0,     0,     0,     0,     0,    0);
fRTEI(lADCEnd+30000, 0,0,     0,     0,     0,     0,    0);

NLS_STATUS nStatus = fRTEBFinish();
if( nStatus != NLS_SUCCESS )
{
    // -> error handling
}

// get synchroneous with real time
// (send an 'empty' event block, containing a 'wait for wake
// up' event ...
nStatus = fRunPause( lPause_us );
if( nStatus != NLS_SUCCESS )
{
    // -> error handling
}

// ... and wait)
nStatus = fRTWaitForWakeup();
if( nStatus = NLS_SUCCESS )
{
    // -> error handling
}

// feedback should now be available; let's check it:
if( !bFeedbackReceived )
{
    // something went wrong, no feedback!
    return SEQU_ERROR;
}

// feedback was received
// -> continue ...

return nStatus;
}

// -----
// receive():
// -----
NLS_STATUS Myseq::receive(SeqLim* pSeqLim, SeqExpo*, SEQData& rSEQData)

```

```

{
    // check whether the received data object is relevant for us
    if( strcmp(rSEQData.getID(), "Feedback_example") )
    {
        return SEQU_NORMAL;
    }

    // retrieve the feedback data
    double dParam1;
    double dParam2;

    // Assuming rSEQData contains a series of double values

    double* pData = (double *)rSEQData.getData();
    dParam1 = *pData;
    dParam2 = *(pData++);
    // add further calculations on received data ...
    ...

    // Set the feedback flag
    bFeedback = true;

    return SEQU_NORMAL;
}

// -----
// runPause(): 'Empty' event block during wait for feedback
// -----
NLS_STATUS Myseq::runPause (long lDuration_us)
{
    fRTEBInit(&sROT_MATRIXUnity);
    //*****
    // Start Time | NCO | SRF | ADC |Gradient Events | Sync
    // (usec) | Event |Event| Event|phase|read|slice| Event
    //*****
    fRTEI(      0,      0,      0,      0,      0,      0,      0, &sWakeUp);
    fRTEI(lDuration_us, 0,      0,      0,      0,      0,      0);

    NLS_STATUS nStatus = fRTEBFinish();
    if( nStatus != NLS_SUCCESS )
    {
        // -> error handling
    }
    return nStatus;
}

```

SeqData communication from a sequence to ICE

Note that the SeqData mechanism can be used as well for sending data FROM the sequence TO Ice, e.g. in order to supply additional information for the reconstruction. For this purpose, all you have to do (on the sequence side) is to prepare the SeqData object and “measure” the object; in ICE you have

to receive the data appropriately. This section describes the required steps.

On the sequence side:

- add the relevant header files:

```
#include "MrServers/MrMeasSrv/SeqIF/libRT/sSYNCDATA.h"
#include "MrServers/MrMeasSrv/SeqIF/libMES/SEQData.h"
#include "MrServers/MrMeasSrv/SeqIF/libRT/include/sROT_MATRIX.h"
```

- add the ICE program for receiving the data (in ::prepare):

```
rSeqExpo.AddAdditionalIceProgramFileName("%CustomerIceProgs%\IceProgramSeqDataExample");
```

- add the code for preparing (where appropriate, e.g. in ::prepare) and sending the data paket (in ::run):

```
NLSStatus NlsStatus = SSEQU_NORMAL;
// create scan object
sSYNCDATA DataScan("DataScan");
// create data object and set a unique ID
SEQData Data;
Data.setID("SomeUniqueID");

// the data to be transferred to ice
std::string message ("some important message for ice");

// allocate space
bool bSuccess = (Data.allocData((int32_t)message.length()) != 0) &&
(Data.getLength() == (int32_t) message.length()) &&
Data.getData();

if( !bSuccess )
{
    UTRACE(Info, 0, ptModule <<< ": serialization failed to produce valid
data");
    NLSStatus NlsError = SSEQU_ERROR;
    return NlsError;
}

// copy data
Data.setData( (void*)(message.data() ), (int32_t) message.length() );

// prepare data scan
NlsStatus = DataScan.setData(Data);
if( ! NlsStatus.isSuccess() )
{
    UTRACE(Info, 0, ptModule << ": DataScan.setData(Data) failed");
    return NlsStatus;
}

NlsStatus = DataScan.prep();
```

```

if( ! NlsStatus.isSuccess() )
{
    UTRACE(Info, 0, ptModule << ": DataScan.prep() failed");
    return NlsStatus;
}

// send data scan
sROT_MATRIX dummyRotMat;
NlsStatus = fRTEBInit( dummyRotMat );
NlsStatus = fRTEI(0, &DataScan);
NlsStatus = fRTEI(1000, 0, 0, 0, 0, 0, 0, 0, 0 );
NlsStatus = fRTEBFinish();

```

On the ICE side:

- create a new ICE program (e.g. IceProgramSeqDataExample.ipr) for processing the data (in case of an additional ICE program, as in the example above, only the configurator of your library has to be mentioned).

- program your functor, e.g. by deriving from `IceScanFunctors`; in addition you have to derive from `ISyncMeasData` and implement the method `UpdateScan`:

```

#include "MrServers/MrVista/include/Parc/FunctorEnv.h"
class ICEIDEAFUNCTORS__API SeqDataExampleFunctor : public IceScanFunctors, public
ISyncMeasData
{
...
...
    virtual IResult UpdateScan(ISEQDataEvent* data);
...
}

```

- a dummy implementation could be:

```

#include "MrServers/MrVista/Parc/PipeLink/Functors/SEQDataEvent.h"
IResult SeqDataExampleFunctor::UpdateScan(ISEQDataEvent* data)
{
    SEQData& rSEQData = data->getSEQData();

    // check whether the received data object is relevant for us
    if( strcmp(rSEQData.getID(), "SomeUniqueID") )
    {
        return I_OK;
    }

    // extract data length
    int32_t dataLen = rSEQData.getLength();

    // extract message
    std::string message( (const char *) rSEQData.getData(), dataLen );

    ICE_OUT("received the following message from the sequence: " << message);

```

```

        return I_OK;
    }

    - adapt the configurator:

IResult SeqDataExampleConfigurator::Compose( ProtocolComposer::Toolbox& toolbox )
{
    using namespace ProtocolComposer;

    toolbox.Init( "IRIS" );

    // -----
    // Insert Functor in Functor chain
    // -----
    // retrieve name of corresponding pipeService for the source Functor:
    MrFunctor sourceFunctor = toolbox.FindFunctor("source");
    std::string pipeName = sourceFunctor.PipeServiceName();
    MrPipeService pipeService = toolbox.FindPipeService(pipeName.c_str());
    // <FunctorIdName>, <FunctorClassName>@<DllName>
    MrFunctor SeqDataExampleFunctor = pipeService.CreateFunctor("SeqDataExample",
"SeqDataExampleFunctor@IceSeqDataExample");

    // no need to insert functor anywhere!

    return I_OK;
}

```

- adapt the ICE makefile to include libMes

Note:

- sending the data requires some time, which has to be added to the sequence duration (in the example it is fixed to 1000us).
- while the regular ADCs are measured according to their sequence time, the data scan described here is sent as soon as it is processed, i.e. usually right at the beginning of the sequence run (most of the sequence is preprocessed and stored in buffers).
- the ICE program's sink `updateScan` does NOT have to be connected, i.e. insertion of the functor into the pipeline is unnecessary.

k-Space Sampling in NUMARIS/4

The goal of most MR imaging sequences is to fill k-space in an efficient manner. The size of the sampled k-space and the sampling density is determined by the desired resolution and the desired image size without wrap-around artifacts.

Before going into the details of programming sequence timing, a brief summary is presented of commonly used terms of k-space sampling in *NUMARIS/4*. First, we define common keywords used for sampling k-space and point out some restrictions of this concept. This summary is to ensure that the specific terminology is used in a consistent manner in the subsequent sections of this chapter.

Then we show how the common k-space definitions are mapped to protocol parameters for each direction of k-space. Finally, we describe how the UI supports k-space segmentation and explain some “special behavior” of the protocol parameters and the UI software.

Common definitions of k-space

In k-space, there are three directions: readout or frequency encoding (RO), phase encoding (PE) and 3d phase encoding (3D). For each direction of k-space, the sequence and the ICE program need information about the following characteristics of k-space and the image:

γ [MHz/T]

The gyromagnetic ratio of the relevant nucleus. Note that the units of measure here is MHz/T.

ImageSamples

The number of pixels of the final image in one dimension.

ImageSize [mm]

The size of the final image.

OverSampling [%/100]

If the sampling density of k-space should be increased in order to avoid wrap-around artifacts, this is done with the oversampling parameter. An OverSampling of 100% means that the k-space is sampled sufficiently dense to image objects of twice the ImageSize without leading to wrap-around artifacts (see following figure).

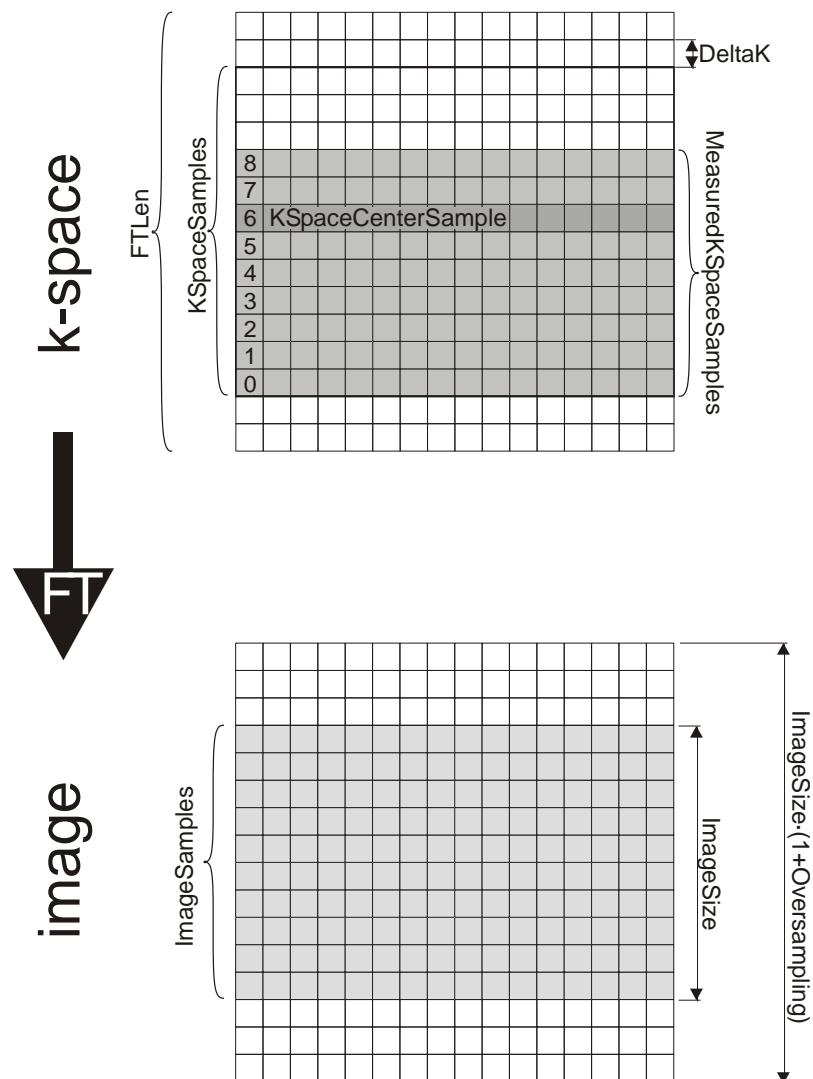


Figure D.2.3: Common K-Space Definition:
 $\text{ImageSamples} = 10$, $\text{OverSampling} = 0.6 = 60\%$, $\text{FTLen} = 16$, $\text{ReducedResolution} = 0.75 = 75\%$,
 $\text{KSpaceSamples} = 12$, $\text{PartialFourier} = 0.75 = 6/8$, $\text{MeasuredKSpaceSamples} = 9$.

FTLen

The number of k-space samples processed by the Fourier transform (k-space to image domain):

$$\text{FTLen} = \text{ImageSamples} \cdot (1 + \text{OverSampling})$$

ReducedResolution [%/100]

The number of acquired k-space samples is reduced in order to reduce the scan time. This reduction is realized by removing the outer k-space samples (symmetrically on both ends of the sampled k-space region). A ReducedResolution of 50% means that the spatial resolution of the image is half as large as the value given by $\text{ImageSize}/\text{ImageSamples}$.

$$\text{KSpaceSamples} = \text{FTLen} \cdot \text{ReducedResolution}$$

KSpaceSamples

The number of acquired k-space samples in one dimension.

 PartialFourier [%/100]

The number of acquired k-space samples can also be reduced asymmetrically. This means that k-space samples are skipped only at one end of the sampled k-space region. A PartialFourier of 75% means that only 50% of the, e.g., upper half of k-space is actually measured by the sequence. Theoretically, this technique does not produce artifacts and does not reduce the spatial resolution of the image.

 MeasuredKSpaceSamples

The number of k-space samples actually measured by the sequence, i.e. KSpaceSamples reduced by PartialFourier.

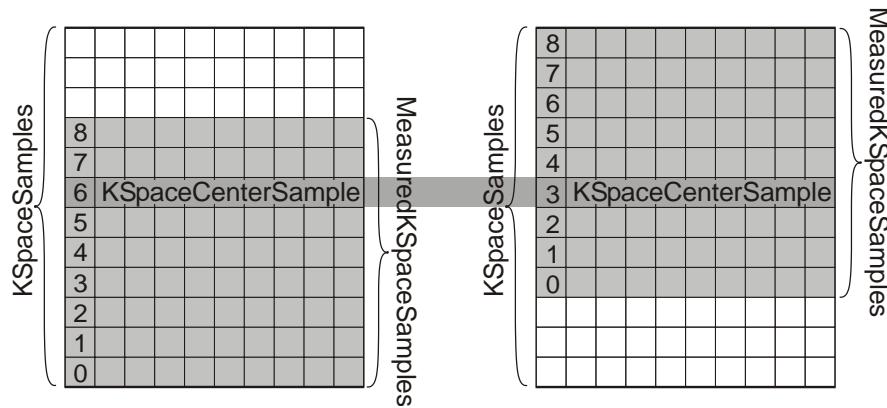
$$\text{MeasuredKSpaceSamples} = \text{KSpaceSamples} \cdot \text{PartialFourier}$$

 SampleIndex

The index of a particular k-space sample. This index increments from 0 to (MeasuredKSpaceSamples - 1).

 KSpaceCenterSample

The SampleIndex of the k-space sample that contains the k-space center. The k-space center is defined as the point in k-space that is measured without any gradient moments applied. This index must be between zero and (MeasuredKSpaceSamples - 1) (see following figure).



FigureD.2.4: KSpace Center Sample: Note that the SampleIndex of the KSpaceCenterSample here changes from 6 to 3 if the lower part of k-space is omitted as when measuring partial Fourier.

 DeltaK [1/m]

$$\text{DeltaK} = \frac{2\pi \cdot 10^3}{\text{ImageSize} \cdot (1 + \text{OverSampling})} = 2\pi \cdot 10^{-3} \cdot \gamma \cdot \text{DeltaMoment}$$

The distance between two k-space samples (expressed as gradient moment):

$$\text{DeltaMoment} = \frac{10^3 \cdot \text{DeltaK}}{2\pi \cdot \gamma} = \frac{10^6}{\text{ImageSize} \cdot \gamma \cdot (1 + \text{OverSampling})}$$

 DeltaMoment [(mT/m)*μs]

The difference of the moment of the phase encoding gradient for adjacent k-space samples.

□ GradientMoment [$(mT/m) * \mu s$]

This is the integral of the gradient over time. The following gradient moment is required for measuring the sample given by SampleIndex:

`GradientMoment = DeltaMoment · (SampleIndex – KSpaceCenterSample)`

Restrictions for k-space sampling

When dealing with all these quantities, there are also the following variable restrictions (which in some cases are not easy to handle):

- The variable KSpaceSamples is an integer. For sequences with k-space segmentation, only a limited subset of integer values is allowed.*
- The variable FTLen is an integer. To speed up image calculation, only a limited subset of integer values is allowed.*
- KSpaceSamples <= FTLen.
i.e. ReducedResolution is less than or equal to 1.0.*

These restrictions are summarized in the following two equations:

$$\text{FTLen} = \text{ImageSamples} \cdot (1 + \text{OverSampling})$$

$$\text{KSpaceSamples} = \text{FTLen} \cdot \text{ReducedResolution}$$

Given these boundary conditions, there are two problems:

- The variables setting up the restrictions (FTLen and KSpaceSamples) are not part of the UI.*
- The relevant parameters available in the UI (ImageSamples, ReducedResolution, OverSampling) cannot be modified independently of each other.*

There are no general solutions for these problems. The following sections describe the way *NUMARIS* deals with these restrictions.

**Hidden
Oversampling**

Restriction handling in NUMARIS/4

KSpaceSamples and *FTLen* are discrete variables. For segmented sequences, this causes the raster for *KSpaceSamples* to become coarser with increased segmentation factors. At the same time, the user should be allowed to select an arbitrary reduced resolution. A mechanism is needed to fulfill these two constraints at the same time. This mechanism is called "hidden oversampling".

With hidden oversampling, the UI software internally increases the oversampling. The oversampling selected by the user in the UI is treated as a minimum oversampling. The maximum for hidden oversampling is 20%. If a parameter change requires a hidden oversampling of more than 20%, the parameter change is not offered to the user by the UI.

- For sequences using segmented k-space sampling in the 2d phase encoding direction, the UI applies the hidden oversampling algorithm.

Read Out Methods

Common Terms	NUMARIS/4 variable(s) for readout direction
ImageSamples	<code>rMrProt.kSpace().baseResolution()</code>
ImageSize	<code>rMrProt.sliceSeries()[0].readoutFOV()</code>
KSpaceSamples	<code>rMrProt.kSpace().baseResolution() * rSeqLim.getReadoutOSFactor()</code>
OverSampling	<code>rSeqLim.getReadoutOSFactor()</code>
DeltaMoment	<code>fGSLGetRODeltaMoment(pMrProt) / (1+rSeqLim.getReadoutOSFactor())</code>

Note the following issues:

Oversampling:

There is a default two-fold oversampling in the readout direction due to hardware limitations. If the sequence programmer always uses the base resolution and the effective dwell time of the ADC, the RO-oversampling factor within the sequence is not necessary. If changes from the default are necessary, the desired RO-oversampling factor may be defined using the appropriate method of the SeqLim class.

ReducedResolution, MeasuredKSpaceSamples, PartialFourier:

There is no UI parameter to control these parameters in the readout direction. This is due to the fact that the readout direction defines the base resolution of the image and other parameters are not necessary or complicate this definition. However, these parameters can be modified within the sequence by reducing the number of ADC samples and moving the k-space center within the ADC sampling window. The UI parameter coming closest to a partial Fourier factor is `rMrProt.kSpace().asymmetricEchoAllowed()`. This enables asymmetric sampling if necessary to make the protocol consistent.

Read FoV for all imaging slices are equal:

Although each imaging slice object in the protocol has its own read FoV, phase FoV, and slice thickness, they are the same for all imaging slices. Otherwise, the sequence timing table would have to change during the measurement according to the current slice index. The image reconstruction program would need mechanisms to handle these variations as well.

Phase Encoding Methods

Common Terms	NUMARIS/4 variable(s) for phase encoding direction
ImageSamples	<code>rMrProt.kSpace().baseResolution() * rMrProt.sliceSeries()[0].phaseFOV() / rMrProt.sliceSeries()[0].readoutFOV()</code>
ImageSize	<code>rMrProt.sliceSeries()[0].phaseFOV()</code>
KSpaceSamples	<code>rMrProt.kSpace().phaseEncodingLines()</code>
OverSampling	<code>rSeqLim.getReadoutOSFactor()</code>
ReducedResolution	<code>rMrProt.kSpace().phaseOversampling()</code>
MeasuredKSpace Samples	<code>rMrProt.kSpace().linesToMeasure(...)</code>
KSpaceCenter Sample	<code>rMrProt.kSpace().echoLine(...)</code>
PartialFourier	<code>rMrProt.kSpace().phasePartialFourierFactor()</code>
DeltaMoment	<code>fGSLGetPEDeltaMoment(pMrProt)</code>

Note the following issues:

ImageSamples:

The image samples in the read/phase plane are quadratic.

PartialFourier:

The method does not return a double value, but an enumeration of type *SEQ::PartialFourierFactor*.

MeasuredKSpaceSamples:

Calculating the lines to measure from the protocol is a little bit tricky. The complete interface of the *MrKspace* method is:

```
NLS_STATUS linesToMeasure (long& rlLinesToMeasure,  
                           long increment           = 1,  
                           double relEchoPosition   = 0.5,  
                           long    phaseCorrLinesFor256 = 16) const;
```

A reference to a long value must be passed to let the function calculate the number of measurement lines. With large values for *increment* and *phaseCorrLinesFor256*, it may not be possible to find a valid number of measured lines based on the raster given by *increment* and between the minimum and the maximum allowed value. If this happens, this method will return an error *NLS_STATUS* code and the sequence should react to this error.

If the rough raster given by the *SEQ::PartialFourierFactor* enumerations is not desired, there is a different *MrKSpace* method that uses a partial Fourier factor of type double:

```
NLS_STATUS linesToMeasureWithPFF (long& rLinesToMeasure,  
                                double dphasePartialFourierFactor,  
                                long   increment           = 1,  
                                double relEchoPosition     = 0.5,  
                                long   phaseCorrLinesFor256 = 16) const;
```

KSpaceCenterSample:

Although it is not recommended to use it, the MrKSpace-method *echoLine* takes an optional parameter:

```
long echoLine (double relEchoPosition = 0.5) const
```

Why is it not recommended to modify this parameter? Because a change in the relative position of the echo line within the measured k-space is nothing else than applying phase partial fourier and changing the resolution.

Phase FoVs for all imaging slices are equal.

3D Phase Encoding Methods

Common Terms	NUMARIS/4 variable(s) for phase encoding direction
ImageSamples	<code>rMrProt.kSpace().imagesPerSlab()</code>
ImageSize	<code>rMrProt.kSpace().imagesPerSlab()</code>
KSpaceSamples	<code>rMrProt.kSpace().partitions()</code>
OverSampling	<code>rMrProt.kSpace().sliceOversampling()</code>
ReducedResolution	<code>rMrProt.kSpace().sliceResolution()</code>
MeasuredKSpace Samples	<code>rMrProt.kSpace().partitionsToMeasure(...)</code>
KSpaceCenter Sample	<code>rMrProt.kSpace().echoPartition(...)</code>
PartialFourier	<code>rMrProt.kSpace().slicePartialFourierFactor()</code>
DeltaMoment	<code>fGSLGet3DDeltaMoment(pMrProt)</code>

Note the following issues:

PartialFourier:

This is analogous to phase encoding.

MeasuredKSpaceSamples:

This is analogous to phase encoding, except for function names.

```
NLS_STATUS partitionsToMeasure (
    long& rlPartitionsToMeasure,
    long increment = 1,
    double relEchoPosition = 0.5,
    long phaseCorrLinesFor256 = 16) const;

NLS_STATUS partitionsToMeasureWithPFF (
    long& rlPartitionsToMeasure,
    double dslicePartialFourierFactor,
    long increment = 1,
    double relEchoPosition = 0.5,
    long phaseCorrLinesFor256 = 16) const;
```

KSpaceCenterSample:

This is analogous to phase encoding, except for function names:

```
long echoPartition (double relEchoPosition = 0.5) const;
```

Thickness in 3D:

The slice thickness selected in the UI is the effective thickness of an image of the 3d slab. The slice thickness stored in the protocol is the slab thickness of the excited 3d volume (`rMrProt.sliceSeries()[0].thickness()`).

This handling of the thicknesses for 3d measurements is due to the fact that the UI provides the same handling for planning slices in 3d as in 2d while the protocol offers the sequence programmer the same handling when exciting 3d-slabs as 2d-slices.

Slice thicknesses for all images are equal.

UI support for segmentation in PE and 3D

The allowed number of phase encoding lines (number of *KSpaceSamples*) is specified with the following method of the SeqLim class:

```
NLS_STATUS setPELines (long lMinimum, long lMaximum, long lIncrement, long lDefault);
```

Segmented sequences do not allow the selection of an arbitrary number of *KSpaceSamples*. In general, this number must be an integer multiple of the segmentation factor (e.g. turbo-factor, epi-factor, number of segments). When a segmented sequence is initialized, it must tell the UI which variable increment should be used for the phase encoding lines using an overloaded version of the same SeqLim-method as before:

```
NLS_STATUS setPELines (long lMinimum, long lMaximum, SEQ::Increment eIncrement,
long lDefault);
```

Among others, the UI supports the following values for *SEQ::Increment*:

SEQ::Increment	Variable Increment for KSpaceSamples
SEQ::	rMrProt.fastImaging().turboFactor()
INC_TURBO_FACTOR	
SEQ::INC_EPI_FACTOR	rMrProt.fastImaging().EPIFactor()
SEQ::	rMrProt.fastImaging().segments()
INC_GRE_SEGMENTS	
SEQ::	rMrProt.fastImaging().turboFactor()
INC_TGSE_FACTOR	* rMrProt.fastImaging().EPIFactor()
SEQ::INC_SEGMENTED	rMrProt.fastImaging().turboFactor() * rMrProt.fastImaging().EPIFactor() * rMrProt.fastImaging().segments()

Coming back to our example, we are now able to tell the UI to use a variable turbo factor:

```
rSeqLim.setTurboFactor ( 1, 65, 2, 7);
rSeqLim.setPELines ( 32, 1024, SEQ::INC_TURBO_FACTOR, 252);
```

The turbo factor can now be selected between 1 and 65 while 7 is the default value. The UI now limits the phase encoding lines between 32 and 1024 and in multiples given by the current turbo-factor.

A special kind of k-space segmentation applies for single shot techniques (e.g. HASTE). In this case, the number of lines per segment is equal to one and the number of acquired echos is MeasuredKSpaceSamples. The following lines of code show how a HASTE sequence may ask the UI to display the current turbo-factor (defined as KSpaceSamples/(1+OverSampling) for single shot) as read

only on the Sequence card.

```
rSeqLim.setTurboFactor ( 1, 512, SEQ::INC_SINGLESHOT, 256);
rSeqLim.setPELines      ( 64, 512,                      1, 256);
```

The following SeqLim-methods are available for segmentation in 3D:

```
NLS_STATUS setPartition (long lMin, long lMax, long lIncrement,
                        long lDef);
NLS_STATUS setPartition (long lMin, long lMax, SEQ::Increment
                        eIncrement, long lDef);
```

Note:

Segmentation in 3D is currently not supported by the UI although the SeqLim methods exist.

Real Time Events

As stated in the previous section, real time events form the interface to what finally constitutes a sequence on the scanner hardware: rf pulses, gradients, ADC, etc. Following the object-oriented sequence concept, real time events are provided as a collection of classes (the real time event library, libRT). Their specific class attributes contain physical properties (amplitude, duration, dwelltime, flip angle, etc).

Additionally, real time event classes contain dedicated methods to support common tasks of sequence programming. For example, a prepared (the meaning of 'prepare' in this context is explained below) RF excitation pulse readily provides the required amplitude for the slice selection gradient by its method `.getGSAmplitude()`. Gradient pulse events have a `.check()` method to perform a selfcheck on its parameters (e.g. whether the amplitude exceeds predefined limits), etc.

In this chapter, the handling of real time events is described in the context of building a simple sequence, rather than listing systematically the classes and their methods. This way, the concept of transforming sequence physics into sequence source code should become more clear.

The following types of real time event classes and variants are currently defined:

RF Pulses

class sRF_PULSE

The following variants of rf pulse classes are derived from the base class:

- sRF_PULSE_SINC (Sinc)*
- sRF_PULSE_GAUSS (Gaussian)*
- sRF_PULSE_HYPSEC (Hyperbolic Secant)*
- sRF_PULSE_RECT (Rectangular)*
- sRF_PULSE_TRI (Triangle)*
- sRF_PULSE_EXT (External)*
- sRF_PULSE_LIN (Linear)*
- sRF_PULSE_ARB (Arbitrary)*
- sRF_PULSE_PTX (multi channel transmit)*

Frequency/ Phase Events	<code>□class sFREQ_PHASE</code> These objects set the frequency and phase properties of the NCO. Please note, that the phase is always set as an increment relative to the previous value and not as an absolute value.
Gradient Pulses	<code>□class sGRAD_PULSE</code> To account for the specific requirements of standard gradient encoding schemes, four variant classes are derived from the base class: <code>sGRAD_PULSE_PE</code> (2D phase encoding) <code>sGRAD_PULSE_3D</code> (3D phase encoding) <code>sGRAD_PULSE_RO</code> (readout gradients) <code>sGRAD_PULSE_SIN</code> (sinusoidally shaped ramps) <code>sGRAD_PULSE_ARB</code> (arbitrarily shaped ramps) <code>sGRAD_PULSE_TRAP</code> (trapezoidal gradient)
Synchronization	<code>□class sSYNC</code> Synchronization events are used for different types of triggering, coil detuning or real time control. Derived classes from SYNC are: <code>sSYNC_ECHO</code> <code>sSYNC_EXTTRIGGER</code> <code>sSYNC_FINISH</code> <code>sSYNC_GradOffset</code> <code>sSYNC_HALT</code> <code>sSYNC_NOP</code> <code>sSYNC_OSC</code> <code>sSYNC_PHYSIO1_HALT</code> <code>sSYNC_PHYSIO2_HALT</code> <code>sSYNC_START</code> <code>sSYNC_SUBSEQFINISH</code> <code>sSYNC_SUBSEQSTART</code> <code>sSYNC_SYNC</code> <code>sSYNC_WAKEUP</code> .
Readout	<code>□class sREADOUT</code> This class contains information regarding the ADC readout event.
Slice Position	<code>□class sSLICE_POS</code> This class contains the information on the orientation and position of each slice. An important member is the rotation matrix (which forms a separate class) associated to it.
Rotation Matrix	<code>□class sROT_MATRIX</code> The rotation matrix maps the logical coordinate system (readout, phase and slice) to the physical gradient system (x, y, and z). It is usually handled via the SLICE_POS object.

The handling of real time events is generally performed in four steps (SLICE_POS and ROT_MATRIX deviate a little from this scheme, described later):

Declaration

In the Declaration step, each real time event (e.g. rf pulses, gradient pulses) is created from its class object. To identify individual events, an ident text string may be defined either as an initialization argument or using the `.setIdent()` method. The ident member of a real time event object serves as its ‘name’, which is particularly useful for debugging; the ident is shown in the visualizer of the sequence simulation as well (see also Sequence Visualizer on page 94). However, for rf pulse objects, it is mandatory to set an unique ident for each event as this is the only identifier of a pulse in the MARS.

Configuration

In the configuration step, the desired parameters (amplitudes, rise times, flipangle, etc.) are set for the object, based on the measurement protocol. For most standard situations, helpful class methods are supplied, e.g. for calculation of the slice gradient amplitude from the rf pulse characteristics.

Preparation

In the Preparation step, the events are actually created. In other words, the real time events are prepared on a hardware-related level, so that they are ready to be played out by the DSPs. Each real time event owns one or more prepare methods `.prep()`. Their main task is to load the required waveforms, e.g. the sampling points for rf pulses or gradient ramps. The generic preparation method for some classes can be overloaded by methods that may accept different arguments. Gradients, for example, may be prepared by passing a certain gradient moment as an argument instead of calculating and setting the amplitude.

Note: When applying the `.prep()` method to a gradient, its maximum amplitude and rise time are not considered. This check must be performed separately by applying the `.check()` method inside `SeqIF::prepare` (which should return the NLSStatus if `.check()` does fail). In fact, the user interface relies on this behavior of the sequence when it runs the binary search on a sequence parameter in order to determine the range of allowed values (the ‘soft limits’).

Execution

During sequence execution, real time events are played out within logical units: the real time event blocks. In general, there are two ways to insert a real time event into an event block:

`fRTEI(...)`

The start time of execution (relative to the event block start time), is given as the first argument of the `fRTEI()` function.

`.run() or .run(offsetTime)`

This is a method of the real time event base class (owned by every real time event). The start time must be previously set by the `.setStart-Time()` method. The optional argument defines an offset in time relative to the start time set.

Note:

`fRTEI()` functions or `.run()` methods need not be arranged in chronological order.

Example:

```
// * -----
// * Start Time | NCO | SRF | ADC | Gradient Events | Sync
// * (usec)      |Event|Event|Event|phase|read|slice | Event
// * -----
fRTEI (    1000,    0,    0,    0,    0,    0, &sGS,    0);
```

is equivalent to

```
sGS.setAxis(SEQ::AXIS_SLICE);
sGS.setStartTime(1000);
sGS.run()
```

and is equivalent to

```
sGS.setAxis(SEQ::AXIS_SLICE);
sGS.setStartTime(600);
sGS.run(400)
```

Defining the Coordinate System of the Current Event Block

Slice Position

The slice position object *sSLICE_POS* is declared as an array with length equal to the number of slices. The array index denotes the chronological order of slices (that may be different from the order of slices in the protocol). Thus the actual protocol slice number is stored in each *SLICE_POS* element (*m_lSliceIndex* member). Additional members of *sSLICE_POS* contain slice shift, slice offcenter RO, slice offcenter PE, phase offcenter PE and phase offcenter 3D. Using the prepare method (.prep()), these parameters and the rotation matrix are automatically calculated directly from the protocol settings:

```
bool sSLICE_POS::prep (MrProtocolData::MrProtData *pMrProt,
                      SeqLim                  *pSeqLim,
                      const Slice               &sSlice,
                      const long                lSliceIndex)
```

&sSlice passes the desired slice from the protocol and *lSliceIndex* denotes its (protocol-) index as mentioned above. This might look like:

```
sSLICE_POS     asSLC[K_NO_SLI_MAX];

const Slice &slice = pMrProt->sliceSeries()[lSliceIndex];
if (asSLC[lChronologicSliceNumber].prep (pMrProt,pSeqLim,slice,
lSliceIndex) == false)
{
    // -> Error handling
}
```

To support the common task of preparing the full set of slices given in the protocol, a dedicated function is provided by libSeqUtil: *fSUPrepSlicePosArray()*. The code example below illustrates the basic steps of an implementation. Note that the rotation matrix is handled internally by the *sSLICE_POS* class and needs only to be referenced when opening a real time event block.

Declaration of a *sSLICE_POS* real time event for each slice:

```
sSLICE_POS      asSLC[K_NO_SLI_MAX];
```

Prepare all slices of the protocol:

```
NLS_Status = fSUPrepSlicePosArray (pMrProt, pSeqLim, asSLC)
// -> check NLS_Status
```

Open an event block passing the rotation matrix:

```
fRTEBInit(&asSLC[lChronologicSlice].m_sROT_MATRIX);
fRTEI(...); // sending the real time events
...
// *** closing the event block:
NLS_Status = fRTEBFinish();
// -> check NLS_Status
```

Hint:

Internally, offcenter slice positions (FOV shifts) are realized as followed, when *sSLICE_POS.prep()* is performed:

```
//-----
-- 
// Phase increment for offcenter FOV in the phase-encoding direction
//-----
-- 
setPhaseOffCenterPE( 360.0 * getSliceOffCenterPE()
                     / ( pMrProt->sliceSeries().front().phaseFOV()
                         * (1.0 + pMrProt->kSpace().phaseOversampling()) ) );

//-----
-- 
// Phase increment for offcenter slice position in the 3D-encoding
// direction
//-----
-- 
setPhaseOffCenter3D( 360.0 * getSliceShift()
                     / ( pMrProt->sliceSeries().front().thickness()
                         * (1.0+pMrProt->kSpace().sliceOversampling()) ) );
```

RF Excitation

A simple excitation scheme in *SeqIF::run* might look like ([Figure D.2.5](#)):

```
long   lStartTime_GS      = lArbitraryTime;
long   lStartTime_RF      = lGSStartTime+sGS.getRampUpTime();
long   lStartTime_GSRefoc = sGS.getDuration();

fRTEI(lStartTime_GS      ,          0,          0, 0, 0, 0,    &sGS, 0);
fRTEI(lStartTime_RF      , &sRFNCOSet,&sRF_sinc, 0, 0, 0,          0, 0);
```

```
fRTEI(lStartTime_RF+sRF.getDuration(),&sRFNCONeg, 0, 0, 0, 0, 0, 0);  
fRTEI(lStartTime_GSRefoc, 0, 0, 0, 0, 0, &sGSRefoc, 0);
```

In the following, the necessary configuration steps are described, taking advantage of provided methods that make the sequence programmer's life easier.

NOTE: After having modified external rf pulses, it is necessary to either restart syngo or reboot the host in order for the RF database to be re-read. The RF database is read once when it is first accessed and thereafter kept in memory. (This is only true on the scanner - on a standalone IDEA, RF database changes take effect immediately.)

Support points of external rf pulses need to lie on raster points. The allowed distance between them is 1 us .. 2^16 * 0.1us in steps of 0.1 us.

Selecting and Preparing RF Pulses

First, an rf pulse object is created from the desired rf pulse class (sinc, gauss, etc.). When declaring an rf pulse, an ident string may be passed, which uniquely names the rf pulse object. Alternatively, the `.setIdent` method has to be used. It should be emphasized that each rf pulse must have a unique ident string (in contrast to the other real time events, where the setting of an ident is optional, but recommended). This string is displayed in the UI interface under the System Transmitter/Receiver tab.

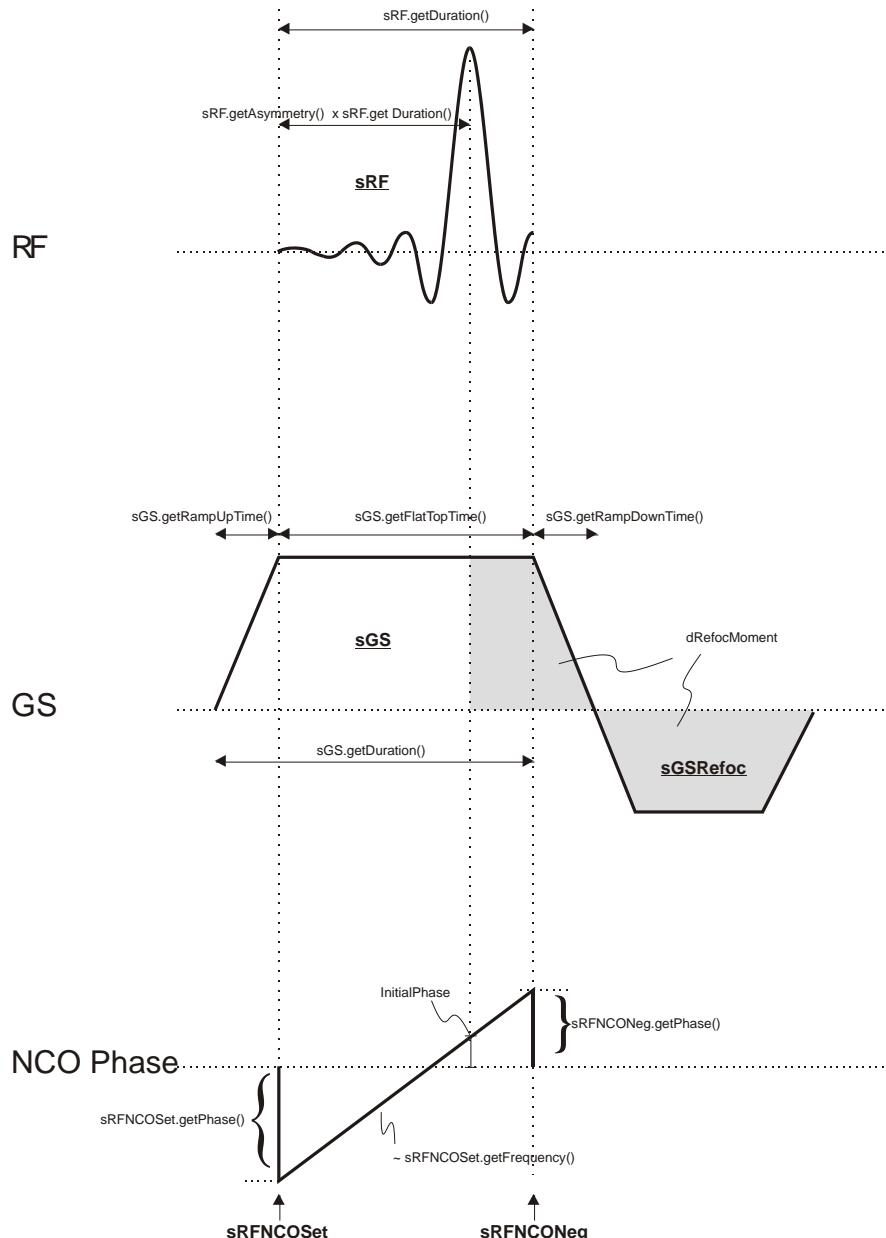


Figure D.2.5: Real time events of an excitation scheme.

The subsequent configuration will in general consist of setting the flip angle (`.setFlipAngle`), the pulse duration (`.setDuration`) and a certain slice thickness (`.setThickness`). Additional parameters are phase (`.setInitialPhase`), number of samples (`.setSamples`), and, for external pulses, the family name (`.setTypeFamilyName`).

```
sRF_PULSE_SINC sRF ("f1_temp1_ex");
...
sRF.setThickness      (dThickness);
sRF.setDuration       (2560);
sRF.setFlipAngle      (pMrProt->flipAngle());
sRF.setInitialPhase   (90);
sRF.setTypeExcitation ();
```

Another example for external pulses:

```
sRF_PULSE_EXT sRF ("se15b130ex");
...
sRF.setThickness      (dThickness);
sRF.setDuration       (2560);
sRF.setFlipAngle      (90.0);
sRF.setInitialPhase   (90);
sRF.setFamilyName     ((char *)"SE2560A90.SE90_12A2_2");
sRF.setTypeExcitation ();
```

The methods `.setTypeExcitation`, `.setTypeRefocussing` and `.setTypeUndefined` allow the addition of information about the rf pulse in terms of sequence physics. This is necessary for the sequence unit test, which needs to know whether the rf pulse in question is meant to be an excitation pulse or a refocusing pulse. The sequence simulator also depends on this information: excitation pulses reset the calculated gradient moments, refocusing pulses invert them and pulses labeled as undefined do not affect them.

Finally, the preparation of the rf pulse (`.prep`) will calculate and load its shape, and calculate the transmitter voltage and the required slice gradient amplitude. The latter values are available for further processing by the corresponding methods `.getTransmitterVoltage` and `.getGSAmplitude`.

```
If (! sRF.prep (pMrProt, pSeqExpo) ) {
    return (sRF.getNLSStatus());
}
```

Note:

Each rf pulse event owns a generic prepare method:

```
bool sRF_PULSE_EXT ::prepGeneric (
    MrProtocolData::MrProtData *pMrProt,
    SeqExpo                 *pSeqExpo)

bool sRF_PULSE_SINC::prepGeneric (
    MrProtocolData::MrProtData *pMrProt,
    SeqExpo                 *pSeqExpo)
...
...
```

If only the prepare of the base class is called (*.prep*), as in the example above, it will directly call the generic prepare of the derived pulse class. Additionally, for each pulse type a specific prepare method exists:

```
bool sRF_PULSE_EXT::prepExternal (
    MrProtocolData::MrProtData *pMrProt,
    SeqExpo                  *pSeqExpo)

bool sRF_PULSE_SINC::prepSinc (
    MrProtocolData::MrProtData *pMrProt,
    SeqExpo                  *pSeqExpo)

bool sRF_PULSE_SINC::prepSinc (
    MrProtocolData::MrProtData *pMrProt,
    SeqExpo                  *pSeqExpo,
    double                   dEmpiricalFactor)

...
```

As you see, the type specific prepare methods are sometimes provided with different interfaces, allowing additional parameters to be passed. For a detailed list of methods, please refer to the reference manual.

Important Methods of RF Pulses (base class primarily)

The following methods of class *sRF_PULSE* are of special interest for subsequent calculations (e.g. gradients). Note that valid parameters are only available after the rf pulse was prepared.

.getDuration

Duration of the rf pulse in μ sec.

.getAsymmetry

rf pulses may be created with an asymmetric shape. The asymmetry factor gives the center of the pulse relative to its duration. Accordingly, a symmetric pulse has an asymmetry factor of 0.5 (see [Figure D.2.5](#)). This value is considered for the calculation of refocusing gradients, echo time and the sequence unit test.

.getGSAmplitude

This method delivers the required slice gradient amplitude, calculated from the specified slice thickness. Note, that each rf pulse class must calculate this (safety relevant!) parameter correctly.

Using Gradient as Slice Selective Gradient

The following example demonstrates how slice selective excitation can be accomplished with only a few lines of code, exploiting the provided methods of the corresponding real time event classes:

Creation of slice and slice-refocusing gradient:

```
sGRAD_PULSE sGS      ("sGSLiSel");
sGRAD_PULSE sGSRefoc ("sGSLiRef");
```

Define the timing:

```
sGS.setRampUpTime      (1000);
// ***** NOTE: sRF.getDuration() % GRAD_RASTER_TIME ==0 required:
sGS.setDuration        (1000+sRF.getDuration());
sGS.setRampDownTime    (1000);

sGSRefoc.setRampUpTime (1000);
sGSRefoc.setDuration   (2000);
sGSRefoc.setRampDownTime (1000);
```

Prepare slice gradient with correct amplitude:

```
if (! sGS.prepAmplitude(sRF.getGSAmplitude())) /*! EGA-04; EGA-02 !*/
{
    // -> error handling
}
```

Calculate refocusing moment and prepare GSRefoc:

```
double dGSRefocMoment = - sGS.getAmplitude() * (sGS.getFlatTopTime() -
sRF.getAsymmetry() * sRF.getDuration()
+ 0.5 * GS.getRampDownTime());
if (! sGSRefoc.prepMomentumTOT(dGSRefocMoment))
{
    // -> error handling
}
```

Check required gradient performance:

(important for binary search to find soft limits of slice thickness)

```
if (!(sGS.check()) || !(sGSRefoc.check()))
{
    // slew rate(s) or amplitude(s) too high => slice thickness can not be realized
    // -> error handling
}
```

Using Negative Slice Selection Gradients

Inverting the polarity of the slice gradient is easily accomplished by the following `.setRequiredGSPolarity` method of the rf pulse class. This should be done before any further calls of `.getGSAmplitude`:

```
sRF.setRequiredGSPolarity(-1.0);
```

All other methods of `sRF_PULSE` will be affected by this setting. In our example nothing else has to be changed!

Realizing Slice Offcenter Excitation

Slice shift and pulse properties for slice offcenter excitation are handled by the `.prepSet` and `.prepNeg` methods of the `sFREQ_PHASE` class. Two instances of `sFREQ_PHASE` are used to set and reset the frequency and phase of the NCO immediately before and after sending the excitation pulse. The required frequencies and phases are calculated by `.prepSet` and `.prepNeg`, based on the slice position object and the rf pulse (which are passed as arguments).

```
sFREQ_PHASE sRFNCOSet("RFNCOSet"); // event to set frequency and phase
sFREQ_PHASE sRFNCONeg("RFNCONeg"); // event to reset frequency and phase
...
sRFNCOSet.prepSet(asSLC[1ChronologicSlice], sRF) ; // prepare to set frequency
and phase
sRFNCONeg.prepNeg(asSLC[1ChronologicSlice], sRF) ; // prepare to reset frequency
and phase
```

Internally the methods `.prepSet` and `.prepNeg` calculate the frequency and phase as follows:

sRFNCOSet.prepSet:

$$f = \gamma \cdot \Delta x \cdot G_{\text{Slice}} + 0,5$$

$$\text{Phase} = -\left(f \cdot \frac{360}{10} \cdot T_{\text{Pulse}} \cdot \text{Asym}_{\text{Pulse}}\right) + \text{Initialphase}$$

$$f = \gamma \cdot \Delta x \cdot G_{\text{Slice}} + 0,5$$

$$\text{Phase} = -\left(f \cdot \frac{360}{10} \cdot T_{\text{Pulse}} \cdot (1 - \text{Asym}_{\text{Pulse}})\right) - \text{Initialphase}$$

sRFNCONeg.prepNeg:

f:	Frequency (absolute offset to the center frequency)
Phase	Phase (relative value, added to the current phase)
γ :	Larmor constant (= <code>RFPulse.getLarmorConst()</code>)
Δx :	Slice offcenter (= <code>SlicePos.getSliceShift()</code>)
G_{Slice} :	Slice gradient amplitude (= <code>RFPulse.getGSAmplitude()</code>)
T_{Pulse} :	rf pulse duration (= <code>RFPulse.getDuration()</code>)
$\text{Asym}_{\text{Pulse}}$:	rf pulse asymmetry factor (= <code>RFPulse.getAsymmetry()</code>)
Initialphase:	rf pulse initial phase (= <code>RFPulse.getInitialPhase()</code>)

Note: `.prepNeg` will set the frequency back to zero. The frequency calculation in the first formula is just needed for the phase calculation in the second formula.

RF Pulse Amplitudes and Valid Flags

The rf pulse transmitter amplitudes shown in the user interface (the prepare function is running in 'ContextPrepareForBinarySearch') are calculated under the assumption that the reference amplitude of the previous adjustment is still valid. After the user has pressed the 'Apply' button, the actual reference amplitude is determined, and now the true rf pulse amplitudes can be calculated (the prepare function is running in 'ContextPrepareNormal'). However, the user interface provides a mechanism to explicitly set the rf amplitude of each pulse to a fixed value (thereafter displayed with an exclamation mark). Internally, the so-called "valid flag" of a rf pulse is set, indicating that it may not be changed any more.

The **reference amplitude** is the input voltage to the transmitter (RFPA) required to produce a 180 degree rotation for a 1 msec rectangular pulse. All rf pulses are scaled to this. External pulses should be scaled such that the peak amplitude of the support is 1.0. Within the external file, the pulse should be defined independent of time/duration, as it is specified in the sequence. This allows the same set of support points to be used with different durations and therefore different bandwidths.

The transmitter amplitudes of rf pulses are calculated on the host computer only. The MARS computer receives them as an array together with the protocol (for safety reasons). The total number of rf pulses which can be applied within a sequence is 128.

Setting Constant Phase for a RF Pulse

Setting a constant phase for a rf pulse is achieved by the methods `.increasePhase` and `.decreasePhase`. These methods are identical for the `sRF_PULSE` class and the `sREADOUT` class and are described in the next section ('Data Acquisition').

Data Acquisition

Now we add a data acquisition scheme to the example above as depicted in [Figure D.2.6](#). It consists of an `sADC` event, a readout gradient `sGRO` preceded by a prephasing gradient `sGRPrephase`, and the phase events `sADCNCOSet` and `sADCNCONeg`. The start time calculations demonstrate how a flexible timing may be implemented: the timing is adapted relative to certain event parameters. The `lShiftADC` variable considers a shift of the ADC with respect to the start time of the read out gradient. This is needed, for example, if the duration of the ADC (columns * dwelltime) is not equal to the plateau time of the readout gradient (a multiple of 10 microseconds, due to the fixed gradient raster time).

```
long lStartTime_GRO          = lStartTime_GSRefoc + sGSRefoc.getTotalTime();
long lStartTime_GRPrephase = lStartTime_GRO - sGRPrephase.getTotalTime();
long lStartTime_ADC          = lStartTime_GRO + sGRO.getRampUpTime() + lShiftADC

fRTETI(lStartTime_GRPrephase, 0, 0, 0, 0, &sGRPrephase, 0, 0);
fRTETI(lStartTime_GRO,           0, 0, 0, 0, &sGRO, 0, 0);
fRTETI(lStartTime_ADC, &sADCNCOset,      0, &sADC, 0, 0, 0, 0);
fRTETI(lStartTime_ADC+sADC.getRoundedDuration(), &sADCNCONeg, 0, 0, 0, 0, 0, 0);
```

The following subsections demonstrate how to prepare and configure these real time events.

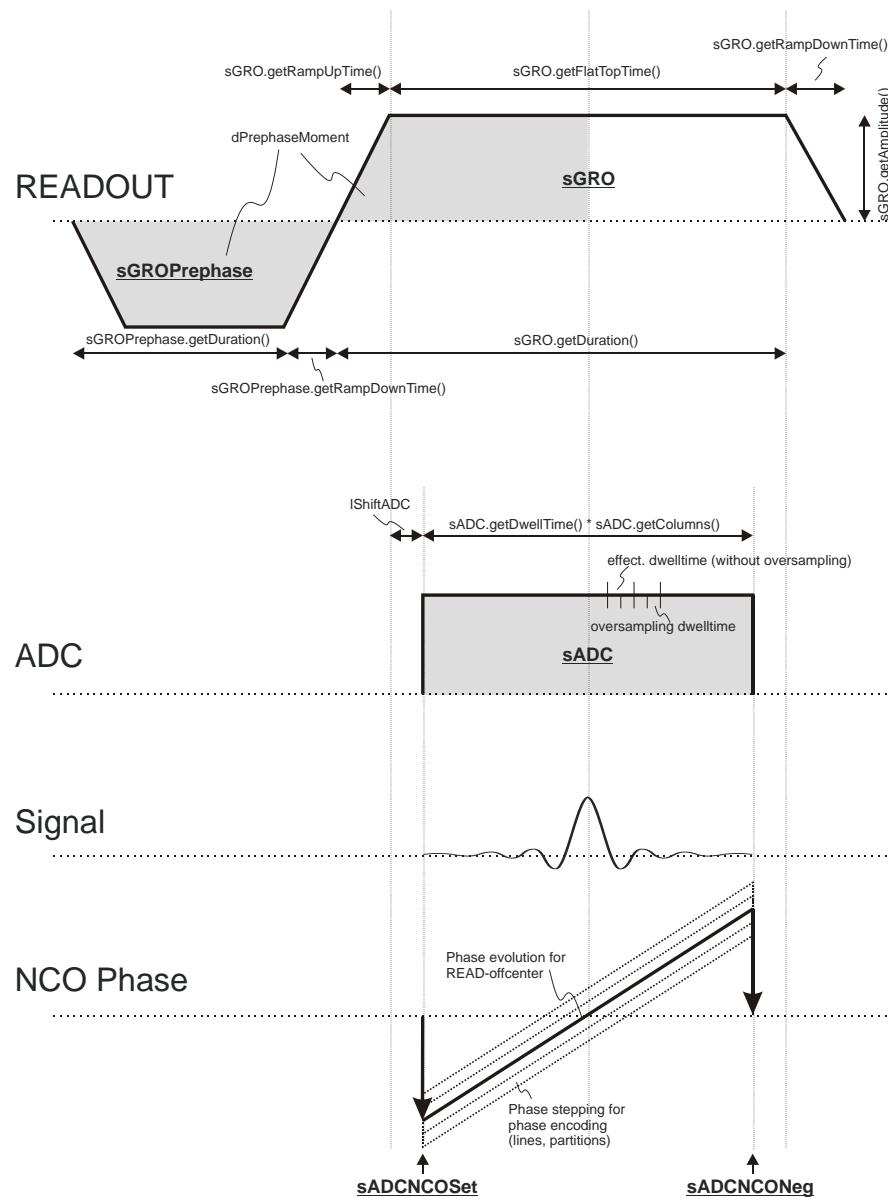


Figure D.2.6: Real time events of a data acquisition scheme.

Configuring and Preparing the ADC

An ADC event is created from the `sREADOUT` class. The ADC is prepared by passing the base resolution and the effective dwelltime (`.prep(const long lColumns, const long lDwellTime)`). At this point, the `Mdh` entry denoting where the central ADC column is intended may be specified:

```
sREADOUT sADC ("sADC");

sADC.prep( pMrProt->kSpace().baseResolution(),
            (long) (pMrProt->rxSpec().effDwellTime(pSeqLim->
                getReadoutOSFactor())[0] + 0.5) );
m_ADC.Mdh.setKSpaceCentreColumn( pMrProt->
            kSpace().baseResolution()/2 ); // typical value
```

Since the return type of the `.effDwellTime` method is double, it must be cast to long type (after adding 0.5 for safer rounding). Note that the `.eff DwellTime` is an array, which allows mixed bandwidth in multiecho sequences.

Note:

The base resolution and the effective dwelltime are specified without oversampling (true number of samples = `base_resolution * oversampling`, real dwelltime = effective dwelltime/oversampling). In fact, the oversampling factor is not a protocol parameter and `sADC.prep` will usually be the only place in the sequence where it is needed as an argument. The oversampling factor can only be specified through its default value (typical value: 2) in the `SeqLim` object. However, the protocol holds the real (or physical) dwelltime (without oversampling).

Due to hardware restrictions, the ADC and the readout gradient operate with different time rasters (i.e. dwelltime and gradient raster time). For many applications, it is important to center the ADC and readout plateau. This can be achieved as in the following lines of code, resulting in time delay `lShiftADC` between the start time of the ADC and the read out gradient:

```
long lRequiredFlatTop = fSDSRoundUpGRT(sADC.getDuration());
// fSDSRoundUpGRT() rounds up to the next multiple
// of 10us (GRT = gradient raster time)

// clever (avoids many problems later):
if (lRequiredFlatTop % (2*GRAD_RASTER_TIME))
{
    lRequiredFlatTop += GRAD_RASTER_TIME;
}
long lShiftADC = (long) ((lRequiredFlatTop-sADC.getDuration())/2);
```

Setting the NCO phase is done similarly to rf pulse events and is treated in a subsection below (Realizing 2D-/3D-Offcenter Measurements).

Using Gradient as Readout Gradient

Following the example above, a readout gradient and its prephasing gradient are set and prepared.

```
sGRAD_PULSE_RO      sGRO("sGRO");
sGRAD_PULSE_RO      sGROPrephase("sGROPrephase");

sGRO.setRampUpTime (1000);
sGRO.setDuration    (1000+lRequiredFlatTop );
// lRequiredFlatTop: see ADC code example above
sGRO.setRampUpTime (1000);

sGRPrephase.setRampUpTime   (1000);
sGRPrephase.setDuration     (2000);
sGRPrephase.setRampDownTime (1000);

if (!sGRO.prepRO(pMrProt,sADC.getDwellTime()))           /* EGA */
{
    // error handling
}
double dPrephaseMoment = - sGRO.getAmplitude() * ( 0.5 * sGRO.getRampUpTime() +
lShiftADC
+ sADC.Mdh.getKSpaceCentreColumn()* sADC.getDwellTime()/1000.0 );

if (!sGRPrephase.prepMomentumTOT(dPrephaseMoment) )
{
    // error handling
}
if (!sGRO.check() || !sGRPrephase.check())
{
    // FOV can not be realized.
    // error handling
}
```

The important methods here are:

.prepRO (MrProt *pMrProt, const double dDwellTimeWithoutOS)

Calculate and set the readout amplitude. This is done as follows:

```
setAmplitude(1.0E9/(dFOVRO*dGAMMA*dDwellTimeWithoutOS));
```

.prepMomentumTOT(const double dMomentum)

Calculate and set the prephasing gradient amplitude for the specified momentum and the previously set duration.

.check

Check that the gradients amplitudes and slew rates do not exceed the system specifications. This is particularly important so that the user interface can determine the graphically displayed parameter ranges (soft limits).

Using a Negative Readout-Gradient Amplitude

In order to invert the readout gradient (e.g. in EPI), the method `.prepAmplitude(-sGRO.getAmplitude())` is called. This must be done after `.prepRO` was called and before the prephasing moment is calculated. Additionally, the ICE program must be told to mirror this line (relative to the center column) by setting the `MDH_REFLECT` flag.

```
if (! sGRO.prepAmplitude(-sGRO.getAmplitude())) /*! EGA-01 !*/
{
    error handling
}
sADC.Mdh.addToEvalInfoMask(MDH_REFLECT);           /*! EGA-01 !*/
```

Realizing 2D-/3D-Offcenter Measurements

The NCO phase events `sADCNCOSet` and `sADCNCNeg` for the ADC are prepared for each line and thus the `.prepSet` and `.prepNeg` methods are called inside `SeqIF::run` immediately before the real time events are played out:

```

long lEchoLine      = pMrProt->kSpace().echoLine();
// Line in k-space center

long lEchoPartition = pMrProt->kSpace().echoPartition();
// Partition in k-space center

sFREQ_PHASE      sADCNCOSet ("sADCNCOSet");
sFREQ_PHASE      sADCNCNeg ("sADCNCNeg");

sADCNCOSet.prepSet (asSLC[1Slice],   sADC,    sGRO,lLine-lEchoLine,   lPartition-
lEchoPartition);
                                         /*! EGA-06 !*/
sADCNCNeg.prepNeg (asSLC[1Slice],   sADC,    sGRO,lLine-lEchoLine,   lPartition-
lEchoPartition);
                                         /*! EGA-06 !*/

```

This is how it works:

```

sADCNCOSet.prepSet (sSLICE_POS &SlicePos,
                     sREADOUT   &Readout,
                     sGRAD_PULSE &GradPulseReadout,
                     const long  lLineNumber,
                     const long  lPartitionNumber):

```

$$f = \gamma \cdot \Delta_{RO} \cdot G_{RO} + 0,5$$

$$\text{Phase} = -f \cdot \frac{360}{10^6} \cdot T_{ADC} \cdot k_{ADCCentre}$$

$$\Delta\text{Phase} = \Delta_{PE} \cdot \text{Line} + \Delta_{3D} \cdot \text{Partition}$$

will set the frequency to f and the phase to $\text{Phase}+\Delta\text{Phase}$ according to the following formula.

γ :	Larmor constant (= <code>Readout.getLarmorConst()</code>)
Δ_{RO} :	Slice offcenter readout (= <code>SlicePos.getSliceOffCenterRO()</code>)
Δ_{PE} :	Offcenter phase (= <code>SlicePos.getSliceOffCenterPE()</code>)
Δ_{3D} :	Offcenter phase (= <code>SlicePos.getSliceOffCenter3D()</code>)
G_{RO} :	Readout amplitude (= <code>GradPulseReadout.getAmplitude()</code>)
T_{ADC} :	ADC duration (= <code>Readout.getDuration()</code>)

```

kADCCentre: relative ADC center column
( = Readout.Mdh.getKSpaceCentreColumn() /
Readout.getColumns() )

Line: current ky line ( = lLineNumber )

Partition: current kz line ( = lPartitionNumber )

sADCNCOSet.prepNeg (sSLICE_POS &SlicePos,
                     sREADOUT &Readout,
                     sGRAD_PULSE &GradPulseReadout,
                     const long lLineNumber,
                     const long lPartitionNumber);

```

will set the frequency back to zero and the phase to *Phase-ΔPhase* according to the following formula.

$$f = \gamma \cdot \Delta_{RO} \cdot G_{RO} + 0,5$$

$$\text{Phase} = -f \cdot \frac{360}{10^6} \cdot (T_{\text{rounded, ADC}} - T_{\text{ADC}} \cdot k_{\text{ADCCentre}})$$

$$\Delta\text{Phase} = \Delta_{PE} \cdot \text{Line} + \Delta_{3D} \cdot \text{Partition}$$

γ :	Larmor constant (= Readout.getLarmorConst())
Δ_{RO} :	Slice offcenter readout (= SlicePos.getSliceOffCenterRO())
Δ_{PE} :	Offcenter phase (= SlicePos.getSliceOffCenterPE())
Δ_{3D} :	Offcenter phase (= SlicePos.getSliceOffCenter3D())
G_{RO} :	Readout amplitude (= GradPulseReadout.getAmplitude())
T_{ADC} :	ADC duration (= Readout.getDuration())
$T_{\text{rounded, ADC}}$:	Rounded ADC duration (= Readout.getRoundedDuration())
$k_{\text{ADCCentre}}$:	relative ADC center column (= Readout.Mdh.getKSpaceCentreColumn() / Readout.getColumns())
Line:	current ky line (= lLineNumber)
Partition:	current kz line (= lPartitionNumber)

Setting Constant Phase for an ADC

To demonstrate the handling of the ADC phase, we look at the RF spoiling implementation of the FLASH template sequence. The actual phase setting is accomplished by using the `.increasePhase (double)` and `.decreasePhase (double)` methods of the rf pulse and ADC phase objects respectively. It is important to note that this constant phase is set after the `.prepSet` and `.prepNeg` methods are called:

```
sRFNCOSet.prepSet (asSLC[1ChronologicSlice], sRF) ;
/*! EGA-05 !*/
sRFNCONeg.prepNeg (asSLC[1ChronologicSlice], sRF) ;
/*! EGA-05 !*/

sADCNCOSet.prepSet (asSLC[1ChronologicSlice], sADC, sGradReadout, 1Line-
1EchoLine,1Partition-1EchoPartition) ;      /*! EGA-06 !*/
sADCNCONeg.prepNeg (asSLC[1ChronologicSlice], sADC, sGradReadout, 1Line-
1EchoLine,1Partition-1EchoPartition) ;      /*! EGA-06 !*/

// For every RF-pulse the phase difference to the previous pulse is incremented
// by an increasing multiple of 50 deg (= RFSPOIL_INCREMENTdeg)
dRFSpoilIncrement += RFSPOIL_INCREMENTdeg ;
dRFSpoilPhase     += dRFSpoilIncrement ;

// keep phase smaller than 20 Pi (= RFMAXPHASEdeg)
dRFSpoilPhase = fmod(dRFSpoilPhase, (double) RFMAXPHASEdeg);
dRFSpoilIncrement = fmod(dRFSpoilIncrement, (double) RFMAXPHASEdeg);

sRFNCOSet.increasePhase (dRFSpoilPhase);
sRFNCONeg.decreasePhase (dRFSpoilPhase);
sADCNCOSet.increasePhase (dRFSpoilPhase);
sADCNCONeg.decreasePhase (dRFSpoilPhase);
```

Filling the Measurement Data Header (Mdh)

The Mdh (measurement data header) plays a key role in the concept that the ICE program performs the image reconstruction without information about the actual loop structure of the sequence. Instead, the Mdh header of the incoming measurement data is evaluated line by line. This requires that each acquired line is labeled via Mdh, defining which slice, partition, repetition, etc. it belongs. Further information (e.g. the k-space center column) may be necessary for more sophisticated operations (partial fourier, ramp sampling, etc.).

In addition, the reconstruction process is controlled to some extend by Mdh flags. An example is the `.Mdh.setPhaseFT(true)` method, which will notify the ICE program that all lines in a slice are acquired (including signal averaging by an additional acquisitions loop) and thus the phase FT can be initiated.

Note that the Mdh entries are also accessed by the sequence unit test, e.g. for checking calculated echo timing against the KSpaceCentreColumn entry of the Mdh.

The Mdh class is a member of the `sREADOUT` class and is accessed via the `sADC` object as demonstrated in the following list of important Mdh entries. The ICE program relies on these values to properly sort each ADC line into the raw data matrices:

Line sorting (loop counters):

```
sADC.Mdh.setCrep (lRepetition);           // the current repetition
sADC.Mdh.setCphs (lPhase);                 // the current phase
sADC.Mdh.setCacq (lAverage);               // the current aquisition
sADC.Mdh.setCscl (lChronologicSlice);     // the current slice
sADC.Mdh.setClin (lLine);                  // the current line
sADC.Mdh.setCpar (lPartition);              // the current partition
sADC.Mdh.setKSpaceCentreLineNo(lCtrLin);   // the central line
sADC.Mdh.setKSpaceCentrePartitionNo(lCtrPart); // the central partition
```

Information on k-space sampling schemes:

```
.setKSpaceCentreColumn      (int value);
.setFirstScanInSlice        (bool flag);
.setLastScanInSlice         (bool flag);
.setLastScanInConcat        (bool flag);
.setLastScanInMeas          (bool flag);
```

Controlling the reconstruction progress:

EvalInfoMask:

```
void setEvalInfoMask        (unsigned long ulEvalBit);
void addToEvalInfoMask      (const unsigned long ulEvalInfoMask)
void deleteFromEvalInfoMask (const unsigned long ulEvalInfoMask)
void toggleEvalInfoMask    (const unsigned long ulEvalInfoMask)
```

The `EvalInfoMask` is a collection of bitflags to control the reconstruction process. For example, setting the `MDH_ONLINE` flag will pass the ADC line to the so called online reconstruction process. Most flags can be easily accessed by using corresponding `.setXXX` and `.isXXX` functions: `.setKSpaceCentreLine(true)` has the same result as `.addToEvalInfoMask(MDH_ZEROINE)`. The latter flag can be requested by `.isKSpaceCentreLine()`.

Phase Encoding

For the example above, add a phase encoding gradient ([Figure D.2.7](#)):

```

sGRAD_PULSE_PE    sGPE("sGPE");

long   lPEDuration      = sGRPrephase.getDuration();
long   lPERampUpTime    = sGRPrephase.getRampUpTime();
long   lPERampDownTime = sGRPrephase.getRampDownTime();

// prepare largest line in k-space (lEchoLine-1 is k-space centre line) ...
if(! sGPE.prepPE (lPERampUpTime, lPEDuration , lPERampDownTime,
                  pMrProt, SEQ::DIR_ASCENDING, 0.0,
                  lLinesToMeasure - lEchoLine -1) /*! EGA-03; EGA-01 !*/
{
    // -> error handling
}

// check for gradient specifications
if(! sGPE.check() )
{
    // -> error handling
}

// ... and again for lower end of k-space
if(! sGPE.prepPE (lPERampUpTime, lPEDuration, lPERampDownTime,
                  pMrProt, SEQ::DIR_ASCENDING, 0.0, -lEchoLine) )
/*! EGA-03; EGA-01 !*/
{
    // -> error handling
}

// check for gradient specifications
if(! sGPE.check() )      // error handling

// -----
// fSEQRun():
// -----
long   lStartTime_GPE = lStartTime_GRO - sGRPrephase.getTotalTime();
long   lEchoLine       = pMrProt->kSpace().echoLine();
// line in the K-Space center

```

```

// set phase gradient to current line
if(! sGPE.prepPE (pMrProt,lLine-lEchoLine ) )
{
    // -> error handling
}
...
// insert into real time event block
RTEI (lStartTime_GPE      , 0, 0, 0, &sGPE, 0, 0, 0);

```

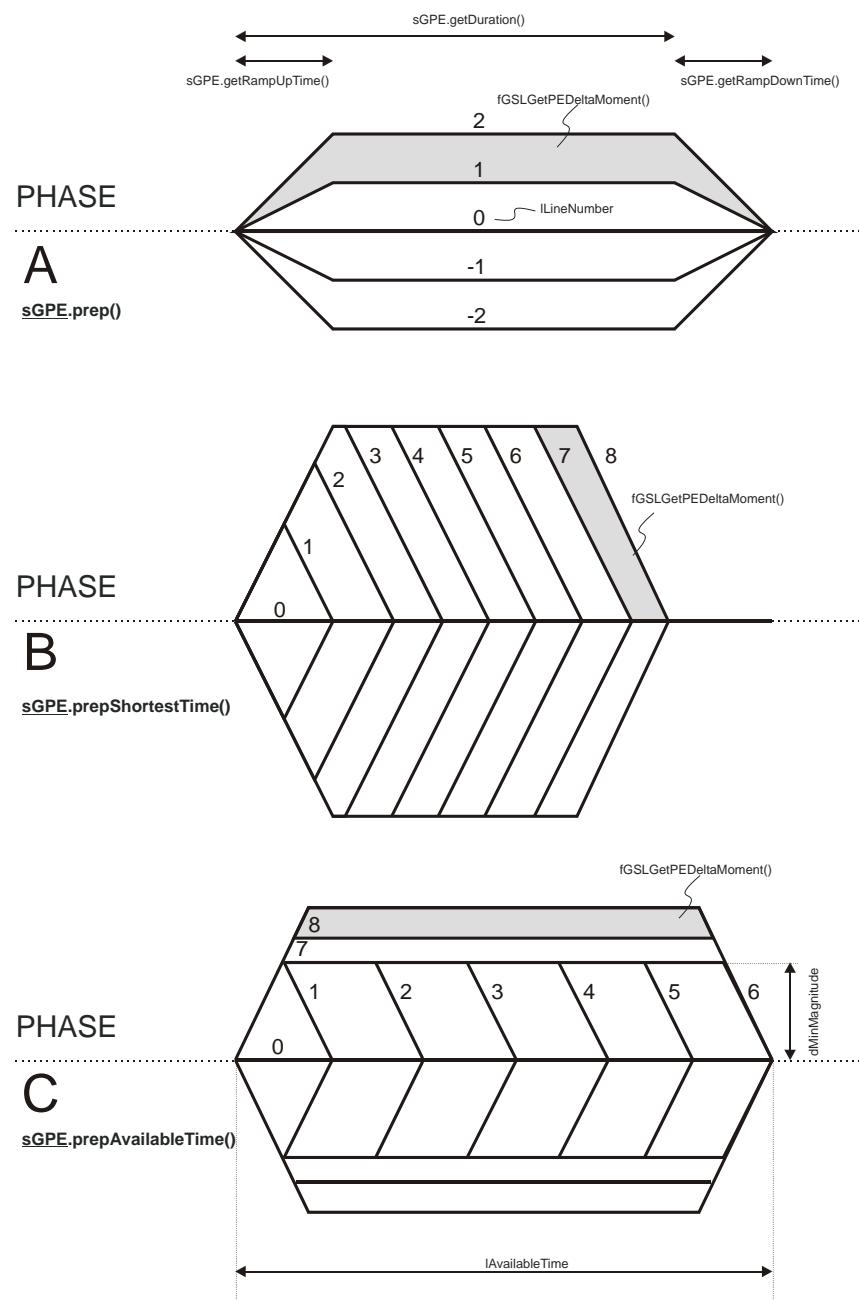


Figure D.2.7: Three different schemes for phase encoding tables.

Traditional phase encoding schemes use constant gradient ramp times and durations for all entries. The recommended approach to achieve this is to prepare the phase encoding event first in *prepare* to set the ramp times and other general parameters. The line number given as an argument of *.prepPE* is defined to be zero in the k-space center: In this way, only a 'brief' *.prepPE* is necessary in *run* or *runKernel* to calculate the specific amplitude of each encoding step inside the lines loop.

```

bool sGRAD_PULSE_PE::prepPE (const long lRampUpTime,
                             const long lDuration,
                             const long lRampDownTime,
                             MrProt *pMrProt,
                             const SEQ::Direction eDirection,
// ASCENDING or DESCENDING
                             const double dOffsetMomentum,
                             const long lLineNumber);

bool sGRAD_PULSE_PE::prepPE (
    MrProtocolData::MrProtData *pMrProt,
    const long                 lLineNumber);

```

Notes:

□ The required gradient moment is determined by

*LineNumber * PEDeltaMoment.*

In this context, there is a library function

```
double fGSLGetPEDeltaMoment (
    MrProtocolData::MrProtData *pMrProt);
```

which returns the *PEDeltaMoment*:

```
PEDeltaMoment = 1/(gamma*FOV_PE*(1+PhaseOversamplingFactor))
```

□ **Direction = Ascending:** the gradient moment increases when incrementing the line number.

Direction = Descending: the gradient moment decreases when incrementing the line number (e.g. used in spin echo sequences, when switching the phase encoding table before the refocusing pulse).

Additionally, the following variants of *.prepPE* exist:

.prepPEShortestTime

This method will prepare the table to be as short a duration as possible ([Figure D.2.7](#): B). Each phase encoding gradient step will be driven with maximum slew rate, which allows for shortest possible timings and high gradient amplitudes near the k-space center.

```

bool sGRAD_PULSE_PE::prepPEShortestTime (
    MrProtocolData::MrProtData *pMrProt,
    const SEQ::Direction eDirection,
```

```

        const double           dOffsetMomentum,
        const long            lLineNumber);

bool sGRAD_PULSE_PE::prepPEShortestTime (
    MrProtocolData::MrProtData *pMrProt,
    const long lLineNumber);

```

.prepPEUseAvailableTime

This method will keep the gradient amplitude as low as possible by using a longer duration ($< lAvailableTime$) rather than increasing the amplitude ([Figure D.2.7: C](#)). As it is not reasonable to use arbitrary low amplitudes, a minimum gradient amplitude ($dMinMagnitude$) has to be defined (due to the digital filter). This approach is attractive in situations where the phase encoding time is not critical and thus lower gradient noise and reduced physiological stimulation may be gained in return.

```

bool sGRAD_PULSE_PE::prepPEUseAvailableTime (
    MrProtocolData::MrProtData *pMrProt,
    const SEQ::Direction      eDirection,
    const double               dOffsetMomentum,
    const long                lLineNumber,
    const long                lAvailableTime,
    const double              dMinMagnitude)

bool sGRAD_PULSE_PE::prepPEUseAvailableTime (
    MrProtocolData::MrProtData *pMrProt,
    const long                lLineNumber,
    const long                lAvailableTime)

```

Using Gradient as Phase Encoding Gradient for 3D

Phase encoding for 3D works is completely analogous to 2D with methods available for all the stepping schemes and the moment calculation:

```

bool sGRAD_PULSE_3D::prep3D      (
    const long          lRut,
    const long          lDuration,
    const long          lRdt,
    MrProtocolData::MrProtData *pMrProt,
    const SEQ::Direction eDirection,
    const double         dOffsetMomentum,
    const long          lPartitionNumber)

bool sGRAD_PULSE_3D::prep3D      (
    MrProtocolData::MrProtData *pMrProt,
    const long          lPartitionNumber)

bool sGRAD_PULSE_3D::prep3DShortestTime      (
    MrProtocolData::MrProtData *pMrProt,
    const SEQ::Direction eDirection,
    const double         dOffsetMomentum,
    const long          lPartitionNumber)

bool sGRAD_PULSE_3D::prep3DShortestTime      (
    MrProtocolData::MrProtData *pMrProt,
    const long          lPartitionNumber)

bool sGRAD_PULSE_3D::prep3DUseAvailableTime (
    MrProtocolData::MrProtData *pMrProt,
    const SEQ::Direction eDirection,
    const double         dOffsetMomentum,
    const long          lPartitionNumber,
    const long          lAvailableTime,
    const double         dMinMagnitude)

bool sGRAD_PULSE_3D::prep3DUseAvailableTime (
    MrProtocolData::MrProtData *pMrProt,
    const long          lPartitionNumber,
    const long          lAvailableTime)

// calculates DeltaMoment for partition:
double fGSLGet3DDeltaMoment(MrProt *pMrProt);

```

Traveling k-space using Gradient Moments

In this section, a list of helpful methods to achieve common gradient parameters is provided.

Specifying individual maximum amplitude/minimum rise times

For each gradient pulse, a specific maximum gradient amplitude (mT/m) and minimum rise time ($\mu\text{s}/(\text{mT/m})$) can be set. The default values after initialization are the nominal maximum amplitude and absolute minimum rise time of the GPA.

```
void sGRAD_PULSE::setMaxMagnitude (const double dMaxMagnitude);
const double sGRAD_PULSE::getMaxMagnitude (void ) const;

void sGRAD_PULSE::setMinRiseTime (const double dMinRiseTime);
const double sGRAD_PULSE::getMinRiseTime (void ) const;
```

Realizing specific gradient moments

Several methods are available for preparation of a specific gradient moment, based on either the total gradient time (= TOT, i.e. including ramp times), or during the plateau time (= FLT, flat top).

Two methods set the calculated amplitude and rise time without regarding restrictions (maximum amplitude, minimum rise time). In these cases, the gradient must be checked later with the `.check` method.

```
// Rut: ramp up time
// Rdt: ramp down time
bool sGRAD_PULSE::prepMomentumTOT (const double dMomentum);
bool sGRAD_PULSE::prepMomentumTOT (const long lRut,
                                    const long lDuration,
                                    const long lRdt,
                                    const double dMomentum);

bool sGRAD_PULSE::prepMomentumFLT (const double dMomentum);
bool sGRAD_PULSE::prepMomentumFLT (const long lRut,
                                    const long lDuration,
                                    const long lRdt,
                                    const double dMomentum);
```

In contrast, the following methods do consider these gradient restrictions. If the desired values cannot be realized, `FALSE` is returned. The resulting gradient pulses are symmetric in terms of equal ramp up and ramp down times.

To set `dTotalMomentum` within `dAvailableTime`:

```
bool sGRAD_PULSE::prepSymmetricTOTExactMomentum (
    const double dTotalMomentum,
    const double dAvailableTime);

bool sGRAD_PULSE::prepSymmetricTOTExactMomentum (
    const double dTotalMomentum,
    const double dAvailableTime,
    const double dUseMinMagnitude);
```

```
bool sGRAD_PULSE::prepSymmetricFLTExactMomentum (
    const double dFlatTopMomentum,
    const double dAvailableTime);
```

To achieve the maximum moment within dAvailableTime:

```
bool sGRAD_PULSE::prepSymmetricTOTMaxMomentum (
    const double dAvailableTime,
    const double dSign);
bool sGRAD_PULSE::prepSymmetricFLTMaxMomentum (
    const double dAvailableTime,
    const double dSign);
```

If the available time is insufficient, *FALSE* is returned and a *NLS_STATUS* code (*GSL_GRAD_AREA_NOT_POSSIBLE*) is set.

To realize *dTotalMomentum* with the shortest possible time (subject to the gradient raster time):

```
bool sGRAD_PULSE::prepSymmetricTOTShortestTime (
    const double dTotalMomentum);
bool sGRAD_PULSE::prepSymmetricFLTShortestTime (
    const double dFlatTopMomentum);
```

Overlapping gradient ramps

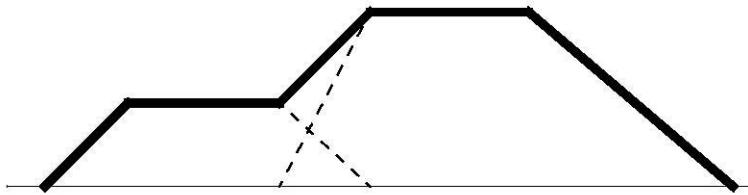


Figure D.2.8: For overlapping gradients the magnitude is summarized

Consider the case of two overlapping gradients on one logical axis as illustrated above. The ramp down of the first gradient and the ramp up of the second gradient coincide, resulting in a flatter effective ramp between the two. This must be considered when applying the `.check` method to the gradient events; the check might fail for an individual ramp due to exceeding slew rate or stimulation limits, but the effective ramp may succeed!

For this reason, the check method of gradient events has two optional parameters, which allow passage of the amplitudes of the overlapping gradients:

```
bool check (double dAmplitudeLeftNeighbor = 0.0,
            double dAmplitudeRightNeighbor = 0.0);
```

Practical Aspects of Implementation

Definition of gradient duration

The gradient duration is defined as ramp up time plus the plateau time. The ramp down time is **not** included.

Gradient ramp down outside the event block

Ramp Outside Event Block

It is possible to have a gradient ramp down outside the event block. Actually, this will always happen if the final event of an event block is a gradient pulse. Since the duration of a gradient pulse is defined by ramp up time plus plateau time (excluding ramp down time), the event block length in this case is determined by gradient start time plus duration. The beginning of the gradient ramp down will then coincide with the beginning of the subsequent event block! In previous versions care had to be taken in case the subsequent event block had a different slice orientation. In this case, the ramp down from the previous event block could be mapped to an ‘arbitrary’ gradient axis. Note that this is NOT an issue anymore, since the gradient events are calculated directly inside the fRTEI call (or `.run()` for that matter); i.e. subsequent changes of the rotation matrix have no effect on already inserted RT events. To avoid ramp down gradients outside the event block, simply insert an empty event with a start time that coincides with the end of the ramp down gradient:

```

lDurationEventBlock = sGRAD.getStartTime() + GRAD.getDuration() +
sGRAD.getRampDownTime();
// note: sGRAD.getStartTime() requires sGRAD.setStartTime(),
// which is not mandatory
fRTEI(lDurationEventBlock, 0, 0, 0, 0, 0, 0, 0);

```

Maximum number of overlapping gradients on one (logical) axis

Overlapping Gradients

It is possible for a maximum of two gradients to overlap in time on one logical gradient axis. Their gradient amplitudes are simply added. This might occur due to an additional gradient ramp down from the previous event block.

Gradients may not overlap with HALT-Bit

At the time the measurement is stopped, e.g. by a *SYNC_PHYSIO_HALT* event, all gradients must be down at zero amplitude.

Gradient raster time

Gradient timing must be rounded to a fixed raster time, defined as *GRAD_RASTER_TIME*. Helpful functions are *fSDSRoundUpGRT(ITime)* and *fSDSRoundDownGRT(ITime)*, which will round up/down *ITime* to a multiple of the gradient raster time. This rounding must be considered when calculating RF, NCO or ADC timings, e.g. centering the ADC and the readout gradient.

Manual coil tuning

lead/hold time

Usually, the coil tuning bits are set automatically by *libRT*. While in previous versions it was necessary to set the coil bit manually in the previous event block if the time between the beginning of the event block and the start time of an rf pulse was smaller than the required lead time, the *libRT* can handle this situation as well.

Nevertheless, you can always check for the required lead (tune) and hold (detune) time via *libSeqSysProp::getCoilCtrlLead* and *libSeqSysProp::getCoilCtrlHold*.

Minimum times between rf pulses and Readout

RX4PROXY

Minimum times between RF Pulses and Readout events as well as between two Readout events are provided by the *SysProperties* class (defined in */n4/pkg/MrServers/MrImaging/libSeqSysProp/SysProperties.h*). It has the following methods. The given implementation gives an idea about the current values in *NUMARIS/4* (may be subject of change in future releases).

```
int32_t getMinDurationBetweenReadoutAndRFPulse (double dRealDwellTime);
```

```
int32_t getMinDurationBetweenRFPulseAndReadout ();
```

```
double getMinDurationBetweenReadoutAndReadout (double dRealDwellTime) ;  
  
int32_t getMinDurationBetweenFrequencyPhaseEvents () ;
```

Note: *dRealDwellTime* denotes the dwell time with oversampling. If the distance between two readouts with different bandwidths is requested, the dwelltime of the first readout must be passed in the argument.

Two FREQ_PHASE events can be sent with 0 time distance if they can be merged, i.e. if they have the same frequency.

Critical size of real time event blocks

The processing time of the real time event blocks must be faster than the execution time of the previous event block if they are to be played out in real time. A large number of very short event blocks might fail in this respect due to the overhead that is caused just by creating and processing a new event block. Vice versa, excessively long and complex event blocks might be fatal as well. In both cases, the sequence will abort with a FIFO underflow error. It is not possible to give any quantitative thresholds, as many variables are involved.

E.3 Beyond the Timing Table

Reordering

A sequence must acquire a line of data (an ADC event) for every line and partition of k-space. The ADC object itself contains the complete data required for the read out direction of k space. To sample the data for the whole k-space, a nested loop over lines and partitions is required:

```
long lLin;
long lPar;
for (lLin = 0; lLin < lLinesToMeasure; lLin++) {
    for (lPar = 0; lPar < lPartitionsToMeasure; lPar++) {
        // Execute Kernel for raw data line
        // defined by lLin and lPar
        ...
    }
}
```

Reordering

Many sequences do not measure lines and/or partitions in linear ascending order but in a different order. Also, more flexible loop structures can be desirable (e.g. changing the loop hierarchy from line-in-partition to partition-in-line). The mechanism which handles the order of line and partition indices (or numbers) and which ensures that each line/partition index combinations appears only once is called "reordering".

The line- and partition-loop can be replaced by a single loop over a *reordering index*. The line- and partition-index is then a function of the reorder index:

```
long lReoId;
for (lReoId = 0; lReoId < lNumReorderIndices; lReoId++) {
    lLin = fGetReorderedLine      (lReoId);
    lPar = fGetReorderedPartition (lReoId);

    // Execute Kernel for raw data line
    // defined by lLin and lPar
    ...
}
```

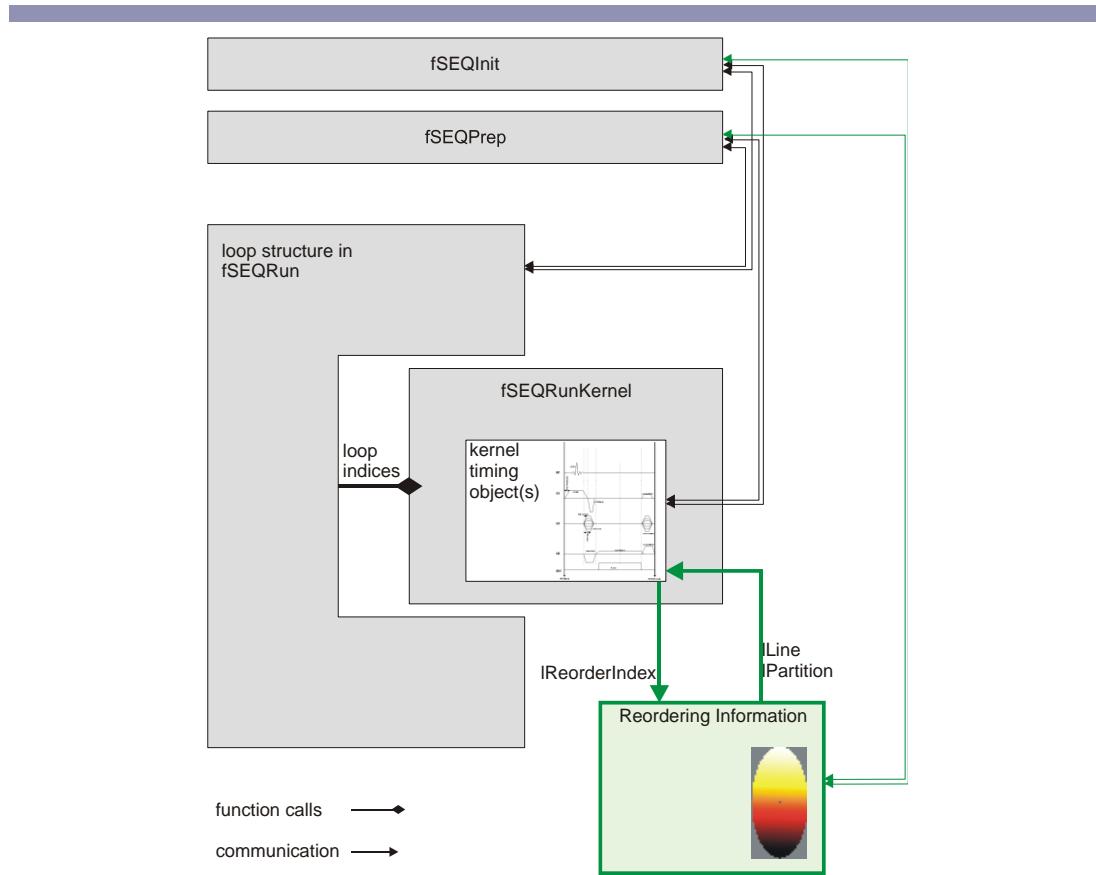


Figure D.3.1: Adding a Reordering Object to the Sequence Library:

The reordering information is calculated and stored by a separate software component "Reordering Information". The kernel timing objects get their data using a standard interface. The reordering object is configured and prepared mainly by sequence prepare(...) and initialize(...).

Base Class Reorderinfo

ReorderInfo

This is the standard interface that is used to calculate reordering schemes. It provides basic schemes for non-segmented sequences and handles storage and access of reordering data. A table is used to map from the reorder index to the respective line and partition index (and additional information).

The base class ReorderInfo offers the following features:

- Various reordering schemes for non-segmented sequences
- Elliptical scanning
- "Private" Partial Fourier (PF) factors
- Support of sequence-check(...)

The following example illustrates how the class ReorderInfo can be implemented into a sequence.

1. Create an instance of the class ReorderInfo:

```
ReorderInfo REOInfo;
```

2. Define the loop hierarchy and the line/partition reordering:

```
REOInfo.setReorderMode (
    PAR_IN_LIN,
    LINEAR_ASCENDING,
    LINEAR_ASCENDING
);
```

3. Initiate the reordering calculation:

```
if (!REOInfo.reorder3dE (rMrProt, rSeqLim)) {
    // Error handling
    ...
}
```

4. Replace `rMrProt.kSpace()` references for information about PE/3D by references to REOInfo throughout the sequence, according to the following table:

<code>rMrProt.kSpace()</code>	ReorderInfo
lLine/lPartition loops	lREOIndex-loop
lLine counter	REOInfo.getLinNo (lREOIndex)
lPartition counter	REOInfo.getParNo (lREOIndex)
<code>rMrProt.kSpace().linesToMeasure(...)</code>	REOInfo.getLinesToMeasure ()
<code>rMrProt.kSpace().PartitionsToMeasure(...)</code>	REOInfo. getPartitionsToMeasure ()
<code>lLine - rMrProt.kSpace().echoLine()</code>	REOInfo.getLinNoCenterZero (lREOIndex)

<code>lPartition =</code>	<code>REOInfo.getParNoCenterZero</code>
<code>rMrProt.kSpace().echoPartition()</code>	<code>(lREOIndex)</code>
<code>rMrProt.kSpace().echoLine()</code>	<code>REOInfo.getKSCenterLin()</code>
<code>rMrProt.kSpace().echoPartition()</code>	<code>REOInfo.getKSCenterPar()</code>
<code>lLine == rMrProt.kSpace().echoLine()</code>	<code>REOInfo.isKSCenterLin</code> <code>(lREOIndex)</code>
<code>lPartition ==</code>	<code>REOInfo.isKSCenterPar</code>
<code>rMrProt.kSpace().echoPartition()</code>	<code>(lREOIndex)</code>

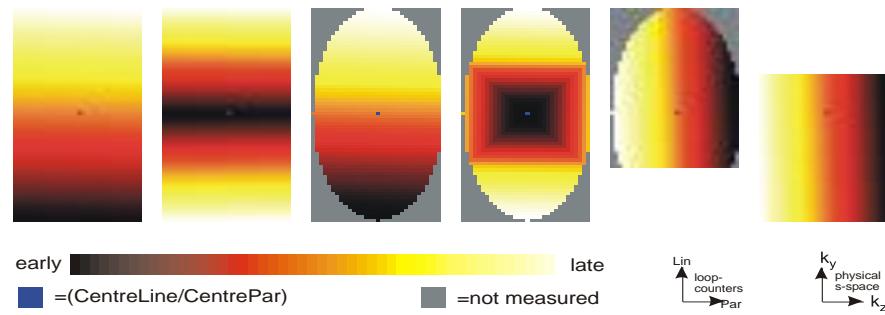
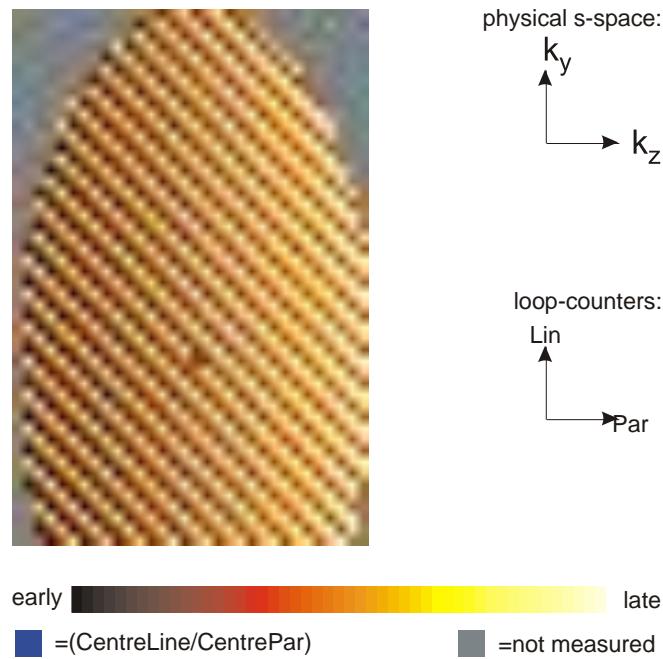


Figure D.3.3: Some Examples of Reordering Scemes that can be used in MR sequences

Deriving Classes From Reorderinfo

This section describes how a customized reordering class can be derived from the base class `ReorderInfo`. The shown example realizes a *checkerboard* reordering scheme with variable step size.



SeqUT-Bitmap (example) for 3d-sequence:
scanning scheme: checkerboard, step-size 3
elliptical scanning activated
lower part of k-space omitted with 6/8 phase
partial fourier

5. Declare custom reordering class as sub-class of `ReorderInfo`:

```
class ReorderInfoCheckerboard : public ReorderInfo
{
public:
    ReorderInfoCheckerboard (
        long lMaxNoOfReorderIndices = -1,
        long lCalcBufferSize = -1
    );

    virtual ~ReorderInfoCheckerboard();
    virtual bool reorderCheckerboard (
        MrProt& rMrProt,
        SeqLim& rSeqLim
    );

    void setlStepSize (long lValue);
}
```

```

protected:
    long m_lStepSize;

private:
    ReorderInfoCheckerboard (
        const ReorderInfoCheckerboard& right
    );

    ReorderInfoCheckerboard& operator= (
        const ReorderInfoCheckerboard& right
    );
};

```

6. Implement constructor and destructor:

```

ReorderInfoCheckerboard::ReorderInfoCheckerboard (
    long lMaxNoOfReorderIndices,
    long lCalcBufferSize)
: ReorderInfo (lMaxNoOfReorderIndices, lCalcBufferSize)
, m_lStepSize (2)
{
}

ReorderInfoCheckerboard::~ReorderInfoCheckerboard()
{
}

```

7. Implement the set-method for the step size:

```

void ReorderInfoCheckerboard::setlStepSize (long lValue) {
    m_lStepSize = lValue;
}

```

8. Implement the new reordering scheme:

```

bool ReorderInfoCheckerboard::reorderCheckerboard (
    MrProt& rMrProt,
    SeqLim& rSeqLim)
{
    static const char* const ptModule
        = {"ReorderInfoCheckerboard::reorderCheckerboard"};

    long lI;
    long lStep;

    // -----
    // m_lStepSize smaller than 2 -> use the std reordering
    // -----
    if (m_lStepSize < 2) {
        return ReorderInfo::reorder3dE (rMrProt, rSeqLim);
    }
}

```

```

}

// -----
// Calculate common k-space data using method of base class
// -----
if (!prepareCalculation (rMrProt, rSeqLim)) {
    UTRACE(Info, 0, ptModule << ": prepareCalculation failed!");
    return false;
}

// -----
// To ensure a real checkerboard structure, the number of
// lines used for the following for-loops must be odd
// -----
long lNumLines = getLinesToMeasure();
if ((lNumLines % 2) == 0) {
    lNumLines++;
}

long lTotalNum = lNumLines * getPartitionsToMeasure();

// -----
// Calculate reordering
// Note: Not performed during soft limits calculation.
// -----
if (!rSeqLim.isContextPrepForBinarySearch()) {
    for (lStep = 0; lStep < m_lStepSize; lStep++) {
        for (lI = lStep; < lTotalNum; += m_lStepSize) {
            long lPar = lI / lNumLines;
            long lLin = lI % lNumLines;

            // Try to append the current line and partition
            // to the reordering array.
            //

            // If the current lin/par-pair is outside the
            // range given by getLinesToMeasure() and
            // getPartitionsToMeasure() and
            // m_bEllipticalScanning the current lin/par-
            // pair is NOT added to the reordering array.

            if (!addLinPar3dE (lLin, lPar)) {
                UTRACE (Info, 0, ptModule << ": addLinPar3dE failed!");
                return false;
            }
        }
    }
}

// -----
// Automatically set FFT-flags

```

```
// -----
if (!setAllFFTFlags()) {
    UTRACE (Info, 0, ptModule << ": setAllFFTFlags failed!");
    return false;
}

// -----
// Finished with success:
// -----
return true;
}
```

9. In the sequence: Create an instance of the class ReorderInfoCheckerboard (instead of ReorderInfo):

```
ReorderInfoCheckerboard REOInfo;
```

10. Define reordering parameters:

```
REOInfo.set1StepSize (3);
REOInfo.setOmitLowerPartOfKSpaceIfPF (true, true);
```

11. Initiate calculation of reordering data:

```
if (!REOInfo.reorderCheckerboard (rMrProt, rSeqLim)) {
    // Error handling
    ...
}
```

Finally, the sequence will now use the custom reordering scheme.

LibKSpace - library for calculating the k-space trajectory

Overview

The library “*libKSpace*” is a new modular framework for calculating the k-space trajectory for an MRI sequence. The library is intended as an alternative and future replacement for the currently used *ReorderInfo*-classes.

For most applications, the k-space trajectory can be described entirely by a table of integer numbers (the so-called *sampling table*) and a procedure to translate this table into a trajectory. In case of Cartesian 2D-measurements, the table contains the phase encoding line indices. These indices are arranged in the order in which the phase encoding lines will be measured. For a radial sampling scheme, however, the table contains the spoke indices instead.

Loop index	Line index
0	16
1	15
2	17
3	14
4	18
...	...

For 3D-measurements, an additional index is required (either the slice index or the partition index). Therefore, the sampling table has one more column.

Loop index	Line index	Partition index
0	16	0
1	15	0
2	17	0
3	14	0
4	18	0
...

Throughout this chapter, the word "line" refers to a "phase encoding line" and the word "k-space point" refers to a point in the ky-kz-plane.

Doxxygen documentation und tutorial

The classes and methods of the libKSpace have *doxygen*-compatible code comments which are incorporated in the help file "IDEA.chm". Additionally, there is a tutorial for the libKSpace which can be found in the following folder:

```
\n4\pkg\MrServers\MrImaging\libKSpace\doc\tutorial\
```

The tutorial consists of several examples, which introduce the most important concepts and classes of libKSpace. The best approach is to first read this chapter here. Then, you should start working with the doxygen documentation and the tutorial.

Note: The examples of the tutorial are also included in the help file "IDEA.chm".

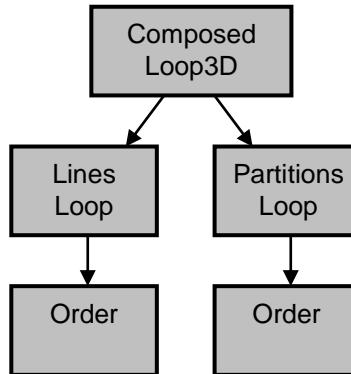
Module concept

The most important design feature of the libKSpace is the division of larger algorithms into several sub-steps which are performed by different modules. Thus, it is easier to replace or extend parts of an algorithm.

A simple example of a module is the module *OrderStrategy* which will be explained in more detail later. Basically, the module allows reordering a series of integer numbers. The following examples show some basic modes:

Order mode	Order table
Ascending order	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Descending order	9, 8, 7, 6, 5, 4, 3, 2, 1, 0
Centric-down order	5, 4, 6, 3, 7, 2, 8, 1, 9, 0
...	...

In order to limit the complexity and coding size of each module, many modules will delegate part of their task to child-modules. The parent-child relationship of all modules can be visualized in a tree structure. A 3D-measurement with a defined loop hierarchy (e.g. lines-in-partitions) may serve as an example. In this case, one *OrderStrategy* instance can be used for the lines loop, while another *OrderStrategy* instance is used for the partitions loop. The module tree would then look like this:



A special module class is the class *KSpaceSampling* which serves as an interface to the MR sequence. At the same time, this instance is the root module of the tree structure. All other modules should be kept hidden from the sequence.

The methods prepare() and calculate()

All modules are derived directly or indirectly from the base class *GeneralModule*. This class defines the following two methods that need to be called before a module can be used:

```

virtual bool prepare()
virtual void calculate()
  
```

The method *prepare()* checks only if the configuration of the module is valid. Examples for checks are:

- The number of lines / partitions / segments must be positive (larger than 0).*
- The number of measured lines / partitions must be divisible by the number of segments (This segmentation condition is only required for certain sampling schemes).*
- Elliptical scanning is not allowed in conjunction with segmentation (only for certain sampling schemes).*

If calculations are required (e.g. to determine the number of measured lines), the module will carry out only those steps that are really required to perform the checks. In general, the *prepare()*-method should be performance-optimized.

The method *calculate()* must not be called before the preparation of a module. It will perform all residual calculation steps to ensure that the module is ready for use.

Approximately, one can say that the two methods *prepare()* and *calculate()* are related to the different prepare-contexts of a sequence. In the case of the *binary search context*, the sampling class instance will only be *prepared*, while it will be *calculated* in the *normal context* and in the *context for scan time calculation*.

A module keeps track of its preparation status by setting internal flags after a successful *prepare()*- or *calculate()* call. These flags can be read by calling the methods *isPrepared()* and *isCalculated()*.

Whenever a module is re-configured (e.g. one of the `set()`-methods of the module is called), the preparation status will be reset. Then, a new `prepare()`- and `calculate()`-call might be required.

Usually, `prepare()`- and `calculate()`-calls are simply cascaded down to child-modules. However, please note that there are situations in which a `prepare()`-method of a parent-module must call the `calculate()`-methods of a child-module. For example, the segmentation condition can only be checked, if the number of measured lines or partitions is known. And this check requires typically the calculation of the *masking* modules (which will be introduced later).

Visualizing and debugging modules

Since each module can contain multiple child-modules, a tree structure is a suitable representation of the module relationships. The modules have methods such as `setChild()` or `getChildAt()` to build and investigate the tree structure. However, the tree representation is intended for visualization and debugging purposes only.

Furthermore, the methods `getName()` and `getDebugInfo()` allow retrieving the name and a string representation of the current state of each module.

The class `Viz` offers simple visualization methods for most of the module types and for the tree structure in text format.

Example for output of Viz::printModules():

Structure:

```
- "Segmented Sampling 3D" (class KSpace::Tutorial::Example10::SegmentedSampling)
  (prep: (x), calc: (x))

  - "Loop" (class KSpace::ComposedLoop3D)
    (prep: (x), calc: (x), mode: LINE_IN_PARTITION_LOOP)

  - "Lines Loop" (class KSpace::Loop2D)
    (prep: (x), calc: (x), elements: 36, measured: 24, segments: 4, mode: S...)

  - "Mask" (class KSpace::CombinedMask2D)
    (prep: (x), calc: (x), active: (x), elements: 36)

    - "Mask0" (class KSpace::PATMask)
      (prep: (x), calc: (x), active: (x), elements: 36, PAT factor: 2...)

    - "Mask1" (class KSpace::PFFMask)
      (prep: (x), calc: (x), active: (x), elements: 36, omitting lowe...)

  - "Order" (class KSpace::OrderStrategy)
    (prep: (x), calc: (x), number of elements: 6, center element: 0, mo...)

  - "Segmentation" (class KSpace::SegmentationStrategy)
    (prep: (x), calc: (x), number of elements: 24, center element: 12, ...)

  - "Partitions Loop" (class KSpace::Loop2D)
    (prep: (x), calc: (x), elements: 8, measured: 8, segments: -1, mode: EL...)

    - "Mask" (class KSpace::PFFMask)
      (prep: (x), calc: (x), active: (x), elements: 8, omitting lower k-s...)

    - "Order" (class KSpace::OrderStrategy)
      (prep: (x), calc: (x), number of elements: 8, center element: 4, mo...)
```

Deriving sub-classes of modules

Application-specific sampling classes typically require specialized modules. The derivation of sub-classes from existing modules is preferred over the alteration or extension of the standard modules.

New parameters which are potentially required to determine the specialized behavior, should be retrieved and altered with *get()*- and *set()*-methods. The *set()*-methods must implicitly call the method *resetPreparation()*.

The methods `prepareInternal()` and `calculateInternal()`

In many cases, the behavior of *prepare()* must be altered to add new constraints and consistency checks. Nevertheless, the original checks of the base class should still be performed. Therefore, the

prepare()-method of the base class should be generally called implicitly. However, a direct call of the base method *prepare()* will lead to errors because the *prepared*-flag would then be set before the checks of the sub-class are performed.

To overcome this problem, there are two methods: *prepare()* and *prepareInternal()*. The user of a module will call the public *prepare()*-method, which will then call the protected method *prepareInternal()* and set the *prepared*-flag accordingly.

Code of the method *GeneralModule::prepare()*:

```
bool GeneralModule::prepare (void)
{
    resetPreparation();

    bool bResult = prepareInternal();

    setPrepared (bResult);
    return bResult;
}
```

The same approach is realized for *calculate()* which calls the protected method *calculateInternal()*.

When deriving a sub-class, the methods *prepareInternal()* and *calculateInternal()* must be overridden instead of *prepare()* and *calculate()*. Furthermore, the methods *prepareInternal()* and *calculateInternal()* of the base class should be typically invoked implicitly in most cases.

Example for overriding *prepareInternal()*:

```
bool PATMask::prepareInternal (void)
{
    if (!Mask2D::prepareInternal()) return false;
    if (m_iPATFactor <= 0) {
        throw KSpaceException ("PAT factor is 0 or negative", __FILE__, __LINE__);
    }
    return true;
}
```

The method *addDebugInfo()*

The debugging information retrieved by *getDebugInfo()* is generated by the virtual protected method *addDebugInfo()*. Sub-classes can add information by overriding the method *addDebugInfo()*. Usually, the *addDebugInfo()*-method of the base class should be invoked implicitly.

Example for overriding addDebugInfo(...):

```
void Mask2D::addDebugInfo (DebugInfoContainer& rContainer) const
{
    Mask::addDebugInfo (rContainer);
    rContainer.addInfo ("elements") << getNumberOfElements ();
}
```

Standard modules

There are various modules provided with the standard distribution of IDEA. The following sections shall give an overview.

OrderStrategy

Order modules are used to reorder any kind of index. This can be for example the line, partition, segment or shot index.

The interface for order modules is *IOrderStrategy*. The main task is to arrange the integer numbers {0, 1, ..., N-1} in a specified way. The standard class *OrderStrategy* allows the following modes:

Order mode	Order table
Ascending order	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
Descending order	9, 8, 7, 6, 5, 4, 3, 2, 1, 0
Centric-down order	5, 4, 6, 3, 7, 2, 8, 1, 9, 0
Centric-up order	4, 5, 3, 6, 2, 7, 1, 8, 0, 9

The integer series that is shown for each mode is the so-called *order table*, which represents the following mapping:

loop index → *element index*

Each index between 0 and N-1 appears exactly once. Therefore, the inverse mapping will always exist. The *inverse order table* for a centric-down mapping looks for instance as follows:

9, 7, 5, 3, 1, 0, 2, 4, 6, 8

Usage

The standard module *OrderStrategy* requires the following data:

- Number of elements N (e.g. number of lines or partitions)
- Center element index

□ Mode (e.g. ascending, descending, centric-down, ...)

After preparation and calculation, the order table can be retrieved by calling `getOrderTable()`. The inverse order table can be retrieved by calling `getInverseOrderTable()`.

Deriving a custom order module

The principal method of order modules is the protected virtual method `calculateOrder()`, in which the order table is calculated (the result must be stored in the member `m_aiOrderTable`).

Example for deriving a custom order module:

```
class CustomOrder : public IOrderStrategy {
public:
    CustomOrder (void) {}

protected:
    virtual void calculateOrder (void) {
        int iI;
        int iCounter = 0;

        for (iI = 0; iI < m_iNumberOfElements; iI += 2) {
            m_aiOrderTable [iCounter++] = iI;
        }

        for (iI = 1; iI < m_iNumberOfElements; iI += 2) {
            m_aiOrderTable [iCounter++] = iI;
        }
    };
}
```

(The order for 9 elements will be: 0, 2, 4, 6, 8, 1, 3, 5, 7)

SegmentationStrategy

The interface class for segmentation modules is `ISegmentationStrategy`. These modules are used for the calculation of a *segmentation table* which represents the following mapping:

$$\text{element index} \rightarrow \text{segment index}$$

The standard segmentation algorithms are implemented in the class `SegmentationStrategy`. The following modes are available (examples shown for 9 elements):

Segmentation mode	Segmentation table
Serial segmentation	0, 0, 0, 1, 1, 1, 2, 2, 2
Centric-out segmentation	2, 1, 1, 0, 0, 0, 1, 2, 2

Each element within a segment has a unique *sub-segment-index* (or shorter: *sub-index*). Here is an example for the centric-out segmentation:

Element index	Segment index	Sub-index
0	2	0
1	1	0
2	1	1
3	0	0
4	0	1
5	0	2
6	1	2
7	2	1
8	2	2

Usage

After preparation and calculation, the methods `getSegmentationTable()` and `getSubIds()` are used to retrieve the segment indices and the sub-indices.

It is possible to rearrange the order of the segments by using the method `setSegmentOrder(...)` which expects an order module as argument. The following example shows the effect for 14 elements and 7 segments (serial segmentation):

- Ascending order: 0, 0, 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6
- Centric-down order: 5, 5, 3, 3, 1, 1, 0, 0, 2, 2, 4, 4, 6, 6

Deriving a custom segmentation module

The method `calculateInternal()` calls the virtual protected method `calculateSegmentIndices()`, which is the method that should be overridden.

It is essential to guarantee that the actual segment sizes match the sizes defined in the vector `m_aiSegmentSizes`. The segment sizes are calculated in the method `calculateSegmentSizes()`. An example for a derived SegmentationStrategy is given in the tutorial.

Masking modules

Masking modules are used to realize k-space undersampling (e.g. partial Fourier, elliptical scanning, PAT or Caipirinha).

Usage

After setting all parameters that are required by the mask (e.g. number of elements, PF-factor, etc.), the mask can be prepared and calculated. Then, the method *isMeasured(...)* tells whether or not a k-space line should be measured.

In case of 2D-masks, the *isMeasured(...)*-method expects only one argument (e.g. the line index). In case of 3D-measurements, the *isMeasured(...)*-method expects two arguments (i.e. line and partition index).

A 3D-mask can be composed of two 2D-masks - one for the lines direction, the other one for the partitions direction (see *ComposedMask3D*).

Example for ComposedMask3D:

```

Mask1
      | x x . x .
      +-----+
Mask2 x | x x . x .
      x | x x . x .
      . | . . . .
      x | x x . x .
      x | x x . x .  ComposedMask
      . | . . . .

```

Furthermore, it is possible to combine multiple masks. The classes *CombinedMask2D* and *CombinedMask3D* realize a logical AND-combination (i.e. k-space points will be measured only if the method *isMeasured(...)* returns *true* for each of the masks).

Existing masking modules

The following masking modules are provided by libKSpace:

- PATMask** (to realize PAT acceleration in one dimension)
- CaipirinhaMask** (to realize PAT-squared or Caipirinha sampling schemes)
- PFMask** (to realize partial Fourier sampling in one dimension)
- EllipticalMask** (to realize elliptical scanning)

Deriving a custom masking module

Typically, the public virtual method *isMeasured(...)* must be overridden.

Additionally, it may be required to override *prepareInternal()* and *calculateInternal()* and to implement new getters and setters.

Example for deriving a custom masking module:

```

// This mask creates patterns like
//
//   x x x . . . x x x . . . x x x . . . x x x
//   (for period set to 3)
//
class CustomMask : public Mask2D {
public:
    CustomMask (void) {}

    int getPeriod (void) {
        return m_iPeriod;
    }

    void setPeriod (int iPeriod) {
        m_iPeriod = iPeriod;
        resetPreparation();
    }

protected:
    int m_iPeriod;

    virtual bool isMeasured (int iIndex) const {
        return (iIndex / m_iPeriod) % 2;
    }

    virtual bool prepareInternal (void) {
        if (!Mask2D::prepareInternal()) return false;

        if (m_iPeriod <= 0) return false;

        return true;
    }

    virtual void addDebugInfo (DebugInfoContainer& rContainer) const {
        Mask2D::addDebugInfo (rContainer);

        rContainer.addInfo ("period", m_iPeriod);
    }
};

```

Loop modules

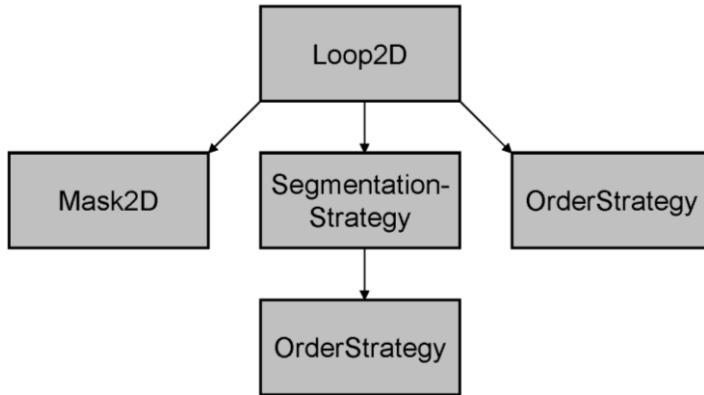
Loop modules define which k-space points shall be measured and how often and in which order those k-space points shall be visited. To carry out this task, loop modules typically use a few child-modules.

The interface for loops in 2D-measurements is *ILoop2D*. A loop module creates a series of integer numbers. This series defines the order in which k-space lines (or partitions) shall be measured. In contrast to order modules, elements can be left out or visited multiple times.

The standard implementation Loop2D uses child-modules to perform its calculations.

- An optional 2D masking module
- An optional segmentation module
- An order module (not optional)

Graphical representation of a *Loop2D* module:



At first, the masking module removes elements which should not be measured. The residual elements are then divided into multiple segments. In the next step, the elements of each segment are reordered separately. Finally, a nested loop over segment-index and sub-segment-index defines the order in which all elements are visited.

A 3D-loop module creates a series of integer pairs (e.g. line and partition index). There are two standard implementations. The class *ComposedLoop3D* allows composing a 3D-loop out of two 2D-loops (one for the line indices and one for the partition indices). The second standard implementation is the *FlexibleLoop* which is described in more detail in the doxygen documentation.

Usage

A loop module is configured by setting the number of elements (i.e. the number of lines or partitions) and the child-modules to be used.

After preparation and calculation, the method *getElementForLoopIndex(...)* allows retrieving any element that is visited by the loop.

Deriving a custom loop module

Typically, it is necessary to override the public virtual methods *getNumberOfMeasuredElements()* and *getElementForPosition(...)*.

Additionally, it may be required to override *prepareInternal()* and *calculateInternal()* and to implement new getters and setters.

The module KSpaceSampling

The module *KSpaceSampling* is a special module that serves as interface to the sequence.

A sequence owns one or multiple sampling class instances which are used to calculate the sampling table and attributes required for image reconstruction.

The abstract class *KSpaceSampling* is derived from *GeneralModule* and defines a few additional methods.

- ❑ *configureAndPrepare(...)* Sets the given parameters and checks the consistency of these parameters
- ❑ *getTotalNumberOfKSpacePoints()* Returns the length of the sampling table
- ❑ *getKSpacePointForLoopIndex(...)* Allows retrieving the k-space point that will be measured at the given loop index
- ❑ *getSamplingInfo()* Returns an object that contains information about the sampling that is typically required by the sequence (and that needs to be exported in the SeqExpo-object in sequence-prepare)
- ❑ *findValidParams(...)* Tries to determine consistent parameters

The following example shows how a concrete sampling class instance (*ExampleSampling*) can be created, configured, prepared and calculated. Finally, an enumerator (*ExampleSampling::Enumerator*) is used to step through the sampling table:

Example usage for sampling class:

```
ExampleSampling Sampling;
SamplingParams Params;

Params.m_iNumberOfLines      = 30;
Params.m_iNumberOfSegments = 5;

... specify more parameters here ...

if (Sampling.findValidParams (Params, Params) && Sampling.prepare (Params)) {
    Sampling.calculate();

    ExampleSampling::Enumerator MyEnumerator (Sampling);

    while (MyEnumerator.moveToNext()) {
        int iLineId      = MyEnumerator->getLineId();
        int iPartitionId = MyEnumerator->getPartitionId();

        cout << iLineId << "\t" << iPartitionId << endl;
    }
}
```

SamplingParams

All the parameters that are required for calculating the sampling table are stored in a compact *SamplingParams* object. The following list shows a few parameters that are typically required for the definition of a sampling trajectory:

- Number of lines*
- Number of partitions*
- Lines order mode*
- Number of segments*
- Segmentation mode*
- ...

The advantage of having a parameter class that is independent of the protocol is that the sequence can decide how the protocol shall be interpreted. Furthermore, the set of sampling parameters can be easily extended by deriving a sub-class.

Deriving a custom sampling class

Custom sampling classes should be preferably derived from the class template *SamplingTable*. This template provides functionality to append single elements or entire 2D- or 3D-loops in the sampling table. Furthermore, the FT-flags and PAT-flags can be automatically calculated by means of the methods *setFTFlags(...)* and *setPATFlags(...)*. Moreover, the inner enumerator class is automatically generated with the template.

The following example shows how a simple sampling class for 2D-measurements can be implemented. The class supports PAT and measures the lines in centric-down order.

Example implementation of a simple sampling class:

```
class SimpleSampling : public SamplingTable <KSpacePoint> {
public:
    Loop2D          m_Loop;
    PATMask         m_PATMask;
    OrderStrategy   m_Order;

    simpleSampling (void) {
        setName ("Simple Sampling");
        setChild ("Loop", m_Loop);

        // set up module structure
        m_Loop.setMask      (m_PATMask);
        m_Loop.setOrderStrategy (m_Order);
    }

    virtual SamplingInfo getSamplingInfo (void) {
        checkPreparation();

        SamplingInfo Result;
```

```

    // calculate sampling information based on mask
    Result.setInfoFor (m_PATMask);

    return Result;
}

protected:
    virtual void configure (const SamplingParams& rParams) {
        KSpaceSampling::configure (rParams);

        // set parameters
        m_Loop.setNumberOfElements           (rParams.m_iNumberOfLines);
        m_PATMask.setPATFactor              (rParams.m_iLinesPATFactor);
        m_PATMask.setNumberOfReferenceElements (rParams.m_iNumberOfReferenceLines);
        m_Order.setMode                     (OrderStrategy::ORDER_MODE_CENTRIC_DOWN);
    }

    virtual bool prepareInternal (void) {
        if (!SamplingTable <KSpacePoint>::prepareInternal()) return false;

        // the loop will implicitly prepare the mask and the order strategy
        if (!m_Loop.prepare()) return false;

        return true;
    }

    virtual void calculateInternal (void) {
        SamplingTable <KSpacePoint>::calculateInternal();

        // the loop will implicitly calculate the mask and the order strategy
        m_Loop.calculate();

        // fill the sampling table with the loop
        append (m_Loop);

        // these methods set the flags that will be used to fill the MDHs later
        setFTFlags (m_Loop.getNumberOfElements(), 1);
        setPATFlags (&m_PATMask, NULL);
    }
};

```

Sequence and libKSpace

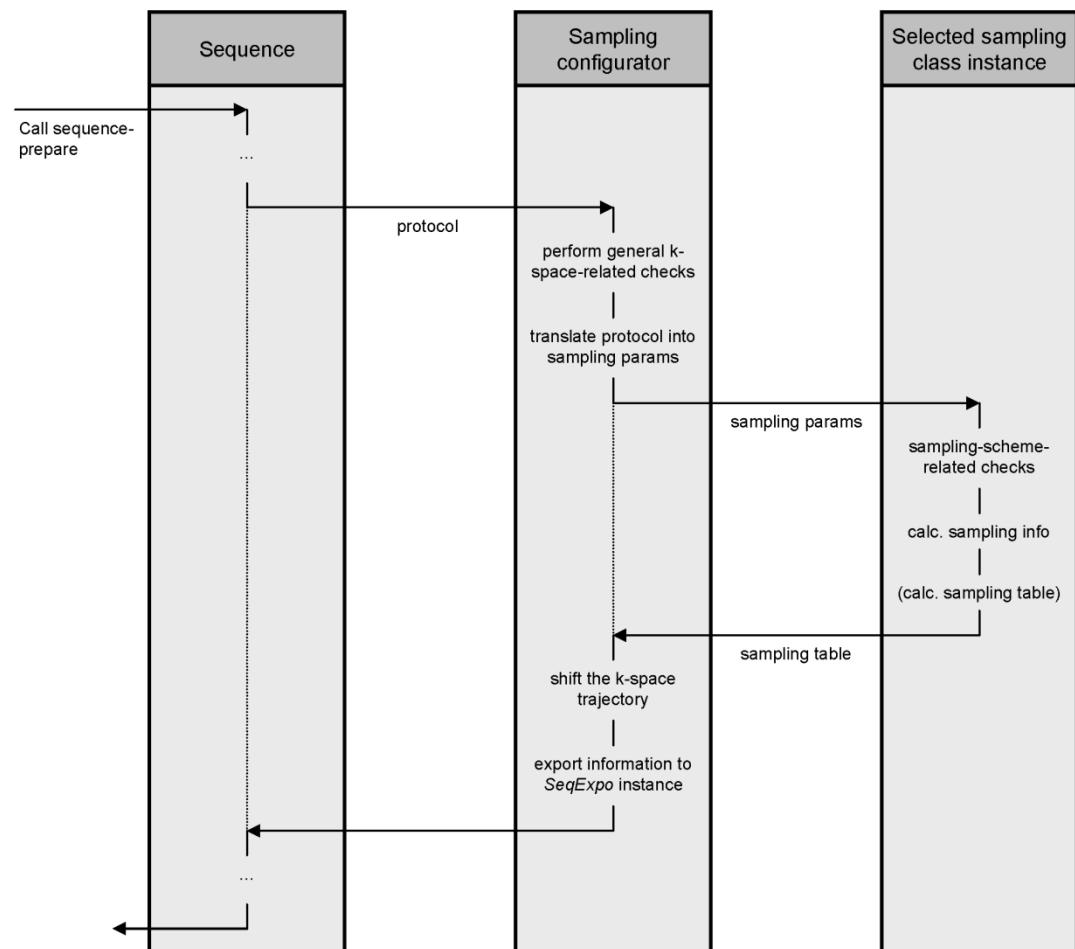
The class SamplingConfigurator

To keep the size of the code of the main sequence class at a reasonable level, it is advantageous to have an additional class that covers all k-space related parts of the sequence. This class is called *SamplingConfigurator*. The main tasks are:

- ❑ Finding the correct sampling scheme and the according sampling class (only if the sequence works with multiple sampling schemes)

- Translating the MR protocol into a *SamplingParams* instance
- Setting sampling-related parameter limits in sequence-initialize
- Performing basic sampling-related checks (e.g. is the number of coils sufficiently large for the adjusted PAT factor)
- Shifting the k-space trajectory such that the first measured line index and the first measured partition index are always 0 (requirement by some ICE programs and the sequence unit test)
- Performing sampling-related exports to the *SeqExpo* object

The following figure illustrates how the sampling configurator is called during sequence prepare.



Example code for usage of sampling configurator in sequence-prepare:

```
NLSStatus Status = m_SamplingConfigurator.prepare (rMrProt, rSeqLim, rSeqExpo);  
if (Status.isError()) {  
    return Status;  
}  
  
SamplingInfo& rInfo = m_SamplingConfigurator.getSamplingInfo();
```

For performance reasons, the sampling table will be calculated only in the contexts *normal* and *scan time calculation*. However, if the preparation was successful, a *SamplingInfo* instance can be retrieved also in the *binary search* context. The following list contains some of the data that is stored in the *SamplingInfo* object (please look-up the complete list in the doxygen documentation):

- The total number of k-space points in the sampling table*
- The number of lines or partitions (total k-space matrix size)*
- The number of measured lines or partitions (PF, PAT, etc. incorporated)*
- The center line or partition index*
- The index of the first measured line or partition*
- The index of the first PAT reference line or partition*
- ...

The base class *SamplingConfigurator* provides the basic functionality for the translation of the MR protocol into a sampling parameter object. However, the selection of the appropriate sampling class must be implemented in an application-specific sub-class since the base class *SamplingConfigurator* does not hold any sampling class instance.

Please read the source code of the demo sequence *FLASH* (in particular the code of the class *FlashSamplingConfigurator*) as an example for a concrete implementation of a sampling configurator.

The class StdCheckSampling

The class *StdCheckSampling* allows calculating the part of the trajectory that should be processed in sequence-check. The class analyzes the sampling table that will be used in sequence-run (passed as *KSpaceSampling* instance) and determines which parts are the most critical. *StdCheckSampling* is itself derived from *KSpaceSampling* and can therefore be used in the same way as the main sampling class is used.

Example code for usage of the class *StdCheckSampling*:

```
m_CheckSampling.setSampling (*m_SamplingConfigurator.getSampling());  
  
if (!m_CheckSampling.configureAndPrepare (*m_SamplingConfigurator.getParams())) {  
    return SEQU_ERROR;  
}  
  
m_CheckSampling.calculate();
```

More details about the class *StdCheckSampling* can be found in the doxygen documentation.

iPAT

iPAT (integrated Parallel Acquisition Techniques) reduces the total imaging time by skipping k-space lines (or partitions). To reconstruct these undersampled k-space raw data, appropriate array coils and special image reconstruction programs are required.

For 3D-sequences iPAT may be applied to phase encoding and partition direction at the same time. Acceleration in both directions is described in the chapter iPAT².

Basic Implementation

iPAT Mode

There are two different iPAT image reconstruction algorithms which have been implemented in the standard ICE-Program and can be selected by the user: *GRAPPA* and *mSENSE*.

The main difference between the two algorithms is, that the GRAPPA ("GeneRALized Partially Parallel Acquisition") reconstruction takes place before the raw data are Fourier transformed, whereas in mSENSE ("modified SENSE"), reconstruction is performed after Fourier transformation. Due to these properties, GRAPPA has been referred to as a SMASH ("SiMultanious Acquisition of Spatial Harmonics") related method, whereas mSENSE is considered to be a SENSE ("SENSitivity Encoding") related method.

Both reconstruction methods can be selected as *iPAT mode* in the user interface.

If nothing is specified in the sequence, the iPAT mode is set to "None" which means that no iPAT reconstruction will be performed. The iPAT mode (and the UI sub card for iPAT) can be activated by adding the following lines in the sequence *initialize(...)*-method:

```
// Activate PAT mode:  
rSeqLim.setPATMode (  
    SEQ::PAT_MODE_NONE,  
    SEQ::PAT_MODE_SENSE,  
    SEQ::PAT_MODE_GRAPPA  
) ;
```

iPAT Acceleration Factor

The UI parameter *iPAT acceleration factor* defines how many k-space lines are skipped (e.g. a factor of 3 means that only every third line is measured). The scan time will be reduced by the same factor. This parameter can be activated in *initialize(...)* as follows:

```
// Set minimum, maximum, increment and default value of the  
// acceleration factor:  
//  
//          (min, max, inc, def)  
rSeqLim.setAccelFactorPE ( 2, 4, 1, 2);
```

This parameter will be displayed on the UI card only if the iPAT mode is set to GRAPPA or mSENSE. The minimum value of this parameter should be larger than 1 because an acceleration factor of 1 means

basically that iPAT is not active (iPAT mode is "None").

The k-space center line must always be measured. Accordingly, only k-space lines which satisfy the following condition will be acquired:

```
((lLineId - lCenterLineId) % lAccelFactorPE) == 0
```

where *lAccelFactor* is the acceleration factor in phase encoding direction defined by the user and the *lCenterLineId* is the index of the center k-space line.

The set of lines which fulfill the equation above are called *PAT scans*.

iPAT Reference Lines

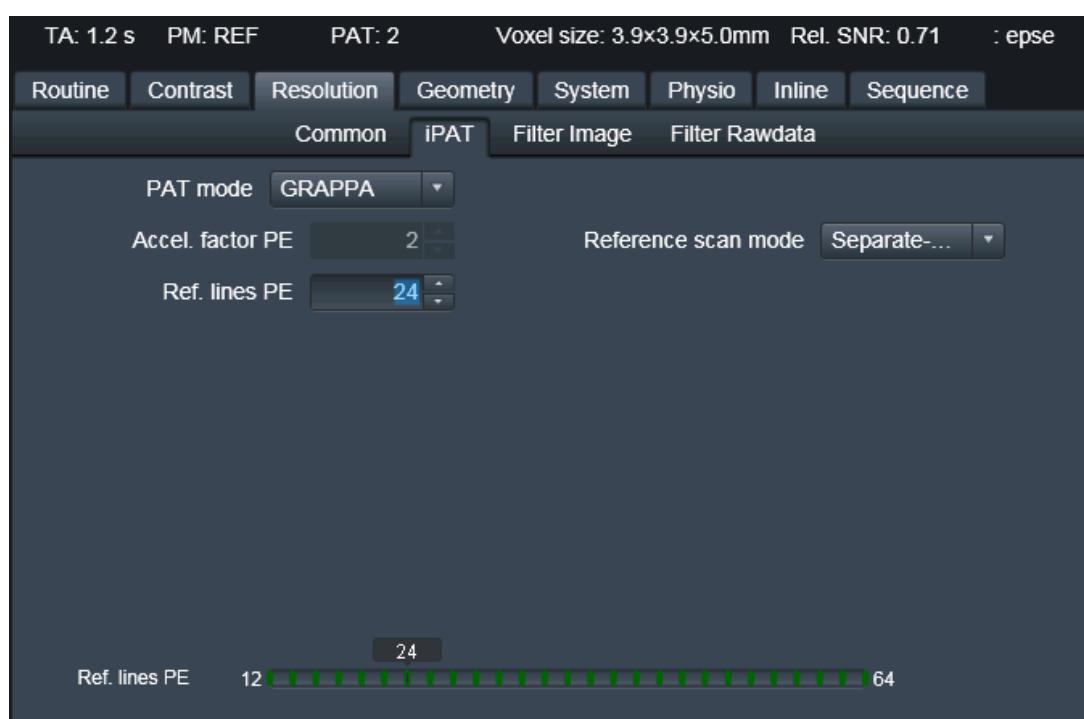
Both GRAPPA and mSENSE methods need the sensitivity maps of the active array coils for image reconstruction. Usually, low resolution maps are sufficient to extract the required coil sensitivities. For this purpose, a central region of the k-space is fully sampled (i.e. without skipping lines). The set of lines in this non-undersampled region is called *iPAT reference lines*. We use the UI parameter *reference line number* to define the size of this fully sampled k-space region. The following lines show how this UI parameter can be activated in *initialize()*:

```
// Activate UI-parameter reference line number;
//           (min, max, inc, def);
rSeqLim.setRefLinesPE ( 2, 256, 1, 24);
```

Equivalent to the iPAT acceleration factor, the UI parameter *reference line number* can only be activated if the iPAT mode is not "None".

The reference region is located around the k-space center. The first and last reference line (included) are:

```
minPATRefLinNo = lCenterLineId - NoOfPATRefLines/2
maxPATRefLinNo = lCenterLineId + (NoOfPATRefLines-1)/2
(NoOfPATRefLines = number of reference lines)
```



iPAT Reference Scan Mode

The reference lines can be acquired in "*inplace reference scan mode*" or "*extra reference scan mode*". If the "*inplace reference scan mode*" is set, the reference lines and imaging scans are measured together. Therefore, properties such as echo time, bandwidth, TE and TR are equal. If possible, these additionally acquired lines will be used to improve the SNR of the finally reconstructed images.

In "*extra reference scan mode*" the data for the reference lines are assumed to have a different image contrast. Therefore, these lines are only used to calculate the coil sensitivity maps for iPAT image reconstruction.

Inplace reference lines should be acquired using the same kernel and kernel parameters as the imaging scans. If a particular k-space line is both a PAT line and a reference line, this line needs to be acquired only once. The disadvantage of inplace reference scans is that the kernel cannot be optimized separately for the reference and the imaging scan.

Extra reference scans can use a different kernel (or different kernel parameters) which results in a different image contrast. This kernel can be optimized for the reference scan and therefore be faster. The disadvantage is that all reference lines have to be acquired (even if some of them have to be acquired again as PAT line for the imaging scan).

The sequence programmer determines if the reference scans acquired are inplace or extra reference scans and sets the UI-parameter and Mdh flags accordingly. According to the settings of the UI-parameters and the Mdh flags, the ICE program can insert the reference scans into the final image in order to improve the SNR of the images.

Mdh for reference lines

To tell the ICE program whether a measured k-space line is a PAT reference line and whether this reference line can be used as an image scan, two Mdh flags are provided:

- PATRefScan**: must be set for all reference lines (*inplace* or *extra*).
- PATRefAndImaScan**: must be set only for *inplace* reference scans.

The following two methods are used to set these flags:

```
Mdh.setPATRefScan      (bool) ;
Mdh.setPATRefAndImaScan (bool) ;
```

The Mdh flag of *PATRefScan* or *PATRefAndImaScan* will be set if the corresponding input bool parameter is true.

Protocol parameters

The protocol class provides the following iPAT parameters:

```
SEQ::PATSelMode PATMode = MrProt.PAT().PATMode() ;
// defined enum values of SEQ::PATSelMode:
// SEQ::PAT_MODE_NONE
// SEQ::PAT_MODE_GRAPPA
// SEQ::PAT_MODE_SENSE

long lAccelFactPE = MrProt.PAT().AccelFactPE();
long lAccelFact3D = MrProt.PAT().AccelFact3D();
long lRefLinesPE = MrProt.PAT().RefLinesPE();
long lRefLines3D = MrProt.PAT().RefLines3D();

SEQ::PATRefScanMode PATRefScanMode
= MrProt.PAT().RefScanMode();
// defined enum values of SEQ::PATRefScanMode:
// SEQ::PAT_REF_SCAN_UNDEFINED (if PATMode == NONE)
// SEQ::PAT_REF_SCAN_INPLACE
// SEQ::PAT_REF_SCAN_EXTRA
// Note: this parameter is not displayed in the UI
```

Export parameters

The ICE program needs some further information on the structure of the reduced phase encoding scheme: the number of the first reference line and the number of the first actually scanned line (i.e. the first iPAT scan) have to be set into the YAPS buffer. In case iPAT is switched off, both parameters should be reset to the default values as shown in the example below.

```
// set YAPS paramteters
// Background: The ICE program needs to be informed (via YAPS)
// about the k-space positions of the first measured line.
// Without iPAT, this value is assumed to be zero.
// However, with iPAT, the first line(s) might be 'gaps'!.
// Since the reference lines are stored in a separate object,
```

```
// the k-space position of the first reference line
// is required, too.

if (rMrProt.PAT() .PATMode() != SEQ::PAT_MODE_NONE) {
    // both values are provided by the Reorder Base Class
    rSeqExpo.setFirstFourierLine
        (pReorderInfo->getMinPATLinNo());

    rSeqExpo.setFirstRefLine
        (pReorderInfo>getMinPATRefLinNo()) ;

} else {
    // reset to default
    rSeqExpo.setFirstFourierLine ( 0);
    rSeqExpo.setFirstRefLine      (-1);
}
```

**ReorderInfo
Class Support
for iPAT**

iPAT Support by the Reorder Base Class

To support the undersampled k-space acquisition scheme, the class ReorderInfo provides some basic functions:

```
long getNoOfPATLines();
long getNoOfPATPartitions();
```

Returns the number of PAT lines/partitions which should be acquired (without reference lines/partitions in "gaps").

```
long getNoOfPATRefLinesInGap();
long getNoOfPATRefPartitionsInGap();
```

Returns the number of additional reference lines/partitions in the "gaps" of the PAT lines/partitions (for inplace reference scan mode).

```
long getPATAccelerationFactorPE();
long getPATAccelerationFactor3D();
```

Returns the acceleration factor.

```
long getNoOfPATRefLines();
long getNoOfPATRefPartitions();
```

Returns the number of reference lines/partitions.

```
bool isPATRefScan();
```

Returns true for all PAT reference lines.

```
bool isPATRefAndImaScan();
```

This function returns also true for all PAT reference lines. For extra reference line mode, this function must be overloaded by sequence programmer to return false.

```
bool acquireForPAT();
```

Returns true if a k-space line is a PAT line or a reference line.

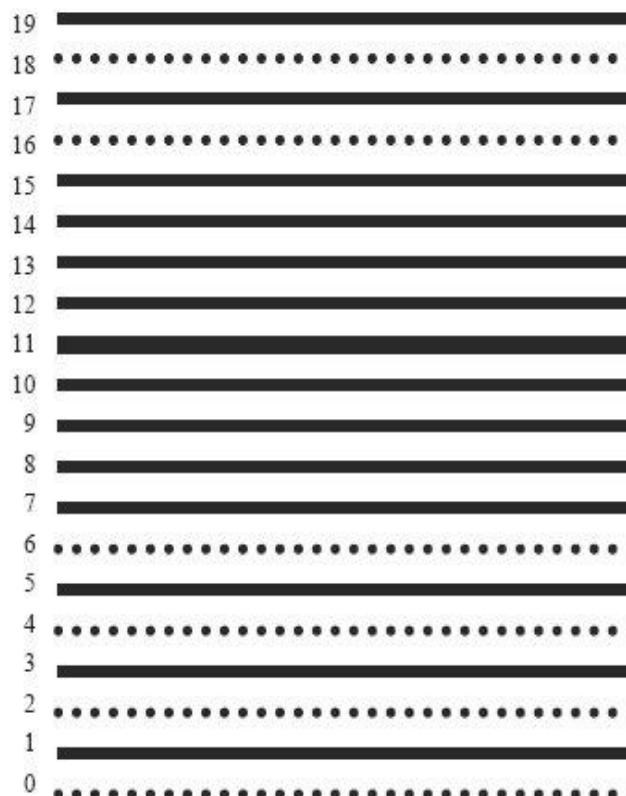
```
long getMinPATLinNo();
long getMinPATParNo();
```

Returns the index of the first scanned line/partition.

```
long getMinPATRefLinNo();
long getMinPATRefParNo();
```

Returns the index of the first reference line/partition.

Before using the methods above, the function *prepareCalculation()* of the ReorderInfo class must be called to initialize the internal parameters.



*Figure D.3.4: An example of k-space acquisition scheme using iPAT with the following parameters:
Acceleration factor = 2; Number of reference lines = 8; Reference scan mode = inplace; maximum
line number = 19; K-space center line = 11; PAT lines = 1, 3, 5, 7, 9, 11, 13, 15, 17, 19; Reference lines
= 7, 8, 9, 10, 11, 12, 13, 14; PATRefAndImaScan = 7, 8, 9, 10, 11, 12, 13, 14*

iPAT Implementation (inplace reference scan mode)

**Sequence
Modification for
iPAT with
'Inplace
Reference Scan
Mode'**

The implementation of iPAT for sequences which use the ReorderInfo Base Class and standard seqloop, the modification is very straight forward:

1. In *initialize()*:

Initialize the iPAT parameters iPAT mode, acceleration factor and reference line number:

```
// activate PAT mode:  
rSeqLim.setPATMode (  
    SEQ::PAT_MODE_NONE,  
    SEQ::PAT_MODE_SENSE,  
    SEQ::PAT_MODE_GRAPPA  
) ;  
  
// set minimum, maximum, increment and default value of the  
// acceleration factor and reference lines:  
rSeqLim.setAccelFactorPE (2, 4, 1, 2);  
rSeqLim.setRefLinesPE (2, 256, 1, 24);
```

2. in *prepare(...)*:

Call the function Seqloop.setLinesToMeasure() to set the correct number of lines to be measured in seqloop, for example:

```
long lLinesToMeasurePAT = Reorder.getNoOfPATLines()  
    + Reorder.getNoOfPATRefLinesInGap();  
lLinesToMeasurePAT *= Reorder.getPartitionsToMeasure();  
mySeqLoop.setLinesToMeasure (lLinesToMeasurePAT);
```

3. Some additional checks in *prepare(...)*:

The following code example contains a number of restrictions which have to be regarded when using the default iPAT support (Reorder base class and ICEProgram2D):

```
//  
// Check some iPAT related restrictions:  
//  
// Check that a sufficient number of coils are selected (should  
// only be checked in ContextNormal to avoid trouble with UI-handling).  
long lNumberOfUsedRxChannels = rMrProt.coilInfo().Meas().getNumOfUsedRxChan();  
  
if (rSeqLim.isContextNormal()) {  
    if (lNumberOfUsedRxChannels < rMrProt.PAT().AccelFactPE()) {  
        return SEQU_PAT_NO_OF_RX_CHANNELS_TOO_LOW;  
    }  
}
```

```

// Number of reference lines should be at least 6*accel.factor:
if (rMrProt.PAT().RefLinesPE() < rMrProt.PAT().AccelFactPE()*6) {
    return SEQU_ERROR;
}

// iPAT is not possible with elliptical scanning
// (currently not supported by ICE)
if ( (rMrProt.kSpace().dimension() == SEQ::DIM_3) &&
    (rMrProt.kSpace().ellipticalScanning())) {
    return SEQU_ERROR;
}

// SENSE is not possible with 'save uncombined images'
if ( (rMrProt.PAT().PATMode() == SEQ::PAT_MODE_SENSE) &&
    (rMrProt.uncombImages() == true)) {
    return SEQU_ERROR;
}

// SENSE is not possible with other reconModes than 'Magnitude'
if ((rMrProt.PAT().PATMode() == SEQ::PAT_MODE_SENSE) &&
    (rMrProt.reconstructionMode() != SEQ::RECONMODE_MAGNITUDE)) {
    return SEQU_ERROR;
}

// SENSE is not possible with 'save unfiltered images'
if ( (rMrProt.PAT().PATMode() == SEQ::PAT_MODE_SENSE) &&
    (rMrProt.normalizeFilter().saveUnfiltered() == true)) {
    return SEQU_ERROR;
}

// SENSE requires a noise adjust scan
if ( (rMrProt.PAT().PATMode() == SEQ::PAT_MODE_SENSE) &&
    (rMrProt.noiseAdjust() == false)) {
    if (rSeqLim.isContextPrepForMrProtUpdate()) {
        rMrProt.noiseAdjust(true);
    } else {
        return SEQU_ERROR;
    }
}

```

4. in *runKernel(...)*:

Set additional Mdh flags for reference scans in sequence kernel, for instance:

```

Kernel.ADC.Mdh.setPATRefScan (Reorder.isPATRefScan (Index));
Kernel.ADC.Mdh.setPATRefAndImaScan (
    Reorder.isPATRefAndImaScan(Index)
);

```

As discussed before, the function `isPATPefScan(...)` should return true for all reference lines and false otherwise. `isPATRefScanAndImaScan(...)` should return true only for inplace reference lines and otherwise false.

Modification of ReorderInfo for iPAT with 'Inplace Reference Scan Mode'

Usually, the sequence to be modified uses a derived ReorderInfo class to acquire k-space lines with more advanced reordering schemes. To add the iPAT functionality to this kind of sequences, the most convenient way is to modify the derived ReorderInfo class.

5. overload the function `prepareCalculation()`:

If iPAT is activated, the total number of lines to be measured (`LinesToMeasure`) is reduced. Therefore, we should do some additional checks to see if the reordering is still possible with iPAT. For segmented sequences for example, `LinesToMeasure` must be dividable by the number of segments. Therefore:

```

bool ReorderInfoXYZ::prepareCalculation (MrProt& rMrProt,
                                         SeqLim& rSeqLim)
{
    static const char* ptModule = {"ReorderInfoGRE::prepareCalculation"};

    bool rtn = ReorderInfo::prepareCalculation (
        rMrProt,
        rSeqLim
    );

    // If iPAT is not activated, then return:
    if (rMrProt.PAT().PATMode() == SEQ::PAT_MODE_NONE) {
        return rtn;
    }

    // If the returned value is false, also return:
    if (!rtn) {
        return rtn;
    }

    // Do some initialization if required
    ...

    // Do some additional checks if desired, for instance:
    // Check that number of lines to be measured is dividable
    // by the number of segments:

    long lLinesToMeasurePAT
        = getNoOfPATRefLinesInGap() + getNoOfPATLines();

    if ((lLinesToMeasurePAT % m_lMeasuredSegments) != 0) {

```

```
        return false;
    }

    // If everything is ok, return true
    return true;
}
```

6. Modify the reorder function `reorderXYZ(...)` which will be called from the sequence:

Use simply the function `acquireForPAT()` to check if the given k-space line should be acquired or not and then insert this line to the reordering list by using the function `setLinNo()` and `setParNo()`:

```
bool ReorderInfoXYZ::reorderXYZ (
    MrProt& rMrProt,
    SeqLim& rSeqLim)
{
    // Do some initialization. Keep in mind that, in case of
    // PAT, the number of lines which should be acquired
    // is reduced:

    long lLinesToMeasurePAT
        = getNoOfPATRefLinesInGap() + getNoOfPATLines();

    ...

    // before using setLinNo and setParNo to insert a k-space
    // line to the measurement, use acquireForPAT to check if
    // this line needs to be acquired:

    if (acquireForPAT (lLin,lPar)) {
        setLinNo (lReorderCounter, lLin);
        setParNo (lReorderCounter, lPar);
    }

    ...
}
```

iPAT Implementation (extra reference scan mode)

Modification of Sequences Using the 'Extra Reference Scan Mode'

To write a sequence with 'extra reference scan mode', the following additional steps should be done:

1. Modify the sequence to acquire the extra reference scans.

The index of the minimum and maximum line number can be obtained by using the ReorderInfo function:

```
long getMinPATRefLinNo();
long getMaxPATRefLinNo();
```

As has been discussed above, for extra reference scans, we can use any arbitrary kernel and kernel parameters. If the Mdh flag has been correctly set, these lines will be not used as image scans.

2. Overload the method

```
bool isPATRefAndImaScan();
```

to return false and use the following Mdh function to set the Mdh flags correctly for extra reference scans:

```
// -----
// Set PAT flags
// -----
Mdh.setPATRefScan      (Reorder.isPATRefScan (lIndex));
Mdh.setPATRefAndImaScan (Reorder.isPATRefAndImaScan (lIndex));
```

3. Check maximum Reference line number:

To avoid allocating to much memory space in the ICE reconstruction program, the extra reference scans will be saved in the gaps of the PAT lines. The following code calculates the available space in the gaps of the PAT lines and check if there is enough space for the extra reference scans. This check should be called in *prepare(...)*:

```
bool fPATcheckMaxNoOfReferenceLines (
    ReorderInfo& rReorderInfo,
    MrProt&       rMrProt)
{
    long lNLineMeas = rReorderInfo.getMaxLineNumber() + 1;
    long echoLineOrig = rReorderInfo.getKSCenterLin();
    long m_peftLen = rMrProt.kSpace().PEFTLen();
    long m_AF = rReorderInfo.getPATAccelerationFactorPE();

    long lEinruettelPreCutoff = m_peftLen/2-echoLineOrig;
```

```
if (lEinruettelPreCutoff>=0) {
    lEinruettelPreCutoff = 0;
} else {
    lEinruettelPreCutoff = -lEinruettelPreCutoff;
}

long lEinruettelPostCutoff
= m_peftLen/2-(lNLinMeas-echoLineOrig);
if (lEinruettelPostCutoff >= 0) {
    lEinruettelPostCutoff = 0;
} else {
    lEinruettelPostCutoff = -lEinruettelPostCutoff;
}

// Number of remaining lines after einruetteln
long lMaxNoOfReferenceLines
= lNLinMeas
- lEinruettelPreCutoff
- lEinruettelPostCutoff;

// Number of remaining reference lines after einruetteln
lMaxNoOfReferenceLines -= lMaxNoOfReferenceLines/m_AF;

return ( rMrProt.PAT().RefLinesPE()
<= lMaxNoOfReferenceLines);
}
```

iPAT²

iPAT²

For 3D-sequences it is possible to apply iPAT to the 3D-phase encoding direction as well. This feature is also called iPAT². In order to distinguish between both phase encoding directions we use the terms acceleration in PE-direction ("lines"-direction) and acceleration in 3D-direction ("partitions"-direction). In the current software iPAT² is restricted to the iPAT mode "GRAPPA", i.e. the UI-parameters for iPAT² are only displayed, if "GRAPPA" is selected and the sequence mode "3D" is set.

The implementation of 3D-acceleration is quite analogue to the previously discussed PE-acceleration. iPAT² may represent the acceleration in both encoding directions (PE and 3D), or acceleration in 3D-direction only. In the latter case, the acceleration factor for PE is set to 1.

Accordingly, the UI-parameters are initialized in *initialize(...)* as follows:

```
// for PAT in both PE and 3D direction:  
rSeqLim.setAccelFactorPE (1, 4, 1, 2);  
rSeqLim.setAccelFactor3D (1, 4, 1, 2);  
rSeqLim.setRefLinesPE (2, 256, 1, 24);  
rSeqLim.setRefLines3D (2, 256, 1, 24);
```

Please note, that now the minimum value for the PE-acceleration is set to 1 (in order to allow 3D-acceleration without PE-acceleration). As a consequence the sequence has to prevent in *prepare(...)*, that iPAT is selected but both acceleration factors are set to 1:

```
if (rMrProt.PAT().PATMode() != SEQ::PAT_MODE_NONE) {  
    if ((rMrProt.PAT().AccelFactPE() == 1) &&  
        (rMrProt.PAT().AccelFact3D() == 1)) {  
        return SEQU_ERROR;  
    }  
}
```

Setting the YAPS entries for both lines and partition is demonstrated in the code example below. It contains a presentation of the sampling scheme for iPAT² (inplace reference scan mode).

```
// set YAPS parameters  
// Background: The ICE program needs to be informed (via YAPS)  
// about the k-space positions of the first actually measured  
// line and partition. Without iPAT, these values are assumed  
// to be zero. However, with iPAT, the first  
// line(s)/partition(s) might be gaps'!.  
// Since the reference lines are stored in a separate  
// object, the k-space positions of the first reference line/  
// partition are required, too.  
  
if (rMrProt.PAT().PATMode() != SEQ::PAT_MODE_NONE) {  
  
    // Note: For AccelPE=1 (i.e. accel. only in partition  
    // direction), 'FirstRefLine' is still required to define  
    // the range of reference partitions in line direction
```

```
// (see acq. scheme below). Thus we set it now (without
// regard of AccelFactPE):

rSeqExpo.setFirstRefLine (
    rReorderInfo.getMinPATRefLinNo()
);

if (rMrProt.PAT().AccelFactPE() > 1) {
    rSeqExpo.setFirstFourierLine (
        pReorderInfo->getMinPATLinNo ()
    );
} else {
    // no accel in PE direction -> set default values
    rSeqExpo.setFirstFourierLine (0);
}

if (rMrProt.PAT().AccelFact3D() > 1) {
    rSeqExpo.setFirstFourierPartition (
        rReorderInfo.getMinPATParNo()
    );
    rSeqExpo.setFirstRefPartition (
        rReorderInfo.getMinPATRefParNo()
    );
} else {
    // no accel in partition direction
    // -> set default values
    rSeqExpo.setFirstFourierPartition (0);
    rSeqExpo.setFirstRefPartition (-1);
}

} else { // PATMode 'none'
    // reset YAPS parameters
    rSeqExpo.setFirstFourierLine      (0);
    rSeqExpo.setFirstRefLine         (-1);
    rSeqExpo.setFirstFourierPartition (0);
    rSeqExpo.setFirstRefPartition   (-1);
}
```

The 'prepAndUpdateProt' mechanism

A protocol is considered inconsistent if *prepare(...)* fails when called in the context *ContextPrepForBinarySearch*. Apart from the user changing the protocol in the UI, the system is frequently confronted with the task to find new, valid protocol parameters. This will generally happen in one of the following situations:

- ❑ A protocol should run in as many different hardware environments as possible. A protocol may become inconsistent, for example, if a certain TE value cannot be realized on the current gradient set.
- ❑ The GSWD (**G**radient **S**afety **W**atch **D**og) modifies the protocol within a search for parameters to avoid stimulation.
- ❑ A default protocol has to be created, e.g. if a new sequence is inserted into the protocol tree.

The *prepAndUpdateProt* mechanism tries to convert an inconsistent protocol back into a consistent one. This section explains how this is accomplished by *Sequence::prepAndUpdateProt(...)*, with a graphical description given in [Figure D.3.19](#). The goal here is to understand the sequence behavior in these situations and to point out some rules for sequence design that must be taken into account.

As a starting point, suppose that *prepare(...)* failed in one of the cases mentioned above. In order to get back to a valid protocol, two approaches are provided. The first explicitly asks the sequence for support. If the latter fails, a second function attempts to adapt TE and TR values in an iterative process similar to the binary search algorithm of the user interface. Here are the two functions in detail:

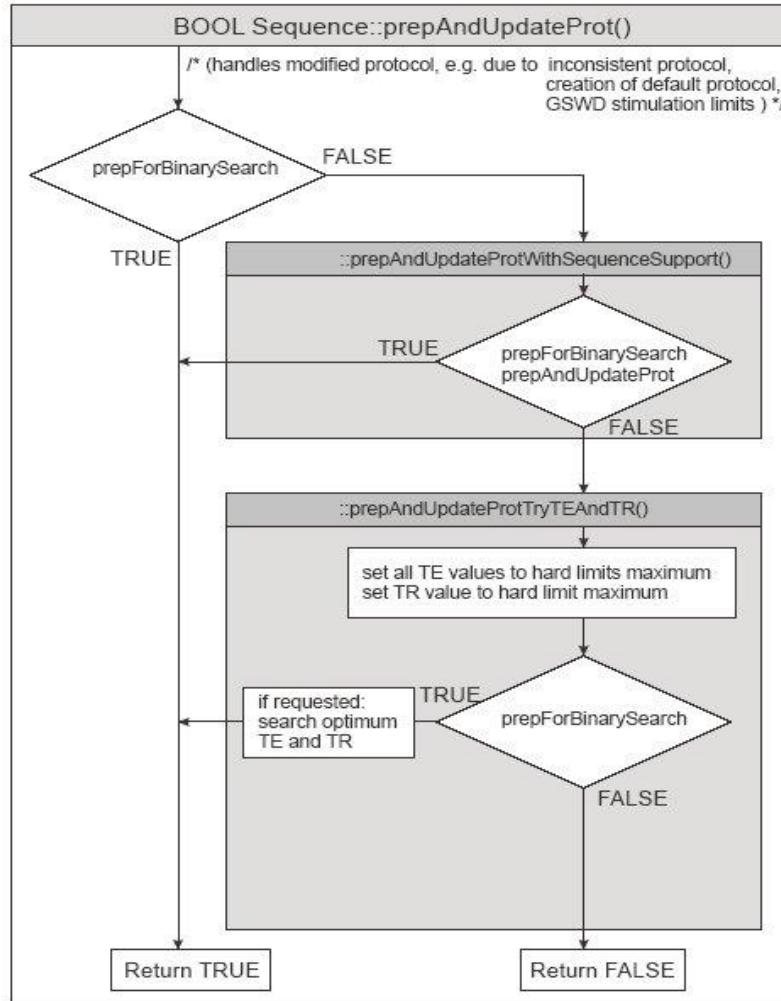


Figure D.3.19: The prepAndUpdateProt mechanism.

❑ Sequence::prepAndUpdateProtWithSequenceSupport (MrProt& rMrProt);

ContextPrep- ForMrProt- Update

Prepare(...) is called once more in ContextPrepForBinarySearch, but additionally the mode ContextPrepForMrProtUpdate is set. The sequence now has the chance to find valid parameters itself. This is one of the very few exceptions where the sequence is allowed to directly set protocol parameters. The following code example is taken from the TSE product sequence and demonstrates a possible implementation in prepare(...):

```

// so far, the sequence has already
//      - calculated the required values for TE,TR,TI
//      - checked if those values match the protocol, and
//          if not, set the flag bOtherT**Needed
//      - all preparing steps have succeeded
lStatus = SEQU_NORMAL;

if (bOtherTENeeded || bOtherTINeeded || bOtherTRNeeded) {

```

```

if (rSeqLim.isContextPrepForMrProtUpdate()) {
    UTRACE (Info, 0, ptModule << ":isContextPrepForMrProtUpdate()==true\n");

    if (bOtherTENeeded) {
        for (lJ=0; lJ < rMrProt.contrasts(); lJ++) {
            rMrProt.te() [lJ] = getMyTE (alNeededTE[lJ]);
        }

        bOtherTENeeded = false;
        UTRACE (Info, 0, ptModule << ": all TE values were adapted.\n");
    }

    if (bOtherTINeeded) {
        rMrProt.ti() [0] = lNeededTI;
        bOtherTINeeded = false;
        UTRACE (Info, 0, ptModule << ": TI adapted.\n");
    }

    if (bOtherTRNeeded) {
        rMrProt.tr() [0] = lNeededTR;
        bOtherTRNeeded = false;
        UTRACE (Info, 0, ptModule << ": TR adapted.\n");
    }
}

if (bOtherTENeeded || bOtherTINeeded || bOtherTRNeeded) {
    bNeedOtherTETITR = true;
    lStatus          = SEQU_ERROR;
}

return lStatus;
}

```

If the sequence supports `ContextPrepForMrProtUpdate` and accordingly `prepare(...)` returns true, the system assumes the new parameters to be valid. There is no further check!

```

Sequence::prepAndUpdateProtTryTEAndTR
    MrProt& rMrProt,
    bool bOptimizeTEAndTR
);

```

First, TR and all TE values are set to their maximum values defined in SeqLim. These parameters are hopefully valid - if the prepare fails again, we ultimately dismiss. In the following scenario, TE and TR are reduced as close as possible to the originally desired values (whether this last step is performed is controlled setting the flag `bOptimizeTEAndTR`).

Consider a few implications of this strategy for the sequence design:

- ❑ The sequence should support the `prepAndUpdateProtTryTEandTR`- approach by setting valid TE and TR hard limits (maximum values) in `SeqLim`. In other words, the sequence must run with these maximum TE and TR values. This requires careful selection of hard limits, especially for multi-contrast sequences.
- ❑ The creation of a default protocol is generally accomplished by taking the default sequence parameters set in the `SeqLim` class. In most cases, this will result in a valid protocol. However, there are circumstances where this approach will fail. An example is the TSE sequence, where TE is limited to discrete values (TE has to be a multiple of the echo spacing). As the minimal echo spacing is dependent on the hardware performance, it is impossible to set a default TE in `SeqLim` that is valid on every system. In these cases, the sequence must support the process of finding valid parameters (see code example above).
- ❑ The GSWD will try to modify the gradient rise time or FOV to stay below the stimulation limit. For many good-natured sequences, this can be accomplished by relying on `prepAndUpdateProtTryTEandTR(...)`. The good-natured assumption, loosely stated, is that TE and TR will increase smoothly with increased gradient demands (increased acquisition time, decreased FOV). However, the TSE example above demonstrates a case where this is certainly not applicable, and a sequence support of `ContextPrepForMrProtUpdate` is inevitable.

The Sequence Unit Test

`libSeqUT` was written in order to perform a unit test for sequences. The goal is to find safety-relevant (internally identified as EGA - Entwicklungsprojekt Gefährdung Analyse [Project Development Risk Analysis]) errors as well as some other errors that are not safety-relevant, but will also produce an unwanted result. These test cases are associated to NUT keys (for Not-Unit-Test-relevant keys).

The basic theory of operation is as follows: the sequence is a .dll file that is linked by the measurement process, i.e. the MesSer of a MAGNETOM installation, or the SequenceTestFrame in a standalone installation, which simulates the measurement process. If the unit test is to be executed, the sequence calls the function `fSEQTest`, which is contained in the `libSeqUT` .dll file. `LibSeqUT` is called just before a run-time event block has been finished. It has access to the run-time event list and will analyze the queued events. `LibSeqUT` is only called on the Intel host if the unit test has been activated, which can be done for the debug and/or the release version (before VE11A only the debug version was possible); note that while the release version is much faster, it is highly recommended to simulate and test the debug version as well in a regular interval, since debug assertions can be an important tool to find possible mistakes. Note that `libSeqUT` cannot be called on the MARS. When all tests are completed, the results will be written into a result file. An interactive intervention of the user is not necessary.

Using the Sequence Unit Test

Each sequence or SBB module must be prepared in a specific way before the unit test can be executed in order to identify those parts of the module that are safety-relevant. SBB functions and pulse sequences must contain a call of the unit test function fSEQTest as the last executable statement in each event block. The prototype of the function is defined as follows:

```
void fSEQTest (
    MrProt&           rMrProt,
    SeqLim&          rSeqLim
    SeqExpo&          rSeqExpo
    unsigned long      ulOriginFunction,
    long               lEventBlockNr,
    long               lLine,
    long               lSliceIndex,
    long               lEcho,
    long               lPartition
);
```

The arguments *rMrProt*, *rSeqLim*, and *rSeqExpo* are references to buffers managed by the sequence. *ulOriginFunction* is used to identify the name of the C++-function from which fSEQTest is called. Possible values for *ulOriginFunction* are listed in the file libSeqUT.h. *lEventBlockNr* is used to pass an arbitrary value to the function fSEQTest to identify the event block. The four parameters *lLine*, *lSliceIndex*, *lEcho* and *lPartition* are used to identify the current line, slice, echo and partition numbers that are used within the event block from which fSEQTest is called. Note that *lSliceIndex* does not specify the current slice loop counter, only the protocol index of the current slice.

Some examples of how fSEQTest is called are given below:

Example 1: Call of fSEQTest from fSBBRSats.cpp

```
fRTEBInit (&asSLC[1RSat].m_sROT_MATRIX);
...
fRTEI ( 0, 0, 0, 0, 0, 0, &sPS[1RSat], 0 );
fRTEI ( sPS[1RSat].lRut,&asRSatzSet[1RSat], 0, 0, 0, 0, 0, 0 );
fRTEI ( sPS[1RSat].lRut, 0, &asRSat[1RSat], 0, 0, 0, 0, 0 );
...
...
mSEQTest(rMrProt,rSeqLim,rSeqExpo,RTEB_ORIGIN_fSBBRSats,0,0,1RSat,0,0);
lStatus = fRTEBFinish();
```

In this first example, the value RTEB_ORIGIN_fSBBRSats identifies the originating function as fSBBRSats.cpp. The event-block number is zero, because there is only one event block within fSBBRSats.c. *lLine*, *lEcho* and *lPartition* are identified as zero because these values are not meaningful in the context of RSat functions. *lSlice* is identified as IRSat, i.e. the RSat number, so that fSEQTest can test for the correct slice-selection gradient amplitude corresponding to this RSat.

Example 2: Call of fSEQTest from a 3D pulse sequence

```

long lSliceIndex = assLC[lSlice].getAnatomicSliceNumber();
...
fRTEBInit (&assLC[lSlice].m_sROT_MATRIX);
...
fRTEI (lT+ sSRF01.lDuration,&sSRF01.zNeg, 0, 0, 0, 0, 0, 0, 0);
fRTEI (lT+= 1500, 0, 0, 0, 0, &sPUL02, 0, 0);
fRTEI (lT+= 200, 0, 0, 0, 0, &sTAB01, 0, &sTAB02, 0);
fRTEI (lT+= 1640, 0, 0, 0, 0, &sPUL03, 0, 0);
...
mSEQTest (rMrProt,rSeqLim,rSeqExpo,
ulTestIdent,50,lLineNumber,lSliceIndex,0,lPartitionNumber);
lStatus = fRTEBFinish();

```

In this second example, the value ulTestIdent identifies the originating function (usually *run(...)*, *runKernel(...)*, or *check(...)*). "50" is an arbitrary value that gives the event block a unique label. lLineNumber, lSliceIndex, lEcho, and lPartitionNumber identify the values of the loop counters during the measurement. These values are used by fSEQTest to verify the gradient amplitudes, RF, and ADC or NCO events.

Note that in both examples, fSEQTest is called by the macro *mSEQTest*. This is done to enable the function calls only when the host version (debug or release) of the sequence is executed with the unit test program. The unit test functionality is therefore not possible with the MARS version (*.so) of the sequence.

When the sequence run is finished, the macro

```
mSEQTest (rMrProt,rSeqLim, rSeqExpo,RTEB_ORIGIN_fSEQRunFinish,0,0,0,0,0);
```

must be called to terminate the unit test. Now libSeqUT will write the test report to a result file in the sequence directory. This file can be read with a HTML browser.

RF pulses used for excitation or refocussing must be marked within the pulse sequences. This information enables the unit test to analyze the evolution of gradient moments. Therefore, the methods shown in the following table must be used for the corresponding SRF pulse objects:

SRF Pulse Type Event	Method
SRF excitation pulses	setTypeExcitation()
SRF refocussing pulses	setTypeRefocussing()

The activity of libSeqUT is controlled by bit no. 24 (0x1000000) of the Sequence Debug Mask. This bit can be passed down to libSeqUT from the Sequence Test Frame.

List of test cases

Error messages of EGA-relevant tests:

- Incorrect readout gradient polarity (AmplSignErr)
- Incorrect readout gradient amplitude (AmplValErr)
- Incorrect slice-select gradient amplitude (RFAmplValIntErr)
- Slice thickness defined in RF pulse structure not the same as protocol value (WrongSliceThickErr)
- Incorrect slice-select gradient amplitude (for external RF pulse) (RFAm-plValExtErr)
- Incorrect slice-select gradient polarity (RFNegAmpLerr)
- Start of READOUT event without a parallel NCO event (StrtNoParaINCO-Err)
- Incorrect NCO frequency at the start of the READOUT event (FreqNotE-quOmegaErr)
- Start of RF_PULSE event without a parallel NCO event (StrtRFNoParaIN-COErr)
- Incorrect NCO frequency at the start of the RF_PULSE event (FrequNotE-quOmegaRFErr)
- Difference between extreme values of 2D phase-encoding moments are incorrect (2DKyMomentsErr)
- Difference between extreme values of 3D phase-encoding moments are incorrect (2DKyMomentsErr)
- Incorrect Ky phase-encoding gradient polarity (values should be decreasing) (MomentsKyIncreasErr)
- Incorrect Kz phase-encoding gradient polarity (values should be decreasing) (MomentsKzIncreasErr)
- Incorrect Ky phase-encoding gradient polarity (values should be increasing) (MomentsKyDecreasErr)
- Incorrect Kz phase-encoding gradient polarity (values should be increasing) (MomentsKzDecreasErr)
- Could not find the FlowSensitivity value in rSeqLim.dFlowSensitivity (NoValueInSEQNameErr)
- Mismatch between rSeqLim.dFlowSensitivity and the sequence name (FlowSensitErr)
- Mismatch between NCO frequency of the READOUT when comparing event block A and B (FqNCOFreqRoErr)
- Mismatch between NCO phase of the READOUT when comparing event block A and B (FqNCOPhaseRoErr)
- Mismatch between NCO frequency of the RF_PULSE when comparing event block A and B (FqNCOFreqRFErr)
- Mismatch between NCO phase of the RF_PULSE when comparing event block A and B (FqNCOPhaseRFErr)

Error messages of non-EGA-relevant tests:

- No gradient activity in the center of the ADC event (NoGrInXErr)
- Readout gradient activity is not in the readout channel (GrNotInRoErr)

- ❑ More than one active gradient in the center of the ADC event (*MoreGrInX-Err*)
- ❑ Label of the readout gradient is not RTEIDENT_PulseRO (*WrongLabelErr*)
- ❑ Readout gradient (RTEIDENT_PulseRO) activity in the wrong channel (*GrInWrongChanErr*)
- ❑ Readout gradient (RTEIDENT_PulseRO) activity without an ADC event (*GrWithoutRoErr*)
- ❑ Invalid gradient activity in the center of the RF_PULSE event (*NoGrInXR-FErr*)
- ❑ Slice-select gradient (RTEIDENT_PulseSS) is not in the slice-select chan-nel (*GrNotInRoRFErr*)
- ❑ More than one active gradient at the center of the RF_PULSE event (*MoreGrInXRFErr*)
- ❑ Slice-select gradient (RTEIDENT_PulseSS) has zero or negative ampli-tude (*AmplSignRFErr*)
- ❑ The label of the slice-select gradient is not RTEIDENT_PulseSS (*RFWrongLabelErr*)
- ❑ Slice-select gradient event on first RF pulse in fSBBDB (*RFAmplSBBDB-BErr*)
- ❑ Incorrect slice-select gradient amplitude in fSBBDB (*RFAmplValSBBDB-BErr*)
- ❑ Internal RF pulse not supported in test of fSBBRSat (*RFIntInSBBRSatErr*)
- ❑ Internal RF pulse not supported in test of fSBBTSat (*RFIntInSBBTSatErr*)
- ❑ Slice-select gradient (RTEIDENT_PulseSS) is not in the slice-select chan-nel (*RFGrInWrongChanErr*)
- ❑ Overlapping slice-select gradients (RTEIDENT_PulseSS) (*OverlapPulSSErr*)
- ❑ Slice-select gradient (RTEIDENT_PulseSS) without parallel RF_PULSE (*GrWithoutRFErr*)
- ❑ End of READOUT event without a parallel NCO event (*EndNoParaNCO-Err*)
- ❑ Unrecognized CSat code (*UnknownCSatCodeErr*)
- ❑ Incorrect NCO frequency in function fSBBNoiseMeas (*FrequNotZeroErr*)
- ❑ NCO frequency for a non-selective RF pulse is not zero (*FrequNotZeroN-SRFErr*)
- ❑ No readout gradient found for calculation of NCO frequency of READOUT event (*CalcFrequNCOErr*)
- ❑ An unexpected READOUT event was found (*RoInWrongFuncErr*)
- ❑ End of RF_PULSE event without a parallel NCO event (*EndRFNoParaN-COErr*)
- ❑ Error during the calcuation of the parameters of OmegaRF (*CalcRF-FrequNCOErr*)
- ❑ The NCO frequency in this origin function is not equal to zero (*RFFrequNotZeroErr*)
- ❑ Timing not correct (*TimingSpinEchoErr*)
- ❑ Timing not correct (No ADC event) (*TimingSpinEchoNoADCErr*)
- ❑ Pulse in the wrong channel (*WrongPulSliceChErr*)
- ❑ More than one label TabKy or TabKz found (*MoreLabelsErr*)
- ❑ Actual Ky-Moment is not equal to already stored one (*KyMomentNotEqualToAlreadyStoredErr*)
- ❑ Actual Kz-Moment is not equal to already stored one (*KzMomentNotEqualToAlreadyStoredErr*)
- ❑ Actual stored Ky-Moment is equal to initialization value (*KyMomentEqual-ToInitErr*)
- ❑ Actual stored Kz-Moment is equal to initialization value (*KzMomentEqual-ToInitErr*)
- ❑ Current minimum measured line number is not equal to expected one (*CurMinMeasLinNumNotEqualToExpOneErr*)
- ❑ Current maximum measured line number is not equal to expected one (*CurMaxMeasLinNumNotEqualToExpOneErr*)

- Current minimum measured partition number is not equal to expected one
(CurMinMeasParNumNotEqualToExpOneErr)
- Current maximum measured partition number is not equal to expected one
(CurMaxMeasParNumNotEqualToExpOneErr)
- Minimum measured line number is not equal to expected one
(TotMinMeasLinNumNotEqualToExpOneErr)
- Maximum measured line number is not equal to expected one (Tot-MaxMeasLinNumNotEqualToExpOneErr)
- Minimum measured partition number is not equal to expected one
(TotMinMeasParNumNotEqualToExpOneErr)
- Maximum measured partition number is not equal to expected one (Tot-MaxMeasParNumNotEqualToExpOneErr)
- Phase encoding moments have non-uniform increments between lines (StepErr2D)
- Phase encoding moments have non-uniform increments between partitions (StepErr3D)
- The expected energy is not equal to the actual energy (EnergyErr)
- The expected measurement time is not equal to the actual measurement time (ms)
(MeasTimeErr)
- The nature of the timing forces a polarity change from - to + (PolarityMinusPlusErr)
- The nature of the timing forces a polarity change from + to - (Polarity-PlusMinusErr)
- Unusual timing of events prevents correct analysis of phase encoding polarity
(UnusualTimingStructureErr)
- The polarity of Ky phase-encoding is not defined (ExpectedPolarityKyIs-NotDefinedErr)
- The polarity of Kz phase-encoding is not defined (ExpectedPolarityKzIs-NotDefinedErr)
- The time interval from an RF event to the next ADC event is too short (RFToADCTimeTooShort)
- The time interval from an ADC event to the next RF event is too short (ADCToRFTimeTooShort)
- Calculated or Expected RF energy is less than or equal to zero (ZeroRfEnergy)
- The expected TR value in rMrProt is not equal to the actual TR value (TRClockErr)
- The expected TI value in rMrProt is not equal to the actual TI value (TIClockErr)
- The ramp up time of the gradient is greater than the duration. (RampUp-TooLong)
- The moment for the GS gradient is not rephased. (IGsMoment-NotRephasedErr)
- The moment for the GR gradient is not rephased. (IGrMoment-NotRephasedErr)
- The moment for the GP gradient is not rephased. (IGpMoment-NotRephasedErr)
- The Acquisition counter in the ADC Mdh is inconsistent. (lInconsistentAc-qCounterErr)

Using libSBB

When programming sequences, one often creates blocks of timing functionality that are not unique to one sequence, but can/should be reused for other sequences as well. One example for such a block is a regional saturation pulse.

IDEA is a very flexible sequence development environment, but with this comes a little more

workload for the sequence programmer, since this block must be included in all sequences to be supported. To reduce this load, so-called *sequence building blocks* (SBB) were implemented. An SBB encapsulates a piece of code that can be reused in different sequences. It acts like a sequence within a sequence.

For the example mentioned above, there is a *SBBRSat* class. This class is coded one time and can then be reused in any sequence without copying the source code to the sequence.

In every sequence where a regional saturation pulse needs to be applied, an occurrence of the *SBBRSat* is derived and the object is prepared, which can then be used anywhere in the *run(...)* function.

As with the regional saturation pulse, the SBB can contain any kind of sequence code. To name a few examples of SBBs:

- all types of sat pulses*
- preparation pulses (STIR/dark blood/tagging/IR/SR)*
- water excitation*
- gradient music*
- insertion of fill times*

Usually the SBB builds a wrapper for an event block, where the inside parts of the event block can be configured via some specific methods. To take advantage of this concept, a base class (*SeqBuildBlock*) exists from which all SBBs are derived and which provides some common methods. These will be described on pages 277 ff (E.3) in more detail.

Beside the event block SBBs, there are helper SBBs available to perform repetitive operations, like:

- playing out complex loop structures*
- calculation of fill times*

All SBBs are contained in the libSBB library. Here is a list of the most relevant SBBs:

- SeqBuildBlock***
Base class for all SBBs.
- SBBBinomialPulses***
Performs a water excitation pulse.
- SBBCSat***
Inserts frequency selective pulses (water, fat, silicone).
- SBBDB***
Inserts dark blood preparation pulses.
- SBBEPIReadOut***
Performs a EPI readout echo train with a gradient reversal.

SBBExcitation**SBBGatingBase** **SBBIRns**

Inserts non slice selective IR or SR pulses.

 SBBIRsel

Inserts slice selective IR or SR pulses.

 SBBList

Keeps track of a list of SBBs for easier preparation.

 SBBMSat

Inserts magnetization transfer pulses.

 SBBNavigator **SBBNoiseMeas** **SBBOptfs** **SBBPATRefScan** **SBBPhaseStabScan**

Performs a phase stabilization scan.

 SBBPrepPulse **SBBPulseSequal** **SBBInvert** **SBBRSat**

Inserts regional saturation pulses.

 SBBSpoilGrad

Creates a spoiler gradient with customizable moment.

 SBBSteadyStateTrigger

Waits for the next trigger while maintaining magnetization in steady-state.

 SBBTokTokTok

Creates 'tok' sounds for patient introduction.

 SBBTSat

Inserts travelling saturation pulses.

SeqConcat

SeqLoop *)

Sequence loop class to realize various kinds of loop structures.

The following methods are functions, not classes, but also belong to the libSBB:

fSBBECGFillTimeRun *)

Performs a wait for physiological triggering.

fSBBFillTimeRun *)

Inserts a specified time slot into the event block.

fSBBMeasRepetDelaysPrep *)

Calculates the total measurement time for a sequence with multiple measurements.

fSBBMeasRepetDelaysRun *)

Inserts pause time between multiple measurements of a sequence according to the delay time specified in the UI.

fSBBOscBitRun *)

Inserts an OSC bit.

*) These SBBs are not derived from SeqBuildBlock base class.

One major benefit of SBBs is that they can be handled in a linked list. The constructor and destructor of the *SeqBuildBlock* base class take care about inserting and deleting the SBB into the list maintained by the *SBBList* class. The list is very helpful for preparing all SBBs with one call, as shown in the next example.

An RSat can be constructed and linked with a CSat by calling

```

SBBList myList;
SeqBuildBlockRSat myRSat (&myList);
SeqBuildBlockCSat myCSat (&myList);
// Now we prepare both:
if (!myList.prepSBBAll()) {
    cout << "An error has occurred while preparing SBBs";
    cout << endl;
}

```

The Base Class SeqBuildBlock

This section describes only a few but important member functions of the SeqBuildBlock base class.

The *SeqBuildBlock* base class has a pure virtual `prep` and `run` function. Therefore, every derived class must provide these functions. As in a sequence, the `prep` function prepares all gradients and RF pulses and calculates the timing while the `run` function sends the real time events during execution.

To interact with its environment, there are some functions available (see also [Figure D.3.20](#)):

□ `getDurationPerRequest`

Returns the duration in μs of the SBB without any ramps lasting outside of the SBB. This value has to be calculated by the SBB and stored in `m_ISBBDurationPerRequest_us`.

□ `getRFInfoPerRequest`

Returns an instance of `MrProtocolData::SeqExpoRFInfo` containing local SAR values and the global RF energy in Ws sent by the SBB. The SAR/energy information must be calculated by the SBB and stored in `m_RFInfoPerRequest`.

□ `setFlipAngle(<angle in degree>)`

sets the `m_dFlipAngle_deg` member, which can be used to calculate the flip angles of the SBB's RF pulses.

□ `getRampTimeOutsideSBB`

Returns the time in μs the last ramp is outside of the SBB. This value has to be calculated by the SBB and stored by `setLastRampTime-OutsideSBB(<RampTime in μs >)`. The function `setLastRamp-DownOutsideSBB` is called with `true`, if `<RampTime in μs >` is greater than 0, otherwise it is called with `false`.

□ `getRequestsPerMeasurement`

Returns how often the run function is called for one measurement. This value has to be calculated by the SBB and stored in `m_lRequestsPerMeasurement`.

If one wants the SBB to change its gradient amplitudes and/or rise times depending upon the gradient mode selected in the UI, the function `updateGradientPerformance(eGradMode)` must be supported. In this function, the rise times and gradient amplitudes of up to 10 gradient groups can be adapted. This function is especially helpful in case of the GSWD detects the protocol will exceed the patient peripheral nerve stimulation limit. Usually, the sequence calls this function when it detects a change in the gradient mode, either due to a change in the UI or due to a possible stimulation situation.

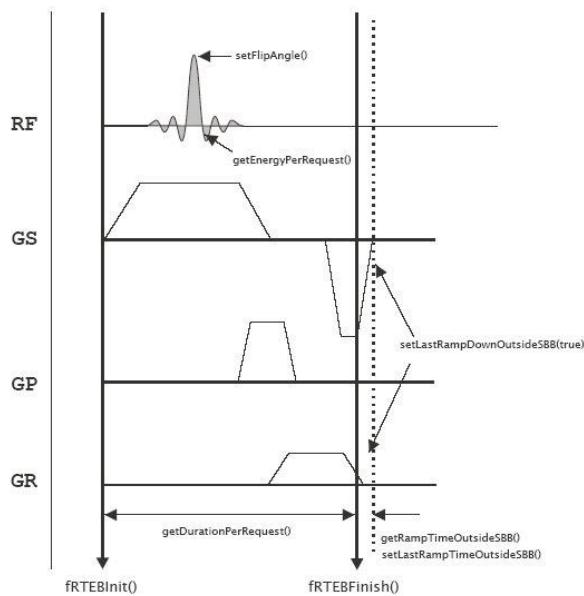


Figure D.3.20: Schematic timing table of a *SeqBuildBlock* and useful methods available through the base class.

Deriving from SeqBuildBlock

To create your own SBB derived from the *SeqBuildBlock* base class, you should at least support the above mentioned values and functions. To easily utilize the list handling, simply call the base class constructor from within your constructor.

```
MySeqBuildBlock::MySeqBuildBlock (SBBLList* pSBBLList,
<additional variables> :
    <member1> (<default value>) ,
    <member2> (<default value>) ,
    SeqBuildBlock(pSBBLList)
{
    // some general initialization
}
```

Then you can take advantage of the *SBBLList::prepSBBAll* method.

Using sequence building blocks

To use a SBB, you generally perform the following steps:

- Create an object of the SBB in the sequence header.*
- Call some member functions of the SBB for global initialization in sequence initialize(...).*
- Call some member functions of the SBB for special preparation in sequence prepare(...).*
- Call the prep function of the SBB from sequence prepare(...).*
- Call the run function of the SBB from sequence run(...) or runKernel(...).*

E.4 UI Sequence Interface

Introduction

The behavior of measurement sequences (e.g. timing, loop structure, etc.) is described by a relatively large set of adjustable parameters which is called the *measurement protocol*.

Inconsistent protocols

The parameter values cannot be chosen arbitrarily. On the one hand, there are technical and physiological limitations (e.g. maximum gradient amplitudes, nerve stimulation, etc.). On the other hand, certain parameter combinations are not allowed for logic reasons (e.g. PAT with only one receiver coil). Hence, there are protocols that do not represent an executable sequence. These protocols are called *inconsistent*.

The sequence user interface (UI) allows to modify the measurement protocol without leading to inconsistencies. The sequence class supports this task by providing a consistent default protocol and a consistency check. The overall strategy to avoid inconsistent protocols is therefore:

- The user starts with a consistent (default) protocol.*
- The user can modify only one parameter at a time. This parameter can only be set to values for which the consistency check succeeds.*
- For certain parameters, the UI offers methods to implicitly adjust other parameters if the consistency check does not succeed immediately (see solve-handlers).*

In the graphical user interface (i.e. EXAM and POET), the parameter limits are visualized by a bar at the bottom of the window. The adjustable values are marked with different colors.

- Green: The parameter can be set to the particular value without changing other parameters.*
- Red: The parameter can be set to the particular value. However, other parameters must be adjusted to maintain consistency.*
- Gray: The parameter cannot be set directly to the particular value.*

Besides the described colors, the user will sometimes also see values that are marked yellow. In this case, multiple values are drawn at the same position in the bar, where some values should have been red and other values should have been green.

Usually, the dependencies among parameters will depend on the particular sequence. This chapter describes how the UI behavior can be adapted to take special dependencies into account.

UI code only on WIN32 platform

Please note that the sequence UI exists only on the host system. Therefore, all code dealing with the UI must be enclosed in the following preprocessor directives:

```
#ifdef WIN32
    // ... UI code ...
#endif
```

Standard Interface

As already mentioned, the sequence must provide a consistent default protocol and a method to

check the consistency of a protocol. These two tasks are fulfilled by the following two methods of the sequence class:

```
NLSStatus initialize (
    SeqLim& rSeqLim
);

NLSStatus prepare (
    MrProt& rProt,
    SeqLim& rSeqLim,
    SeqExpo& rSeqExpo
);
```

The class *SqLim* stores absolute limits (or *hard limits*), increments and default values for numeric parameters. For selection parameters, *SqLim* will hold the selectable options.

The class *MrProt* contains the measurement protocol (all parameter values that can be modified).

The class *SqExpo* (sequence export buffer) is used to store information that depends on the protocol and is calculated by the sequence (e.g. the scan time).

The default protocol must be consistent!

The method *initialize(...)* will define parameter hard limits, default values, etc. in the provided *SqLim* object. The protocol generated from the default values must be consistent. Otherwise, loading the sequence (e.g. in POET) will not be possible.

Prepare context

In the context of the UI framework, the method *prepare(...)* performs a consistency check for the specified protocol. For performance optimization, the developer should then perform only critical tasks instead of a complete preparation. The sequence can inquire the *preparation context* by calling the following method:

```
bool rSeqLim.isContextPrepForBinarySearch();
```

This method will return true if the *prepare(...)*-call is intended as consistency check for the so-called *binary search*. Basically, this algorithm is a *trial-and-error* method. A more detailed explanation is given later in this chapter.

The protocol must not be modified in *prepare(...)*!

Please be aware, that modifying the protocol or parameter limits is not allowed during the sequence *prepare*. (There are only few exceptions from this rule - for example when parameters on the *special card* are initialized with default values.)

Protocol parameter types

There are different parameter types in the protocol:

- Enumeration type
- Boolean type
- Numeric types
- Numeric array types
- In the following sections, these types are explained.

Enumeration type parameters

Enumeration type parameters have a defined set of possible values. Usually, these parameters correspond to drop-down lists on UI parameter cards. You can typically find the definition of these enumerations in the following header file (or one of the files included by this header file):

```
\n4\pkg\MrServers\MrProtSrv\MrProt\SeqDefines.h
```

The selectable options can be specified in the method *initialize(...)*. Here is an example:

```
NLSStatus initialize (SeqLim& rSeqLim) {
    ...
    rSeqLim.setFatSuppression (
        SEQ::FAT_SUPPRESSION_OFF, // default option
        SEQ::FAT_SATURATION,
        SEQ::FAT_EXCITATION
    );
    ...
}
```

The first option specified in *initialize(...)* is always used as default option. If only one option is specified, the parameter will not be shown in the graphical UI.

Boolean type parameters

Boolean type parameters are treated similarly to enumeration type parameters. However, they can have only two different values, *true* or *false*. Usually, Boolean type parameters correspond to check boxes in the UI.

The two possible values are typically given as members of the enumeration *SEQ::Switch*. Here is an example that shows how the selectable values can be specified in *initialize(...)*:

```
NLSStatus initialize (SeqLim& rSeqLim) {
    ...
    rSeqLim.setIntro (
        SEQ::ON, // default value
        SEQ::OFF
    );
    ...
}
```

The first option specified in *initialize(...)* is used as default option. If only one option is specified, the parameter will not be shown in the graphical UI.

Numeric type parameters

Numeric type parameters are integer or floating point numbers. These values have a limited range (*hard limits*) which is defined by an absolute minimum x_{min} and an absolute maximum x_{max} . Furthermore, each parameter has a limited accuracy which is represented by an *increment value* x_i . This value together with the minimum value defines a raster of selectable values (i.e. $\{x_{min}, x_{min} + x_i, x_{min} + 2x_i, \dots\}$).

The following example shows how the limits for a numeric type parameter are specified in *initialize(...)*:

```
NLSStatus initialize (SeqLim& rSeqLim) {
    ...
    rSeqLim.setFlipAngle(
        10, // minimum [degrees]
        90, // maximum [degrees]
        1, // increment [degrees]
        25 // default value [degrees]
    );
    ...
}
```

Numeric array type parameters

Numeric type parameters can be also arranged in arrays. Each of the elements within an array has its own limited range and accuracy.

The following example shows how the limits for a numeric array type parameter are specified in *initialize(...)*:

```
NLSStatus initialize (SeqLim& rSeqLim) {
    ...
    rSeqLim.setTE(
        0, // array index
        500, // minimum [µs]
        60000, // maximum [µs]
        10, // increment [µs]
        10000 // default value [µs]
    );
    ...
}
```

UILink Framework

UI parameters

It is necessary to distinguish between two different sets of parameters. On the one hand, there are the protocol parameters which are stored in *MrProt*. These parameters are visible to the sequence in *prepare(...)*, *check(...)* and *run(...)*. On the other hand, there are the UI parameters which are displayed in the graphical user interface and which can be directly edited by the user.

Although many parameters are identical in both sets, UI parameters and protocol parameters are not necessarily related one-to-one. For example, the UI parameter *aspect ratio* is related to two protocol parameters at the same time. A change of the *aspect ratio* will result in a change of *phase-FoV* and the *number of phase-encoding lines* (in order to keep the *phase resolution* constant).

For the same reason, a numeric type parameter could be theoretically represented by a drop-down list of the UI.

The so-called *UILink framework* defines an interface between the protocol parameters and the UI parameters.

UILink objects

Each parameter on the UI parameter cards is represented by a *UILink* object. The base class for these objects is *MrUILinkBase*, which defines common properties. For example, each UI parameter can have:

- Label identifier string (displayed on the left side of the parameter in the UI)*
- Tooltip text*
- ...

Depending on the particular parameter type, additional information is required. For example, a numeric parameter will have a numeric value and a numeric limits, etc. Therefore, specialized *UILink* classes (and class templates) are derived from *MrUILinkBase* to represent different parameter types (typedefs written in brackets):

- MrUILinkArray*
- MrUILinkLimited <long>(or: LINK_LONG_TYPE)*
- MrUILinkLimited <double>(or: LINK_DOUBLE_TYPE)*
- MrUILinkSelection <unsigned int>(or: LINK_SELECTION_TYPE)*
- MrUILinkSelection <bool>(or: LINK_BOOL_TYPE)*

Array type parameters actually require two *UILink* objects - an *outer* and an *inner* object. The *outer* object keeps the information about the array size and is of class *MrUILinkArray*. The *inner* object represents the actual data (e.g. *LINK_LONG_TYPE*).

Creating UILink objects

New *UILink* objects are created in the *initialize(..)*-method. The following information is required:

- UILink class for the parameter type*
- MR name tag (defines location in the UI)*

A list of the most important *MR name tags* is given at the end of this chapter. A complete list can be found in the file:

```
\n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\MrStdNameTags.h
```

The following expression is used to create new parameters:

```
UILink-class* pUILink = _create <UILink-class> (
    SeqLim*      pSeqLim,
    const char*   sMrNameTag
);
```

An array type parameter is created by means of the following expression:

```
MrUILinkArray* pUILink = _createArray <UILink-class> (
    SeqLim*      pSeqLim,
    const char*   sMrNameTag,
    long          lMaxSize,
    UILink-class*& rpElement
);
```

The additional argument *lMaxSize* defines the maximum allowed number of elements for this array parameter. *rpElement* is used as export reference to the *inner UILink* object. The *outer UILink* object is returned directly.

The following lines show how a *UI parameter* of *long* type is created on the first position of the sequence *special card*.

```
LINK_LONG_TYPE* pUILink = _create <LINK_LONG_TYPE> (
    &rSeqLim,
    MR_TAG_SEQ_WIP1
);
```

Accessing existing *UILink* objects

The expressions to access existing *UILink* objects are similar to those to create *UILink* objects. Again, the following information is needed:

- UILink class for the parameter type*
- MR name tag (defines location in the UI)*

The following expression is used to access an existing parameter:

```
UILink-class* pUILink = _search <UILink-class> (
    SeqLim*      pSeqLim,
    const char*   sMrNameTag
);
```

In case of an array type parameter, the previous expression will return the outer *UILink* object (of class *MrUILinkArray*). The *inner UILink* object can be retrieved by means of the following expression:

```
UILink-class* pUILink = _searchElm <UILink-class> (
    SeqLim*      pSeqLim,
    const char*   sMrNameTag
);
```

UI handler functions

The *UILink* objects do not store information themselves. Instead, there are so-called *UI handler functions* that provide the information on request (e.g. *get-value handlers*, *set-value handlers*, *get-label-id handlers*, ...). This approach comes with several advantages.

At first, there is no redundant information and therefore no potential mismatch with the protocol. The *get-value handlers* will always derive the UI parameter value from protocol parameters. Accordingly, *set-value handlers* will directly modify protocol parameters.

Furthermore, the UI behavior can be adapted to the particular sequence by replacing the standard *handler functions* by specialized implementations. This process is called *registration* of new *handler functions*.

The following list shows the most important *handler function types* with a short explanation.

Handlers for all UI elements

- ❑ *get-value* Returns the UI parameter value. This value is derived from the current protocol.
- ❑ *set-value* Sets the UI parameter value. The protocol will be modified accordingly.
- ❑ *get-label-id* Returns the identifier string to be displayed.
- ❑ *get-unit-id* Returns the unit identifier string to be displayed.
- ❑ *is-available* Returns whether or not the parameter can be modified in the UI.
- ❑ *get-tooltip* Returns the tooltip text to be displayed.
- ❑ *try* Checks the consistency of the new protocol. This handler will be invoked after setting values or resolving conflicts.
- ❑ *Solve* Tries to resolve a parameter conflict by modifying other protocol parameters. This handler is usually invoked if the try-handler does not succeed.

Handlers for numeric parameters

- ❑ *get-precision* Returns the number of digits to be displayed.
- ❑ *get-limits* Returns all possibly selectable values.

Handlers for enumeration type parameters

- ❑ *get-options* Returns all possibly selectable options.
- ❑ *format* Returns the option text for a selection value.

More detailed descriptions of each of the handler functions are given later in this chapter.

Implementing UI handler functions

UI handler functions are global functions. The first argument is always a pointer to the respective UI element which provides access to the *UILink* framework. As an example, the signature of a *get-tooltip handler* for a long parameter is given here:

```
unsigned int getTooltip (
    LINK_LONG_TYPE* const pUILink,
    char*           arg_list[],
    long            lIndex
);
```

pUILink is a pointer to the *UILink* object for which this handler is registered. Other *UILink* objects must be requested by means of *_search* (or *_searchElm* for array type parameters). The following example shows an example implementation for a customized *get-tooltip handler*:

```
unsigned int getTooltip (
    LINK_LONG_TYPE* const pUILink,
    char*           arg_list[],
    long            lIndex)
{
    LINK_LONG_TYPE* pBaseResUILink = _search <LINK_LONG_TYPE>
        (pUILink, MR_TAG_BASE_RESOLUTION);

    LINK_DOUBLE_TYPE* pTEUILink = _searchElm <LINK_DOUBLE_TYPE>
        (pUILink, MR_TAG_TE);

    long   lBaseRes = pBaseResUILink->value (0);
    double dTE      = pTEUILink->value (0);

    static char sToolTip [1024];
    sprintf (sToolTip, "Base res: %i, TE: %f", lBaseRes, dTE);
    arg_list [0] = sToolTip;

    return MRI_STD_STRING;
}
```

This code requests the *UILink* objects for *base resolution* and *TE*. Subsequently, the *get-value handlers* are called and the acquired values are formatted in a string which shall be used as tooltip text. The return value *MRI_STD_STRING* indicates that the text pointed to by *arg_list [0]* should be displayed.

More details about the mechanism to return string values is given in the section "Resource Identifiers".

Registering UI handler functions

The *UI handler functions* must be registered to the *UILink* objects during initialization of the sequence. The following lines demonstrate how a pointer to the customized *get-tooltip handler* can be registered to the *UILink* object for *base resolution*:

```
LINK_LONG_TYPE* pBaseResUILink =
    _search <LINK_LONG_TYPE>(&rSeqLim, MR_TAG_BASE_RESOLUTION);

if (pBaseResUILink != NULL) {
    pBaseResUILink->registerToolTipIdHandler (getTooltip);
}
```

As shown here, a *SeqLim* object can also be passed to *_search* for accessing *UILink* objects.

Please note that *initialize(...)* may be called multiple times. However, *UILink* objects will actually exist only in the final *initialize(...)*-call. Therefore, the obtained pointers to *UILink* objects must be checked for *NULL*.

Keeping original UI handler functions

The methods to register a new *UI handler function* always return a pointer to the previously registered *handler*. Often, it is necessary to keep this function pointer.

For example, a customized *solve handler* might be able to solve sequence specific conflicts. However, general conflicts should still be solved by calling the original *solve handler*.

WIPParameterToolkit (WPT)

The mechanism for adding UI parameters described above involves redundant tasks. Therefore, the WIPParameterToolkit was developed, which provides simple methods to add WIP parameters. For an example please check the FLASH sources.

The tool will automatically register standard implementations of the following UI handlers:

- GetValue
- SetValue
- GetLimits
- GetLabelID
- GetUnitID
- GetOptions
- Format

It is recommended to define your WIP parameter positions with an enum (e.g. in the header file of your sequence):

```
enum WIPParamPositions {
    WIPParamLong    = 0,
    WIPParamDouble  = 1,
    WIPParamSelection = 7,
    WIPParamBool    = 8
};
```

The WIP parameters must be created in the sequence init method (preferably by the associated sequence UI link object). For the host side, it is possible to register customized handler methods.

```
rTool.createLongParameter (WIPParamLong, rSeqLim, "WIP long", "unit", lMin, lMax,
lInc, lDef);
rTool.createDoubleParameter (WIPParamDouble, rSeqLim, "WIP double", "unit 2", 2,
dMin, dMax, dInc, dDef);
rTool.createSelectionParameter (WIPParamSelection, rSeqLim, "WIP selection");
rTool.addOption (WIPParamSelection, "first option");
rTool.addOption (WIPParamSelection, "second option");
rTool.addOption (WIPParamSelection, "third option");
ifdef WIN32
    rTool.registerSolveHandler (WIPParamLong, solveHandlerLong);
endif
```

Please consult the doxygen documentation of the WIPParameterToolkit for a full description on supported WIP UI elements, and have a look at the FLASH example.

Some important notices:

- The create-methods must be called under Windows AND Linux.
- The prepare-method of the WIPParameterTool must be called in each prepare and before WIP-parameters are read.
- In sequence-prepare, the values must be read from the protocol (e.g. rTool.getLongValue (rMrProt, WIPParamLong);)

- In UI-handlers, the values should be read from the UI (e.g. rTool.getLongValue(WIPParamLong);)

Soft Limits and Binary Search

As described in the introduction, the UI needs to calculate the range of selectable values for each parameter on the currently displayed parameter card in EXAM or POET. The principal algorithm is described in this section.

For each parameter, there are *hard limits* which define the maximum allowed parameter range. These limits are defined in *initialize(...)* and cannot be exceeded in any circumstance.

However, the UI will allow parameter changes only if the protocol remains consistent. Therefore, the parameters are restricted by additional limits which are protocol dependent. These limits are called *soft limits*.

Enumeration and Boolean type parameters

The *soft limits* are calculated only for UI parameters. In case of a drop-down list (or a check box), the procedure follows basically a *trial-and-error* approach. At first, the UI will call the *get-options* handler to determine all possibly selectable options. Then, all options are tested separately:

- The current protocol is copied.*
- The set-value handler is used to set the parameter to the option to be tested.*
- The try handler is used to check the consistency of the modified protocol.*
- If this test is positive, the particular option will be added to the soft limits.*
- The original protocol is restored.*

Please note that a drop-down list will only be displayed if there is at least one selectable alternative option to the currently active option.

The above strategy works fine if only a few options must be tested. However, for parameters with a large number of options, this strategy will significantly affect the UI performance. Therefore, numeric parameters must be treated differently.

Numeric type parameters

The standard algorithm for calculating the *soft limits* of numeric parameters is the so-called *binary search*. This algorithm is described in the standard literature.

A prerequisite for this algorithm is that the set of values for which the protocol is consistent must be *convex* (parameter-wise).

Convexity implies basically the following property: If there are two parameter values x_1 and x_2 for which the protocol is consistent, then the protocol will also be consistent for any parameter x between x_1 and x_2 .

Briefly, the algorithm searches the lower and upper *soft limit* separately by the method of *nested intervals*. Thus, the run-time of the algorithm is proportional to the logarithm of the number of adjustable values.

The major advantage of this approach is that the *binary search* reacts flexible to modifications of the sequence behavior (e.g. timing).

Numeric array type parameters	Each element of numeric array type parameters will be treated separately in the <i>binary search</i> .
--------------------------------------	--

Resource Identifiers

Some of the *UI handlers* return texts (e.g. *get-label-id*). Since these texts are often available in different languages, so-called *resource identifiers* (or *resource IDs*) are returned instead of strings. The *resource IDs* can be found in the file:

```
\n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\StdProtRes\StdProtRes.h
```

The according strings which will be actually displayed can be found in the same folder in the file:

```
\n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\StdProtRes\StdProtRes.rc
```

Some of the strings contain inserts of the following form:

`%n!format!`

Here, *n* is a positive integer number and *format* is a printf-like format specification. These inserts serve as placeholders for further *resource IDs* or string values.

The most widely used format specifications are:

- S*: String variable
- r*: Resource ID
- d*: Decimal value

The additional values must usually also be provided by the *UI handlers*. Therefore, the according *UI handlers* have an argument that is called *arg_list*. This array is of type *char*[]*. However, the array can also be filled with *resource IDs* or 32-bit numbers since the format specifier tells how the value should be interpreted.

The number *n* written in front of the format specification tells the array position of the value to be inserted, starting with 1. "%n!S!" will be replaced by the text *arg_list* [*n*-1]; i.e. "%1!S!" inserts a string from *arg_list*[0].

The following example has the result "This is test":

```
arg_list [0] = MRI_STD_EXAMPLE_TEST;      // "test"
return      MRI_STD_EXAMPLE_RETURNID; // "This is %1!r!"
```

If the inserted values contain placeholders themselves, the according positions are calculated relative to the position of the currently inserted value. Therefore, if *arg_list* [*m*] contains the placeholder "%n!S!", this placeholder will be replaced by the text *arg_list* [*m*+1+*n*-1], i.e. *arg_list*[*m*+*n*] (the *m*+1 is necessary since the format string itself is at position *m*, all parameters start at *m*+1).

The following example has the result "This is a big test":

```
arg_list [0] = MRI_STD_EXAMPLE_BIG;      // "a big %1!r!"
arg_list [1] = MRI_STD_EXAMPLE_TEST;      // "test"
```

```
return      MRI_STD_EXAMPLE_RETURNID; // "This is %1!r!"
```

A custom string can be returned as demonstrated in the following example:

```
static char sText [40];
sprintf (sText, "This is a custom text");
arg_list [0] = sText;
return MRI_STD_STRING; // "%1!S!"
```

Description of UI handler functions

Each of the UI handler functions has a signature similar to the following example:

```
typedef bool (*PFctIsAvailable) (
    MrUILinkBase* const pUILink,
    long           lIndex
);
```

The first argument *pUILink* is always a pointer to the *UILink* object for which this handler is called.

The last argument is always *lIndex*, which has a meaning only for array type parameters. The index tells the position of the displayed element within the array.

Additional arguments for the *UI handler functions* are explained in the specific descriptions given within this section.

As shown in the example, there are type definitions for function pointers to *handler functions*. Since the function signature can depend on the particular *UILink* class, there are multiple type definitions for the same UI handler. For example:

```
LINK_LONG_TYPE::PfctGetValue
LINK_DOUBLE_TYPE::PfctGetValue
LINK_SELECTION_TYPE::PfctGetValue
LINK_BOOL_TYPE::PfctGetValue
```

get-value

```
typedef Data-type (*PFctGetValue) (
    UILink-class* const pUILink,
    long           lIndex
);
```

The *Data-type* depends on the particular *UILink-class*:

- ❑ *UILink-class: LINK_LONG_TYPEData-type: long*
- ❑ *UILink-class: LINK_DOUBLE_TYPEData-type: double*
- ❑ *UILink-class: LINK_SELECTION_TYPEData-type: unsigned*
- ❑ *UILink-class: LINK_BOOL_TYPEData-type: bool*

DESCRIPTION: This handler returns the current value of the UI parameter. The value is actually calculated from the currently set protocol.

In some cases, the parameter unit and value need to be converted. For example, timing values are usually stored in microseconds but displayed in milliseconds.

For *LINK_SELECTION_TYPE* parameters, the *unsigned* value is interpreted as *resource ID*. If an option is not predefined in "StdProtRes.h", the index of the currently selected option must be translated into a *resource ID*. This can be done by using the *SET_MODIFIER* macro:

```
unsigned uiSelectedIndex = 2; // just for example
unsigned uiResourceId = MRI_STD_STRING;
SET_MODIFIER(uiResourceId, uiSelectedIndex);
```

Finally, the handler should return *uiResourceId*.

REGISTRATION: `PFctGetValue registerGetValueHandler (
 PFctGetValue pHandlerFct
);`

CALL: `Data-type value (long lIndex) const;`

Note: This method will call the *get-value* handler only if *isAvailable(...)* returns true.

set-value

```
typedef Data-type (*PFctSetValue) (
    UILink-class* const pUILink,
    Data-type          NewVal,
    long               lIndex
);

```

□The *Data-type* depends on the particular *UILink-class* (see description for *get-value handler*).

DESCRIPTION: The UI will invoke this handler to modify UI parameters. This modification can be in the context of the binary search or due to a real user input.

Internally, the handler function will modify protocol parameters to realize the UI parameter value given by *NewVal*.

In some cases, the parameter unit and value need to be converted. For example, timing values are usually stored in microseconds but displayed in milliseconds.

The method will return the actually set value because this value might differ from the desired new value. For example, timing values often need to be on a certain fixed raster. In this case, the nearest value on the raster might be actually set.

As in the case of the *get-value handler*, the *unsigned* value for *LINK_SELECTION_TYPE* parameters is interpreted as *resource IDs*. In case, the option is not predefined in "StdProtRes.h", the *GET_MODIFIER* macro can be used to translate the *resource ID* into the option index:

```
unsigned uiSelectedIndex = GET_MODIFIER (uiResourceId);
```

REGISTRATION: PFctSetValue registerSetValueHandler (
 PFctSetValue pHandlerFct
);

CALL: Data-type value (Data-type NewVal, long lIndex) const;

Note: This method will call the *set-value handler* only if *isEditable(...)* returns true.

get-label-id

```
typedef unsigned (*PFctGetLabelId) (
    UILink-class* const pUILink,
    char*           arg_list[],
    long             lIndex
);

```

DESCRIPTION: This handler function returns the label string, which should be currently displayed.

The return value is evaluated as *resource ID*. The meaning of *arg_list* and the returned *resource ID* are explained in the section "Resource Identifiers".

REGISTRATION: PFctGetLabelId registerGetLabelIdHandler (
 PFctGetLabelId pHandlerFct

```
) ;

CALL:     unsigned labelId (char* arg_list[], long lIndex) const;
```

get-unit-id

```
typedef unsigned (*PFctGetUnitId) (
    UILink-class* const pUILink,
    char*           arg_list[],
    long            lIndex
);
```

DESCRIPTION: This handler function returns the unit identifier string (e.g. "ms"), which should be currently displayed.

The return value is evaluated as *resource ID*. The meaning of *arg_list* and the returned *resource ID* are explained in the section "Resource Identifiers".

The following predefined *resource IDs* are available:

- MRI_STD_UNIT_SEC[s]**
- MRI_STD_UNIT_MS[ms]**
- MRI_STD_UNIT_MM[mm]**
- MRI_STD_UNIT_DEGREES[deg]**
- MRI_STD_UNIT_HZ[Hz]**
- MRI_STD_UNIT_MHZ[MHz]**
- MRI_STD_UNIT_V[V]**
- MRI_STD_UNIT_PERCENT[%]**
- MRI_STD_UNIT_HZ_PER_PIXEL[Hz/Px]**
- MRI_STD_UNIT_CM_PER_SECOND[cm/s]**
- MRI_STD_UNIT_PPM[ppm]**
- MRI_STD_B_VALUE_UNIT[s/mm²]**

REGISTRATION: PFctGetUnitId registerGetUnitIdHandler (
 PFctGetUnitId pHandlerFct
);

CALL: unsigned unitId (char* arg_list[], long lIndex) const;

If no *get-unit-id* handler is registered, this method will return the default value *MRI_STD_EMPTY*.

is-available

```
typedef bool (*PFctIsAvailable) (
    UILink-class* const pUILink,
    long lIndex
);
```

DESCRIPTION: This handler function is invoked by the UI to decide whether a parameter should be displayed or not.

Note: To permanently hide the parameter, call the *MrUILinkBase* member function *unregister()* rather than implementing an *is-available* handler. An *is-available* handler is necessary if the display mode depends on the actual protocol setting.

REGISTRATION: `PFctIsAvailable registerIsAvailableHandler (
 PFctIsAvailable pHandlerFct
);`

CALL: `bool isAvailable (long lIndex) const;`

get-tooltip

```
typedef unsigned (*PFctGetToolTipId) (
    UILink-class* const pUILink,
    char* arg_list[],
    long lIndex
);
```

DESCRIPTION: If a *get-tooltip* handler has been registered for a UI parameter, a tooltip appears when the mouse cursor hovers over the corresponding UI element.

The return value is evaluated as *resource ID*. The meaning of *arg_list* and the returned *resource ID* are explained in the section "Resource Identifiers".

If the method returns 0, no tooltip is shown.

REGISTRATION: `PFctGetToolTipId registerGetToolTipIdHandler (
 PFctGetToolTipId pHandlerFct
);`

CALL: `unsigned toolTipId (char* arg_list[], long lIndex) const;`

try

```
typedef bool (*PFctTry) (
    UILink-class* const pUILink,
    void* pAddMem,
    const MrProtocolData::MrProtData* pOrigProt,
    long lIndex
);
```

DESCRIPTION: The *try handler* is called to perform a consistency check of the currently set protocol. This is for example necessary during the *binary search*.

The argument *pOrigProt* is a pointer to the original protocol (i.e. the protocol before the parameter modification).

The default *try handler* is *MrUILinkBase::stdTryHandler*. Basically, the sequence *prepare(...)*-method is called in the context "binary search". If this method returns success, the try handler will return true (and vice versa).

A customized *try handler* is required only in exceptional cases. Some sequences perform intricate calculations in the *prepare(...)*-method. In other cases, the binary search needs to check a large number of modified protocols.

In these cases, the computational effort of a binary search can be very high. Accordingly, it might be useful to provide a reduced consistency check (at least for some UI parameters).

REGISTRATION: `PFctTry registerTryHandler (`
 `PFctTry pHandlerFct`
`) ;`

CALL: `bool tryProt (`
 `void* pAddMem,`
 `const MrProtocolData::MrProtData* pOrigProt,`
 `long lIndex`
`) const;`

solve

```
typedef unsigned (*PFctSolve) (
    UILink-class* const           pUILink,
    char*                         arg_list[],
    void*                          pAddMem,
    const MrProtocolData::MrProtData* pOrigProt,
    long                           lIndex
);
```

DESCRIPTION: This handler function is called if a parameter value could not be set directly. Then, this handler tries to modify additional UI parameters to re-establish protocol consistency.

This handler will be called during the *binary search* or if the user selects a parameter value in the "red zone" of the parameter bar.

The return value is evaluated as *resource ID*. The meaning of *arg_list* and the returned *resource ID* are explained in the section "Resource Identifiers".

However, generally, the user should return *MRI_STD_CONFIRMATION_MSG*. Then, the UI will automatically create a suitable resolution text that informs about the implicitly made parameter changes.

If the handler returns *0*, the parameter conflict could not be solved (i.e. no consistent protocol could be found).

Note: The modified protocol will not be automatically checked for consistency. Therefore, it can be reasonable to verify the solution by calling the *try handler*.

REGISTRATION: `PFctSolve registerSolveHandler (`
 `PFctSolve pHandlerFct`
`) ;`

CALL: `unsigned solve (`
 `char* arg_list[],`
 `void* pAddMem,`
 `const MrProtocolData::MrProtData* pOrigProt,`
 `long lIndex`
`) const;`

get-precision

```
typedef unsigned (*PFctGetPrecision) (
    UILink-class* const pUILink,
    long lIndex
);
```

DESCRIPTION: This handler informs the UI about the number of fractional digits to be displayed. This handler exists only for the *UILink* classes *LINK_DOUBLE_TYPE* and *LINK_LONG_TYPE*.

If no *get-precision* handler is registered, 6 fractional digits are used for *double* parameters and 0 fractional digits for *long* parameters.

REGISTRATION: PFctGetPrecision registerGetPrecisionHandler (
 PFctGetPrecision pHandlerFct
);

CALL: unsigned precision (long lIndex) const;

get-limits

```
typedef bool (*PFctGetLimits) (
    UILink-class* const pUILink,
    std::vector <MrLimit <Data-type> >& rLimits,
    unsigned long& rulVerify,
    long lIndex
);
```

DESCRIPTION: This *handler function* is called to define the initial search space for the *binary search*. This initial search space can actually consist of multiple intervals or points. The binary search will then treat these intervals or points separately.

The argument *rLimits* is used as export object. The vector can be filled with intervals or points. The template class *MrLimit <TYPE>* offers the following methods to specify intervals or points:

```
❑void setLonely (
    TYPE Value,
    Color Col = GREEN
);
```

Makes this limit an interval with only one point.

```
❑bool setEqualSpaced (
    TYPE Low,
    TYPE High,
    TYPE Incr,
    Color Col = GREEN
);
```

Defines an interval from Low to High. Only points which are on an equally spaced raster will be checked during binary search. The raster spacing is given by Incr.

```
❑bool setBase2 (
    TYPE Low,
    TYPE High,
    Color Col = GREEN
);
```

Specifies a limit interval beginning with Low and ending with High. Each selected value is two times the preceding value. The values Low and High are aligned to a power-of-2 raster (e.g. a Low value of 260 will be turned into 256).

Each of the methods has the argument *Col*, which is used to specify whether the according interval or point should be displayed as *green* or *red* zone in the UI parameter bar.

The argument *rulVerify* is also an export reference to specify how the intervals or points should be treated to find the parameter *soft limits*. You can assign one of the following values:

□VERIFY_BINARY_SEARCH

All intervals that were marked green are verified by means of the binary search algorithm.

This means that the binary search will separate red and green zones within these intervals.

This method is "medium" time consuming.

□VERIFY_SCAN_ALL

All individual values in all specified intervals are verified one by one. This method is "very" time consuming and will slow down the system.

□VERIFY_OFF

The UILink Framework does not verify the specified limit intervals at all. Instead, the framework will use the color codes specified in this method. This method results in the lowest computational effort.

The *handler* returns true if at least one interval or point was specified. Otherwise, the *handler* returns false.

Example for a get-limits handler:

```
bool exampleGetLimits (
    LINK_LONG_TYPE* pThis,
    std::vector <MrLimit <long> >& rLimits,
    long& rulVerify,
    long lPos)
{
    rulVerify = LINK_LONG_TYPE::VERIFY_OFF;
    rLimits.clear();

    MrLimit<long> Interval;

    Interval.setEqualSpaced (0, 5, 1);
    rLimits.push_back (Interval);

    Interval.setEqualSpaced (16, 20, 2, MrLimit <long>::RED);
    rLimits.push_back(Interval);

    Interval.setLonely (30, MrLimit <long>::RED);
    rLimits.push_back(Interval);

    return true;
}
```

The preceding example would result in the following parameter limits:

Green: 0, 1, 2, 3, 4, 5

Red: 16, 18, 20, 30

```
REGISTRATION: PFctGetLimits registerGetLimitsHandler (
    PFctGetLimits pHandlerFct
);
```

CALL:

```
bool getLimits (
    std::vector <MrLimit <Data-type> >& rLimitVector,
    unsigned long&                      rulVerify,
    long                                lIndex
) const;
```

get-options

```
typedef bool (*PFctGetOptions) (
    UILink-class* const      pUILink,
    std::vector <unsigned>& rOptions,
    unsigned long&           rulVerify,
    long                    lIndex
);
```

DESCRIPTION: This handler can be registered to *LINK_SELECTION_TYPE* and *LINK_BOOL_TYPE* parameters and corresponds to the *get-limits handler* for numeric type parameters.

The exported vector *rOptions* is filled with *resource IDs* that represent the selectable options. If these options are not predefined in "StdProtRes.h", the *resource IDs* must be created by using the *SET_MODIFIER* macro. This macro is explained in the description of the *get-value handler*.

As in the case of the *get-limits handler*, the options can be marked *green* or *red*. Since no parameter bar is shown for drop-down lists, the *red* color is substituted by brackets around the option.

To macro *SET_RED* can be used to mark an option *red*. Here is an example:

```
unsigned uiResourceId = MRI_STD_FAT_SUPPRESSION_NONE;
SET_RED (uiResourceId);
rOptions.push_back (uiResourceId);
```

The argument *rulVerify* is an export reference to specify how the intervals or points should be treated to find the parameter *soft limits*. You can assign one of the following values:

VERIFY_SCAN_ALL

All individual options are verified one by one to decide whether the option needs to be displayed as green or red.

VERIFY_OFF

The options are shown as specified and not verified again.

The *handler* returns true if at least one option was specified. Otherwise, the *handler* returns false.

REGISTRATION: `PFctGetOptions registerGetOptionsHandler (
 PFctGetOptions pHandlerFct
);`

CALL: `bool getOptions (
 std::vector <unsigned>& rOptions,
 unsigned long& rulVerify,
 long lIndex
) const;`

format

```
typedef int (*PFctFormat) (
    LINK_SELECTION_TYPE* const pUILink,
    unsigned uiOption
    char* arg_list[],
    long lIndex
);
```

DESCRIPTION: This handler can be registered to *LINK_SELECTION_TYPE*. The task is to return the text which represents the option given by *uiOption*.

The argument *uiOption* must be interpreted as *resource ID*. The macro *GET_MODIFIER* can be used to translate this *resource ID* into the according option index. This is explained in the description for the *set-value handler*.

The returned value must also be interpreted as *resource ID*. The meaning of *arg_list* and the returned *resource ID* are explained in the section "Resource Identifiers".

REGISTRATION: PFctFormat registerFormatHandler (
 PFctFormat pHandlerFct
);

CALL: int format (
 unsigned uiOptionId,
 char* arg_list[],
 long lIndex
);

Additional Methods of MrUILinkBase

At first, the class *MrUILinkBase* and its sub-classes offer methods to call the registered *handler functions*. However, *MrUILinkBase* offers a few more methods that are required for the implementation of *handler functions*. These methods are described in this section.

prot

```
MrProt& prot();
const MrProt& prot() const;
```

DESCRIPTION: These member functions return a reference to the current protocol.

EVAprot

```
MrIEVAProtocol*& EVAprot();
const MrIEVAProtocol*&() const;

(MrIEVAProtocol is a class in namespace MED_MRES_UI_EVAProtocol)
```

DESCRIPTION: These member functions return a reference to the current *evaluation protocol*. The *evaluation protocol* is optional and holds all parameters that are intended to be used by the image calculation system to perform an on-line image evaluation.

The evaluation protocol will be described in a further chapter of this manual.

To avoid naming conflicts, the evaluation protocol implementation is encapsulated in its own namespace (*MED_MRES_UI_EVAProtocol*).

Within the *UILink handler functions* you can reference the evaluation protocol with either of these functions.

sequence

```
Sequence& sequence() const;
```

DESCRIPTION: This member function returns a reference to a *Sequence* object which is just a wrapper of the actual sequence. The following example demonstrates how the real implementation of the particular sequence can be accessed:

```
Sequence& rSequence = pUILink->sequence();
Flash* pSeq = dynamic_cast <Flash*> (rSequence.getSeq());
```

Note: The returned pointer can be *NULL*.

seqLimits

```
SeqLim& seqLimits() const;
```

DESCRIPTION: This method allows to reference the sequence parameter *limits*.

seqExports

```
SeqExpo& seqExports() const;
```

DESCRIPTION: This method allows to reference the sequence export buffer.

unregister

```
void unregister();
```

DESCRIPTION: This method removes a parameter permanently from the UI. Implicitly, all handler functions are deregistered. The method must be called in the sequence *initialize(...)*-method.

isEditable

```
bool isEditable (long lIndex) const;
```

DESCRIPTION: This method tells whether or not the UI element can be currently edited. The method will return *false* only in one of the following cases:

- ❑ If the protocol is opened in READ_ONLY mode. This mode can also be inquired directly by calling the method *viewMode()*.
- ❑ *isAvailable()* returns *false*.
- ❑ No set-value handler is registered.
- ❑ No get-limits handler is registered.

List of search keys

Parameter Name	command to retrieve the addresses of the correspondent UILink-Parameter
Allowed delay (to reduce SAR)	LINK_DOUBLE_TYPE* pAllowDelay = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_ALLOWED_DELAY)
Navigator Phase-Encoding Direction	LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SG_PE_DIR,MR_TAG_NAVIGATOR_ARRAY)
Navigator Normal	LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_SG_NORMAL,MR_TAG_NAVIGATOR_ARRAY)

STIR (Fat Suppressed IR)	<code>LINK_BOOL_TYPE* pStir = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_STIR)</code>
2D-Interpolation	<code>LINK_BOOL_TYPE* p2DInterpol = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_2D_INTERPOLATION)</code>
Acquisition Delay	<code>LINK_LONG_TYPE *pAcqDelay = _search<LINK_LONG_TYPE>(pSeqLim, MR_TAG_ACQUISITION_DELAY);</code>
Acquisition Duration	<code>LINK_LONG_TYPE *pAcqDur = _search<LINK_LONG_TYPE>(pSeqLim, MR_TAG_ACQUISITION_DURATION);</code>
Actual delay (to reduce SAR)	<code>LINK_DOUBLE_TYPE* pActDelay = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SAR_DELAY)</code>
Adj Init(Reset Adjust Volume)	<code>LINK_SELECTION_TYPE* pAdjInit = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_ADJ_INIT)</code>
Adj Volume Description of Orientation	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_ORI_DESCR,MR_TAG_ADJ_VOL)</code>
Adj Volume Dimension-Phase	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_PDIM,MR_TAG_ADJ_VOL)</code>
Adj Volume Dimension-Readout	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_RDIM,MR_TAG_ADJ_VOL)</code>
Adj Volume Dimension-Slice-Select	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SDIM,MR_TAG_ADJ_VOL)</code>
Adj Volume First Angle of Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_ALPHA,MR_TAG_ADJ_VOL)</code>
Adj Volume In-Plane Rotation	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_ROTATION,MR_TAG_ADJ_VOL)</code>
Adj Volume Orientation	<code>LINK_GRAD_DIR_TYPE* pOri = _search<LINK_GRAD_DIR_TYPE>(pSeqLim,MR_TAG_ORIENTATION,MR_TAG_ADJ_VOL)</code>
Adj Volume Position	<code>LINK_VECTOR_TYPE* pVect = _search<LINK_VECTOR_TYPE>(pSeqLim,MR_TAG_POSITION,MR_TAG_ADJ_VOL)</code>
Adj Volume Second Angle of Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_BETA,MR_TAG_ADJ_VOL)</code>
Adjust Reset	<code>LINK_BOOL_TYPE* pAdjReset = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_ADJ_RESET)</code>
Adjust Water Suppression	<code>LINK_BOOL_TYPE* pAdjWatSuppr = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_ADJ_WATER_SUPPRESSION)</code>

Adjust with Body Coil	<code>LINK_BOOL_TYPE* pAdjWithBC = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_ADJ_WITH_BODY_COIL)</code>
Adj-Volume Reset	<code>LINK_BOOL_TYPE* pAdjValReset = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_ADJ_VOL_RESET)</code>
Angio Direction	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_PCANGIO_FLOW_DIR)</code>
Angio Mode	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE >(pSeqLim,MR_TAG_PCANGIO_FLOW_MODE)</code>
Angio Number of encodings	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_PCANGIO_ENCODINGS)</code>
Angio Reconstruction	<code>LINK_BOOL_TYPE* pLong = _searchElm<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_PCANGIO_RECON)</code>
Angio Time to k-Space Center	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_TIME_TO_CENTER)</code>
Angio TOF-Flow Direction	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_TOF_FLOW_DIR)</code>
Angio TOF-Inflow	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_TOF_INFLOW)</code>
Angio Velocity	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_PCANGIO_FLOW_VELOCITY)</code>
Assume Silicon	<code>LINK_BOOL_TYPE* pAssSilicon = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_ADJ_ASSUME_SILICON)</code>
Asymmetric Echo	<code>LINK_SELECTION_TYPE* pAsymEcho = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_ASYMMETRIC_ECHO)</code>
Averages	<code>LINK_LONG_TYPE* pAvg = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_AVERAGES);</code>
Averaging Mode	<code>LINK_SELECTION_TYPE* pAvrMode = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_AVERAGING_MODE)</code>
Bandwidth (per Pixel)	<code>LINK_DOUBLE_TYPE *pBWperPixel= _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_BANDWIDTH);</code>
Base Resolution	<code>LINK_LONG_TYPE* pBaseRes = _search< LINK_LONG_TYPE> (pSeqLim, MR_TAG_BASE_RESOLUTION);</code>

Care Bolus	<code>LINK_BOOL_TYPE* pCareBolus = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_CARE_BOLUS)</code>
Coil Elements	<code>LINK_LONG_TYPE* pCoilElm = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_COIL_ELEMENTS)</code>
Combined Echoes	<code>LINK_LONG_TYPE*pCombEchoes = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_COMBINED_ECHOES);</code>
CompensateT2Decay	<code>LINK_BOOL_TYPE* pComT2Dec = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_COMP_T2_DECAY)</code>
Concatenations	<code>LINK_LONG_TYPE* pConcat= _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_CONCATENATIONS);</code>
Confirm Frequency Adjustment	<code>LINK_BOOL_TYPE* pConfFreqAdj = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_ADJ_CONF_FREQ_ADJ)</code>
Contrasts	<code>LINK_LONG_TYPE*pContrasts=_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG CONTRASTS);</code>
Crush Slab Groups	<code>LINK_LONG_TYPE *pCRSlabGr=_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_CRUSH_SLAB_GROUPS);</code>
CSI BASE Resolution	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_CSI_BASE_RESOLUTION)</code>
CSI Phase FOV	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_PHASE_FOV)</code>
CSI Phase Resolution	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_PHASE_RESOLUTION)</code>
CSI Readout FOV	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_READOUT_FOV)</code>
CSI Slice Resolution	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_SLICE_RESOLUTION)</code>
CSI Slice Thickness	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> pSeqLim,MR_TAG_CSI_SLICE_THICKNESS)</code>
CSI VOI FoV Phase	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_VOI_FOV_PHASE)</code>
CSI VOI FoV Readout	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_VOI_FOV_READ)</code>
CSI VOI FoV Slice	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_CSI_VOI_FOV_SLICE)</code>
CSI VOI On/Off	<code>LINK_BOOL_TYPE* pBool = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_CSI_VOI_ON)</code>

Dark Blood	<code>LINK_BOOL_TYPE* pDarkBlood = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_DARK_BLOOD)</code>
Decoupling Duration	<code>LINK_DOUBLE_TYPE* pDCDur = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_DECOUPLING_DURATION)</code>
Decoupling FlipAngle	<code>LINK_DOUBLE_TYPE* pDCDur = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_DECOUPLING_FLIP_ANGLE)</code>
Decoupling Pause	<code>LINK_DOUBLE_TYPE* pDCPause = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_DECOUPLING_PAUSE)</code>
Decoupling Total Duration	<code>LINK_DOUBLE_TYPE* pDCTotalDur = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_DECOUPLING_TOTAL_DURATION)</code>
Decoupling Type	<code>LINK_SELECTION_TYPE* pDCType = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_DECOUPLING_TYPE)</code>
Delay in TR	<code>LINK_DOUBLE_TYPE* pDelayInTR = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_DELAY_IN_TR)</code>
Delay/Pause between consecutive measurements	<code>LINK_DOUBLE_TYPE *pMeasDelayTime= _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_MEASUREMENT_DELAY_TIMES);</code>
Delta Frequency	<code>LINK_LONG_TYPE *pDelFreq = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_DELTA_FREQUENCY);</code>
Delta Frequency Main	<code>LINK_LONG_TYPE *pDeltaFreq = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_DELTA_FREQUENCY_MAIN);</code>
Delta Frequency Opt1	<code>LINK_LONG_TYPE *pDelFreqOPT1 = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_DELTA_FREQUENCY_OPT_1);</code>
Diffusion B-value	<code>LINK_LONG_TYPE* pBValue = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_BVALUE)</code>
Diffusion Reconstruction	<code>LINK_LONG_TYPE* pDiffRecon = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_DIFF_RECON)</code>
DiffusionMode	<code>LINK_SELECTION_TYPE* pDiffMode = _search<LINK_SELECTION_TYPE> pSeqLim,MR_TAG_DIFF_MODE)</code>

Dimension	<code>LINK_SELECTION_TYPE* pDim = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_DIMENSION)</code>
Dimension-Ex	<code>LINK_SELECTION_TYPE* pDimEx = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_DIMENSION_EX)</code>
Dixon method	<code>LINK_SELECTION_TYPE* pDixon = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_DIXON)</code>
Echo Spacing	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_ECHO_SPACING)</code>
Elliptical Scanning	<code>LINK_BOOL_TYPE* pElliptScan = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_ELLIPTICAL_SCANNING)</code>
Endless Measurement	<code>LINK_BOOL_TYPE* pMeasEndl =_search<LINK_BOOL_TYPE> (pSeqLim, MR_TAG_ENDLESS_MEASUREMENT);</code>
EPI-Factor	<code>LINK_LONG_TYPE*pEpi = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_EPI_FACTOR);</code>
Excitation Pulse	<code>LINK_SELECTION_TYPE* pExcPulse = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_EXCIT_PULSE)</code>
Fat Saturation Mode	<code>LINK_SELECTION_TYPE* pFatSatur= _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FAT_SAT_MODE)</code>
Fat Suppression	<code>LINK_SELECTION_TYPE* pFatSuppr = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FAT_SUPPRESSION)</code>
FFT Scale Factor	<code>LINK_DOUBLE_TYPE *pFFTScale = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_FFT_SCALE_FACTOR);</code>
Filter bool	<code>LINK_BOOL_TYPE* pBool = _searchElm<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_FILTER_BOOL_0,MR_TAG_FILTER_LIST)</code>
Filter Numeric 0	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FILTER_NUM_0,MR_TAG_FILTER_LIST)</code>
Filter Numeric 1	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FILTER_NUM_1,MR_TAG_FILTER_LIST)</code>
Filter Selection	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FILTER_SEL,MR_TAG_FILTER_LIST)</code>
Flip Angle	<code>LINK_DOUBLE_TYPE*pflipAng= _search<LINK_DOUBLE_TYPE> (pSeqLim, MR_TAG_FLIP_ANGLE);</code>

Flow Compensation	<code>LINK_SELECTION_TYPE* pLong = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FLOW_COMP)</code>
FOV in phase encoding direction	<code>LINK_DOUBLE_TYPE* pPhaseFOV = _search<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_PHASE_FOV);</code>
FOV in readout-direction	<code>LINK_DOUBLE_TYPE* pReadFOV= _search<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_READOUT_FOV);</code>
Gradient Mode	<code>LINK_SELECTION_TYPE* pGradMode = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_GRADIENT_MODE)</code>
Grid Tag/Line Angle	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_TAG_ANGLE)</code>
Grid Tag/Line Tag	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_TAG)</code>
Grid Tag/Line-Tag Distance	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_TAG_DIST)</code>
Image Numbering for Main Orientation Cor	<code>LINK_SELECTION_TYPE* pImageNumbCor = _search<LINK_SELECTION_TYPE> pSeqLim,MR_TAG_IMAGE_NUMB_COR)</code>
Image Numbering for Main Orientation Sag	<code>LINK_SELECTION_TYPE* pImageNumbSag = _search<LINK_SELECTION_TYPE> pSeqLim,MR_TAG_IMAGE_NUMB_SAG)</code>
Image Numbering for Main Orientation Tra	<code>LINK_SELECTION_TYPE* pImageNumbTra = _search<LINK_SELECTION_TYPE> pSeqLim,MR_TAG_IMAGE_NUMB_TRA)</code>
Image Numbering Precedence for the 3 Main Orientations (Multi-Slice-Multi-Angle)	<code>LINK_SELECTION_TYPE* pImageNumb = _search<LINK_SELECTION_TYPE> pSeqLim,MR_TAG_IMAGE_NUMB_MSMA)</code>
Images per Slab	<code>LINK_LONG_TYPE* pImagesPerSlab=_search <LINK_LONG_TYPE>(pSeqLim, MR_TAG_IMAGES_PER_SLAB);</code>
Intro	<code>LINK_BOOL_TYPE* pIntro = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_INTRO)</code>
iPat Accel. Factor 3D	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_PAT_ACC_3 D)</code>

iPat Accel. Factor PE	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_PAT_ACC_P E)</code>
iPAT Mode	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_PAT_M ODE)</code>
iPat Ref. Lines 3D	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_PAT_LINES_ 3D)</code>
iPat Ref. Lines PE	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_PAT_LINES_ PE)</code>
Magn. Preparation	<code>LINK_SELECTION_TYPE* pMagnPrep = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_INVERSION)</code>
Manual Reference Amplitudes Reset	<code>LINK_BOOL_TYPE* pManRefAmpl = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_TX_MAN_REF_AMPL_RESET)</code>
Measurements	<code>LINK_LONG_TYPE* pMeas = _search<LINK_LONG_TYPE>(pSeqLim, MR_TAG_MEASUREMENTS);</code>
Minimum Rise Time	<code>LINK_SELECTION_TYPE* pMinRiseTime = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_RISE_TIME)</code>
MTC	<code>LINK_BOOL_TYPE* pMTC = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_MTC)</code>
Multi Slice Mode	<code>LINK_SELECTION_TYPE* pMultiSliceMode = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_MULTI_SLICE_MODE)</code>
Multiple Series	<code>LINK_BOOL_TYPE* pMultSeries = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_MULTIPLE_SERIES)</code>
Navigator Coronal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_POS_COR,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Description of Slice Group Orientation	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_ORI_DESCR,MR_TAG_NAVIGATOR_ARRA Y)</code>

Navigator Distance Factor	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SG_DISTANCE_FACTOR, MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator First Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_ORI_ALPHA,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Groups Position	<code>LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_SG_POSITION,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Groups Size	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SG_SIZE,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Param-Card Position	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_POSITION2,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Sagittal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_POS_SAG,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Second Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_ORI_BETA,MR_TAG_NAVIGATOR_ARRAY)</code>
Navigator Transverse component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_POS_TRA,MR_TAG_NAVIGATOR_ARRAY)</code>
No of Diffusion Directions	<code>LINK_LONG_TYPE* pDiffDir = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_NUMBER_DIFF_DIRS)</code>
No of Diffusion Weightings	<code>LINK_LONG_TYPE* pNoDiffWgt =_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_BVALUE_SIZE)</code>
NOE Count	<code>LINK_LONG_TYPE* pNOECount = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_NOE_COUNT)</code>
NOE Delay	<code>LINK_DOUBLE_TYPE* pNOEDelay = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_NOE_Delay)</code>
NOE Duration	<code>LINK_SELECTION_TYPE* pNOEDuration = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_NOE_DURATION)</code>

NOE FlipAngle	<code>LINK_DOUBLE_TYPE* pNOEFlipAngle = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_NOE_FLIP_ANGLE)</code>
NOE Type	<code>LINK_SELECTION_TYPE* pNOEType = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_NOE_TYPE)</code>
Noise Adjust Switch	<code>LINK_BOOL_TYPE* pNoiseAdjSw = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_NOISE_ADJ UST)</code>
Noise Level for BV	<code>LINK_LONG_TYPE* pNoiseLvl =_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_DIFF_NOISE_LEVEL)</code>
Nucleus Main	<code>LINK_SELECTION_TYPE* pNucleus = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_NUCLEUS_MAIN)</code>
Nucleus Opt	<code>LINK_SELECTION_TYPE* pNucleusOpt = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_NUCLEUS_OPT_1)</code>
Number of RSats	<code>LINK_LONG_TYPE *pRSatCount=_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_RSAT_COUNT);</code>
Number of Slice /Slab groups	<code>LINK_LONG_TYPE *pSGCount=_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_SG_COUNT);</code>
Offset frequency for binomial pulse (fat or water exciation)	<code>LINK_DOUBLE_TYPE* pOffsBinPuls = _search<LINK_DOUBLE_TYPE > (pSeqLim, MR_TAG_BINOM_PULSE_DELTA_FREQ)</code>
phase cycling type	<code>LINK_SELECTION_TYPE* pPhaseCycType = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_TYPE_OF_PHASE_CYCL)</code>
phase encoding type	<code>LINK_SELECTION_TYPE* pPhaseEncType = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_TYPE_ OF_PHASE_ENC)</code>
Phase Oversampling	<code>LINK_DOUBLE_TYPE* pPhaseOvs=_search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_PHASE_OVERSAMPLING)</code>
Phase Partial Fourier Factor	<code>LINK_SELECTION_TYPE* pPartFourierFactor = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_PHASE_PARTIAL_FOURIER)</code>
Phase Resolution	<code>LINK_DOUBLE_TYPE* pPhaseResolution=_search< LINK_DOUBLE_TYPE> (pSeqLim, MR_TAG_PHASE_RESOLUTION);</code>

PhaseStabilize	LINK_BOOL_TYPE* pPhaseStab = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_PHASE_STABILIZE)
Physio Threshold 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_THRESHOLD)
Physio Threshold 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_THRESHOLD)
Physio Arrhythmia Detection	LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_ARRHYTHMIA_DETECTION)
Physio Average Cycle in ms 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_AVG_CYCLE_MS)
Physio Average Cycle in ms 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_AVG_CYCLE_MS)
Physio Calculated Phases 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_CALCULATED_PHASES)
Physio Calculated Phases 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_CALCULATED_PHASES)
Physio Gate off 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_FIRST_GATE_OFF)
Physio Gate off 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_GATE_OFF)
Physio GateOn 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_FIRST_GATE_ON)
Physio GateOn 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_FIRST_GATE_ON)
Physio GateRatio 1	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_GATE_RATIO)
Physio GateRatio 2	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_GATE_RATIO)
Physio Number of physiological measured phases	LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_PHYSIO_PHASES)
Physio Respiration Phase 1	LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FIRST_RESPIRATION_PHASE)
Physio Respiration Phase 2	LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SECOND_RESPIRATION_PHASE)

Physio SignalMode 1	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_FIRST_SIGNAL_MODE)</code>
Physio SignalMode 2	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SECOND_SIGNAL_MODE)</code>
Physio Std.Deviation of Average Cycle in ms 1	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_STD_DEV_MS)</code>
Physio td.Deviation of Average Cycle in ms 1	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_STD_DEV_MS)</code>
Physio TriggerDelay	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_TRIGGER_DELAY)</code>
Physio TriggerDelay	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_TRIGGER_DELAY)</code>
Physio TriggerPulses 1	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_TRIGGER_PULSES)</code>
Physio TriggerPulses 2	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_TRIGGER_PULSES)</code>
Physio TriggerWindow	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_TRIGGER_WINDOW)</code>
Physio User Acquisition Window 1	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_FIRST_ACQUISITION_WINDOW)</code>
Physio User Acquisition Window 2	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE> (pSeqLim,MR_TAG_SECOND_ACQUISITION_WINDOW)</code>
Pixel Size	<code>MrUILinkString* pString = _search<MrUILinkString> (pSeqLim,MR_TAG_PIXEL_SIZE)</code>
Position Mode Selection	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_POSMODE_SEL)</code>
Preparation Scans	<code>LINK_LONG_TYPE*pPrepScan = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_PREPARATION_SCANS);</code>
Protocol Name	<code>MrUILinkString* pString = _search<MrUILinkString> (pSeqLim,MR_TAG_PROTOCOL_NAME)</code>
PSat neg Orientation	<code>LINK_GRAD_DIR_TYPE* pOri = _search<LINK_GRAD_DIR_TYPE> (pSeqLim,MR_TAG_ORIENTATION,MR_TAG_PSAT_NEG)</code>
PSat neg Position	<code>LINK_VECTOR_TYPE* pVect = _search<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_POSITION,MR_TAG_PSAT_NEG)</code>
PSat neg Thickness	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SAT_THICKNESS,MR_TAG_PSAT_NEG)</code>

PSat pos Orientation	<code>LINK_GRAD_DIR_TYPE* pOri = _search<LINK_GRAD_DIR_TYPE> (pSeqLim,MR_TAG_ORIENTATION,MR_TAG_PSAT_POS)</code>
PSat pos Position	<code>LINK_VECTOR_TYPE* pVect = _search<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_POSITION,MR_TAG_PSAT_POS)</code>
PSat pos Thickness	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SAT_THICKNESS,MR_TAG_PSAT_POS)</code>
PSat/TSat-GAP	<code>LINK_DOUBLE_TYPE* pSatGap = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG PTSAT_G AP)</code>
PSat/TSat-Selection	<code>LINK_SELECTION_TYPE* pPsatTsat = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG PTSAT_SEL)</code>
PSat/TSat-Selection-Ex	<code>LINK_SELECTION_TYPE* pPsatTsatEx = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG PTSAT_SEL_EX)</code>
PSat/TSat-Thickness	<code>LINK_DOUBLE_TYPE* pSatThick= _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG PTSAT_THICK)</code>
Receiver Gain	<code>LINK_SELECTION_TYPE* pReodr = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG RECEIVER_GAIN)</code>
Reconstruction Mode	<code>LINK_SELECTION_TYPE* pReconMode = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG RECON_MODE)</code>
Regular Saturation Regions Groups Position	<code>LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG SG POSITION,MR_TAG REGULAR SAT_L IST)</code>
Regular Saturation Regions Param-Card Position	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG POSITION2,MR_TAG REGULAR SAT_LIS T)</code>
Regular Saturation Regions Coronal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG POS COR,MR_TAG REGULAR SAT_LIST)</code>
Regular Saturation Regions Description of Slice Group Orientation	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG ORI_DESCR,MR_TAG REGULAR SAT_LIS T)</code>

Regular Saturation Regions Distance Factor	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SG_DISTANCE_FACTOR, MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions First Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_ALPHA,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Groups Size	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE>(pSeqLim,MR_TAG_SG_SIZE,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Sagittal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_SAG,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Second Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_BETA,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Transverse component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_TRA,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Normal	<code>LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE>(pSeqLim,MR_TAG_SG_NORMAL,MR_TAG_REGULAR_SAT_LIST)</code>
Regular Saturation Regions Phase-Encoding Direction	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_SG_PE_DIR,MR_TAG_REGULAR_SAT_LIST)</code>
Rel. SNR	<code>rUILinkString* pString = _search<MrUILinkString>(pSeqLim,MR_TAG_SNR)</code>
Remove Oversampling	<code>LINK_BOOL_TYPE* pRmOS = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_REMOVE_OVER)</code>
Reordering Mode	<code>LINK_SELECTION_TYPE* pReodr = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_reordered)</code>
Restore Magnet	<code>LINK_BOOL_TYPE* pREsMagn = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_RESTORE_MAGNET)</code>
RF Bandwidth	<code>LINK_LONG_TYPE *pRFBW = _search<LINK_LONG_TYPE>(pSeqLim, MR_TAG_RF_BANDWIDTH);</code>

RF-Pulse Type	<code>LINK_SELECTION_TYPE* pRFPulseType = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_RFPULSE_TYPE)</code>
RF-Spoiling	<code>LINK_BOOL_TYPE* pRFSpoiling = _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_RF_SPOILING)</code>
RX Reset	<code>LINK_BOOL_TYPE* pRXReset = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_RX_RESET)</code>
Saturation Mode	<code>LINK_SELECTION_TYPE* pSatMode = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_SAT_MODE)</code>
Saturation Mode EX	<code>LINK_SELECTION_TYPE* pSatModeEX = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SAT_MODE_EX)</code>
Save Uncombined Images	<code>LINK_BOOL_TYPE* pSAveUncomb= _search<LINK_BOOL_TYPE> (pSeqLim,MR_TAG_SAVE_UNCOMBINED)</code>
Scan Time	<code>MrUILinkString* pString = _search<MrUILinkString>(pSeqLim,MR_TAG_SCAN_TIME)</code>
Scanning Sequence	<code>MrUILinkString* pString = _search<MrUILinkString> (pSeqLim,MR_TAG_SCANNING_SEQUENCE)</code>
Scanning Sequence inkl. Pathname	<code>MrUILinkString* pString = _search<MrUILinkString> (pSeqLim,MR_TAG_SCANNING_SEQUENCE_PATH)</code>
Segments	<code>LINK_LONG_TYPE* pSegments = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_SEGMENTS);</code>
Shift Absolute value	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SEQ_RESERVED1)</code>
Shift Direction 1	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SEQ_RESERVED2)</code>
Shift Direction 2	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SEQ_RESERVED4)</code>
Shift Direction 3	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SEQ_RESERVED5)</code>
Shim Mode	<code>LINK_SELECTION_TYPE* pShimMode = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SHIM_MODE)</code>

Simultaneous excitation	<code>LINK_SELECTION_TYPE* pTagSimExt = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SIMULT_EXCIT)</code>
Slice exitation order (series mode)	<code>LINK_SELECTION_TYPE* pSliceExt = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SERIES_MODE)</code>
Slice Oversampling	<code>LINK_DOUBLE_TYPE* pSIOvs = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SLICE_OVERSAMPLING)</code>
Slice Partial Fourier Factor	<code>LINK_SELECTION_TYPE* pSlicePartFourier = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SLICE_PARTIAL_FOURIER)</code>
Slice Resolution	<code>LINK_DOUBLE_TYPE* pSIRes = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SLICE_RESOLUTION)</code>
Slice(Slab) Normal	<code>LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_SG_NORMAL,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Phase-Encoding Direction	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_SG_PE_DIR,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Coronal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_POS_COR,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Description of Slice Group Orientation	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_ORI_DESCR,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Distance Factor	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SG_DISTANCE_FACTOR, MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) First Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_ORI_ALPHA,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Groups Postition	<code>LINK_VECTOR_TYPE* pVect = _searchElm<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_SG_POSITION,MR_TAG_SLICE_GROUP_LIST)</code>

Slice(Slab) Groups Size	<code>LINK_LONG_TYPE* pLong = _searchElm<LINK_LONG_TYPE>(pSeqLim,MR_TAG_SG_SIZE,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Param-Card Position	<code>LINK_SELECTION_TYPE* pSel = _searchElm<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_POSITION2,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Sagittal component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_SAG,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Second Angle of Slice Group Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_BETA,MR_TAG_SLICE_GROUP_LIST)</code>
Slice(Slab) Transverse component of the Position Vector/ Phase Offcenter	<code>LINK_DOUBLE_TYPE* pDbl = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_TRA,MR_TAG_SLICE_GROUP_LIST)</code>
Spectroscopy CSI FinalMatrix InterPolatedResolutionRead	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_CSI_INTERPOL_RES_1)</code>
Spectroscopy CSI FinalMatrix Phase	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_CSI_INTERPOL_RES_2)</code>
Spectroscopy CSI FinalMatrix Slice	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_CSI_INTERPOL_RES_3)</code>
Spectroscopy VOI Coronal component of the Position Vector	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_COR,MR_TAG_VOI)</code>
Spectroscopy VOI Description of Vol Orientation	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_ORI_DESCR,MR_TAG_VOI)</code>
Spectroscopy VOI Dimension-Phase	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_PDIM,MR_TAG_VOI)</code>
Spectroscopy VOI Dimension-Readout	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_RDIM,MR_TAG_VOI)</code>
Spectroscopy VOI Dimension-Slice-Select	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SDIM,MR_TAG_VOI)</code>
Spectroscopy VOI First Angle of Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_ALPHA,MR_TAG_VOI)</code>
Spectroscopy VOI GSP- Orientation	<code>LINK_GRAD_DIR_TYPE* pOri = _search<LINK_GRAD_DIR_TYPE>(pSeqLim,MR_TAG_ORIENTATION,MR_TAG_VOI)</code>

Spectroscopy Vol GSP-Position	<code>LINK_VECTOR_TYPE* pVect = _search<LINK_VECTOR_TYPE>(pSeqLim,MR_TAG_POSITION,MR_TAG_VOI)</code>
Spectroscopy VOI In-Plane Rotation	<code>LINK_LONG_TYPE* pLong = _search<LINK_LONG_TYPE>(pSeqLim,MR_TAG_ROTATION,MR_TAG_VOI)</code>
Spectroscopy VOI Param-Card Orientation	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_ORIENTATION2,MR_TAG_VOI)</code>
Spectroscopy VOI Param-Card Position	<code>LINK_SELECTION_TYPE* pSel = _search<LINK_SELECTION_TYPE>(pSeqLim,MR_TAG_POSITION2,MR_TAG_VOI)</code>
Spectroscopy VOI Sagittal component of the Position Vector	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_SAG,MR_TAG_VOI)</code>
Spectroscopy VOI Second Angle of Orientation	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_ORI_BETA,MR_TAG_VOI)</code>
Spectroscopy VOI Transverse component of the Position Vector	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_POS_TRA,MR_TAG_VOI)</code>
Suppress permanent AdjFre	<code>LINK_BOOL_TYPE* pSupPermAdjFre = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_ADJ_FRE_NOT_PERMANENT)</code>
TD	<code>LINK_DOUBLE_TYPE *pTD = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TD);</code>
TD (for imaging)	<code>LINK_DOUBLE_TYPE *pTD = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TD_IMAGING);</code>
TE	<code>LINK_DOUBLE_TYPE *pTE = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TE);</code>
TE (for Imaging)	<code>LINK_DOUBLE_TYPE *pTE = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TE_IMAGING);</code>
Thickness of Slices/Reconstructed Partitions	<code>LINK_DOUBLE_TYPE*pSliceThick=_search<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_SLICE_THICKNESS);</code>
Thickness of Slices/Slabs (GSP)	<code>LINK_DOUBLE_TYPE* pSlabThickGSP=_search<LINK_DOUBLE_TYPE>(pSeqLim,MR_TAG_SG_THICKNESS);</code>
TI	<code>LINK_DOUBLE_TYPE *pTI = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TI);</code>
TI (for imaging)	<code>LINK_DOUBLE_TYPE *pTI = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TI_IMAGING);</code>

TR	<code>LINK_DOUBLE_TYPE *pTR = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TR);</code>
TR (for Imaging)	<code>LINK_DOUBLE_TYPE *pTR = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TR_IMAGING);</code>
TSat Orientation	<code>LINK_GRAD_DIR_TYPE* pOri = _search<LINK_GRAD_DIR_TYPE> (pSeqLim,MR_TAG_ORIENTATION,MR_TAG_TSAT)</code>
TSat Position	<code>LINK_VECTOR_TYPE* pVect = _search<LINK_VECTOR_TYPE> (pSeqLim,MR_TAG_POSITION,MR_TAG_TSAT)</code>
TSat Thickness	<code>LINK_DOUBLE_TYPE* pDbl = _search<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_SAT_THICKNESS,MR_TAG_TSAT)</code>
Turbo-Factor	<code>LINK_LONG_TYPE *pTurbo = _search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_TURBO_FACTOR);</code>
TX Amplitude Correction Factor	<code>LINK_DOUBLE_TYPE*pTXAmpl= _search<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TX_CORRECTION_FACTOR);</code>
TX frequency	<code>LINK_DOUBLE_TYPE *pTxf = _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TX_FREQUENCY);</code>
TX Manual Reference Amplitudes	<code>LINK_DOUBLE_TYPE *pTxMan = _searchElm<LINK_DOUBLE_TYPE> (pSeqLim,MR_TAG_TX_MAN_REF_AMPL);</code>
TX Reference Amplitude	<code>LINK_DOUBLE_TYPE *pTxRef= _searchElm<LINK_DOUBLE_TYPE>(pSeqLim, MR_TAG_TX_REFERENCE_AMPLITUDE);</code>
TX Reset	<code>LINK_BOOL_TYPE* pTransReset = _search<LINK_BOOL_TYPE>(pSeqLim,MR_TAG_TX_RESET)</code>
Vector Size	<code>LINK_LONG_TYPE *pVectSize=_search<LINK_LONG_TYPE> (pSeqLim, MR_TAG_VECTOR_SIZE);</code>
Water Suppression	<code>LINK_SELECTION_TYPE* pWaterSuppr = _search<LINK_SELECTION_TYPE> (pSeqLim,MR_TAG_WATER_SUPPRESSION)</code>

For a complete list, please refer to \n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\MrStdNameTags.h.

Direct Protocol Access:

This section shows by example how to access a protocol that is opened by the UI from a separate process. Protocol parameters can be read from and written into the opened protocol.

The protocol is accessed via the MrProtAccess library. Protocol parameters are addressed via their UI name tags as defined in

```
\n4\pkg\MrServers\MrProtSrv\MrProtocol\UILink\MrStdNameTags.h
```

This is the same interface that is used by the parameter cards in the standard user interface.

First, a connector object has to be connected to the opened protocol via a protocol ID. All communication with the protocol subsequently uses this connector object. On a standalone IDEA installation one can connect to a protocol that has been opened in POET. When loading a protocol into poet, the protocol ID is displayed in the *PoetProtAccessService* output window.

Mapped protocol with ID <-2> into process.

The ID is always -2 for the first loaded protocol.

On the host one can get the ID of the protocol in the online-editor (this is the protocol currently opened by the queue) using the *MriClientProtocolController*:

```
MriClientProtocolController sController;
Controller.GetOnlineProtId (ProtID);
```

Single parameters can be read out of the protocol using the connector object. When reading many parameters at once performance is improved by reading the parameters first into a cache object that is set up to hold a subset of protocol parameters. After synchronizing the cache with the protocol, these parameters can be read out of the cache into a Data object. Parameters are written directly into the protocol using the connector object.

Parameters can be single values (e.g. *MR_TAG_BANDWIDTH* in a single echo sequence), or collections (e.g. *MR_TAG_SLICE_GROUP_LIST*). Collections can be made up out of groups or single values. The number of steps needed to reach a single value member is the depth of a collection. For example, *MR_TAG_SLICE_GROUP_LIST* has depth 2 since it is a collection of slice groups and each slice group consist of elements, as e.g. *MR_TAG_POS_COR*, that are single values. In order to load a collection completely (i.e. all its single value elements) into the cache, its entry in the cache must be set up with sufficient depth. It is not possible to modify a collection directly in the protocol, instead its single values must be set individually.

Note: The programmer should make sure that his application does not modify protocols outside of the intended scope. He should take care to always end the application before leaving. In addition, he should limit the life time of the application to the life time of a sequence or ICE program, and/or check the identity of the protocol before manipulating it.

E.5 Advanced Sequence Programming

Arbitrary RF Pulses

The libRT offers several classes for common internal rf pulse shapes like "Linear", "Triangle", "Rectangular", "Gaussian", "Sinc", "Hyperbolic secant". In addition, there are two other classes available: "External" and "Arbitrary".

While "External" reads the information about the RF profile from an external file generated using

PulseTool (see section D.4), "Arbitrary" pulses can be used to calculate the profile from within the sequence during runtime. This might be useful if one wants to create spatially selective rf pulses (e.g. for grid tagging or for 2D rf pulses).

To create an arbitrary rf pulse shape by calculating the support points on the fly within the sequence, an array of type `sSample` is used, where the amplitude is stored as complex float values (magnitude and phase in radians) for every sampling point. The magnitude values must be between 0 and 1, the phase value between 0 and 2π . The array may not contain more than `IRF_MAX_ENVELOPE_ELEMENTS` (currently 8192) elements.

The restriction of amplitude values to the range 0..1 implies, that negative values can only be achieved by using a phase shift of π in the phase array. Example: A sinus pulse shape would decompose into:

```
Amp = abs (sin(x)), Phase = ( sin (x) > 0 ? 0 : π )
```

The number of sample points in the array may be specified using the `setSamples()` method and the total pulse duration in μs is passed to the class using `setDuration()` (see [Figure D.5.1](#)). These two methods determine the 'dwell' time of the arbitrary pulse. The minimum dwell time is 0.5 μs .

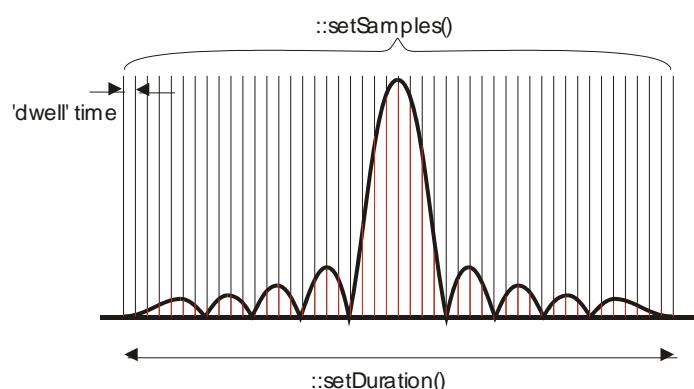
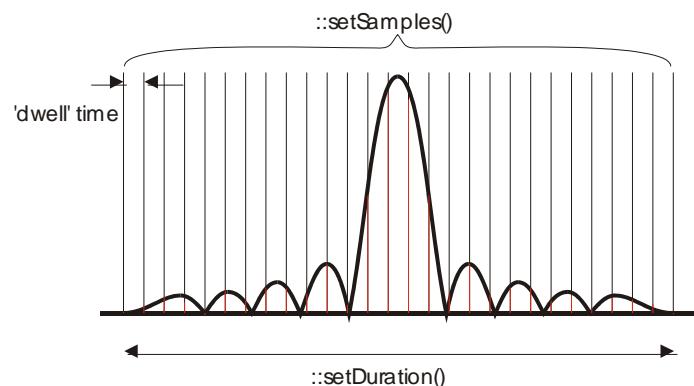


Figure D.5.1: Arbitrary rf pulse shapes with different 'dwell' times.

In addition, the effective amplitude integral of the pulse has to be calculated for proper SAR calculations. The effective amplitude integral is the magnitude of the sum of all complex amplitudes.

To complete the settings, the following parameters must be specified: initial phase, flip angle, slice thickness, and (for the unit test) the type of the pulse (excitation, refocusing, unknown). Then the rf pulse can be prepared with

```
prepArbitrary (pMrProt, pSeqExpo, *ssSample, EffectiveAmplitudeIntegral)
and used in the fSEQRun portion of the sequence.
```

Example code:

```
#include "MrServers\MrMeasSrv\SeqIF\libRT\libRT.h"
ssSample myRFPulseArray[IRF_MAX_ENVELOPE_ELEMENTS]
sRF_PULSE_ARB myRFPulse;

mySequence::prepare(....)
{
...
// here we calculate an rf pulse with a simple half sine
double dEffectiveAmplIntegral = 0.;
double dRealAmpl = 0., dImagAmpl = 0.;
for (int i=0;i<IRF_MAX_ENVELOPE_ELEMENTS; i++)
{
    myRFPulseArray[i].flAbs =float(sin(3.1415926/
        double(IRF_MAX_ENVELOPE_ELEMENTS*i)));
    myRFPulseArray[i].flPha = float(0.);
    dRealAmpl += (myRFPulseArray[i]).flAbs *
        cos ((myRFPulseArray[i]).flPha);
    dImagAmpl += (myRFPulseArray[i]).flAbs *
        sin ((myRFPulseArray[i]).flPha);
}
dEffectiveAmplIntegral = sqrt(dRealAmpl*dRealAmpl +dImagAmpl*dImagAmpl);
if (!myRFPulse.prepArbitrary(pMrProt,pSeqExpo,
    dEffectiveAmplIntegral))
{
    cout << "ERROR: "<<myRFPulse.getNLSStatus()<<endl;
    return (false);
};
myRFPulse.setStartTime(<start time of the rf pulse>);
```

```
...
    return (true);
}

mySequence::run(..)

{
    myRFPulse.run();
}
```

Arbitrary Gradient Shapes

In some applications, non-trapezoidal gradient shapes are desired, but the standard preparation methods of `libRT` support only trapezoidal gradients. However, you can define and use arbitrary gradient shapes, just alike the arbitrary rf pulse shapes (see previous chapter).

For this, the class `sGRAD_PULSE_ARB` is provided.

The method

```
void sGRAD_PULSE_ARB::setRampShape (const float *paflShape, const long lRampUpPoints, const long lRampDownPoints = 0, bool bLastValueNEZero = false);
```

allows for the definition of arbitrary shapes. The shape array values have to be in the range [-1 .. +1].

Physiological Triggering

NUMARIS/4 allows the triggering of sequences by cardiac or respiratory signals. The available trigger modes must be allowed in `fSeqInit`:

```
pSeqLim->addPhysioMode (SEQ::SIGNAL_NONE, SEQ::METHOD_NONE);
pSeqLim->addPhysioMode (SEQ::SIGNAL_CARDIAC,
                         SEQ::METHOD_TRIGGERING);
pSeqLim->addPhysioMode (SEQ::SIGNAL_RESPIRATION,
                         SEQ::METHOD_TRIGGERING);
```

The actual setting chosen by the user is available from the protocol:

```
static SEQ::PhysioSignal FirstSignal = SEQ::SIGNAL_NONE;
static SEQ::PhysioMethod FirstMethod = SEQ::METHOD_NONE;
static SEQ::PhysioSignal SecondSignal = SEQ::SIGNAL_NONE;
static SEQ::PhysioMethod SecondMethod = SEQ::METHOD_NONE;
pMrProt->physiology().getPhysioMode (FirstSignal, FirstMethod,
                                       SecondSignal, SecondMethod);
```

Here, `PhysioMethod` refers to the method used for sequence control (`METHOD_TRIGGERING`). `PhysioSignal` identifies the port in the physiological monitoring unit (PMU) where the signal is received, i.e., ECG (`SIGNAL_CARDIAC`) or respiration (`SIGNAL_RESPIRATION`). `SIGNAL_CARDIAC` will enable the ECG leads, the pulse trigger device and the external trigger in the user interface. `SIGNAL_RESPIRATION` will enable the “Resp/Trigger” mode.

Usually the SeqLoop sends the `sSYNC` event which suspends the sequence execution until a trigger is received. However, one can also directly call the respective function.

Optical Trigger Signal Output

A sequence programmable optical trigger signal output exists in the control cabinet (Trig_out). It can be made available by the local service engineer.

The trigger signal output can be programmed as EXTTRIGGER and can be used e.g. for the triggering of stimulation devices in functional imaging experiments. In the product ep2d_bold sequence this optical trigger output is played out at the beginning of each volume block (duration 10 msec).

Code example:

```
//. -----
//. Sync object: EXTTRIGGER
//. -----
static sSYNC_EXTTRIGGER      sExtTrigger ("Ident_Ext") ;
.

.NLS_STATUS fSEQPrep (...)

..

//. -----
//. Preparation of the trigger bit
//. -----
// Prepare the trigger bit
sExtTrigger.prep      ( 200 ) ; // duration: 200us
.

static NLS_STATUS fSEQRunKernel (...)

.

fRTEI( 0, 0, 0, 0, 0, 0, 0, &sExtTrigger);

AA = ROTSENS * theMouseData.getA().getValue();
BB = ROTSENS * theMouseData.getB().getValue();
CC = ROTSENS * theMouseData.getC().getValue();
Quaternion QMouse (0.0, AA,BB,CC);
double dd = AA * AA + BB * BB + CC * CC;
if (dd < 1.0)
{
    QDeltaMouse.setScalar_s(sqrt(1.0 - dd));
}
else
{
    dd = 1.0 / sqrt(dd);
    QDeltaMouse.setScalar_s(0.0);
    QDeltaMouse.setImag_x (QDeltaMouse.getImag_x() * dd);
    QDeltaMouse.setImag_y (QDeltaMouse.getImag_y() * dd);
    QDeltaMouse.setImag_z (QDeltaMouse.getImag_z() * dd);
}
```

E.6 Spectroscopy

Introduction

This section describes features of the product spectroscopy sequences. A typical spectroscopy sequence requires fewer interdependent parameters than an imaging sequence and hence is easier to implement. In most cases, it is easier to write a spectroscopy sequence virtually from scratch rather than modifying an imaging sequence.

Common Concepts

There are several sequence design features that are common to the product spectroscopy sequences. Most of these concepts are true for all the sequences, except as noted.

SEQUENCE CARD

The product spectroscopy sequences have a Sequence tab card that is different from the imaging Sequence tab card (only one tab rather than two subtabs). This card is enforced by using:

```
rSeqLim.setSequenceCard (SEQ::SEQUENCE_CARD_SPECTROSCOPY);
```

PREPARATORY SCANS

The number of preparatory or dummy scans are defined explicitly in the Sequence tab rather than internally calculated as in the imaging sequences:

```
rSeqLim.setPreparingScans (0, 16, 1, 4);
```

Separate executions of the *runKernel* are performed with the ADC explicitly turned off using *fRTSetReadoutEnable*.

ADC SAMPLING

The product spectroscopy sequences begin ADC sampling after the spoiler following the last slice selective rf pulse for svs or csi and at the specified TE time for the fid sequence. The sampling process and ADC duration is controlled by several factors:

- ❑ *Bandwidth. This is the total bandwidth of the measurement in Hz (twice the Nyquist frequency), specified by:*

```
rSeqLim.setBandWidth (0, 1000, 2500, 10, 1000);
```

- ❑ *Vector Size. This is the final number of points that are normally stored in the Local Database.*

The signal is measured with a two-fold oversampling that is default in the product sequences and is normally removed by the Ice program. If kept, then the number of points stored will be two times the Vector Size:

```
rSeqLim.setVectorSize (256, 2048, SEQ::BASE2, 512);
rSeqLim.setReadoutOSFactor (2.);
rSeqLim.setRemoveOversampling (SEQ::YES, SEQ::NO);
```

In order to use the product postprocessing software, the Vector Size must be a binary number.

❑*Extra sampling points.* As mentioned earlier, the ADC sampling for the product svs and csi sequences begins at the end of the spoiler gradient following the final slice selective rf pulse with the peak signal occurring at the specified TE. If the TE is sufficiently long, then there may be extra time between the beginning of the sampling and the peak signal. If so, these extra points before the TE time can be used together with extra points after the two-fold oversampled signal to remove a truncation artifact that occurs during the removal of the oversampling. These extra points are specified in the Measurement Data Header parameters setPreCutoff/setPostCutoff:

```
m_sAdc.getMDH().setPreCutOff(usSamplesBeforeEcho);
m_sAdc.getMDH().setPostCutOff(usSamplesAfterEcho);
```

and are processed in one of the functors in the Ice program IcePrgSpectroscopy. The extra points can be confirmed by examining the Mdh during simulation of the sequence.

WATER SUPPRESSION

The standard water suppression technique used in the product sequences is the Water Elimination Technique, (WET) technique (Ogg, et al, Journal of Magnetic Resonance B 104, 1 – 10 (1994)). Water suppression is selected in the Contrast > Common tab using the Water Suppression parameters. The options are activated with the SeqLim method setWaterSuppression and the bandwidth of the pulses using the method setRfBandwidth:

```
rSeqLim.setWaterSuppression (SEQ::WATER_SATURATION,
    SEQ::WATER_SUPPRESSION_OFF, SEQ::WATER_SUPPRESSION_WEAK,
    SEQ::WATER_SUPPRESSION_RF_OFF);
rSeqLim.setRfBandwidth (20, 60, 5, 35);
```

The option WATER_SATURATION provides the most complete water suppression when the pulse amplitude optimization is performed (see below). The option WATER_SUPPRESSION_WEAK will retain some water signal in the measured data. This may be necessary in certain situations to perform the coil combination for phased array receiver coils. The option WATER_SUPPRESSION_RF_OFF is available. In this case, the spectrum is acquired with the WET code present, but with no rf pulses active. This allows a reference dataset to be acquired for eddy current correction.

Currently, the WET technique is hard-coded in all the product sequences. However, there is no internal adaptation of the pulses based on field strength, so it is necessary to optimize the pulse amplitudes to the Adjust Volume. This optimization is achieved as a prescan measurement specified

by the *AdjustWaterSuppression* checkbox:

```
rSeqLim.setAdjWatSup (SEQ::ENABLE, SEQ::DISABLE);
```

The optimization process will be executed automatically if the checkbox is selected or it can be manually initiated from the Options Adjustment menu. A STEAM sequence is executed using a single Gaussian pulse at the transmitter frequency followed by three volume selective pulses (defining the Adjust Volume) and measuring the resulting signal at a TE of 135 ms. Eleven correction factors for the Gaussian pulse are tested and the factor producing the best water suppression is applied to the WET pulses in the main sequence. The correction factor will be applied to the rf pulses if the Flip Angle Correction flag is set for the rf pulse:

```
rf_ws1.setFlipAngleCorrection();
```

It is recommended to always perform this adjustment to produce the best possible water suppression.

SPECTRAL SUPPRESSION

For svs and csi sequences, there is also the option for Spectral suppression. This approach uses a MEGA technique with two narrow band rf pulses to saturate the selected signals (MEGA REFERENCE). The option is selected from the Contrast > Common tab. There are two type of suppression available:

```
rSeqLim.setSpectralSuppression (SEQ::SPEC_SUPPR_NONE,  
SEQ::SPEC_SUPPR_LIPID, SEQ::SPEC_SUPPR_LIPID_WATER);
```

The choice of SPEC_SUPPR_LIPID applies both pulses at the Lipid frequency (~1.2 ppm), while the SPEC_SUPPR_LIPID_WATER option applies one pulse at the water frequency and the other at the lipid frequency:

```
rSeqLim.setSpecLipidSupprBandwidth (0.80, 2.50, 0.05, 1.55);  
rSeqLim.setSpecLipidSupprDeltaPos (-6.0, -3.0, 0.01, -3.4);  
rSeqLim.setSpecWaterSupprBandwidth (0.80, 2.50, 0.05, 1.55);  
rSeqLim.setSpecWaterSupprDeltaPos (-0.5, 3.0, 0.01, 0.0);
```

Note the frequencies displayed in the User Interface are defined relative to water at 0.0 ppm rather than 4.7 ppm. For water suppression without lipid suppression, the WET technique is recommended.

SHIM OPTIONS

Shimming to the volume of interest is critical to successful spectroscopy studies. It is normally done to the 1H signal. Gradient offset currents are used to correct for linear distortions and dedicated DC power supplies connected to shim coils are used to correct for second order distortions if present. Two approaches to shimming are possible: manual shimming and interactive shimming.

Interactive Shimming

Interactive shimming

Interactive shimming is initiated from the Options Adjustments dropdown menu. A STEAM sequence is executed using three volume selective pulses (defining the Adjust Volume) and measures the resulting signal at a TE of 135 ms with a frequency resolution of 2 Hz per point. An infinite loop is executed and the results are displayed both graphically and numerically. An optimal shim is indicated by a large integrated magnitude time domain signal (indicated by |P|), a long T2* (calculated assuming a Lorentzian line shape) or a narrow linewidth (indicated by FWHM). Linear corrections are performed through gradient offset currents and second order corrections are available if the Advanced Shim option is installed.

Automatic Shimming

Automatic Shimming

While imaging studies normally use the Standard shim mode, Spectroscopy sequences have additional options:

```
rSeqLim.setAdjShim ( SEQ::ADJSHIM_ADVANCED, SEQ::ADJSHIM_STANDARD,  
SEQ::ADJSHIM_TUNEUP, SEQ::ADJSHIM_ABDOMEN, SEQ::ADJSHIM_BRAIN,  
SEQ::ADJSHIM_BREAST, SEQ::ADJSHIM_PROSTATE );
```

Tuneup. Tuneup shim uses the base shim generated during tune up of the system by Service.

DESS-Based Shimming. A 3D phase map using a DESS (Double Echo Steady State) gradient echo sequence mapping the entire bore. The spatial resolution is scanner-specific, but is approximately 7x7x7 mm³. The phase difference between the two echoes is used to calculate shim currents to optimize the homogeneity to the Adjust Volume of the protocol.

Two options are available:

Standard: A single DESS sequence is executed.

Advanced: Two DESS sequences are executed with opposite gradient polarities.

Uncompensated zero-order eddy currents are averaged out. This is the default selection.

GRE-Based Shimming. A dual TE gradient echo sequence is used to map the bore with different shim volumes. The spatial resolution is region-specific:

Brain: 3D Gradient Echo, 1 axial slab, centered -25 mm Tra, FOV 236.25 mm x 280 mm x 281mm, Matrix 64 PE x 128 RO x 72 SL.

3D Breast: 3D Gradient Echo, 1 axial slab, centered 20 mm Cor, FOV 380 mm x 380 mm x 240 mm, Matrix 93 PE x 192 RO x 48 SL mm.

Abdomen: 2D Gradient Echo, 36 axial slices, Isocenter, 10 mm thickness, 25% distance factor, FOV 379.6875 mm x 450 mm, Matrix 64 PE x 128 RO.

- o Prostate: 2D Gradient Echo, 36 Axial slices, Isocenter, 5.5 mm thickness, 0% distance factor, FOV 355 mm x 355 mm, Matrix 64 PE x 128 RO.

LOOPING (Measurement Data Header)

Looping in the product sequences is performed explicitly rather than using the SeqLoop class of the imaging sequences. The following Mdh parameters are expected by the Icc program:

setCacq (cAverage)	Averages
setCseg (cSegment)	First CSI dimension
setClin (cLines)	Second CSI dimension
setCpar (cPartition)	Third CSI dimension
setCeco (cEcho)	Echoes
setCrep (cRepetition)	Repetitions
setCphs (cPhase)	Reserved: used for Water Reference scan

In addition, the Mdh values *setPreCutoff* and *setPostCutoff* should be used as described earlier to define points before and after the signal to be removed in post-processing.

OTHER COMMON PARAMETERS IN MrProt

Within the product spectroscopy sequences, many of the parameters within *MrProt* are used for the same purpose as in imaging sequences:

```
rSeqLim.setAllowedFrequency ( 40200000, 63850000 );
rSeqLim.setFlipAngle      ( 0.000, 180.000, 1.000, 90.000 );
rSeqLim.setFilterType     ( SEQ::PRESCAN_NORMALIZE );
rSeqLim.setAverages       ( 1,      1024,      1,      1 );
rSeqLim.setTR              ( 0, 1000000, 10000000, 10000, 1500000 );
rSeqLim.setTE              ( 0, 30000,   300000, 1000,   30000 );
rSeqLim.setRepetitions    ( 0,      511,      1,      0 );
rSeqLim.setRepetitionsDelayTime ( 0, 10000000, 100000, 0 );
rSeqLim.addPhysioMode(SEQ::SIGNAL_NONE, SEQ::METHOD_NONE);
rSeqLim.addPhysioMode(SEQ::SIGNAL_CARDIAC, SEQ::METHOD_TRIGGERING);
rSeqLim.addPhysioMode(SEQ::SIGNAL_RESPIRATION, SEQ::METHOD_TRIGGERING);
rSeqLim.setPhases         ( 1, 1, 1, 1 );
```

However, there is a section of parameters that are specific to spectroscopy sequences. These are stored in *rMrProt.Spectroscopy()*.

Another common feature of all spectroscopy sequences is that the images used in the Graphical User Interface can only be distortion uncorrected images; the images must have been acquired using REF mode.

Sequence-Specific Parameters

RF PULSES

For FID sequences, the excitation pulse for ^1H is a non-frequency selective (rectangular) pulse derived from RF_PULSE_RECT with a pulse duration of 0.5 ms. The frequency for rf and ADC objects is the Adjust Frequency value.

For SVS and CSI sequences, volume selective excitation is used. When the *SeqLim* method

```
rSeqLim.isSVSSequence(true);
```

is set to true, then the User Interface allow graphical selection of the Volume of Interest (VOI) in the respective direction:

```
rSeqLim.setVoIPosCor (-150, 150, .1, 0);
rSeqLim.setVoIPosSag (-150, 150, .1, 0);
rSeqLim.setVoIPosTra (-150, 150, .1, 0);

rSeqLim.setVoISizePhase (3, 40, 1, 20);
rSeqLim.setVoISizeReadout (3, 40, 1, 20);
rSeqLim.setVoISizeSlice (3, 40, 1, 20);
```

For CSI sequences, the VOI will be represented as a white box superimposed on top of the CSI grid.

The Slice selective rf pulses used in the product SVS and CSI sequences are read from an external file. Beginning in VE11A, this file is located in %MEASDATA%\extrf_spec.pls, the same directory as the imaging external rf pulse library. The pulse durations for all three pulses adapt based on the respective VOI dimension. A dummy pulse is prepared using the waveform with a default duration. If the pulse is clipped, then the duration for that dimension is increased in increments of 0.100 msec to lower the pulse amplitude. In this way, the pulse duration is kept short so that the bandwidth and the associated slice gradient amplitude is as high as possible. As a result, it is important that the TE defined in the default protocol be able to accommodate the increased durations.

Because of the possible pulse duration variations, the SVS and CSI sequences fix the time between the first two rf pulses to 6.9 ms to maintain the same spectral patterns.

DELTA FREQUENCY

The volume selection graphics on the User Interface and stored in *MrProt* define the location for the desired rf excitation volume. The corresponding rf center frequencies will be calculated as offsets to the transmitter frequency (determined by Adjust Frequency). For ^1H , this will normally be the water resonant frequency. Due to the chemical shift differences, the water at one location has the same resonant frequency as the metabolite at a different location. As a result, the rf pulses will excite metabolites at different locations from the graphical position. While this misregistration occurs for excitation pulses used in normal imaging, it is not normally visualized since all rf pulses excite the

same plane. However, for volume selective spectroscopy using rf pulses that excite tissue volumes in different orientations, this misregistration can cause undesirable signal variations. The misregistration is worse when low amplitude slice selection gradients are used, such as for large excitation volumes. To reduce the extent of this misregistration, the rf center frequency can be shifted using the Delta Frequency:

```
rSeqLim.setSliceSelectDeltaFrequency (-5.0, 0.0, 0.1, 0.0);
```

The center frequencies for all slice selective rf pulses will be shifted by this amount. Note the frequencies displayed in the User Interface are defined relative to water at 0.0 ppm rather than 4.7 ppm. The Delta frequency is stored in *MrProt* in the variable

```
rMrProt.getsSpecPara().getdDeltaFrequency()
```

WATER REFERENCE SCANS

For single voxel and CSI scans, a water reference scan can be enabled on the Sequence card. This measurement will perform the sequence but remove the water suppression pulses, leaving the gradient pulsing and ADC sampling. This water reference scan can be used for intensity correction, eddy current correction, or phased array coil combination.

```
rSeqLim.setAutoRefScan ( SEQ::AUTO_REF_SCAN_OFF,  
                         SEQ::AUTO_REF_SCAN_INLINE, SEQ::AUTO_REF_SCAN_NO_INLINE,  
                         SEQ::AUTO_REF_SCAN_SAVE_ALL );  
rSeqLim.setAutoRefScanNo ( 1, 6, 1, 1 );
```

The number of averages for the reference scan may be set independently of the number of averages for the water suppressed scans. AUTO_REF_SCAN_INLINE will acquire the water unsuppressed scan and use it in the data processing but not store the unsuppressed scan. AUTO_REF_SCAN_NO_INLINE will acquire and store the water unsuppressed scan but not use it in the data processing. AUTO_REF_SCAN_SAVE_ALL will acquire the unsuppressed scan and use it in the data processing and store in a separate series in the Local database.

STEAM SEQUENCES

There are both single voxel and CSI sequences that use three 90° volume selective rf pulses to excite the VOI, producing a stimulated echo. For these sequences, there is a mixing time TM defined, which is the time between the second and third rf pulses. This is stored in the TD array of *MrProt* but is activated in the User Interface using:

```
rSeqLim.setTD(0, 10000, 100000, 1000, 10000);
```

CSI SEQUENCE-SPECIFIC FEATURES

CSI sequences have a number of features that are unique to them. Several parameters in *MrProt* are specific for all CSI sequences:

□ 2D versus 3D CSI

The product sequences support both 2D- and 3D-CSI. For 2D-CSI, the encoding is perpendicular to the slice direction. The choice is controlled by the parameter:

```
rSeqLim.setDimension(SEQ::DIM_2, SEQ::DIM_3);
```

□ Field Of View and Encoding Steps

The Field of View is displayed in the Graphical User Interface as a grid superimposed in yellow on the VOI box in white. This display is activated by:

```
rSeqLim.isCSISequence(true);
```

The FOV center will always be coincident to the center of the corresponding VOI dimension.

The FOV dimensions are defined by:

```
rSeqLim.setReadoutFOV      (100.0, 300.0, 1.0, 120.0);
rSeqLim.setPhaseFOV        (100.0, 300.0, 1.0, 120.0);
rSeqLim.setSliceThickness(10.0, 40.0, 1.0, 20.0); //for 2D
rSeqLim.setSlabThickness (55, 300); //for 3D
```

There are two differences in the method that the resolution is controlled compared to imaging sequences. The first is that the scan resolution for each CSI dimension is explicitly defined in the User Interface (rather than using the Phase/Slice resolution percentages). This is specified using:

```
rSeqLim.setBaseResolution(8, MAX_CSI_ENCODE,
                         SEQ::INC_NORMAL, 16);
rSeqLim.setPELines         (8, MAX_CSI_ENCODE, 1, 16);
rSeqLim.setPartition       (8, MAX_CSI_ENCODE,
                           SEQ::INC_NORMAL, 8);
```

and stored in *MrProt* as

```
rMrProt.kSpace().baseResolution();
rMrProt.kSpace().phaseEncodingLines();
rMrProt.kSpace().partitions();
```

The second difference is that the display resolution (sent to the Local Database) can be different from the scan resolution (acquired). This is specified using:

```
rSeqLim.setFinalMatrixSizeRead (8, MAX_CSI_ENCODE,
                               SEQ::BASE2, 16);
```

```
rSeqLim.setfinalMatrixSizePhase(8, MAX_CSI_ENCODE,
    SEQ::BASE2, 16);
rSeqLim.setfinalMatrixSizeSlice(8, MAX_CSI_ENCODE,
    SEQ::BASE2, 8);
```

and stored in *MrProt* as:

```
rMrProt.spectroscopy().finalMatrixSizeRead();
rMrProt.spectroscopy().finalMatrixSizePhase();
rMrProt.spectroscopy().finalMatrixSizeSlice();
```

If the display resolution is larger than the scan resolution, the Ice program will perform zero filling prior to performing the spatial FT in the particular direction.

Phase encoding is performed in the same manner as in imaging sequences: at the time of the signal detection, the most negative k value for a given dimension is expected to have the lowest loop index (e.g., 0) and the k moments will increase linearly as the loop index increases.

□ Phase Encoding type:

Besides the specification of the number of phase encoding steps in each dimension, it is possible to define the completeness of k space sampling using

```
rSeqLim.setPhaseEncodingType(SEQ::PHASE_ENCODING_FULL,
    SEQ::PHASE_ENCODING_ELLIPTICAL,
    SEQ::PHASE_ENCODING_WEIGHTED);
```

○ Full:

Full encoding acquires a signal following all possible kx-ky-kz combinations for the specified number of averages. This generates the most complete k space, but requires the longest measurement time.

○ Elliptical:

Elliptical encoding superimposes an elliptical filter (spherical if the number of encoding steps in the three dimensions are equal in number) when determining the choice of kx-ky-kz combinations. Combinations where the vector sum of the gradients exceeded the boundary of the filter region are excluded from acquisition. All measured combinations will be acquired for the specified number of averages.

○ Weighted:

Weighted encoding superimposes an Hanning filter in each direction when determining the choice of kx-ky-kz combinations and signal averaging. Only combinations near the center of the k volume are measured for the specified number of averages. As the gradient magnitude increases, fewer averages are measured according to the filter shape. Combinations where the vector sum of the gradients exceeded the boundary of the filter region are excluded from acquisition. If the number of averages equals 1, then Weighted is equivalent to Elliptical.

□ Short Term / Long Term Averaging

For measurements with multiple averages, the phase encoding order can be controlled in a limited way. Two options are provided, based on

```
rSeqLim.setAveragingMode(SEQ::INNER_LOOP, SEQ::OUTER_LOOP);  
rMrProt.kSpace().getucAveragingMode();
```

- *Short Term:*

Short Term averaging performs all signal averaging for a kx-ky-kz combination before moving to the next one.

- *Long Term:*

Long Term averaging acquires a signal from each kx-ky-kz combination before acquiring a second average for any one.

□ Fully Excited VOI

The csi_se sequence is a double echo sequence using optimal control 180 degree rf pulses for volume selection (Mao, et al, *Journal of Magnetic Resonance* 79,1-10, (1969)). Due to the low gradient amplitudes required for volume selection, chemical shift misregistration occurs that distorts spectra in voxels near the edge of the VOI. This distortion can be eliminated using Outer Volume Suppression or OVS:

```
rSeqLim.setOuterVolumeSuppression(SEQ::OFF, SEQ::ON);  
rMrProt.getsSpecPara().getucOuterVolumeSuppression();
```

The technique internally excites larger excitation volumes for the in-plane directions and applies four saturation pulses to suppress the unwanted signal. These saturation pulses are not visible in the user interface, except in reducing the number of allowed variable saturation pulses.

The csi_st sequence uses three 90 degree volume selective rf pulses to excite the VOI, producing a stimulated echo. The csi_slaser uses a 90 degree excitation and two adiabatic full passage rf pulses, all volume selective. Because of the larger bandwidths for the rf pulses for these two sequences, this misregistration is small and can be ignored.

Ice Program

The product Ice program used by all Spectroscopy sequences is different from those used in the imaging sequences. It is referenced in the *prepare* method of the sequences using:

```
rSeqExpo.setICEProgramFilename ("%SiemensIceProgs%\IcePrgSpectroscopy");
```

The program was completely rewritten for VE11 and is now a true functor-based Ice program in the style of the imaging Ice programs. Many Ice functions are used to perform standard operations within the functors. The *IcePrgSpectroscopy.ipr* file calls a configurator file, *IceSpectroF*, which creates the functor chain based on *MrProt* and *SqExpo* parameters and provides some configuration parameters to certain functors. The same Ice program is used for all product sequences; the functor chain is modified as appropriate. Trace macros are incorporated into the functors to print out relevant data.

There are two primary classes of functors used: *SpecScanFunctors*, which operate on the incoming measurement data, and *SpecReconFunctors*, which perform the majority of the data processing (there are two functors from the Imaging Ice program that are also used). While the .h files for these functors and the *IceSpectroF* configurator file are delivered in the */MrSpecAcq/Ice* directory and respective subdirectories, below is a brief summary of the function of each functor:

SpecScan Functors

SpecScanFunctors receive the digitized signal from all receiver coils together with the Measurement Data header (*Mdh*). All incoming lines of data are processed by this set of functors. In addition to using the imaging functor *Flags*, there are five functors that may be present:

1. SpecRoFtFunctor:

This functor is standard and is executed multi-threaded. *SpecRoFtFunctor* is used to remove the extra points that might be present (*Mdh* parameters *PreCutoff* and *PostCutoff*). A complex conjugation is performed to provide the correct input to the *syngoMR* Spectroscopy tab. Also, if specified in *MrProt*, the oversampled data will be removed so that the subsequent functors receive data corresponding to the Vector Size. Refer to the section on ADC Sampling at the beginning of this chapter for the specific parameters in *MrProt* that control this functionality. *TRACE_14* will display the number of extra points as well as the values of the reduced data sets.

2. SpecRawObjProvider:

This functor is standard and is executed single-threaded. *SpecRawObjProvider* is used to create a raw data object based on the parameters from *MrProt* and *SqExpo* and store all signals from all coils provided by *SpecRoFTFunctor* and until the scan is completed. The maximum number of averages is stored in RAM for usage in later functors. For CSI scans, the raw object allocates memory and initializes to zero for the number of averages specified in *MrProt*, even if elliptical or weighted scanning is selected. While this approach will use slightly more RAM than necessary, signal averaging can be done without worrying about the phase encoding type. *TRACE_5* will display the size of the raw data object. *TRACE_14* will display the number of stored averages and the data at the beginning of *ComputeScan*.

3. SpecFreqCorrFunctor:

This functor is optional and is executed single-threaded. If the Online Frequency Correction

```
rSeqLim.setFrequencyCorrectedAccumulation(SEQ::OFF,SEQ::ON);
```

is selected in *MrProt*, *SpecFreqCorrFunctor* performs an online frequency correction by calculating the frequency offset from the Adjust Frequency value and applying to all signals. Both the original and corrected signals are stored. TRACE_15 will display the frequency difference for the first average.

4. *SpecOnlineDisplayFunctor*:

This functor is optional and is executed single-threaded. *SpecOnlineDisplayFunctor* branches from the *SpecRawObjProvider* and receives its own copy of the input data. It will create a "image" for online display at the operator's console. The display consists of three signals form the maximum channel: a cumulative time domain signal, the incoming time domain signal and the magnitude FT of the incoming time domain signal.

5. *SpecLooper*:

This functor is standard and is executed single-threaded. *SpecLooper* is the terminal functor of the line-by-line functor chain and is the beginning of the *SpecReconFunctors* part of the Ice program. It is derived from the imaging functor *ImageLooper* and has the same functionality except for its output. It The data that is passed to the remaining part of the program consists of all columns, lines, channels, partitions, and segments; in other words, this functor supports a 3D-CSI dataset with multiple channels. TRACE_7 will print out the loop counters.

SpecRecon Functors

SpecReconFunctors receive the data for a measurement from *SpecLooper* and process it as a unit. Besides an instance of the *ImageSend* functor configured to send complex data, there are nine possible functors in this part of the functor chain that may be present:

1. *SpecAccuFunctor*:

This functor is standard and is executed multi-threaded. *SpecAccuFunctor* performs the signal averaging of the data. Since the incoming data is sent one average at a time, it is possible for SVS or FID sequences to save each signal as a series in the Local Database using the *SaveSingleAverages* checkbox in *MrProt*:

```
rSeqLim.setSaveSingleAverages(SEQ::OFF, SEQ::ON);
```

Averaging only occurs for a complete phase cycle, based on the number of averages stored in RAM by *SpecRawObjProvider*. Scaling of the data to the number of averages is also performed. TRACE_15 will display the type of phase cycling.

2. *SpecMiniHeadFillFunctor*:

This functor is standard and is executed single-threaded. *SpecMiniHeadFillFunctor* is similar in concept to the imaging functor *MiniHeadFillDecorator*. The standard parameters in the MiniHeader are defined with additional spectroscopy-specific modifications made as appropriate. TRACE_30 will display the contents of the Miniheader.

3. *SpecFinishFunctor*:

This functor is standard and executed multi-threaded. *SpecFinishFunctor* is similar in concept to the imaging functor *ImaFinishDecorator*. The data are converted from the Patient Coordinate System to the DICOM coordinate system. TRACE_30 will print out the current SODA set. For CSI scans, the data are reordered so that the two innermost dimensions are Columns x Lines (necessary for the Send functionality). For 3D-CSI scans, the Partitions dimension is copied into the Phases dimension (normally used for cardiac phases). This is

necessary for sending the entire 3D dataset as a single file since the Send process only allows one partition at a time to be sent.

4. *SpecCSIFilterFunctor*:

This functor is present if the sequence is a CSI sequence and the filter option is selected:

```
rSeqLim.setFilterType(SEQ::HAMMING);
```

It is executed multi-threaded. *SpecCSIFilterFunctor* applies a matched Hamming filter to the CSI data prior to the spatial FT in all directions. This is intended to reduce truncation artifacts that might occur due to the limited sampling in the k directions.

5. *SpecCSIFunctor*:

This functor is present if the sequence is a CSI sequence and is executed multi-threaded. *SpecCSIFunctor* performs the spatial FT to convert CSI data to a voxel-based representation for each receiver channel. The functor supports 1, 2, and 3 spatial transformations. Scaling of the data to the voxel size is also performed. TRACE_24 prints landmark messages.

6. *SpecNormalizeFunctor*:

This functor is present if an array coil is used and is executed multi-threaded. *SpecNormalizeFunctor* is used to normalize the signals from the different array coils to a common value. If Prescan Normalize is selected

```
rSeqLim.setFilterType(SEQ::PRESCAN_NORMALIZE);
```

7. *SpecPhaseCorrFunctor*:

This functor is present if an array coil is used and is executed multi-threaded. *SpecPhaseCorrFunctor* is used to perform a phase correction necessary for the correct combination of phased array coils in the time domain. If a water reference scan is available, then the phase correction factors are taken from the respective coils. If no water reference scan is available, then a zero-order phase correction is performed to the signal from each coil.

8. *SpecCombineFunctor*:

This functor is present if an array coil is used and is executed multi-threaded. *SpecCombineFunctor* performs the combination of the channels dimension (receiver coils). The input data is complex floating point data, scaled and phase corrected. TRACE_27 will print out the control structure.

9. *SpecFatIronQuantFunctor*:

This functor is optional and is used for the Histo pulse sequence for estimation of fat and iron in liver. It is executed single-threaded. *SpecFatIronQuantFunctor* will calculate fat and iron percentages from T2 calculations based on five TE values. Three reports are sent to the Local Database: a summary table, a spectral image based on the first TE time, and a calculated percentage of fat and iron deposition. TRACE_24 prints landmark messages.

If a measurement is aborted, SVS data is processed based on the available data and the results are sent to the Local Database. However, incomplete CSI data (short term averaging, long term averaging less than one complete average) is disregarded.

Spectroscopy Output Files

The Ice program stores the “unprocessed” time-domain signal (complex real/imaginary, floating point) from the tissue volume defined by the sequence (FID, SVS) or for each voxel (CSI). The number of complex samples for each signal is specified as the Vector Size in *MrProt*. To send data out of the syngoMR software, there are three possible approaches:

Direct archive of the data.

From the Browser, it is possible to archive the data to a CD or DVD in the same manner as images. This format can be read back onto the scanner for later reprocessing.

RDA format.

From the syngo Spectroscopy tab, there is an option to Export Raw Data. This process results in a file with the extension .rda. In this file, there is an ASCII header containing many of the measurement parameters, followed by the time-domain signal in binary, floating point real/imaginary pairs.

DICOM format.

From the Patient Browser > Application > DICOM Tools > Export MR Spectroscopy Data, it is possible to send complex data to a PACS node using the Enhanced DICOM format (Dicom 3.0 standard). This is an export process only; it is currently not possible to import a DICOM Spectroscopy file. The service engineer must configure a DICOM node with the logical name MRSPECTRO_STORAGE and the Supported DICOM service of Storage. The receiving node must accept MR Spectroscopy as a supported Storage Class.

The directly archived and RDA data are complex conjugated relative to the DICOM format above, so an inverse FT is necessary to obtain the correct frequency display while the DICOM data requires a forward FT for the correct display.

X-Nucleus Operation

Multinuclear or X-Nucleus sequences for imaging or spectroscopy require additional features in order to function properly. As there are only two product sequences that currently support multinuclear operation and both are spectroscopy sequences, the following discussion will use these sequences as an example. Unless otherwise indicated, this entire discussion would be equally appropriate for multinuclear imaging sequences.

SYSTEM HARDWARE/SOFTWARE REQUIREMENTS

For X-nucleus operation, there are several requirements that must be fulfilled. First is that the *syngoMR* system must support multinuclear operation. This feature is enabled by a Multinuclear license in the license file. With this license present, the following nuclei are recognized in the software: ^1H , ^3He , ^7Li , ^{13}C , ^{17}O (only at 3T), ^{19}F , ^{23}Na , ^{31}P , ^{129}Xe .

The transmitter and receiver software must also operate at the X-nucleus frequency. For X-nucleus operation, the frequency synthesizer will automatically shift to the desired frequency range. A broadband RF power amplifier is installed in the Control Cabinet that is capable of excitation at all frequencies for the above nuclei. All frequency ranges are calibrated by the Service Engineer during Tune-Up of the systems.

Multinuclear operation also requires transmitter and receiver coils that operate at the desired frequency range (it may be the same coil). In the coil file associated with the coil, the nucleus for the coil must be defined (the example below is for a double resonant coil):

```
Misc.tSupportedNuclei = "31P"  
Misc.tCompatibleNuclei = "31P,1H"  
Misc.tDecouplingNuclei = "1H"
```

The coil must be connected to the table in order for the system to support multinuclear operation; in other words, if the coil only operates at ^1H , the system will always be at the ^1H frequency even if the sequence supports other nuclei.

Prior to performing an X-nucleus measurement, it is necessary to set the transmitter frequency and power, analogous to the adjustments necessary for ^1H operation. These adjustments are made using the X-Frequency and X-Transmitter tabs under the Options > Adjustments dropdown. These tabs will only be available if the connected coil supports multinuclear operation. The X-Frequency adjustment is used to set the frequency synthesizer to the X-nucleus frequency. An fid sequence is executed, with a starting frequency based on the current ^1H frequency translated down to the X-nucleus range by the ratio of gyromagnetic ratios. The X-Transmitter performs no measurement; it is only setting the Reference Amplitude and is the output voltage from the transmitter adequate for a 180 degree rotation of a 1 ms rectangular rf pulse. This is used to scale all rf pulses and in the SAR calculation. For double resonant coils, it is recommended to perform the ^1H adjustments first as they will provide a

reference for the X-nucleus adjustments.

SEQUENCE REQUIREMENTS

A sequence can be made to support multinuclear operation by adding a few features:

Nucleus specification

The operator must specify the nucleus of operation for the sequence (¹H is default). This can be achieved by activating the Sequence Nucleus tab using:

```
rSeqLim.setSupportedNuclei(NUCLEI_ALL.get().c_str());
```

The choice will then be limited to the nucleus defined for the coil. To limit the available options to a specific nucleus, the appropriate nucleus string can be provided: NUCLEI_13C, NUCLEI_31P, etc.

Larmor constant retrieval

The scanner supports two transmit paths, allowing for heteronuclear decoupling or Nuclear Overhauser Enhancement (NOE). The second transmit path is currently limited to ¹H. In MrProt, the TXSPEC section has an array of length two that contains the nucleus information (identity, base frequency, reference amplitude, Delta frequency). It is necessary to incorporate the Larmor constant associated with the particular nucleus in all calculations for rf and frequency/phase calculations. Two steps are necessary to perform this:

1. Retrieve the Larmor constant for the selected nucleus. This is normally performed at the beginning of *prepare*:

```
MeasNucleus mainNucleus(rMrProt.getsTXSPEC() .
getasNucleusInfo()[0].gettNucleus().c_str());
```

If a second nucleus is used in the measurement, then the Larmor constant for the second nucleus must be retrieved as well :

```
MeasNucleus txNucleus(rMrProt.getsTXSPEC() .
getasNucleusInfo()[1].gettNucleus().c_str());
```

2. The Larmor constant must be transferred into all rf and frequency/phase objects that are used based on the X-nucleus:

```
MyRf.setNucleus(mainNucleus);
MyFreqSeq.setNucleus(mainNucleus);
MyFreqNeg.setNucleus(mainNucleus);
```

```
MyAdc.setNucleus(mainNucleus);
```

SEQUENCE FEATURES

There are two sequences that support X-nucleus operation. A brief description of their features is below:

FID sequence

FID. The product FID sequence includes several features that are active only when used for X-nucleus scanning.

The excitation pulse is a non-frequency selective pulse. For X-nucleus scanning, three options are available:

- Rectangular: This pulse is derived from RF_PULSE_RECT with a default duration of 0.5 ms;*
- Adiabatic Half-passage: This pulse is an external pulse stored in the rf pulse library %MEASDATA%\extrf_spec.pls, and has a default duration of 2.5 ms;*
- Adiabatic BIR-4: This pulse is calculated in the sequence and has a default duration of 5 ms.*

The pulse duration is stored in the TD array.

A second feature is an option for Nuclear Overhauser Enhancement (NOE). A series of rectangular pulses at the ^1H frequency is applied prior to the primary excitation pulse to saturate the ^1H spins. X-nucleus spins coupled to these saturated spins will undergo a rebalancing of the spin populations, resulting in a larger net magnetization. The parameters for NOE are:

```
rSeqLim.setNOEs      (1, 10, 1, 1);
rSeqLim.setNOEType   (SEQ::NOE_NONE, SEQ::NOE_RECTANGULAR);
rSeqLim.setNOEDuration (1000, 90000, 1000, 5000);
rSeqLim.setNOEDelay   (1000, 100000, 1000, 10000);
rSeqLim.setNOEFlipAngle(0, 180, 1, 90);

rMrProt.getsSpecPara().getlNOEs();
rMrProt.getsSpecPara().getlNOEType();
rMrProt.getsSpecPara().getlNOEDuration();
rMrProt.getsSpecPara().getlNOEDelay();
rMrProt.getsSpecPara().getdNOEFlipAngle();
```

A third feature is ^1H Decoupling. The product sequence supports Continuous Wave and WALTZ-4 decoupling. The decoupling rf pulses are stored in an RF_PULSE_ARB object and executed immediately prior to the ADC sampling. The parameters for Decoupling are:

```
rSeqLim.setDecouplingType(SEQ::DECOUPLING_NONE,
                         SEQ::DECOUPLING_WALTZ4, SEQ::DECOUPLING_CW);
rSeqLim.setDecouplingFlipAngle(0, 360, 1, 180);
rSeqLim.setDecouplingDuration(500, 2000, 500, 1000);
rSeqLim.setDecouplingTotalDuration(10, 100, 10, 50);
rSeqLim.setDecouplingPause( 0, 200, 10, 100 );
```

Csi_fid Sequence

CSI_FID. The product CSI_FID sequence uses a slice selective excitation pulse external pulse stored in the rf pulse library %MEASDATA%\extrf_spec.pls, followed by up to three phase encoding tables in different directions for spatial localization. For ^1H scanning, the WET water suppression is available.

For X-nucleus scanning, proton decoupling as well as NOE are available using the same schemes as in the fid sequence.

OTHER MISCELLANEOUS ISSUES

Unit Test

The Unit Test performs basic tests for RF power and measurement times. Tests of Frequency/Phase and Gradient objects are not performed on the current software version. Checks for gradient amplitudes, RF center frequencies must be done manually using the Event Block dump or with print statements.

E.7 pTX

Introduction

To make full use of the pTX option, the system software was extended to support the following features:

- **Adjustments:** Support for B1/B0 mapping for pulse calculation
- **RF safety:** Extended local SAR supervision for pTX with support for VOP (Virtual observation points) based supervision.
- **Sequence:** A new RF pulse class was added for pTX pulses (`sRF_PULSE_PTX`). A new sequence interface method was added to control pTX pulse calculation (`calculatePTX`) in addition to `initialize`, `prep`, `check`, `run`.
- **Pulse calculation:** A MATLAB based pulse calculation interface is provided to calculate pTX pulses

For IDEA developers, the modifications in the sequence programming interface and the newly provided pulse calculation interface are relevant.

As mentioned above, the `calculatePTX` method was added to the sequence interface. This modifies the sequence behavior:

Host:

- `initialize()` is called when a protocol is opened
- `prepare()` is called during binary search while protocol is edited
- `calculatePTX()` is called after scan/apply was pressed and calculates pTX pulses
- `prepare()` is called to finally prepare the sequence including SAR look-ahead

MaRS:

- The protocol is prepared and run on the MARS computer

Only after `calculatePTX()` the actual pTX RF pulses are available. `calculatePTX()` is called only once on the host computer.

pTX pulse sequence programming

Aim: To extend existing pulse sequences to use computed multi-channel pulses on the scanner.

Two cases have to be distinguished: RF shimming and dynamic pTX.

RF shimming

In general, all sequences can perform global RF shimming using the “B1 shimming” parameter in the Adjustment tabcard in the protocol. No sequence changes are necessary. RF shim settings are not applied to dynamic (arbitrary) pTx pulses.

If it is desired to change the RF shim settings pulse by pulse or, e.g., slice by slice, the following SeqExpo parameter need to be set:

```
rSeqExpo.setbTxScaleFactorsFromSeq(true)
```

to enable the

```
bool setTxScaleFactor(const unsigned long ulTxChannel, const
Complex& scaleFactor)
```

method of all RF pulse objects. This method allows to set an arbitrary RF shim for each RF pulse object.

Dynamic pTX

Three main aspects were changed for pTx programming: A new RF pulse class (**sRF_PULSE_PTX**), calculatepTx() to trigger the pTx calculation and several flags within the sequence to control the pTx pulse calculation. These will be explained in the following:

```
class sRF_PULSE_PTX
```

This class is a container for one arbitrary RF pulse for each transmit channel and arbitrary gradient pulses for each gradient axis. The content of these sub-objects is automatically transferred from the pTx pulse calculation framework base on a calculation ID that will be obtained during the calculatePtx phase of the sequence. It includes the use of prep(), run(), check() and getRFInfo() which supports global and local SAR calculations via VOPs

The calculation ID can be set using the following method:

```
void sRF_PULSE_PTX::setCalculationID(int32_t iCalculationID);
```

Otherwise the RF pulse object can be used as all other RF pulse objects using prep(), run(), check() and, e.g., getRFInfo(). The getRFInfo() method was extended to include local SAR information based on VOPs. Thus, the pTX SAR look-ahead needs to be handles like in any other sequence.

During a normal prepare() phase, the object does not contain the final RF pulse. Thus, at least a preliminary pulse duration needs to be defined to be able to calculate the sequence timing during binary search. Only during the final prepare on the scanner, the actual RF pulse is available including all relevant timing, power and SAR parameters. Thus, it is recommended to supply a worst-case pulse duration during normal prepare and calculate the final timing during the final prepare on the scanner (e.g. when the calculationID() of the pulse object is non-zero).

MrProt

In general, sequences shall not enable pTX for the default protocol to guarantee a working default

protocol also if the system does not have the pTX option. When pTX is enabled in the protocol (see pTX UI card below), the sequence needs to set the pTX flag in the protocol and request B1/B0 maps if required:

```
rMrProt.getTxSpec().getasNucleusInfo()[0].setbHasPTXPulses(true);
rMrProt.getAdjData().setuiAdjB1AcqMode(SEQ::AdjB1Acq value);

SEQ::ADJB1ACQ_STANDARD_SINGLE    Acquire B1 maps of individual channels
SEQ::ADJB1ACQ_NONE              Do not acquire B1 maps

rMrProt.getAdjData().setuiAdjB0AcqMode(SEQ::AdjB0Acq value);

SEQ::ADJB0ACQ_STANDARD          Acquire B0 map
SEQ::ADJB0ACQ_NONE              Do not acquire B0 map
```

To get the number of transmit channels of the connected coil, the following code can be used:

```
#include "MrServers/MrProtSrv/MrProt/CoilSelect/MrTxCoilSelect.h"
#include "MrServers/MrProtSrv/MrProt/MrCoilInfo.h"

MrTxCoilSelect(rMrProt.coilInfo().Meas().getaTxCoilSelectData()[0]).getNum
OfUsedTxChan();
```

Note: Do **not** use the number of transmit channels available on the system since this might not match the number of used transmit channels for the given RF coil!

calculatePTX()

The calculatePTX() method was added to the sequence interface to support inline RF pulse calculation for pTX:

```
calculatePTX(MrProt &rMrProt, const SeqLim & rSeqLim);
```

Within this method, the pTX pulse calculation plug-in and configuration file is selected, parameters are transferred to the pTX pulse calculation plug-in (e.g. from MrProt), the RF pulse(s) are calculated and attached to the sRF_PULSE_PTX objects.

To calculate a pTX pulse, run:

```
getRTController().calculatePTX(int32_t& riCalculationID,
                                PhysParameters& rOutput,
                                const std::string& rsPlugin,
                                const MrProt& rProt,
                                const PhysParameters& rInput);
```

The default for the plugin string is "PhysPtxPulseDesign@PhysPtxPulseDesign". If custom plugins are used, the string needs to be adapted to "PhysPtxPulseDesign@MyPluginDlName".

The plug-in configuration file ("designFile") is specified as part of the rInput object. Using the varargin parameter, additional parameters to extend/modify the configuration file can be added.

```

rInput["designFile"][-] = "PPDconfig_dynpulse_spokes";
PhysVector vVarargin;
vVarargin.push_back("Dpa.rf_zt = 5.0"); // slice thickness in mm
vVarargin.push_back("myParameter = 1");
rInput["varargin"][-] = vVarargin;

```

The configuration files are stored in C:\MedCom\MriProduct\PhysConfig\

The calculation ID can then be passed to the corresponding pulse object:

```
sRF_PULSE_PTX::setCalculationID(int32_t iCalculationID);
```

Note: The RF pulses returned by calculatePTX() do not contain frequency shifts for off-centre slice position. This has to be done by the sequence.

Note: The RF pulses returned by calculatePTX() can be retrieved as .ini files both on the scanner (C:\MedCom\temp\Phys\[...]) and in IDEA (follow "path_output" in user_path.m; typically 'C:\Matlab\PtxPulseDesign.local\data_output').

Using offline RF pulses	To use arbitrary RF pulses a design file that is configured to read external arbitrary pulses needs to be supplied to the pulse calculation (see above). An example is given in \n4\pkg\MrServers\MrPhysicsSrv_src\PtxConfig\ep2d\PPDconfig_ep2d_ini.m. There, the Dpa.main_task is set to 5 (process external pTX pulse). Furthermore, the pulse file name needs to specified (Dpa.import_ExtPulse). This file or a similar one can be used in any sequence to support externally calculated pTx pulses. For details on .ini files see section "pTX pulse interface to the scanner".
--------------------------------	---

Note: The used .ini file can also be selected by the sequence by providing the Dpa.import_ExtPulse parameter from the sequence (see calculatePTX() section).

MATLAB pTX pulse calculation

General requirements	To build and run the MATLAB pTX pulse calculation framework, the following MATLAB products are required:
-----------------------------	--

- MATLAB r2012b (32bit)
- Optimization Toolbox
- Image Processing Toolbox
- MATLAB Compiler

Ensure that MATLAB r2012b comes before any other version in the system PATH (so that typing "MATLAB" in a command window should open MATLAB r2012b).

In general, the system allows to use MATLAB figures and UIs to control the pulse calculation online.

While the UI is open, the sequence will stay in “preparing...” state until all UIs/figures are closed and the pulse calculation is finished. However, due to IT security reasons, there is no Java Virtual machine available on the scanner by default. To enable this feature, the MATLAB runtime environment, which is part of IDEA, needs to be reinstalled on the scanner and the physlifecycle.dll that is also part of idea needs to be replaced on the scanner.

Be aware that this modifies the system software. Use at your own risk!

Adjustments and SAR data

To calculate pTX pulses in simulation, input parameter files containing B0/B1 maps and SAR information are needed. There are two types of input parameters:

- **Phys_UID????_Input.xml:** These files are used as input for the so-called PhysServer, responsible for pulse design. The content is converted to mxArray format and passed to the main MATLAB routine PPD_main.m. The used file is specified using the physshell utility. An example file is provided in /n4/pkg/MrServers/MrPhysicsSrv/PtxData/Phys_UID-108_Input.xml (8 Tx channels). The file can also be obtained from the scanner after a sequence using pTX pulse(s) was run. It has to be present at any time, even if it is not used, as it is used by the physserver for initialization.
- **???DataUser.mat:** These files (AdjDataUser.mat, SarDataUser.mat and SysDataUser.mat) can be read by the MATLAB routine. To use these files, they have to be specified in the configFile.m in parameters Dpa.ADJ_file, Dpa.SYS_file and Dpa.SAR_file. They can also be found on the scanner after pTX adjustments, in C:\MedCom\MriProduct\PhysConfig).

Using and modifying the framework

Below, the pTX pulse calculation framework implemented in MATLAB is described, as well as how to use it in the IDEA simulation environment and on the scanner. **Three use cases can be distinguished:**

1. Using the existing framework as is;
2. Adding functionalities to the framework;
3. Modifying existing functionalities of the framework (not recommended).

Using standard pulse designs

This is the most simple use case, that requires the modification of only one file (in simulation or on the scanner): the config file, here referred to as myConfigFile.m. However, the steps described below are also needed for the more advanced use cases considered later.

A number of pulse designs and optimisation techniques are readily available and can be used by controlling the framework through myConfigFile.m. Examples of such files, calling various pulse

designs, can be found under `\n4\pkg\MrServers\MrPhysicsSrv_src\PtxConfig\`.

A list of most parameters, with the different available options and the default ones, is available here: `\n4\pkg\MrServers\MrPhysicsSrv_src\PtxPulseDesign\PPDconfig_base.m`.

Simulation using MATLAB:

To use a config file within the MATLAB environment, it must be placed in the **pulse design root directory**: `\n4\pkg\MrServers\MrPhysicsSrv_src\PtxPulseDesign`. To start the pulse calculation, open MATLAB and run `PPD_main('myDesignFile')`. (Note the absence of .m file extension.)

The MATLAB routine requires `???DataUser.mat` files to be present, containing B0/B1 maps and SAR information (see “Files and Folders” below). Those files must also be placed in the pulse design root directory, and be referred to in the `designFile.m` in parameters `Dpa.ADJ_file`, `Dpa.SYS_file` and `Dpa.SAR_file`.

Simulation within the IDEA environment:

IDEA and the scanner environment do not directly use the .m files as present in the pulse design root directory. Instead it relies on a compiled .dll version of the framework, called *calculation plugin*. More details on plugins will be provided later; for now, we consider that only the default calculation plugin is installed, as it comes with the IDEA distribution.

To control the plugin with a design file, the latter must be placed in the following directory: `\n4\x86\extsw\MedCom\MriProduct\PhysConfig`. If `???DataUser.mat` are used as input, they must also be placed there.

- In the IDEA shell:

Input data can be provided either with `Phys_UID????_Input.xml` or with a set of `???DataUser.mat`. To prepare the calculation, run `physstart` to start the calculation server. To control the calculation server, the utility `physshell` is used. There, a calculation plug-in can be selected (option 2). To select the input parameter file, go to “6. Input”, “8. Edit rerun calc id” and type in the calculation ID of the config file (e.g. -108 for the example). To select `myDesignFile.m`, use “6. Input”, “5. Edit design file name”. Finally, the calculation can be started with “14. Calculate PTX”. Debug output is written to the logviewer.

- In POET: simulation of a pTx sequence:

In general, a pTx sequence can be simulated as any sequence. To be able to also simulate the pTx pulse calculation, the following restrictions apply:

- The PhysServer must be running (`physstart / physstop`).
- The sequence must refer to the correct calculation plugin (here the default: “`PhysPtxPulseDesign@PhysPtxPulseDesign`”) and to your design file

- (Input["designFile"]["-"] = " myDesignFile"); See "calculatePTX()" section.
- To use adjustment data retrieved from a scanner, the designFile.m must specify the name of the ???DataUser.mat.
 - For initialization of thePhysServer the demo B0/B1 maps (Phys_UID???_Input.xml), which are part of IDEA, need to be loaded. This has to be defined within the sequence according to the demo FLASH sequence:

```
#ifdef WIN32
    if( getenv("MIDEA_HOME") != NULL )
    {
        rInput["CtrlData"]["RerunCalcID"] = -(100 + 8); // 8 = numTx
    }
#endif // WIN32
```

Note: If ???DataUser.mat files are specified in myConfigFile.m, these will overwrite the data in Phys_UID???_Input.xml (be it in IDEA shell or POET).

Pulse design on the scanner:

The file myConfigFile.m must be placed in C:\MedCom\MriProduct\PhysConfig.

As calibration files ???DataUser.mat are generated on the fly by the scanner, they should not be specified in the configuration file; instead the fields Dpa.ADJ_file, Dpa.SYS_file and Dpa.SAR_file should be left blank (e.g. Dpa.ADJ_file = '').

Adding functionalities to the framework

In case the user wants to add their own methods to extend the MATLAB pulse calculation framework, a number of hooks are available to replace parts of the provided pTX pulse calculation algorithms. This allows to use user-provided gradient trajectories and pulse optimization algorithms to calculate an arbitrary pTX RF pulse and associated gradients.

Pulse design steps:

The pulse calculation is done in several steps that can be individually controlled or replaced using the configuration options in myConfigFile.m. Note that any new custom parameters can be specified in the "Dpa" structure in the design file, to be retrieved later in the code.

Using Dpa.main_task, the main operation mode selected. The available operation modes are:

- PPD environment-, installation- and unit-test (1)
- RF shimming (2)
- show dynamic gradient-/RF-design only (3)

- Arbitrary RF pulse calculation (4)
- External RF pulse processing (5)
- User defined RF pulse calculation (6)

The next paragraph is related to option (4) and partly (6) only and describes the workflow of the pulse calculation process:

- The reformatted B0/B1 maps are processed
- *Create target magnetization pattern*
- *Create basic pulse structure and gradient waveform*
- *Small flip-angle optimization*
- *High flip-angle optimization based on small flip-angle result (optional)*
- Evaluate SAR and component protection
- Bloch simulation (optional)
- Return final RF pulse(s) to the sequence

All steps in italic can be replaced by a user-defined hook.

Available user hooks:

- user_path(): extends the PATH variable in MATLAB and specifies directory for logging output
- user_GRFdesign(): specifies the basic pulse structure (i.e. sinc shape for spokes) and the gradient waveform.
- user_SDdesign(): specifies spokes positions in k-space
- user_magnetization(): specifies target magnetization/flip-angle pattern
- user_LFoptimisation(): performs RF pulse optimization for small flip-angles
- user_HFoptimisation(): performs RF pulse optimization for large flip-angles
- user_dynamic_PPD(): replaces the full optimization chain except input data handling, Bloch simulation and final SAR / component protection evaluation.

Examples for all hooks are available here:

\n4\pkg\MrServers\MrPhysicsSrv_src\PtxPulseDesign\library_user\.

Compiling a new pulse calculation plugin:

All the additions made by the user in the library_user directory can be directly used within the

MATLAB environment, so long a correct myConfigFile.m is provided when calling PPD_main. However, **to use them in IDEA or on the scanner**, a new pulse calculation plugin (.dll) must be compiled; let us call it MyPtxPulseDesign.dll.

To do so, open a command window *with elevated rights* and navigate to the pulse design root directory (\n4\pkg\MrServers\MrPhysicsSrv_src\PtxPulseDesign), then run:
"compile.bat MyPtxPulseDesign".

The resulting compiled binary can be found as: \n4\x86\prod\bin\MyPtxPulseDesign.dll. This step can be performed from any command window, it is not necessary to do this within an IDEA shell.

Note: by default, only .m files directly in library_user are compiled. If you want to place some files (e.g. helper functions) in subdirectories within library_user, all these subdirectories must be explicitly declared in compile.bat.

Simulation within the IDEA environment and use on the scanner:

The steps described in the first use case (*Using standard pulse designs*) regarding the addition/editing of a configuration file (myPulseDesign.m) must also be completed here.

In addition, the compiled plugin must be placed in the following directory:

- in IDEA: \n4\x86\delivery\bin;
- on the scanner: C:\MedCom\MriCustomer\lce.

On sequence side, in calculatePTX(), the plugin string should now be specified as: "PhysPtxPulseDesign@MyPtxPulseDesign". Note that only the part after the "@" is affected, i.e. the reference to the library file (.dll).

If one wants to switch between functionalities of the default plugin and the new one, it is of course possible to select different design files, each for a specific application. The custom plugin contains all the functionalities of the default one, in addition to the user-defined extensions. It is therefore compatible with design files defined to control the default plugin, so it is not necessary to call different plugins from the sequence for different applications: specifying different design files is enough.

Note: The plugin file should not be renamed after compilation. Either adapt the plugin string called by the sequence, or recompile the framework with a different name.

Important note: Do not use "PtxPulseDesign.dll" as a name for your custom plugin, especially on the scanner, as it is the name of the default plugin.

Modifying functionalities of the framework (not recommended)

A third use case would arise when the supplied user hooks are not enough for some applications, and the user wants to amend default functionalities of the pulse calculation framework. Note that such situation should occur in rare occasions, and the user is advised to perform those modifications with

great care as the whole framework could be compromised. In general, make sure that the new plugin is still compatible with predating config files and still supports functionalities of the default plugin.

This case is similar to what has been described before, and changes performed on any file of the framework are readily testable within the MATLAB environment. **However, any changes on .m files other than those in library_user would not be taken into account by the compilation.** Indeed, compile.bat builds the default components of the framework not from .m files, but from the associated protected .p files. The resulting custom plugin would be no different from that described above, and the modifications to the framework would not be visible in either IDEA or the scanner.

A necessary step is to first convert MATLAB .m files into .p files, using the "makePcode_standalone.m" script. Then calling compile.bat as described before will compile the modified.p files including your own modifications.

Files and Folders

File	Folder (IDEA)	Folder (System)
myConfigFile.m	\n4\x86\extsw\MedCom\MriProduct\PhysConfig	C:\MedCom\MriProduct\PhysConfig
AdjDataUser.mat SarDataUser.mat SysDataUser.mat	\n4\x86\extsw\MedCom\MriProduct\PhysConfig	C:\MedCom\MriProduct\PhysConfig
RFPulse.ini <i>as input</i>	\n4\x86\extsw\MedCom\MriProduct\PhysConfig	C:\MedCom\MriProduct\PhysConfig
RFPulse.ini <i>output from pulse design</i>	Follow "path_output" in user_path.m; typically 'C:\Matlab\PtxPulseDesign.local\data_output'	C:\MedCom\temp\Phys\[...]
MyPtxPulseDesign.dll	\n4\x86\delivery\bin	C:\MedCom\bin

Hints for the demo sequence FLASH

The FLASH demo sequence now also supports pTx. The FLASH_ptx sequence is considered deprecated.

- ❑ Within POET, you can switch from a standard non-pTX pulse to a pTX pulse on the tab card Sequence – pTX by adding a pTX pulse of type excitation.

- To make the INI Pulse known on the scanner, it has to be copied into the

%CustomerSeq%\RFPulses folder:

C:\MedCom\MrCustomer\seq\RFPulses\pTXRFPulse[iID].ini

SeqLim parameters for pTX sequences

The following options are available on the PTX RFPulse UI card
(\MR_Infra_01\comp\ProtBasic\public\Interfaces\SeqLim.h):

```
// index, min, max, inc, def
rSeqLim.setPTXPulses ( 1, 1, 1, 1);
rSeqLim.setPTXPulseAcceleration ( 0, 1.0, 1.0, 0.1, 1.0);

rSeqLim.setPTXPulseType ( MrProtocolData::PTXPulseType_Excitation );
rSeqLim.setPTXTrajectoryType ( MrProtocolData::PTXTrajectoryType_ExternalIni
);
rSeqLim.setPTXB0CorrectionType ( MrProtocolData::PTXB0CorrectionType_Off );
// index, min, max, inc, def
rSeqLim.setPTXPulseFlipAngle ( 0, 0, 180, 1.0, 90);

// index, min, max, inc, def
rSeqLim.setPTXPulseReadFOE ( 0, 1.0, 1.0, 0.1, 1.0);
rSeqLim.setPTXPulsePhaseFOE ( 0, 1.0, 1.0, 0.1, 1.0);
rSeqLim.setPTXPulseSliceFOE ( 0, 1.0, 1.0, 0.1, 1.0);
rSeqLim.setPTXPulseReadMatrix( 0, 1.0, 1.0, 0.1, 1.0);
rSeqLim.setPTXPulsePhaseMatrix( 0, 1.0, 1.0, 0.1, 1.0);
rSeqLim.setPTXPulseSliceMatrix( 0, 1.0, 1.0, 0.1, 1.0);
```

Example simulation with active PhysService

In order to simulate a pTX sequence including the calculations, the PhysService has to be started first.

- Switch to sequence directory

[vE12U-CD-d] Z:\n4\pkg> cs FLASH

- Switch to pTX system (select a pTX system in given list)

[vE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\FLASH_ptx> sys

- Start PhysService (before simulation)

[vE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\FLASH_ptx> physstart

- Simulate pTX sequence (and use PhysService)

[vE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\FLASH_ptx> sim

- Stop PhysService (after simulation)

[vE12U-CD-d] Z:\n4\pkg\MrServers\MrImaging\seq\FLASH_ptx> physstop

Note that the simulation visualizer has only one icon for the main RF channel; in order to display the additional channels the respective files need to be loaded via Diagrams->Additional->New. For the second RF channel you would have to enter "RF2" in the field for the "New File Identifier" and correspondingly for higher channel numbers. Confirm the two open dialog boxes with "OK"; the additional RF channel should be displayed in the visualizer.

pTX pulse interface to the scanner

To support a correct INI-format, the following MatLab routines are provided to read/write pTX pulses in the INI format:

```
save_pTXRFPulse_toINI.m  
get_pTXRFPulse_fromINI.m
```

Both files can be found here: \n4\pkg\MrServers\MrPhysicsSrv_src\PtxPulseDesign\library_main\in_out\

Example pulses (pTXRFPulse0*.ini) are provided within the FLASH_ptx sequence. Please check the Matlab code for more details and/or try to read the provided pTXRFPulse0.ini via get_pTXRFPulse_fromINI as an example.

Dynamic pTX rf pulse description (INI format)

A regular slice selective (SS) excitation consists of an RF pulse (defined by an amplitude / phase array) and a gradient in slice-selection direction (defined by the GSS gradient value and a rephasing gradient moment). In a generalized manner a pTX excitation pulse consists of an amplitude / phase array for each channel (typically 2-ch, 8-ch, 16-ch) and a shaped gradient, i.e. one gradient array, in general up to three axes. The type of pTX RF pulse is 1D using only one gradient axis, 2D excitation using two gradient axes or 3D pulses, respectively.

It is obvious that a pTX RF pulse now contains more extended information, thus we defined a new format in which all relevant info of the pTX RF pulse is stored in a single file. An example is given in FLASH_ptx\pTXRFPulse0.ini. The pTX pulse can be created using the matlab routines mentioned above. It is feasible to use several different pulses (*with different RFPulseID*) within a sequence, e.g. pTXRFPulse0 / pTXRFPulse21 / pTXRFPulse22.

pTX RF Pulse Format

The INI File Format is commonly known, see for example:

http://en.wikipedia.org/wiki/INI_file

Some special features were implemented in addition to define arrays.

The INI file consists of multiple sections starting with a keyword in square brackets. Inside a section,

blocks there are two types of entries:

scalars comprising of keyword = value

arrays comprising of keyword[index] = value1 value2 ... , index = 0,1,2..

Anything past the hash sign # is treated as a comment.

The following blocks are required in the PTX RF Pulse Format:

```
[pTXPulse], [Gradient], [pTXPulse_ch0] ... [pTXPulse_ch(N-1)],
```

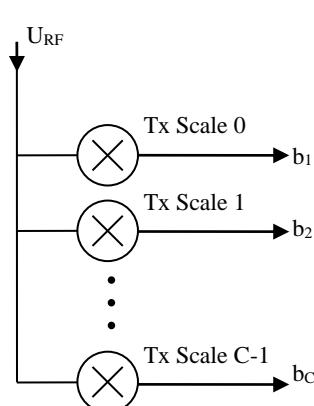
where N is the number of transmit channels used. The first section contains general parameters which are self-explanatory. The other sections contain the gradient and RF arrays. For the RF pulses, the wave form is given as amplitude of the transmitter in [V], and phase from 0 to 2π ; for the gradient wave forms, as Gx, Gy, Gz in mT/m. Note that the number of RF samples must be a multiple of the number of gradient samples (with a factor of 1, 2, 3, 4 or 5); with a standard gradient rastertime of 10us, the RF dwell time can then be 10, 5, 3.33, 2.5 or 2 us.

To make the INI Pulse known on the scanner, it has to be copied into the following %CustomerSeq% folder:

```
C:\MedCom\MriCustomer\seq\RFPulses\pTXRFPulse[iID].ini
```

Apply B1 shimming parameter

In the manual adjustments menu the standard Tx-Scale parameters (e.g. circular or elliptical) can be manually replaced per TX channel by any desired complex value (magnitude and phase), as calculated offline:



Data interface from the scanner

In order to externally calculate dynamic pTX pulses or B1 shimming parameter, usually you need some scanner data (like B0-maps, B1-maps, SAR-data or HW limitations). The following data are exported by the scanner to support your offline pTX calculations. Every time the pulse design is called at the scanner (from any pTX sequence, e.g. the service sequence "dyn_pulse" by selecting "QA2" as excitation pulse at the "special tabcard", or any sequence with B1 shimming enabled¹), the following three Matlab data files are generated:

- Adjustment data:** mainly B1- and B0 maps of slices as defined in the corresponding sequence-protocol:

C:\MedCom\MriProduct\PhysConfig\AdjDataUser.mat

- SAR Data:** VOP data to provide SAR information:

C:\MedCom\MriProduct\PhysConfig\SarDataUser.mat

- System parameter:** basic system data and component protection data:

C:\MedCom\MriProduct\PhysConfig\SysDataUser.mat

To make use of those data files as input for the pulse-design-suite, set the following PPDconfig-Files variables:

```
Dpa.ADJ_file = 'AdjDataUser.mat';
Dpa.SYS_file = 'SysDataUser.mat';
Dpa.SAR_file = 'SarDataUser.mat';
```

Adjustment data

Variable	Description	Type	Unit
image_m	number of rows of the 2D-images (height)	int	pixels
image_n	number of columns of the 2D-images (width)	int	pixels
coils	[optional] number of independent channels (design channels)	int	-
S	2D-array of the sensitivity-profiles (or B1-maps), each of the columns holds a one-dimensional representation (row-by-row) of the B1-image of a channel in the logical coordinate system (LCS). Extends to a 3D-array for multi-slice-data that have to be sorted along the anatomical desired slice index as defined in the UI.	(image_m* image_n) xcoils [xslices], complex	IT/V

¹ Note that in the latter case there is no B0-map available

values_m	image pixel positions (see definition of the LCS) along the image column ² , including lateral shift	1ximage_m, double	mm
values_n	image pixel positions (see definition of the LCS) along the image row ³ , including lateral shift	1ximage_n, double	mm
W	validity-mask-image, corresponding to the sensitivity-images S. Extends to a 3D-array for multi-slice-data.	image_m ximage_n [xslices], logical	-
B0	[optional] 2D-B0map-image, corresponding to the sensitivity-images S, holding the main magnetic field deviations from the main magnetic field strength Bmain. Extends to a 3D-array for multi-slice-data.	image_m ximage_n [xslices], double	Hz
deltaTE	[optional] echo time difference while B0 aquistion	double	ms
slices	[optional] number of images (to be optimized commonly ⁴)	int	-
values_s	[multi-slice only] slice positions (see definition of the LCS), corresponding to the sensitivity-images S	1xslices, double	mm
image_ori	common main image orientation: 0/1/2 = tra/sag/cor	int	-
image_psd	common PE-scanning-direction: 0/1 = standard/swapped	int	-
rois	[optional] number of ROI objects	int	-
R	[optional] ROI-mask-image, corresponding to the sensitivity-images S. Extends to a 3D-array for multi-slice-data. Multiple ROIs are supported and will be merged within the pulse design	image_m ximage_n [xslices] [xrois], logical	-
T	[optional] magnetization target image, corresponding to the sensitivity-images S (will be normalized to alpha). For multi-slices this optionally extends to a 3D-array	image_m ximage_n [xslices], complex	-

SAR data

Variable	Description	Type	Unit
----------	-------------	------	------

² Pixel height then is `(values_m(1)-values_m(end))/(image_m-1)`

³ Pixel width then is `(values_n(end)-values_n(1))/(image_n-1)`

⁴ a single pulsdesign call generally computes one common pulse for all stated slices

ZZ	3D array that contains the averaged (over the 10g environment) E field sensitivity products multiplied with the conductivity values for the preselected VOPs; to be used for rapid local 10g SED calculation for the full model (by choosing the maximum value for all VOPs)	coilsxcoilsxvopnum, complex	-
ZZtype	interpretation of the ZZ-matrices (type), e.g. absolute SED-values [Ws/kg] or relative allowed advantage factor with respect to a CP/EP base excitation	vopnumx1, int	-
GlobalSARAbsValue	absolute global SAR value	double	-
GlobalSARLimit	global SAR limit	double	-
VOPTypeLimit	VOP type limits for...	limitnumx1, double	-
VOPTypeLimitID	...the VOP-types as in Sar.ZZtype	limitnumx1, int	-
indexfield	[optional] Continuous numbering of all voxels representing the body tissue (in a cuboid, all other voxels are zero)	voxels_xxvoxels_yxvoxels_z, int	-
clusterfield	[optional] Assigns each voxel represented by the indexfield-number to a hotspot-cluster	nonempty_voxelsx1, int	[1..hotspotnum]

System parameter

Variable	Description	Type	Unit
bmain	main magnetic field strength	double	T
gmax	maximum gradient strength	double	mT/m
smax	maximum slew rate	double	mT/m/ms
g_step	sampling time of the gradient system	double	ms

rf_step	minimum RF sampling time	double	ms
umax	maximum applicable RF-voltage (depends on the adjustment)	double	V
adjtra	adjust transmitter voltage (rect pulse, 1ms, 180°)	double	V
txScaleFactor	RF-coefficients as passed from the original adjustment (CP or EP)	coilsx1, complex	-

All other Sys-data are related to the internal HW peak component protection.

pTX local SAR supervision

The pTX local SAR supervision is based on Virtual observation points (VOPs). To support this approach, a VOP file is associated with all pTX coil files. The sequence framework transparently supports VOPs. No additional steps are needed to support pTX SAR supervision / look-ahead in sequences.

Creating VOP files

To create VOP files, a MATLAB script is provided with IDEA in

```
\n4\pkg\MrPatientSafetySrv\RFSWD\RFSWD_Utils\VOPCompression\matlab\IDEAExamples\createVOPFile.m
```

See documentation in that file for details.

MR Coordinate Systems

This section describes the definitions and conversion of MR coordinate systems for Numaris/4.

Coordinate systems in the measurement system

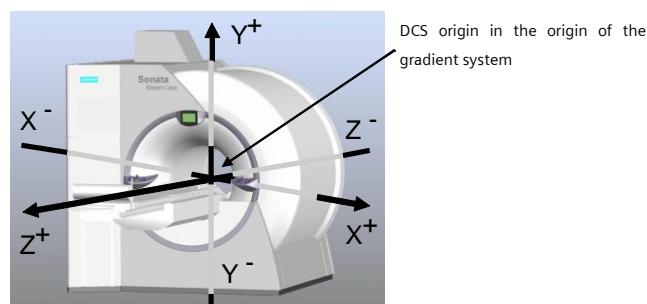
Note: all coordinate systems are right-handed

- Device Coordinate System
- Patient Coordinate System
- Gradient Coordinate System a.k.a PRS Coordinate System
- Table Coordinate Systems
 - TableBoard Coordinate System
 - Table Coordinate System
- Series Block Coordinate System
- Shim Coordinate System

Device Coordinate System (DCS)

Standard systems:

- Positions concerning the gradient system of the scanner are **not** identical to the Gradient Coordinate System (GCS).
- Orientation of the X-, Y- and Z-axis correlate to XYZ-gradients. They are identical in respect to the axis of the B_0 field.
- The DCS origin matches the origin of the gradient system.

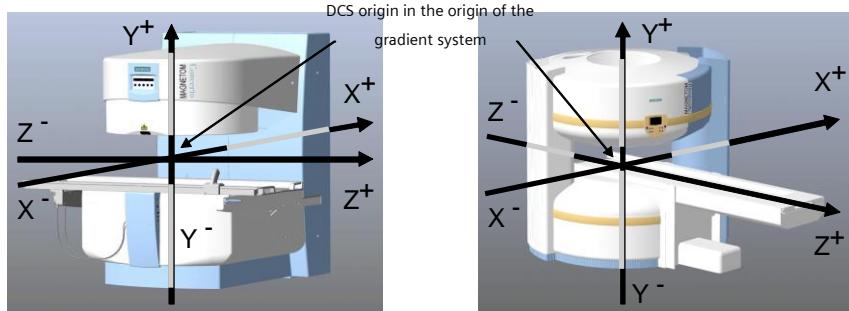


Open systems:

- Positions concerning the gradient system of the scanner are **not** identical to the Gradient Coordinate System (GCS).
- Orientation of X, Y and Z axis correlate to XYZ-gradients. Permutates respective to the axes of

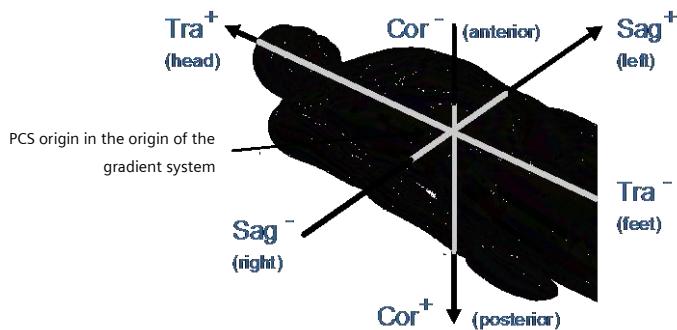
the B_0 field.

- The DCS origin matches the origin of the gradient system.



Patient Coordinate System (PCS)

- The Positions in respect to the patient are independent of the patient's bedding.
- Axes: Sagital (Sag), Coronar (Cor), Transversal (Tra)
- PCS origin matches the origin of the DCS.



- Conversion PCS \leftrightarrow DCS on the basis of the patient positioning
 - head first / supine ($X_{DCS} = \text{Sag}$, $Y_{DCS} = -\text{Cor}$, $Z_{DCS} = -\text{Tra}$)
 - head first / prone ($X_{DCS} = -\text{Sag}$, $Y_{DCS} = \text{Cor}$, $Z_{DCS} = -\text{Tra}$)
 - head first / right lateral ($X_{DCS} = \text{Cor}$, $Y_{DCS} = \text{Sag}$, $Z_{DCS} = -\text{Tra}$)
 - head first / left lateral ($X_{DCS} = -\text{Cor}$, $Y_{DCS} = -\text{Sag}$, $Z_{DCS} = -\text{Tra}$)
 - feet first / supine ($X_{DCS} = -\text{Sag}$, $Y_{DCS} = -\text{Cor}$, $Z_{DCS} = \text{Tra}$)
 - feet first / prone ($X_{DCS} = \text{Sag}$, $Y_{DCS} = \text{Cor}$, $Z_{DCS} = \text{Tra}$)
 - feet first / right lateral ($X_{DCS} = -\text{Cor}$, $Y_{DCS} = \text{Sag}$, $Z_{DCS} = \text{Tra}$)
 - feet first / left lateral ($X_{DCS} = \text{Cor}$, $Y_{DCS} = -\text{Sag}$, $Z_{DCS} = \text{Tra}$)

- “left” instead of “first” for some open systems (e.g. Concerto)

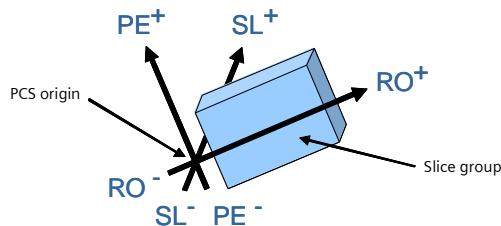
The conversion PCS \Leftarrow DCS is implemented in `MeasPatient`:

```
MeasPatient::transformPCSToDCS()
```

Gradient Coordinate System (GCS)

a.k.a. PRS Coordinate System

- Position respective to the PRS gradients of one slice (-group)
 - PRS-grades arise due to super position of the XYZ-grad.
 - Position of the slice(group) is not considered.
- Axes: Phase-Encoding (PE), readout (RO), Slice (SL)
 - Slice-Axis is also called “Slice Normal Vector”.
- GCS origin matches the origin of the PCS.



- Conversion GCS \Leftarrow PCS on the basis of PRS directions in PCS coordinates (from `libGSI`)

$$\begin{pmatrix} Sag \\ Cor \\ Tra \end{pmatrix} = (PE \quad RO \quad SL) \bullet \begin{pmatrix} P_{Sag} & P_{Cor} & P_{Tra} \\ R_{Sag} & R_{Cor} & R_{Tra} \\ S_{Sag} & S_{Cor} & S_{Tra} \end{pmatrix}$$

The conversion GCS \Leftarrow PCS is implemented in `libGSI: transformGCSToPCS()`

TableBoard Coordinate System (TBCS)

- The position respective to the table board is independent from the movement of the table board.
- The direction of the X-, Y- and Z-axis is analogue to the DCS.
- TBCS origin is defined respective to the table board
 - X = 0 in the middle of the table board
 - Y = 0 at the contact surface of the table board (without coils)
 - Z = 0 at the outer edge of the table board
(at the head side with patient bedding head first)

- Valid coil positions on the table board
 - In X direction: positive and negative
 - In Y direction: *always* positive
 - In Z direction: *always* positive

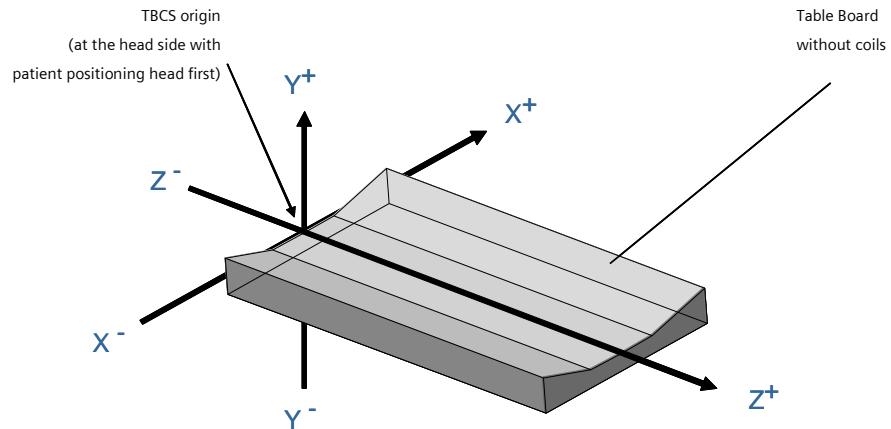


Table Coordinate System (TCS)

- The position in respect to the table depends on the offset of the table board.
- The direction of the X-, Y- and Z-axis is analogue to the DCS.
- TCS origin equates to the origin of the TBCS in home position of the table board.
- Conversion between TBCS and TCS on the basis of the table board offset in TCS coordinates (from `PTABProxy`):
 - $X_{TCS} = X_{TBCS} + X\text{-Offset}_{TCS}$
 - $Y_{TCS} = Y_{TBCS} + Y\text{-Offset}_{TCS}$
 - $Z_{TCS} = Z_{TBCS} + Z\text{-Offset}_{TCS}$

The conversion between TBCS and TCS is implemented in `PTABProxy`:

`PTABProxy::transformTBCSToTCS()`



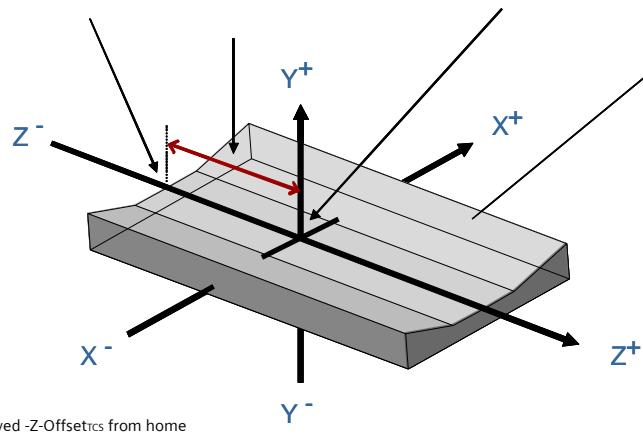
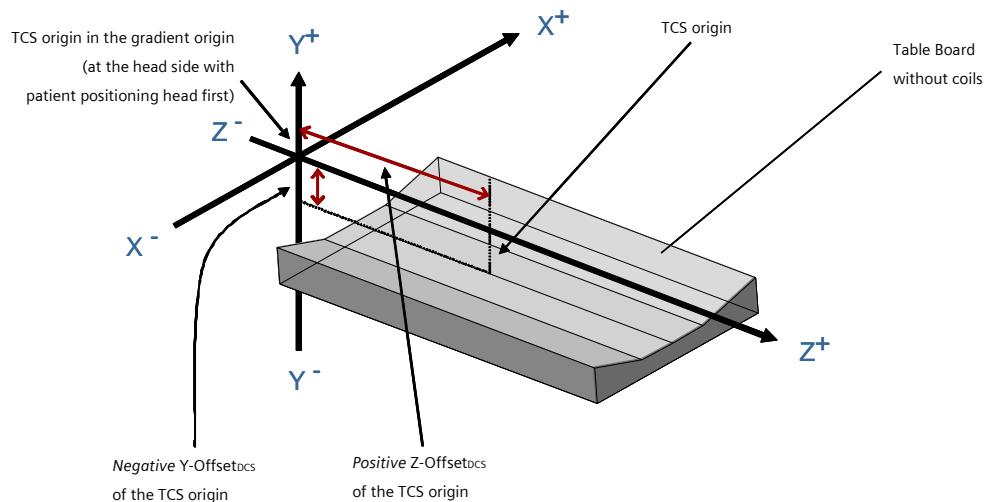


Table moved -Z-Offset_{TCS} from home position into the magnet.

- The TCS origin has a fixed offset to the origin of the DCS.
- Conversion TCS \leftrightarrow DCS on the basis of the fixed offset in DCS coordinates (from MeasPerm)
 - $X_{DCS} = X_{TCS} + X\text{-Offset}_{DCS}$
 - $Y_{DCS} = Y_{TCS} + Y\text{-Offset}_{DCS}$
 - $Z_{DCS} = Z_{TCS} + Z\text{-Offset}_{DCS}$

The conversion TCS \leftrightarrow DCS is implemented in PTABProxy:

```
PTABProxy::transformTCSToDCS ()
```



- The offset of the TCS origin from the origin of the DCS in MeasPerm:

- alPTABZeroDistToMagnetCenter[0..2]
- Maintained for X-, Y- and Z-directions.
- The TCS origin is **not** identical with the light-beam localizer.
- The offset of the light-beam localizer from the origin of the DCS in MeasPerm:
 - lPTABDistanceLMToMagCentX..Z
 - Currently maintained only for Z-direction.

Series Block Coordinate System (SBCS)

- The SBCS models *relative* and *no absolute* scan positions.
- Scan positions respective to the initial patient position within a continuous image series (series block).
 - Axes: Sagital (Sag), Coronar (Cor), Transversal (Tra)
 - Direction of the Sag-, Cor- and Tra- axis is analog to the PCS.
 - Only the scan position in Tra-direction can be adjusted in the protocol.
(e.g. "Scan Position = H 200", 200mm in head direction)
- The SBCS origin equates the position of the table board in TCS coordinates *during the Prepare of the first measurement*.
 - Before the first measurement there is no valid SBCS origin.
 - SBCS origin is evaluated at the beginning (!) of the Prepare.
 - The TBCS origin defines the position of the table board.
- The SBCS origin will become invalid by following events:
 - Registration of a patient
 - Positioning of a patient via the light-beam localizer
 - Movement of the table board into home position
 - Manual movement of the table board (only on systems with tables without motor)
 - Alternation between "Scanner ready" ⇔ "Scanner not ready"
- The next measurement defines a valid SBCS origin
 - At the same time a new series block begins
 - Incompatible images between different series blocks
- Calculation of a position of the table board in TCS coordinates for a scan position in SBCS coordinates

- Goal: Positioning of the slice (group) in DCS origin by a contrarian table movement
- Only possible with a valid SBCS origin
- Conversion of the SBCS origin from TCS → DCS → PCS
- $\text{TableBoard}_{\text{Sag}} = \text{Origin}_{\text{Sag}} - \text{Slice}_{\text{Sag}}$
- $\text{TableBoard}_{\text{Cor}} = \text{Origin}_{\text{Cor}} - \text{Slice}_{\text{Cor}}$
- $\text{TableBoard}_{\text{Tra}} = \text{Origin}_{\text{Tra}} - \text{Slice}_{\text{Tra}}$
- Conversion of the Position from PCS → DCS → TCS

The calculation is implemented in `MrSeriesBlockManager`:

```
MrSeriesBlockManager::TransformSBCS2TablePos()
```

- Calculation of a coil position in PCS coordinates for a scan position in SBCS coordinates.
 - Wanted: Coil position during a planned measurement.
 - Only possible with a valid SBCS origin.
 - Conversion of the coil position from TBCS → TCS *with the position of the table board in the SBCS origin.*
 - Conversion of the coil position from TCS → DCS → PCS
 - $\text{Coil}_{\text{Sag}} = \text{Coil}_{\text{Sag}} - \text{Slice}_{\text{Sag}}$
 - $\text{Coil}_{\text{Cor}} = \text{Coil}_{\text{Cor}} - \text{Slice}_{\text{Cor}}$
 - $\text{Coil}_{\text{Tra}} = \text{Coil}_{\text{Tra}} - \text{Slice}_{\text{Tra}}$

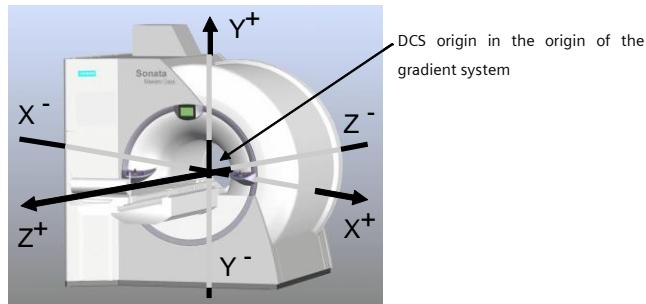
The calculation is implemented in `MrSeriesBlockManager`:

```
MrSeriesBlockManager::TransformTBCS2PCS()
```

Shim Coordinate System (ShimCS)

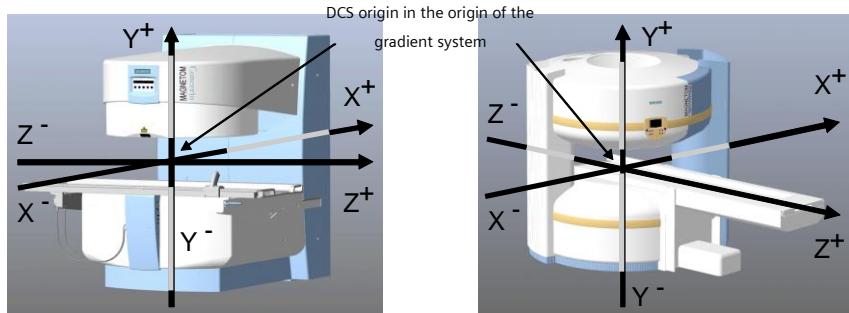
Standard systems:

- Positions in respect to the gradient coils/shim coils are **not** identical to the Gradient Coordinate System (GCS).
- Orientation of the X-, Y- and Z-axis is analogue to the DCS. It is identical to the axes of the B_0 field.
- The ShimCS origin matches the origin of the DCS.



Open systems:

- Positions in respect to the gradient coils/shim coils are **not** identical to the Gradient Coordinate System (GCS).
- Orientation of X-, Y- and Z-axis permutes in respect to the DCS. It is identical to the axes of the B_0 field.
- The ShimCS origin matches the origin of the DCS.



- Conversion between ShimCS and DCS based on the system type (from `SHIMBaseProxy`)
 - $X_{DCS} = (\text{System Type} == \text{Standard}) ? X_{ShimC} : Y_{ShimC}$
 - $Y_{DCS} = (\text{System Type} == \text{Standard}) ? Y_{ShimC} : Z_{ShimC}$
 - $Z_{DCS} = (\text{System Type} == \text{Standard}) ? Z_{ShimC} : X_{ShimC}$
- There is no modular conversion between ShimCS and DCS implemented.

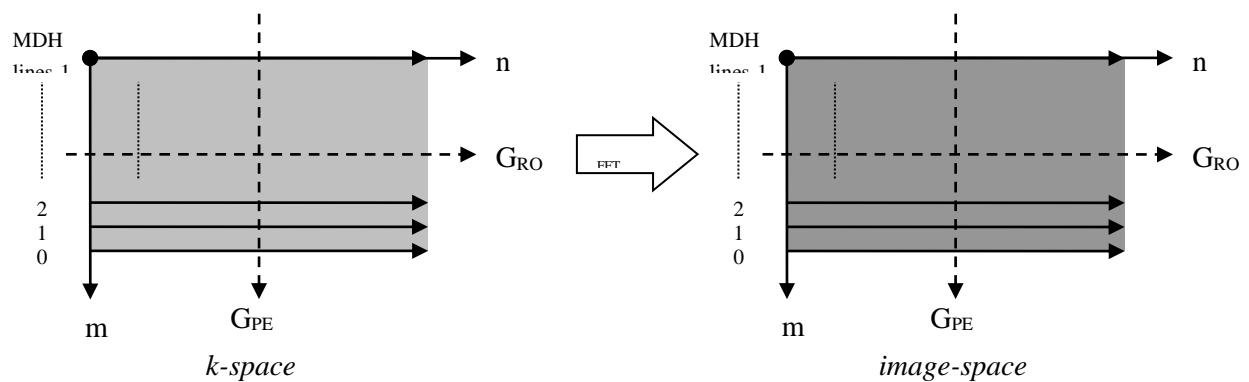
Pulse design coordinate system (LCS)

The pTx-pulse is designed in the gradient based logical-coordinate-system (LCS)⁵ consisting of read-out direction (RO), phase-encoding direction (PE) and slice-selection direction (SS). In the Matlab

⁵ also known as gradient-coordinate-system (GCS) or pe-ro-ss-coordinate-system (PRS)

code, (RO,PE,SS) is usually renamed to (x,y,z).

All 2D-input images for the pulse design need to be delivered in the 2D LCS. They are generated by FT from the scanners raw MEAS-data holding the data-lines as recorded by the ADC. Please follow the MDH-information to arrange the k-space with the correct line sorting, the MDH_reflect parameter etc. Then the 2D-image data have to be organized, following the Matlab mxn-matrix notation (m rows, n columns), like:

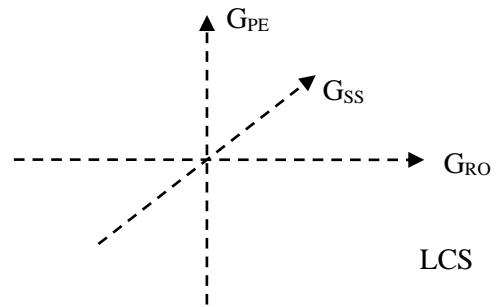


The used Matlab plot-routine *imagesc()* draws the images on the screen as they appear here, with origin upper left and first PE-scan-line 0 at bottom. From this mxn representation of the image-space, 2D-images can also be organized in a 1-dimensional array of size $(m \times n) \times 1$, whose elements are taken column-wise from the 2D-image.

The 3D logical-coordinate-system (LCS) is defined in the following way (each slice of a multi-slice design is treated at its position $G_{ss}=0$). Please note the direction of the SS-coordinate:

$$\circ \quad \vec{G}_{PE} \times \vec{G}_{RO} = \vec{G}_{SS}$$

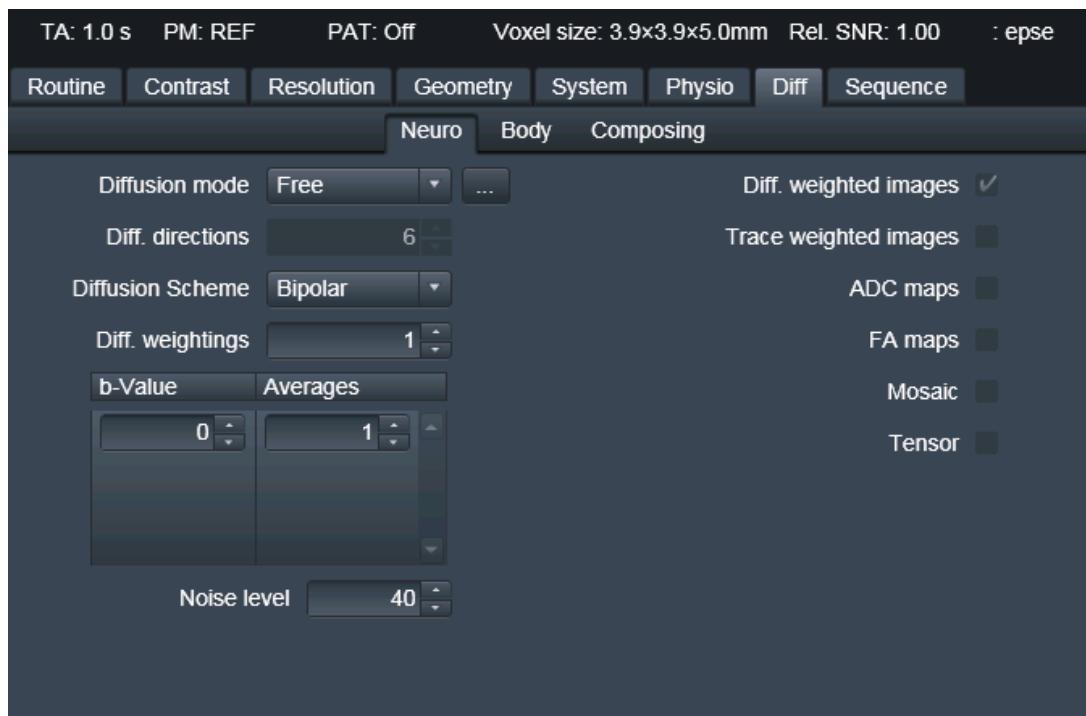
Slice-shifting is performed within the sequence framework.



E.8 User Defined Diffusion Vector Sets

Starting with syngo MR D13D, the DTI (Diffusion Tensor Imaging) license enables the import and export of user defined diffusion vector sets (DVS) in selected product diffusion sequences (e.g. ep2d_diff, resolve). As a prerequisite, a text file (extension: dvs) containing the DVS definitions has to be available in the appropriate directory (C:\Medcom\MriCustomer\seq\DiffusionVectorSets) on the scanner. Multiple files containing different DVS are supported.

Once this directory exists, the UI parameter *Diffusion Mode* on the *Diffusion* card offers an additional entry named *Free*. This *Free* mode supports the same data acquisition and processing options as the *MDDW* mode – for a reasonable tensor estimation, at least six non-colinear diffusion vectors are required. If selected, one button appears right next to the Diffusion mode selection that allows for importing and exporting of user defined DVS.



Please note that images acquired with a user defined DVS will not have the "SIEMENS sequence" mark.

Import

When pressing the import button, a file open dialogue appears that allows a DVS file to be selected. If the file syntax is correct (for details, please see below), all DVS in the file get imported by the sequence and can be selected by choosing the corresponding number of diffusion directions in the UI. Otherwise, an error message pops up providing hints on how to correct the detected syntax errors.

Export

When pressing the export button, a file save dialogue appears that allows a DVS file to be specified. The DVS currently selected in the protocol gets exported using the same DVS file format as described below. Note that exporting SIEMENS internal DVS is not possible.

Protocol

When selecting a user-defined DVS, all related information becomes part of the measurement protocol. Thus, it is possible to rerun the same measurement even if the original DVS file is no longer present. It is even possible to use the "Phoenix"-feature to run the protocol on another scanner without the need to copy the corresponding DVS file. However, please note that the DTI license will still be required.

DVS File Format

It is possible to specify several DVS definitions with different number of directions in each text file. Lines starting with a hash mark (#) are treated as comments and will be ignored. Blank lines will be skipped, too. All commands are not case-sensitive.

The instruction [directions=<number>] starts the definition of a new section, i.e. another DVS which contains <number> directions. Note that this directive must not contain any blanks or tabs. An arbitrary number of DVS can be specified, however each number of directions must be unique within each DVS file. The maximum number of directions supported by one DVS is 1024.

The coordinate system used for playing out the DVS is controlled by the directive CoordinateSystem=<coord>. Two modes are supported:

xyz

The DVS is played out in the physical gradient coordinate system. Diffusion vector directions do not depend on the actual slice orientation. This is the default if no coordinate system directive is specified.

prs

The DVS is played out using the rotation matrix of the current slice, i.e. the phase-read-slice coordinate system is used. Diffusion vector directions are thus linked to the actual slice orientation.

It is possible to specify a comment using the directive Comment=<text>. The description given in <text> will get displayed as a tooltip in the UI.

The directive Vector[<index>]=(<x>,<y>,<z>) defines a diffusion direction vector. <index> specifies the vector index (counting starts at 0), while <x>, <y> and <z> specify the three spatial coordinates. After normalization, it is required that no vector component exceeds the value range [-1.0 ... +1.0].

If the DVS is specified in prs coordinates, it is additionally required that the magnitude of each vector stays within the value range [0.0 ... +1.0].

The normalization algorithm is controlled by the directive `Normalisation=<norm>`. Three different modes are supported:

unity

Each vector gets normalized to unity. This mode will ensure that every direction receives a weighting with the same b-value. This is the default if no normalization directive is specified.

maximum

First a unity normalization gets applied. Afterwards, the whole vector set is scaled such that the largest vector component of the whole set is 1.0. This mode will make best use of the gradient performance and yield the shortest echo times. However, it should be used in the xyz coordinate system only.

none

*No normalisation is applied at all, the vectors are used as provided. This mode is quite useful for implementing e.g. q-space acquisition schemes with a uniform sampling density on different shells. For the internal diffusion gradient calculation, the vector with the largest magnitude is taken as the reference: its amplitude will produce the b-value specified in the UI. The actual b-value corresponding to a user defined diffusion vector can be calculated according to the following equation: $b_{actual} = b_{UI} * \text{Magnitude}^2_{actual} / \text{Magnitude}^2_{maximum}$.*

Example

A valid (though not necessarily meaningful) DVS definition file may look like this:

```
# -----
#           Copyright (C) SIEMENS AG 2011  All Rights Reserved.
# -----
#
# Project: NUMARIS/4
#   File: c:\Medcom\MriCustomer\seq\DiffusionVectorSets\MyVectorSet.dvs
#   Date: Thu Nov 11 11:11:11 2011
#
# Descrip: External vector file for SBBDiffusion
# -----


[directions=6]
CoordinateSystem = xyz
Normalisation = maximum
Comment = my diffusion vector set
Vector[0] = ( 1.0, 1.0, 0.0 )
Vector[1] = ( 2.0, 1.0, 0.0 )
Vector[2] = ( 1.0, -2.0, 0.0 )
Vector[3] = ( 2.0, 1.0, -1.0 )
Vector[4] = ( -2.0, 0.0, 1.0 )
Vector[5] = ( 0.0, -1.0, 2.0 )
```

This files defines a single DVS consisting of six non-colinear vectors given in physical gradient coordinates. It includes a user comment and a `maximum` normalization directive. The latter triggers an internal rescaling of all vectors such that the largest component equals 1.0:

```

Vector[0] = ( 0.790569, 0.790569, 0.000000 )
Vector[1] = ( 1.000000, 0.500000, 0.000000 )
Vector[2] = ( 0.500000, -1.000000, 0.000000 )
...

```

The magnitude of all vectors is thus scaled to 1.118034.

If a *unity* normalization directive would have been used instead, the internal rescaling would yield vectors with a magnitude of 1.0:

```

Vector[0] = ( 0.707107, 0.707107, 0.000000 )
Vector[1] = ( 0.894427, 0.447214, 0.000000 )
Vector[2] = ( 0.447214, -0.894427, 0.000000 )
...

```

Without normalization (mode *none*) no rescaling would take place. Since several components of the example vector set would exceed the value range [-1.0 ... +1.0], a syntax error would pop up when trying to import this DVS.

DICOM Attributes

The following DICOM attributes related to diffusion imaging are stored in a private vendor section of diffusion images:

```

0019;XX0C; SIEMENS MR HEADER; B_value; 1; IS; 1
0019;XX0D; SIEMENS MR HEADER; DiffusionDirectionality; 1; CS; 1
0019;XX0E; SIEMENS MR HEADER; DiffusionGradientDirection; 1; FS; 3
0019;XX27; SIEMENS MR HEADER; B_matrix; 1; FD; 6

```

The most important attribute for later evaluation is the attribute *0019;XX27* which holds the information about the b-matrix used during the acquisition of the diffusion dataset. The six floating point values contain the upper triangle of the b-matrix. Note that the contribution of imaging gradients is considered.

B_value

IS 1

Scalar b-value (s/mm²). Note that this attribute is not filled for images based on multiple diffusion weightings (e.g. apparent diffusion coefficient maps).

DiffusionDirectionality

CS 1

Specifies whether the frame contains directional (e.g. diffusion weighted images) or isotropic information (e.g. trace weighted images).

DiffusionGradientDirection

FD 3

Holds the three values of the effective diffusion encoding direction including the actual polarity for the frame (patient coordinates). The contribution of imaging gradients is considered. Note that this attribute is filled only for images containing directional information.

B_matrix

FD 6

Contains the six independent b-matrix elements (bxx, bxy, bxz, byy, byz, bzz), characterizing the diffusion encoding for this frame (patient coordinates). The data is stored with a precision of 1s/mm². Note that this attribute is filled for diffusion weighted images only.

Special attention is required when accessing the data in private vendor specific sections. An offset which needs to be considered is defined at the beginning of each private DICOM section. This is illustrated in the following example.

Suppose that a given image contains the entry 0019,0010 LO |SIEMENS MR HEADER|. This defines an offset of 10 for all other SIEMENS private attributes of section 0019. Using this offset as the high byte allows accessing all other private SIEMENS attributes of this section, for example:

```
(0019,100c) IS |1000|          # 4, ;B_value
(0019,100d) CS |DIRECTIONAL|    # 12, ;DiffusionDirectionality
(0019,100e) FD |0.7064\0.707812\0.00141138| # 24, ;DiffusionGradientDirection
(0019,1027) FD |500\501\1\502\1\0|      # 48, ;B_Matrix
```

Since the offset is dynamically allocated whenever the image gets modified (e.g. by a PACS system) it has to be looked up explicitly.

For additional information about the DICOM format, please refer to the "DICOM Standard, Part 5, Chapter 7.8: Private Data Elements" (<http://medical.nema.org/dicom>).

IDEA MANUAL

Copyright © Siemens Healthcare GmbH 2018-2019. All rights reserved.



Manufacturer's note:

This device bears a CE mark in accordance with the provisions of Council Directive 93/42/EEC of June 14, 1993 concerning medical devices and the Council Directive 2011/65/EU of June 08, 2011 on the restriction of the use of certain hazardous substances in electrical and electronic equipment.

The CE marking applies only to medical devices which have been put on the market according to the above-mentioned EC Directives. Unauthorized changes to this product are not covered by the CE mark and the related Declaration of Conformity.

Legal Manufacturer
Siemens Healthcare GmbH
Henkestr. 127
91052 Erlangen
Germany

Siemens Healthineers Headquarters
Siemens Healthcare GmbH
Henkestr. 127
91052 Erlangen, Germany
Phone: +49 9131 84-0
siemens-healthineers.com