METK - The Medical Exploration Toolkit

Konrad Mühler and Christian Tietjen

Magdeburg, June 17, 2009

Contents

Acknowledgement

First, we want to thank all people, that helped us to develop that huge bunch of modules:

- All the MeVis people, that provide us enormous support in the forum, via phone and personally, especially Florian Link, Felix Ritter, Olaf Konrad, Milo Hindenach, Wolf Spindler, Tobias Boskamp, Staphan Zidowitz and many more.

- Our students, expecially Mathias Neugebauer, Rocco Gasteiger, Roland Pfisterer and Stefan Hiller for using and testing the METK.
- Our colleagues Alexandra Bear, Ragnar Bade, Steffen Oeltze, Jeanette Cordes, Arno Krüger and Jana Dornheim for their suicidal undertaking of using the METK first
- Our boss Bernhard Preim for giving us free rein to do what we think fit to do and for letting us write this documentation in the inspiring surrounding of home sweet home.

1 Introduction

In the following we will describe concept and realization of the MedicalExplorationToolkit—the METK. The METK was designed for loading, visualizing and exploring segmented medical data sets. It is a framework of several modules in MeVisLab, a development environment for medical image processing and visualization.

MeVisLab provides a lot of basic modules to load and process medical images as well as visualization modules to show segmented objects in 3d. Whole applications can be created based upon these modules connecting them to networks. Using only basics modules, these networks come up complex very fast. MeVisLab provides so called macro modules, to bundle subnetworks. But for many tasks like loading a case or visualize some segmented objects, the user has to create his own macro modules. Hence, we created the METK as a framework with about 50 macro modules. Defining a common data and communication standard, all METK modules can be reused in different applications and new inventions are immediately usable for all users and developers.

The METK helps to develop basic applications very fast. It provides basic facilities like case management as well as advanced visualization techniques like stippling. The main tasks, the METK helps to solve, are:

1. **Case Management: Load and save whole cases of segmented structures**
   e.g. for surgery planning, educational training or intra operative visualization
2. **Basic Visualization in 2D and 3D: Visualize segmented structures in multiple manner**
   e.g. iso surface rendering, stippling, hatching, silhouettes, volume rendering, 2d overlays
3. **Advanced Visualization: Provide advanced medical visualization techniques**
   e.g. ghost views, cutaways, safety margins, labeling techniques, convolution surfaces
4. **Animation: Interactive animations as well as pre-rendered videos**
   Provide interactive animations of 3d scenes and facilities to render reusable videos.

5. **User Interface: Widgets than can be used in medical applications**
   e.g. tree views, automatic structure lists
6. **Synchronization: Changes made in one part of an application should be communicate to all other parts**
   Changing the visualization of a structure in 3D should have also effects in 2D. Or if an object is selected in 3D, the 2D viewer should jump to the corresponding slice.
7. **State saving: The user ought to save different states of the visualization for later use.**
   e.g. as a collection

In the following sections some basics of the METK and all METK modules are described.

2 Application Scenarios

In the following, a few application scenarios are given to show the potential of the METK. You may combine each of this scenarios, because all METK modules use the same database, and will always show synchronized visualizations. At the end of each section, all modules are listed you need to reproduce the functionality. The `METKManager` is always necessary and not stated explicitly.

2.1 Case Building / Segmentation

Well, the first thing you have to do, if you want to use the METK—and you have no existing METK case or any convertible case file (see Section 3.2)—is to setting up a new one. For setting up a new case, or expanding an existing one, there are several modules to support you.

The METK does NOT support particular segmentation algorithms, like LiveWire, Watershed, Mass Spring Models, or whatever. It just helps you to store the retrieved segmentation information in a METK readable manner.

Example: if you have to prepare a clinical dataset and to segment several structures, you first load the original dataset. The dataset has to be declared with `METKAddImage`. Then you have to clamp the image data to the region of interest (ROI) by using `METKAddROI`. As a last step, you can segment all the structures you need with your desired MeVisLabML-modules, and save the segmentation information by using `METKAddStructure`.

Additionally, general information about the patient and the findings may be stored by using `METKPatientObject` and `METKCaseObject`.

Needed modules: `METKCaseObject`, `METKPatientObject`, `METKAddImage`, `METKAddROI`, `METKAddStructure`

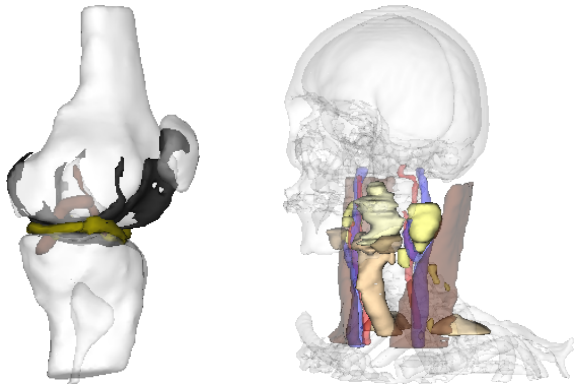
Example network: `METKCaseBuilding.mlab, PatientAndCaseObject.mlab`

2.2 Conventional 3D Rendering

Rendering a case with isosurfaces is one of the simplest things you can do with the METK. The isosurfaces are automatically generated when a new case is loaded. This will take some time, but the surfaces are stored as Open Inventor files and will be reused on a second load.

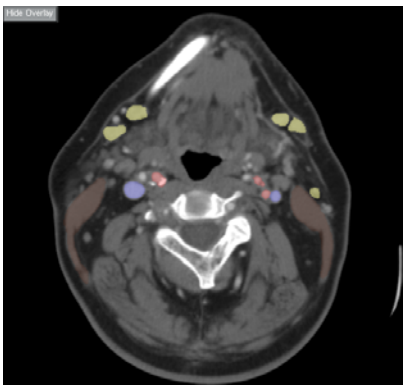
Needed modules: `METKIsoSurface` and `METKViewer3D`

Example network: `METKRenderModes.mlab`



2.3 Conventional 2D Image Display

Displaying the segmentation results in 2D is a bit more complicated, but this is just because to keep the underlying memory management as fast as possible. The desired ROI may be loaded with `METKROISelect` and displayed with `METKViewer2D`. If you want to display the segmentations as a colored overlay, you first have to generate a combined segmentation mask, containing all segmentation results for the given ROI. This is done by using `METKCodedSegmentation`. The generation takes some time, but the result will be stored at your hard drive and will make the future image display much faster, because only one extra image has to be loaded (instead of one image per segmentation). The result of the combined segmentation mask may be displayed with `METKOverlay2D`.

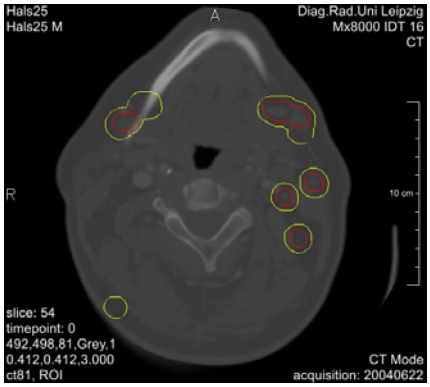


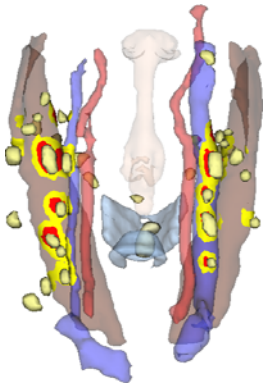
Needed modules: METKCodedSegmentation, METKOverlay2D, METKROISelect, METKViewer2D

Example network: METKImageDisplay.mlab

2.4 Displaying Critical Distances

Displaying critical distances to pathologic structures seems to be one of the most common tasks in medical visualization. The display in 2D and 3D is supported. In both cases, you need to compute a distance transformation, with the pathologic structures as seed points. This can be done by using METKDistanceTransform. In 2D, METKSurfaceDistance2D displays the isolines of the distance transformation. In 3D, the distances may be mapped on user specified structures by using METKSurfaceDistance3D.



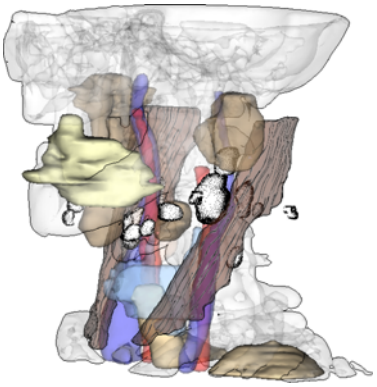


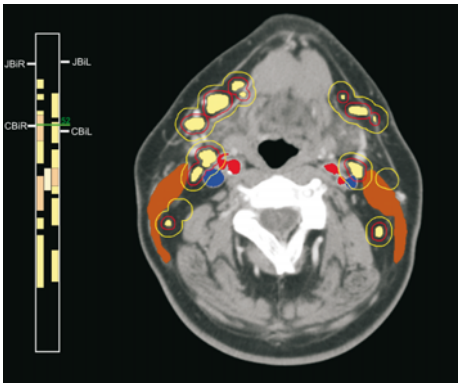
Needed modules: METKDistanceTransform, METKSurfaceDistance3D, METKSurfaceDistance2D, standard 2D image display and 3D rendering modules

Example network: METKSurfaceDistance.mlab

2.5 Illustrative Visualization

Unless our working group adorns oneself with *Illustrative Medical Visualization*, some modules for illustrative, non-photorealistic rendering are obligatory. Thus, we offer modules for silhouette and feature line rendering, stippling and hatching. In 2D, a technique we call *LiftChart* is provided.





Needed modules (2D): METKLiftChart, standard 2D image display modules

Example network (2D): METKLiftChart.mlab

Needed modules (3D): METKSilhouette, METKStippling, METKTexturing, standard 3D rendering modules

Example network (3D): METKRenderModes.mlab

2.6 Interactive Exploration

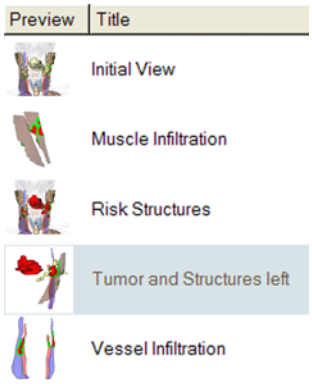
For the interactive exploration of the case data, it is possible to pick a certain structure in 2D and/or 3D. The currently selected object is delivered by METKCSO. To hide or show structures or groups of structures, METKStructureGroupBrowser may be used. METK is also able to compute good viewpoints to every single structure in a case and to generate adjustable animations for introducing a case or presenting at a surgeons tumor board.

Needed modules: METKCSO, METKStructureGroupBrowser, Konrad's Animations, die Kamerapositionierungsgeschichten

Example network: METKBrowser.mlab, PickingAndCSO.mlab, Konrad's Animations, die Kamerapositionierungsgeschichten

2.7 Storing Interesting Views

Sometimes it is a bit hard to find a good view for a specific task, e.g., to show an interesting view between two small, hidden structures. For that purpose, METKCollections are provided. This module enables you to store the current viewing position and all display parameters in a *Collection*. The collection is stored with a description, a thumbnail and a screenshot and may be loaded afterwards.



A slim version of METKCollections is the METKBookmark3D module. This module just stores the current viewing position. The transitions between two stored states may made animated with both modules. In that case, the module UMDAnimation2 is necessary.

Needed modules: METKCollections, METKBookmark3D

Example network: METKCollections.mlab, METKBookmark3D.mlab

2.8 Debugging

If you want to set up a new case, or to import a given case with some of the converters, it would be interesting if every data entry has its right place or to clean up the entries. Therefore, the METK browsers might be helpful, especially the METKHierarchyBrowser and METKExplorer.

If you are developing an own METK application, METKDebug helps you to find out, whether there are unnecessary ObjMgr notificications. In the advanced developing process, METKLogFile dumps the console output in a file. Thus, the end user may send you the file when errors occurred.

Needed modules (2D): METKDebug, METKLogFile, METKHierarchyBrowser, METKExplorer

Example network (2D): LogFileAndDebug.mlab, METKBrowser.mlab

2.9 Already Constructed Applications

The following sections presents four applications, which are based on the METK.

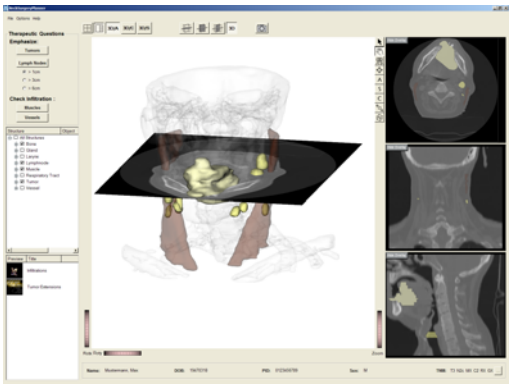
2.9.1 NeckSurgeryPlanner

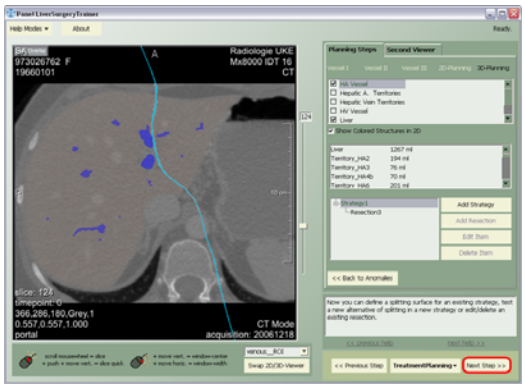
The NeckSurgeryPlanner is a Software-Assistant for Pre-operative Planning and Visualization of Neck Dissections. Neck dissections are carried out for patients with malignant tumors in the head and neck region. The extent of the intervention depends on the occurrence and location of malignant lesions. To ensure the decision about operability and the surgical procedure of the neck dissection, the occurrence and number of enlarged (and probably malignant) lymph nodes, as well as their location in relation to risk structures have to be assessed and visualized.

Up to now neck dissections are often planned on the basis of axial slices of CT or MRI data. However, the detection of enlarged lymph nodes in 2D data is difficult for surgeons. Thus, it is possible that affected lymph nodes are not recognized and eventually a surgery has to be canceled because of previous misinterpretation of the patient's operability. To support the preoperative planning of neck dissections, 3D visualizations are intended to explore pathologic structures. For this purpose, we developed the NeckSurgeryPlanner (NSP) to support the surgeon's decision.

Needed modules: nearly all.

Example network: `NeckSurgeryPlanner.mlab`



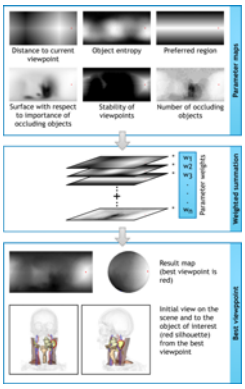


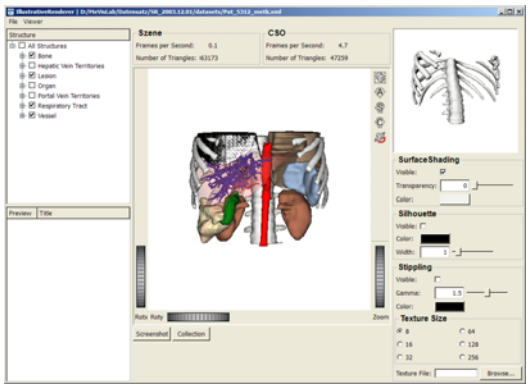
2.9.2 LiverSurgeryTrainer

...

2.9.3 VisibilityDemonstrator

....





2.9.4 IllustrativeRenderer

Lorem ipsum dolor sit amet, consectetur adipisici elit, sed eiusmod tempor incidunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquid ex ea commodi consequat. Quis aute iure reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint obcaecat cupiditat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

3 METK Basics

3.1 Data and Communication

Both, the data and the communication of the METK are controlled by the `METKManager`. The core of the `METKManager` is an ObjMgr. The ObjMgr is a concept of MeVisLab to hold data in three staged hierarchy (objectID, layerID, infoID). These data can be read and written by other modules, connected to the ObjMgr. Changes of single values are communicated to all connected modules. Each module can listen to several entries and so react on changes. For more information read the documentation of the ObjMgr.

To get the data and to communicate with other METK modules each METK module needs to be connected to the `METKManager`. It can be connected directly or via other METK modules' METK output, which leads to a better network design. The `METKManager` loads a case and stores all information about the segmented structures, case information and communication values in its ObjMgr. All METK modules can access and manipulate these data. So, if one module change the color of a structure, are other METK modules can react on this event and can where required adjust their own settings.

Some data, like the Communication object are only for communication. Those values are not stored in the cases and are only hold to send events and messages to other METK modules.

The core of each METK module to connect itself with the METKManager (better: the internal ObjMgr) is an ObjInfo module (please see also the ObjMgr documentation for information about this module).

For more information on the data structure see File format.

3.2 The Cases

The METK was designed to handle medical cases of segmented structures and to visualize them in an appropriate environment. The METKManager can load cases of different types:

- Cases generated with HepaVision
- Cases generated with RhinoVision
- Cases generated with NeckVision
- Cases generated with NeckSegmenter
- METK cases

For how the cases are identified, please see HepaXMLConverter, NeckVisionXMLConverter, StandardXMLConverter and ConvertXML. Additionally, you may set up an new case by using the CaseBuilding modules.

3.3 The 3d Objects

The basic concept of the 3d scenes in MeVisLab and also in the METK is OpenInventor. So all structures are stored in single OpenInventor files. The METKManager creates automatically OpenInventor files for structures, that haven't one. After the first creation, the files are only loaded. That speeds up the loading process as well as the visualization process where the user can fast enable and disable the structures.

The basis of all inventor files is a segmentation mask. The METKManager uses an internal GenerateIVFile module to create all inventor files from their segmentation masks. If there are multiple (non overlapping) objects are stored in one segmentation mask, each structure has additionally an ObjectValue (objID⇒Image⇒ObjectValue) to identify its part of the segmentation mask. This (now called *single coded segmentation mask*[1]) masks are only used by HepaVision.

[1] In contrast to *multi coded segmentation* masks of METKCodedSegmentation

4 File format

5 Visualization concept

- creating IsoSurfaceContainer for each object
- inventor stuff like SoShapePartition

6 Messaging Concept

One of the most challenging problems on working with the METK is that you don't know which of your module in the network is being processed at the moment. When the Loaded event is sent for example, it is not possible to forecast in which order the modules will process the event. Also—depending on the network—user interactions will cause different module to react.

Thus, it is not possible to forecast when an initiated process has been finished by another module. Especially the mixture of python and C++ code makes it very difficult. To overcome these problems, an own messaging system is provided, besides listening to data-changing-events (DCE).

For synchronized communication between two modules, `METKMsgSender` and `METKMsgReceiver` were developed. These two modules are able to send message to each other, which reception (*processing*) and finishing (*done*) must be confirmed. There can be arbitrary senders and receivers, with one restriction: If there are more than one `METKMsgReceiver` listening to the same message, only one receiver will receive the message. It is not defined, which one.

Sender and receiver exchange data by establishing a shared communication channel. If there is no responsible receiver for a message, the transmission is canceled. If there is one, the message data could be sent. After processing the data, the receiver will sent a confirmation to the sender. For more details on sending and receiving data, please have a look into Section 9. The `METKManager` already uses the messaging concept.

7 Build your own METKModule

- basic concept of using the METKObjInfo with loaded and cleanup callbacks
- shrink the number of listened events

8 Module Overview

All modules grouped by their Genre and with the short description sentences.

Module	Short Description
METK	
METKManager	The METKManager is the heart of every METK network. This module must always be included.
METKMisc	
METKCSO	It gives you information about the currently selected object (CSO), the region of interest (ROI) to which it belongs and the selection point.
METKCaseObject	Simple module for setting and getting case info data in an METK case.
METKClipping	...
METKCollections	This module stores the current state of the ObjMgr along with a short comment, a caption, a small thumbnail and a screen shot, referred to as a *Collection*.
METKGenerateVesselIV	...
METKLiftChart	Display the vertical extent of segmentation results.
METKMeasures	This module computes the main axis and volume of a structure as exact as possible.
METKPatientObject	Simple module for setting and getting patient info data in an METK case.
METKPicking	Enables picking of objects in a 3D scene. Is already included in METKViewer3D.
METKStatus	Prompts the current status of the METK.
METKStructureDetails	Displays the details of an existing structure.
METKCaseBuilding	
METKAddImage	Generates database entries for a new image
METKAddROI	Generates database entries for a new ROI
METKAddStructure	Generates database entries for a new structure
METKDebug	
METKDebug	This module logs frequency and amount of ObjMgr events.
METKLogFile	Prompts error messages on errors and writes console content to a file.

METKMessaging	
METKGlobalMessages	Receives global loaded and cleanup messages
METKMsgReceiver	Module to receive data for synchronized message handling
METKMsgSender	Module to send data for synchronized message handling
METKGUI	
METKBookmark3D	Saves position in the viewer and bring you to this positions animated.
METKExplorer	Stand alone application to browse and manipulate a case and its objects.
METKHierarchyBrowser	Browser to visualize the hierarchy in a case. Structures can be selected.
METKInfoWin	Opens and close a small window for messages
METKObjBrowser	Provides a little GUI to browse objects and change their visual parameter
METKSingleObjInfo	This module provides several windows to view and edit the data of single objects or layers.
METKStructureGroupBrowser	This module enables the user to browse through the case data by using the entries ⇒Structure and ⇒StructureGroup.
METKVesselExplorer	Stand alone application to convert single vessel parts to convolution surfaces
METKImageDisplay	
METKCodedSegmentation	Provides the basis of METKOverlay2D as a multi coded segmentation
METKROISelect	Selects one of the available ROIs
METKOverlay2D	This module generates an overlay displaying the visible segmented structures in a 2D viewer.
METKLabeling	
METKLabels	...
METKOpen	
METKCaseOptimizer	Stand alone application to clean up case folders
METKLoadSegMask	Opens the segmentation mask of a structure
METKPackFiles	Compress all segmentation masks in folder.
METKSaveCase	Saves a case to a XML file
METKRenderModes	

METKIsoSurface	Loads the iso surfaces of the segmented structures.
METKSilhouette	Loads the silhouettes of the segmented structures.
METKStippling	Loads the stipple rendering of the segmented structures
METKTexturing	Loads textures coordinates for segmented structures
METKSurfaceDistance	
METKDistanceTransform	Computes a distance transformation for one or more structure groups.
METKSurfaceDistance2D	This module displays two isobars of a distance transformation. Color and value may be specified.
METKSurfaceDistance3D	This module displays two regions of a distance transformation on the segmented surfaces.
METKViewer	
METKViewer2D	The METKViewer2D simply displays a ML images, but it provides some additional features.
METKViewer3D	...
METKViewpoint	
METKCalcCamPos	...
METKCreateIntraOPDummy	...
METKGenerateCamData	...
METKIntraOPViewpoint	...
METKToSolverConnection	...
METKAnimation	
METKAutoFading	...
METKObjXMLWriter	...
METKScriptBuilder	Builds animation scripts from different ObjMgr states
METKScriptInit	...
METKScriptTrigger	...
UMDAnimation2	...
Other important but Non-METK modules	
SoShapePartition	...
GenerateIVFile	...
IsoSurfaceContainer	...
Local/Developer	

ConvertXML	This module converts XML files to the METK format
HepaXMLConverter	This module converts XML files from HepaVision to the METK format
NeckVisionXMLConverter	This module converts XML files from NeckVision (old version) to the METK format
StandardXMLConverter	This module converts XML files from METK to the METK format

9 Module descriptions in alphabetic order

9.1 METKAddImage

Genre:	METKCaseBuilding
Authors:	Christian Tietjen
Short description:	Generates database entries for a new image

Usage

This module sets all necessary entries for a given image in the ObjMgr. It can be used for a new or an existent case.

The name of the image will be retrieved automatically by the image input. The image's name will be used to create an object ID.

Send ObjMgr-Events

ObjMgr-Key	Value	Action
ObjID ⇒Global ⇒ObjectType	Source	Is always a source image
ObjID ⇒Global ⇒ChildIds	–	No children yet
ObjID ⇒Global ⇒Valid	True	Is always true
ObjID ⇒Image ⇒Filename		Name of the file (without path)
ObjID ⇒Image ⇒ImageType	Original	Is always an original
ObjID ⇒Image ⇒ObjectValue	-1	Has no specific value
ObjID ⇒Image ⇒ToWorldMatrix		Transformation matrix
ObjID ⇒Image ⇒Modality	CT/MRT/...	
ObjID ⇒Image ⇒Protocol		
ObjID ⇒Image ⇒VoxelSizeX		Voxel's size in x direction
ObjID ⇒Image ⇒VoxelSizeY		
ObjID ⇒Image ⇒VoxelSizeZ		
ObjID ⇒Image ⇒SizeX		Number of voxel in X direction
ObjID ⇒Image ⇒SizeY		
ObjID ⇒Image ⇒SizeZ		
ObjID ⇒Image ⇒StudyDate		
ObjID ⇒Image ⇒StudyTime		
ObjID ⇒Image ⇒Institution		
ObjID ⇒Image ⇒Manufacturer		
ObjID ⇒Image ⇒ManufacturersModelName		
ObjID ⇒Image ⇒BodyPartExamined		

Inputs and Outputs

image:	Input image for retrieving name, location, size and DICOM tags

Fields

objectID: (string)	Name of the created Object
create: (trigger)	Create all entries

Related modules

METKManager, METKAddROI, METKAddStructure

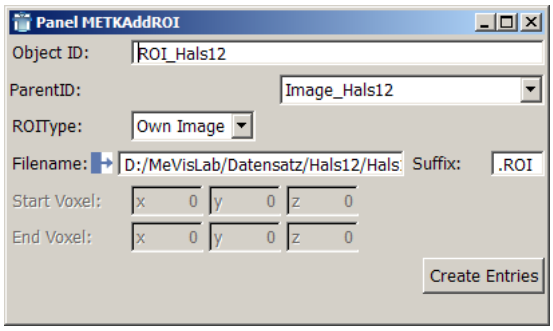
9.2 METKAddROI

Genre:	METKCaseBuilding
Authors:	Christian Tietjen
Short description:	Generates database entries for a new ROI

Usage

This module sets all necessary entries for a ROI in the ObjMgr. It can be used for a new or an existent case. To specify a new ROI, the ROI must be inherited from an image (e.g. generated with METKAddImage). The image input is checked to find the related image. The childIDs entry of the image is set automatically by ROI creation. The name of the image will be retrieved automatically by the image input, but may be edited afterwards. The image's name will be used to create an object ID.

Depending on the chosen ROI type, a new image dataset is saved to the hard drive (by using the given suffix), or the input image is used and clamped by startVoxel and endVoxel.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
ObjID ⇒Global ⇒ObjectType	Source	and
ObjID ⇒Image ⇒ImageType	Original	if so, the field parentID is updated

Send ObjMgr-Events

ObjMgr-Key	Value	Action
ObjID ⇒Global ⇒ObjectType	Result	Is always a resulting image
ObjID ⇒Global ⇒ChildIds	–	No children yet
ObjID ⇒Global ⇒ParentId		Object ID of the parent image
ObjID ⇒Global ⇒Valid	True	Is always true
ObjID ⇒Image ⇒Filename		Name of the file (ROIType ownImage or not set)
ObjID ⇒Image ⇒StartVoxel		Starting voxel (ROIType subregion or not set)
ObjID ⇒Image ⇒EndVoxel		Ending voxel (ROIType subregion or not set)
ObjID ⇒Image ⇒ImageType	Original	Is still an original
ObjID ⇒Image ⇒ObjectValue	-1	Has no specific value
ObjID ⇒Image ⇒ToWorldMatrix		Transformation matrix
ObjID ⇒Image ⇒SizeX		Number of voxel in X direction
ObjID ⇒Image ⇒SizeY		
ObjID ⇒Image ⇒SizeZ		

Inputs and Outputs

image:	Input image for retrieving name, location, size and DICOM tags

Fields

objectID: (string)	Name of the created Object
parentID: (string)	Available parent IDs
ROIType: (string)	Subregion or OwnImage (see Send ObjMgr-Events)
filename: (string)	Depends on ROIType, name of the resulting file
suffix: (string)	Depends on ROIType, name suffix of the resulting file
startVoxel: (int)	Depends on ROIType, clamps the input image
endVoxel: (int)	Depends on ROIType, clamps the input image
create: (trigger)	Create all entries

Related modules

METKManager, METKAddImage, METKAddStructure

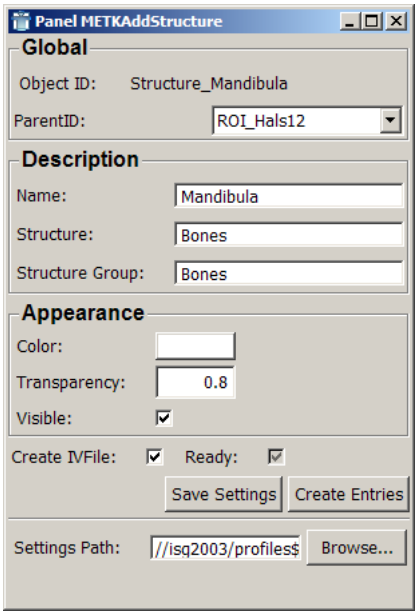
9.3 METKAddStructure

Genre:	METKCaseBuilding
Authors:	Christian Tietjen
Short description:	Generates database entries for a new structure

Usage

This module sets all necessary entries for a structure in the ObjMgr. It can be used for a new or an existent case. To specify a new structure, the structure must be inherited from a existent ROI (e.g. generated with METKAddROI). The image input is checked to find the related ROI. The childIDs entry of the ROI is set automatically by structure creation. The name of the image will be retrieved automatically by the image input. The object ID of the structure will be generated by the name of the structure. The user must ensure that there is no object with the same name (object ID).

The standard entries may be set by this module. The settings may be saved and reused for each structure type. After filling out all entries, the settings may be saved by saveSettings. When entering a new structure, the values will be loaded after the structures type was entered. The isosurface of the structure is created if you want to.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID ⇒Global ⇒ObjectType`	Result	and
`ObjID ⇒Image ⇒ImageType`	Original	if so, the field parentID is updated

Send ObjMgr-Events

ObjMgr-Key	Value	Action
ObjID ⇒Global ⇒ObjectType	Result	Is always a resulting image
ObjID ⇒Global ⇒ParentId		Object ID of the parent image
ObjID ⇒Global ⇒Valid	True	Is always true
ObjID ⇒Image ⇒Filename		Name of the file (ROIType ownImage or not set)
ObjID ⇒Image ⇒ImageType	Segmentation	Is a segmentation result
ObjID ⇒Image ⇒ObjectValue		Value of the segmentation mask
ObjID ⇒Description ⇒Name		Name of the structure
ObjID ⇒Description ⇒Structure		Type of the structure
ObjID ⇒Description ⇒StructureGroup		Group of the structure
ObjID ⇒Files ⇒InventorFile		IV filename (only if created)
ObjID ⇒Appearance ⇒Color		
ObjID ⇒Appearance ⇒Transparency		
ObjID ⇒Appearance ⇒Visible		

Inputs and Outputs

image:	Input image for retrieving name, location, size and DICOM tags

Fields

objectID: (string)	Name of the created Object
parentID: (string)	Available parent IDs
name: (string)	Name of the structure
Structure: (string)	Type of the structure
StructureGroup: (string)	Group of the structure
Color: (color)	Color of the structure
Transparency: (float)	Transparency of the structure
Visible: (bool)	Visible or not
createImageFile: (bool)	Creates a new image file (doesn't use the existent)
createIVFile: (bool)	Creates isosurface
ready: (bool)	Isosurface is created
saveSettings: (trigger)	Save the settings for the current structure type
create: (trigger)	Create all entries
settingsPath: (string)	Path to the settings files

Related modules

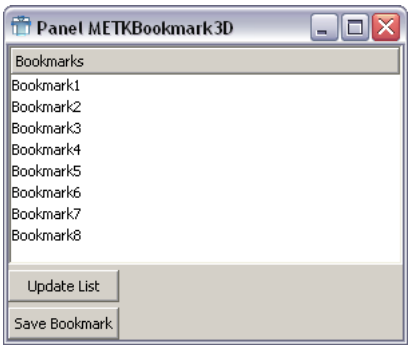
METKManager, METKAddROI, METKAddImage

9.4 METKBookmark3D

Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	Saves position in the viewer and bring you to this positions animated.

Usage

If this module is included in a network together with an UMDAnimation2 module, it can bookmark any position in a viewer. Double clicking a bookmark brings you animated back to the bookmarked position.



Developer Information

The UMDAnimation2 is needed, because the METKBookmark3D generates a script and send it to UMDAnimation via ObjMgr to set the bookmarked positions.

Send ObjMgr-Events

Writes its bookmarks in the `OBJ_BOOKMARKS` object. Sends animation scripts via `OBJ_ANIMATION`⇒ `LAY_ANIMATION_SCRIPT`⇒ `INF_SCRIPT_FULLSCRIPT`

Fields

`updateList:`	Reads values from OBJ_BOOKMARKS and refresh the list view
`saveBookmark:`	Force the viewer to write its current values in the ObjMgr and save this values in a new layer under OBJ_BOOKMARKS
`viewerName:`	Name of the viewer, that position should be bookmarked

Related modules

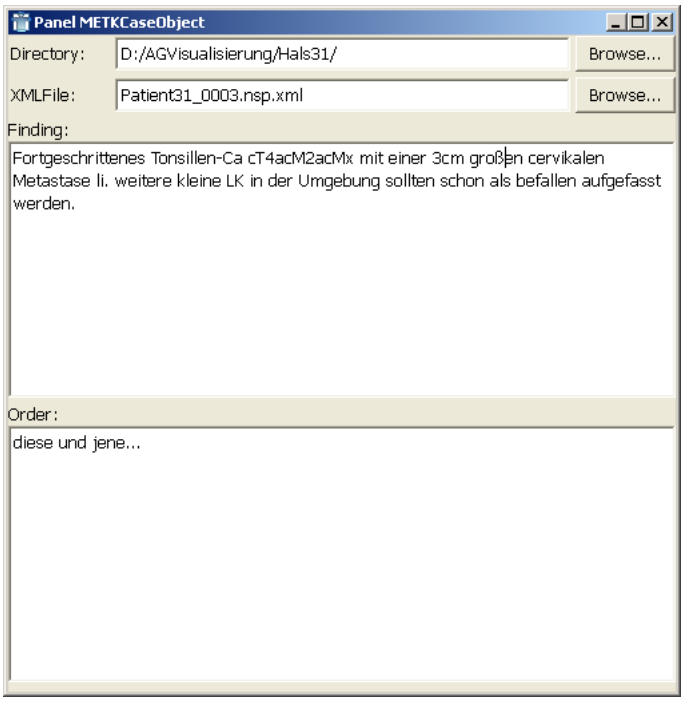
UMDAnimation2, METKViewer3D

9.5 METKCaseObject

Genre:	METKMisc
Authors:	Jana Dornheim
Short description:	Simple module for setting and getting case info data in an METK case.

Usage

Simple module for setting and getting case info data in an METK case.



Listen ObjMgr-Events

The module listen to LAY_CASE and other events, but only the author knows, what there is exactly happen.

Fields

directory:	directory where the case data is located
XMLFile:	the current opened XML file
finding:	the finding indicated by the radiologist or surgeon
order:	filled out by the surgeon to order a segmentation task

Related modules

METKPatientObject

9.6 METKCaseOptimizer

Genre:	METKOpen
Authors:	Konrad Mühler
Short description:	Standalone application to clean up case folders

Usage

Use this module with care and only if you know what you are doing!!!

This is a standalone application (without a network) that parse a METK case file for all files in the case folder. If a filename doesn't exist in the case file, the file is deleted. So all unused files like segmentation mask without object or vessel tree files without corresponding objects will be deleted.

Use only METK files and only, if this XML are the only one in the case folder. Other XML case files in the folder could use other files, that shouldn't be deleted.

Fields

8cm **filename:**	METK xml filename
getUnusedFiles:	Not only get the unused files but also delete them!

Related modules

METKPackFiles

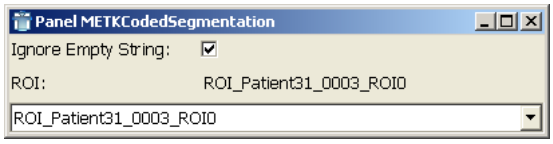
9.7 METKCodedSegmentation

Genre:	METKImageDisplay
Authors:	Konrad Mühler, Christian Tietjen
Short description:	Provides the basis of `METKOverlay2D` as a multi coded segmentation

Usage

Normally, every segmented object is saved as one file. If you want to display all these masks in a View2D for example, you have to add one SoView2DOverlay per mask. This results in a very poor performance. To avoid that, `METKCodedSegmentation` generates one file per current ROI and adds all masks derived from this ROI in only one file. This has to be done only one time. On the next load, the generated file will be loaded automatically.

If there is an CSO object with a corresponding ROI, the coded segmentation will be generated. The output of the module can be used with `METKOverlay2D` for example.



Developer Information

The module checks whether there is a ROI in the current case. If so, the first one is loaded and the generation starts, or simply the pre-generated is loaded. If there are more than one ROI, the computation starts only for the first one found in the ObjMgr. The object values are stored for each objects in its ⇒`Image`⇒`CodedSegObjectValue` devided by comma. Additionally all objects for each voxel value are stored in `CodedSegmentation` ⇒`roiName` ⇒`VOXELVALUE`.

For more information ask Konrad Mühler or wait for a detailed documentation.

Inputs and Outputs

Output:	currentCodedSegmentaiton (MLImage)	Coded segmentation image for the selected ROI

Fields

8cm	(default yes): If ROI is set to [None], the module maintains the previous ROI as input. On NO, the output contains no data.
ignoreEmptyString:	
rois:	List of the available ROIs in the case.
roiSelect:	selected ROI
listenToRoi:	Tag to synchronize ROI selection. All modules with the same tag in `listenToRoi` updating their rois if one of them change its selection.
`(string)`	
objValues:	List of used voxel values for each object
imgValues:	List of all voxel values and their containing objects
forceGeneration:	(re)generate a coded segementation
closeImg:	close the loaded image

ToDo List

- Explain in detail, what a multi coded segmentation is and how it is generated

Related modules

`METKOverlay2D`, `METKROISelect`, `METKViewer2D`

9.8 METKCollections

see **??**

9.9 METKCSO

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	It gives you information about the currently selected object (CSO), the region of interest (ROI) to which it belongs and the selection point

Usage

It gives you information about the currently selected object (CSO), the region of interest (ROI) to which it belongs and the selection point. The output field gives you a pointer to the iso surface object. All values are read only! METKCSO listens for changing values in the ObjMgr database (CSO, SelectedObject).

Listen ObjMgr-Events

ObjMgr-Key	**Value**	**Action**
`objectID` ⇒`LAY_SELECTEDOBJ` ⇒`INF_SELECTION`		Sets the new CSO on outputs
`objectID` ⇒`LAY_SELECTEDOBJ` ⇒`INF_OBJID`		

Inputs and Outputs

Output:	currentlySelectedObject (Inventor)	The currently selected object as plain inventor file (without material)
Output:	currentlySelectedObject WithMaterial (Inventor)	The currently selected object as plain inventor file with material

Fields

`cso:`	ObjectID of currently selected object.
`roi:`	ROI for CSO
`selection:`	Vector of picked point of the CSO
`ignorePickable:`	FALSE, if the flag `objID` ⇒`LAY_APPEARANCE` ⇒`INF_PICKSTYLE` of objects should be taken into account, so that objects, that are marked as "'UNPICKABLE"' are not pickable
`objectMustBeVisible:`	TRUE, if an object must be visible to set it as a CSO

Related modules

??

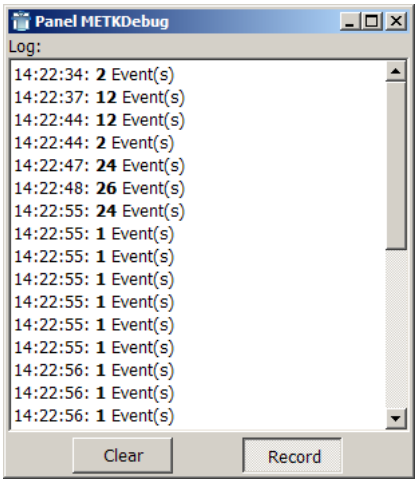
9.10 METKDebug

Genre:	METKDebug
Authors:	Christian Tietjen
Short description:	This module logs frequency and amount of ObjMgr events.

Usage

Very simple module for debugging. Sometimes too many events are sent separately by the ObjMgr. If "'notify"' is touched for every single event, the whole braodcasting mechanism is established. This can slow down an application dramatically. To get an idea of which module is sending unnecessary "'notifies"', this module can be used. It doesn't tell you, which module is sending, but it tells you how much events are sent at one time and at which time.

The events are recorded as long as the "'Record"' button is pressed.



Developer Information

It is not possible to track the module which has released the events.

Fields

11cm	output of the thrown events with time and number
`log:`	
`record:`	starts recording when pressed and stops when released
`clear:`	clear all recorded events from log

Related modules

METKLogFile

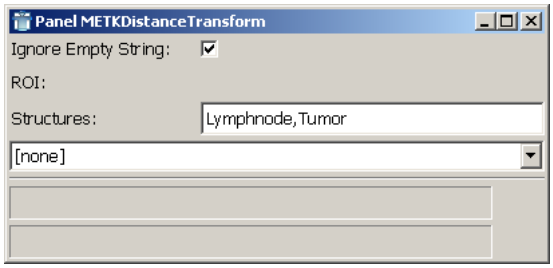
9.11 METKDistanceTransform

Genre:	METKImageDisplay
Authors:	Christian Tietjen
Short description:	Computes a distance transformation for one or more structure groups.

Usage

Computes a distance transformation for one or more structure groups. It is not possible to specify single structures, because the computational effort would be to high. The module searches for all matching `objectID` ⇒`Description` ⇒`StructureGroup` entries.

The panel includes two progress bars. The first is the progress of the distance transformation computation, the second indicates the saving progress.



Inputs and Outputs

Output:	transformationImage (MLImage)	The generated distance transformation image

Fields

ignoreEmptyString:	If ROI is set to `[None]`, the module maintains the previous ROI as input. On NO, the output contains no data.
ROI:	List of the available ROIs in the case. In combination with METKCSO you can always choose the right one.
structures:	The structures used as seed point for the distance transformation. Multiple structures may be specified separated by commas and without whitespaces. Touching this field force a generation of transformation image.
listenToRoi: `(string)`	Tag to synchronize ROI selection. All modules with the same tag in `listenToRoi` updating their rois if one of them change its selection.

Related modules

`METKSurfaceDistance2D`, `METKSurfaceDistance3D`

9.12 METKExplorer

Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	Stand alone application to browse and manipulate a case and its objects.

Usage

The METKExplorer is a stand alone application, that also can be included in a environmental METK-network. It has its own METKManager, so that it can load its own cases. Via an external connection, it can be connected to another METKManager.

With the METKExplorer, the opened case can be browsed using the built-in `METKHierarchyBrowser` and `METKStructureGroupBrowser`.

Main window elements In the left part of the window, you can select an object whose parameter you want to view or change.

Changing the "‘Case type"’ only affects the recommended object names in the "‘Hierarchy"’ tab.

If you uncheck the "‘Use internal ObjMgr"’ flag, you can plug in an external `METKManager`. You must open the case with this METKManager and not with the METKExplorer.

You can choose between two types of object lists: a hierarchy browser and a structure group browser.

If you check "‘Show all objects"’ in the bottom part all objects (not only structures) are shown in the list.

If you check "‘Generate IVFiles on start"’ (default and recommended) the inventor files for all structures are created on startup if they don’t exist.

Clicking the "‘Delete objects without existing IV file"’ button all structures, that have no existing inventor file are deleted. This can be helpful if for some structures no inventor file can be created (occur in some MeVis Distance Service cases)

Hierarchy tab In the "‘Object Name"’ section you can see the object name and its ID. These values can be the same, but in the METK they normally differ. If a new name for an object is set using the "‘Rename Object"’ button, the objectID changes to "‘Structure_"’+object name.

In the "‘Object location"’ section you can delete the object or change its parent. Changes will also be propagated in the ⇒`ChildIds` info of the objects parents.

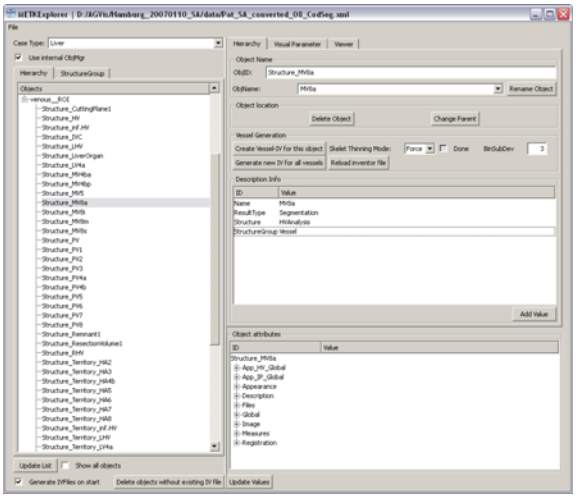
In the "‘Vessel Generation"’ section the standard inventor file can be regenerated using the convolution surfaces of Steffen Oeltze. Some parameter can be set and new convolution surfaces can be generated for all vessel objects in a case in one fell swoop.

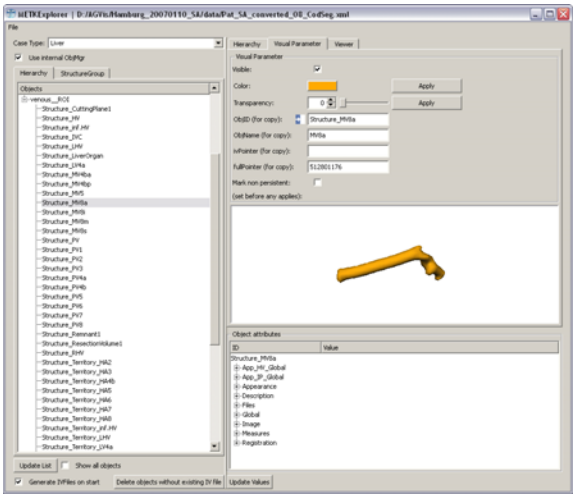
The data of the selected object can be viewed in "‘Description Info"’ and "‘Object attributes"’ (both of type `METKSingleObjInfo`). The values in the "‘Description Info"’ part come from the ⇒`Description` layer of the object and can be edited by double clicking a value. New info values can be added using the button in the bottom right.

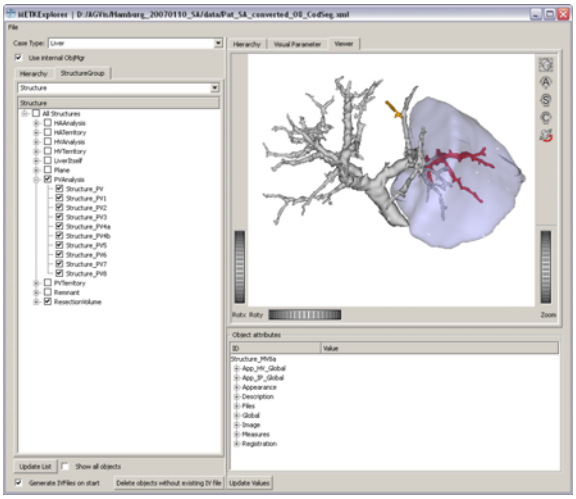
Visual Parameter tab In the "‘Visual Parameter"’ tab some visual parameters like color and transparency can be changed. In the viewer the single object in its visualization style is shown. The values must be applied. The values marked with "‘copy only"’ are read only values, that can be copied to use it elsewhere. If the "‘Mark non persistent"’ flag is set, the applied values are marked as non persistent, i.e. they will not be saved in the case file. **This flag must be set before any changes are applied!**

The controls of the tab are provided by the `METKObjBrowser`.

Viewer tab Here you can see all visible objects together.







Developer Information

Ask Konrad for more information. He himself can't remember all features and tricks using this module.

Fields

The fields are only help fields for the user interface.

ToDo List

- Implement MultiObject facilities from METKStructrueGroupBrowser into METKExplorer

Related modules

METKSingleObjInfo, METKHierarchyBrowser, METKStructureGroupBrowser, METKObjBrowser, **??**

9.13 METKGlobalMessages

Genre:	METKMessaging
Authors:	Christian Tietjen
Short description:	Receives global loaded and cleanup messages

Usage

While loading or closing a case, the METKManager sends messages to inform all METK modules to build or cleanup the internal data structures. The messages are only sent by METKManager! Using this module, the messages could be received via normal scripting. In the normal case, this module should not be used. Instead, the use of the Python class **??** is recommended.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
Communication ⇒GlobalEvents ⇒CaseLoaded	Loaded or Cleanup	Case is loaded or closed

Fields

`loaded:`	Touched on case loaded
`cleanup:`	Touched on case cleaned up

Related modules

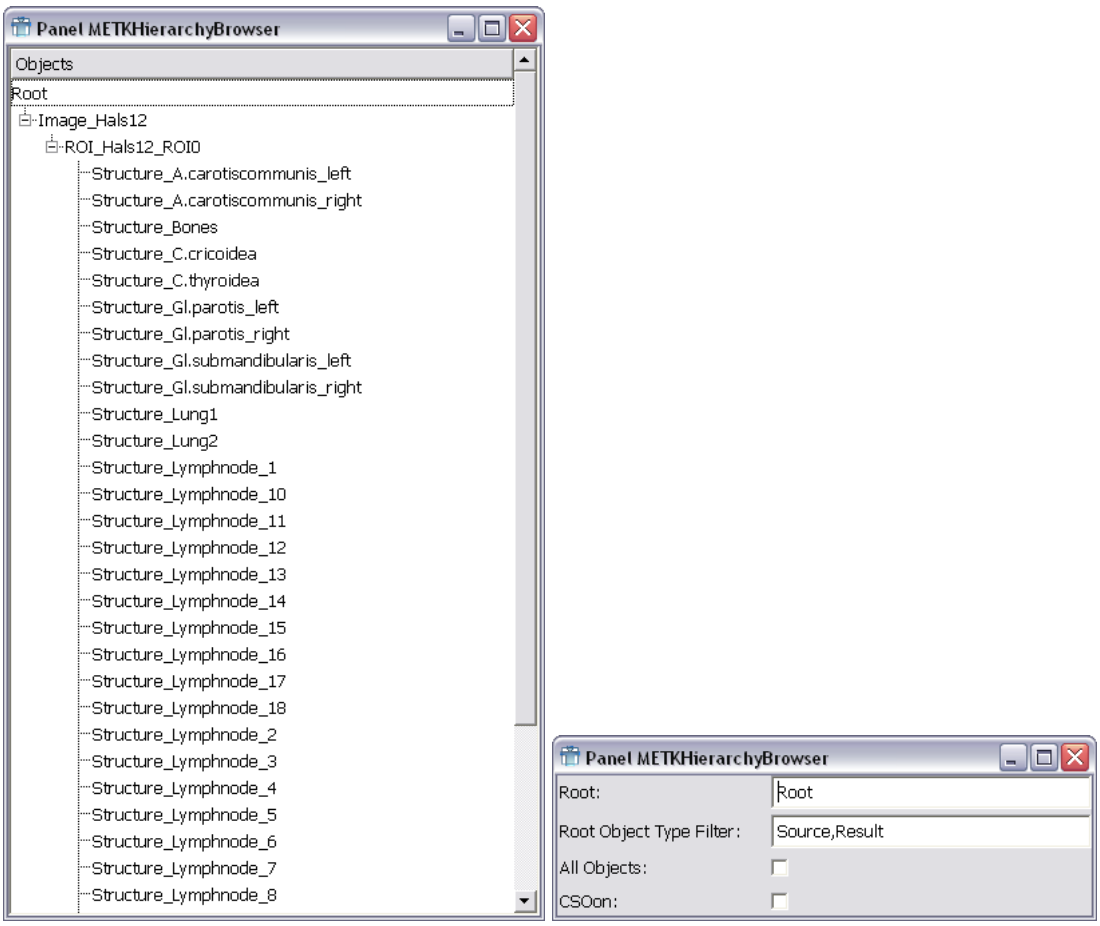
`METKManager`, `METKMsgReceiver`, `METKMsgSender`

9.14 METKHierarchyBrowser

Genre:	METKMisc
Authors:	Konrad Mühler, Christian Tietjen
Short description:	Browser to visualize the hierarchy in a case. Structures can be selected.

Usage

The `METKHierarchyBrowser` provides a folded list of all structures in a case and their parents. The list can be collapsed and single structures can be selected.



Developer Information

If `allObjects` isn't set, only objects are included in the list, that have one of the values listed in `rootObjectFilter` as their `ObjectType` (`ObjectID` ⇒`Global` ⇒`ObjectType`). Standard values are `Source` and `Result`. On the first level under the root only objects are shown, that have no parents or whose parents doesn't exist. Under each object all objects are shown, that are listed in the `ChildIds` of the object (`ObjectID` ⇒`Global` ⇒`ChildIds`).
Attention: An object can be listed as the child of multiple objects (not only one as microorganisms or two like the human)!

If `allObjects` is checked, the values of `rootObjectFilter` are ignored and all objects of the case are included in the list.

If `CSOon` is checked, the currently selected entry is published to all other METK modules as the CSO object (see METKCSO).

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
LOADED-EVENT		Update list view
CLEANUP-EVENT		Clear list view

Send ObjMgr-Events

ObjMgr-Key	Value	Action
CSO		If enabled the current selected structure is set to CSO

Fields

root:	String for the root of the tree
rootObjectFilter:	Possible values of ObjectType entry of an object, that should be included in the list
selectedObjId:	ObjectID of the currently selected object
CSOon:	If checked the current selected structure is set to CSO
allObjects:	If checked rootObjectFilter is ignored and all objects are included in the list
updateView:	Refresh the list view

ToDo List

- update to new loaded and cleanup handling

Related modules

METKStructureGroupBrowser, METKCSO

9.15 METKInfoWin

Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	Opens and close a small window for messages

Usage

This module opens and close a small window with messages. It can be opened and closed via the METKManager. The window can also be used without the METKManager and can be opened directly via the fields.

Developer Information

Known problems Sometimes the close event will be missed :-(

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
OBJ_GUI ⇒LAY_INFOWIN ⇒INF_ISON	TRUE/FALSE	Open or close the window
OBJ_GUI ⇒LAY_INFOWIN ⇒INF_INFOHEADER	String	Set the message header
OBJ_GUI ⇒LAY_INFOWIN ⇒INF_INFOTEXT	String	Set the message text

Fields

infoHeader:	Message header
infoText:	Message text
showWin:	Open/Close the window

Related modules

METKStatus

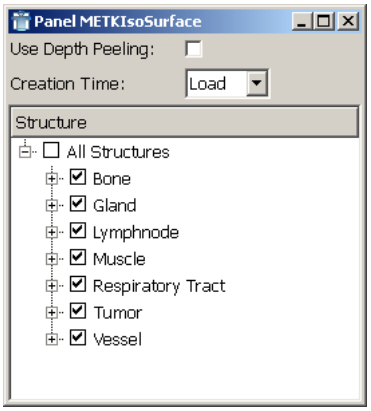
9.16 METKIsoSurface

Genre:	METKRenderModes
Authors:	Konrad Mühler, Christian Tietjen
Short description:	Loads the iso surfaces of the segmented structures.

Usage

Loads the iso surfaces of the segmented structures. The predefined colors and transparency values are used. Per default, the surfaces are created when the loaded event was sent. It is also possible to load the structures when they are treated as visible. Optionally, depth peeling may be used for the display, but depth peeling is not supported by all graphic cards.

All structures can be switched to (in)visible with the internal `METKStructureGroupBrowser`. The same behavior is reached by setting `ObjectID ⇒LAY_APPEARANCE ⇒INF_VISIBLE`.



Developer Information

For each object an `IsoSurfaceContainer` is created. For more information have look at the Visualization concept.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID⇒LAY_APPEARANCE ⇒INF_CLIPPING`		Move objects in the "'to clip group"' or in the "'not to clip"'

Inputs and Outputs

Input:	inClipping (Inventor)	Objects that should be clipped additionally to the structures that marked as clipped.
Output:	outInventor (Inventor)	Scene of visible structures

Fields

`useDepthPeeling`:	Enable depth peeling (Verweis auf Everitt bei NVidia)
`creationTime`:	**`Load`:** Creation after Loaded event. Takes some time at the beginning. **`Visible`:** Creation on general visible. Takes some time while exploring the data.

Related modules

`METKTexturing`, `METKSilhouette`, **??**

9.17 METKKeyStates

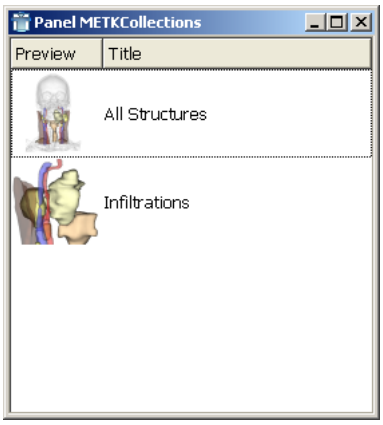
aka METKCollections

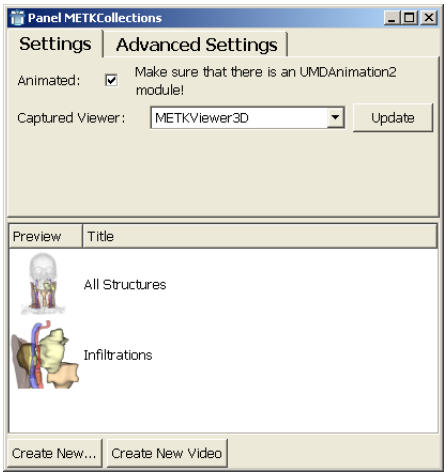
Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	This module stores the current state of the ObjMgr along with a short comment, a caption, a small thumbnail and a screen shot, referred to as a *key state*.

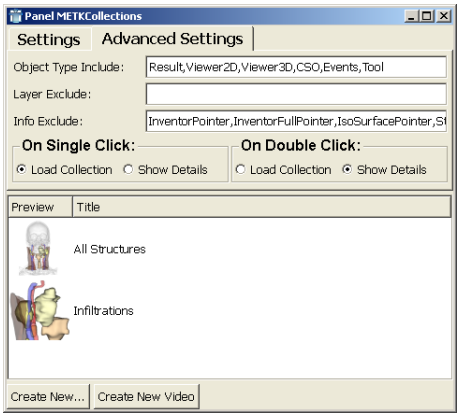
Usage

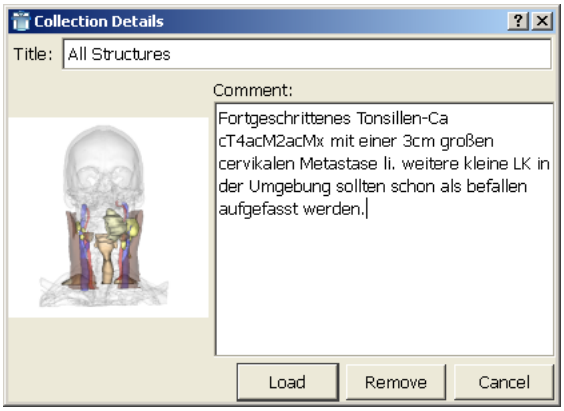
This module stores the current state of the ObjMgr along with a short comment, a caption, a small thumbnail and a screen shot, referred to as a *key state*. If there are more then one 2D or 3D viewer, a viewer to be captured has to be defined. Otherwise, the first viewer found in the ObjMgr is used (3D viewers are preferred). A key state may be accessed later and all ObjMgr entries are set back to the saved key state. In contrast to ObjDump, `METKKeyStates` does not care about non-persistent entries. To sort out unwanted entries, such as pointers to Open Inventor nodes, an excluding mask has to be defined. Also, if you want an object to be included, the object type has to be included (`objectID` ⇒`Global` ⇒`ObjectType`).

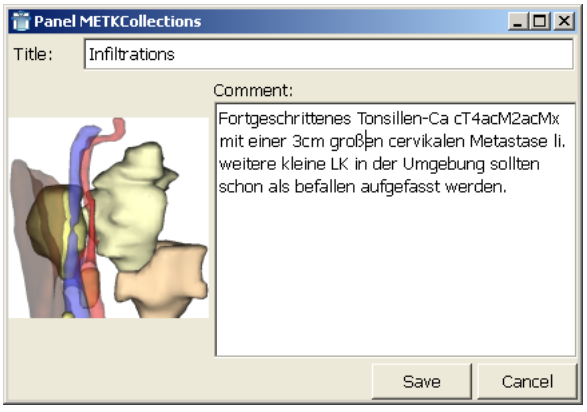
If an UMDAnimation2 module is connected to this module, it is possible to blend smoothly between the current state and the selected key state. Also, a video interpolating all key states may be generated. But there is one drawback: because the key state list is sorted alphabetically, the video also interpolates in the same order.











Listen ObjMgr-Events

ObjMgr-Key	Value	Action
ViewerName ⇒LAY_GLOBAL ⇒INF_GLOBAL_REFRESH	False or "'0"'	The viewer writes its values in the ObjMgr and so the METKKeyState module can proceed with creating a new key state

Send ObjMgr-Events

Send animation scripts, force viewers to refresh and writes screenshot data in viewer objects.

Fields

9cm **animated:**	all transitions are animated. Only possible if an UMDAnimation2 module is connected.
capturedViewer:	list of all connected viewers. The currently selected viewer will be captured.
objectTypeInclude:	only ObjMgr objects with following objects types are saved.
layerExclude:	excludes whole layers of the included objects
infoExclude:	single infos may be discarded
loadCollection:	load the key state on single/double click
showDetails:	only shows the key state's details
create:	create a new key state
renderVideo:	generate a video from the current set of videos

Related modules

METKScriptBuilder, UMDAnimation2, METKViewer3D, ObjDump

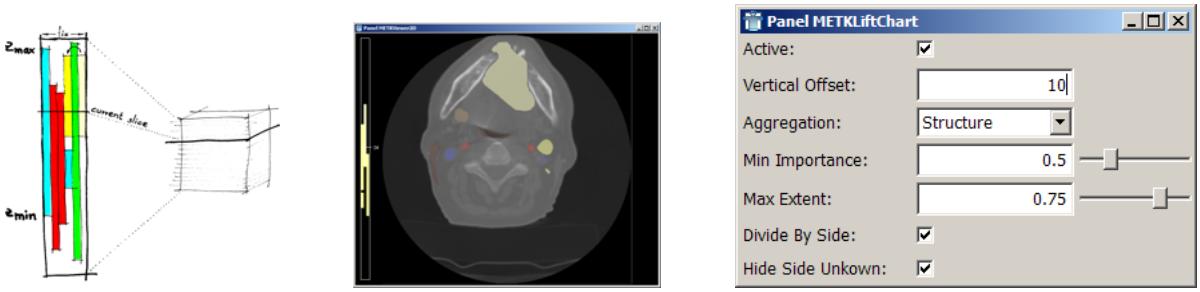
9.18 METKLiftChart

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Display the vertical extent of segmentation results.

Usage

The basic problem of slice based visualization, namely the lack of an overview in cross-sectional images, has been tackled with a 2.5D approach to provide the essential information, the LiftChart LiftChart. A narrow frame attached next to the cross sectional image represents the overall extent of slices in the volume data set. The top and bottom boundary of the frame correspond to the top and bottom slice of the dataset. Each segmented structure is displayed as a bar at the equivalent vertical position inside this frame. Upper bars correspond to higher structures in the body.

The currently displayed slice of the volume dataset is depicted by a horizontal line in the LIFTCHART widget. The slice number is displayed next to this representation. To visualize not only the z-distribution of structures in the volume, but also information about their horizontal position, we developed several arrangements of the bars in the LIFTCHART.



divideBySide and hideSideUnknown: Since some anatomic structures have a defined side, e.g. the left or right, the LIFTCHART may be divided in three parts: one part for structures on the left and on the right side each and one part for structures in the middle.

Aggregation: None The most simple form shows each anatomic structure represented by one bar.

Aggregation: Structure, StructureGroup and All: It is also possible to group bars which belong to the same class of anatomical structures to minimize the horizontal extent of the widget. The lymph nodes are aggregated into one column. The choice whether structures should overlap each other in the LIFTCHART or not depends on the clinical question. To make it possible to emphasize single structure within one aggregated column, the more important structures are drawn at last. Thus, if they are colored differently, they are visible again.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID ⇒Appearance ⇒*`	*	Listen for relevant events
`ObjID ⇒Description ⇒*`	*	Listen for relevant events

Fields

active:	Toggle LIFTCHART display
verticalOffset:	Offset to the top and bottom border
aggregation:	See above.
minImportance:	Slider to blend off unimportant structures. If the Importance tag is not set, the 1 - Transparency is used.
maxExtent:	Slider to blend off too big structures. Many structures range over the whole extent of the dataset. Thus, the provided information is useless.
divideBySide:	See above.
hideSideUnknown:	See above.

Related modules

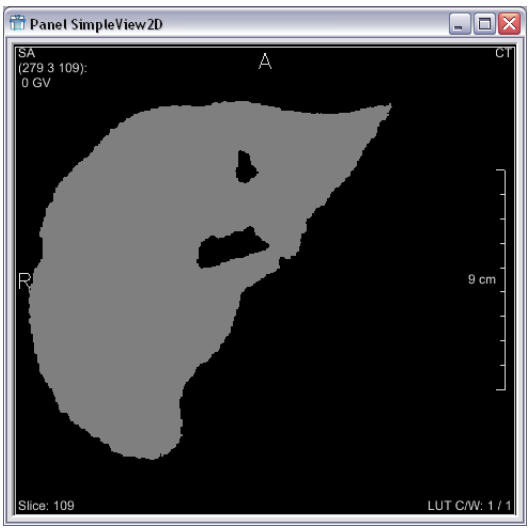
METKOverlay2D

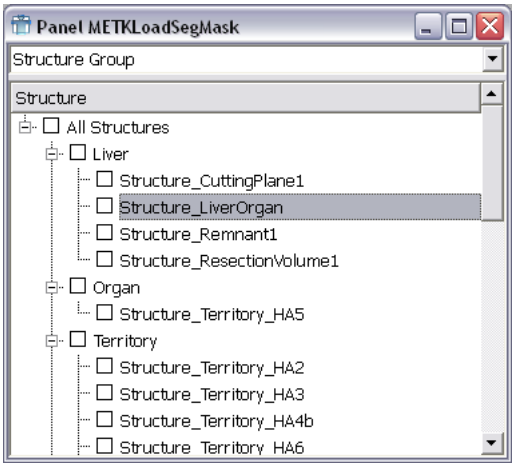
9.19 METKLoadSegMask

Genre:	METKOpen
Authors:	Konrad Mühler
Short description:	Opens the segmentation mask of a structure

Usage

This module opens the segmentation mask of a structure. With a given objectID the mask specified in objID⇒LAY_IMAGE⇒INF_FILENAME is loaded and thresholded by the object value from objID⇒LAY_IMAGE⇒INF_OBJVALUE. If there exists no object value, the image is thresholded by the image min/max value. The output has value 0 for background and value 1 for object voxels. If there are multiple values in the image and no object value is given, the values of the output are scaled to 0...1.





Inputs and Outputs

Output:	mask (MLImage)	Segmentation mask

Fields

`objID:`	ObjectID of the structure the segmentation mask should be loaded.

Related modules

`METKViewer2D`, `METKCodedSegmentation`

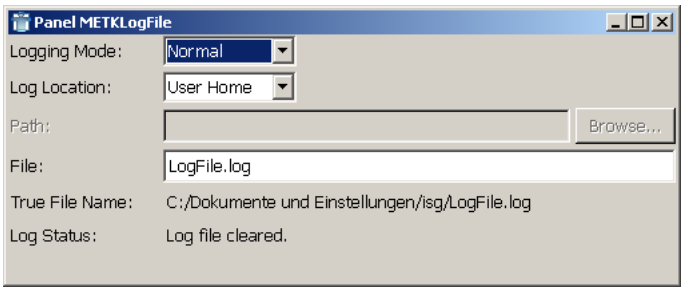
9.20 METKLogFile

Genre:	METKDebug
Authors:	Christian Tietjen, Konrad Mühler
Short description:	Prompts error messages on errors and writes console content to a file.

Usage

Prompts error messages on errors and writes console content to a file. The sensibility depends on the logging mode:

- **`Normal`**: This is the developer mode. In this mode, NO error message prompts on fatal errors. The log file is written and includes the whole console output.
- **`User`**: The is the end user mode. In this mode, an error message prompts on fatal errors. Thus, the user knows when something has gone wrong. The log file is written and includes the whole console output.
- **`Demo`**: Nothing is logged or put out for maximum performance.
- **`Test`**: Logs and traces all errors, but doesn't prompt an error message.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`OBJ_CASE⇒LAY_CASE ⇒INF_CASEDIR`		Potentially change folder to save log file

Fields

`loggingMode:`	The modi described above.
`logLocation:`	Writes the log file into the users home, into the directory where the opened case is located, or into a user defined directory.
`path:`	Path to the log file. Only available when a user defined destination is selected.
`file:`	Name of the log file.
`trueFileName:`	The directory and filename written out.
`logStatus:`	Current status of the log file.

Related modules

METKDebug

9.21 METKManager

Genre:	
Authors:	Christian Tietjen, Konrad Mühler
Short description:	The METKManager is the heart of every METK network. This module must always be included.

Usage

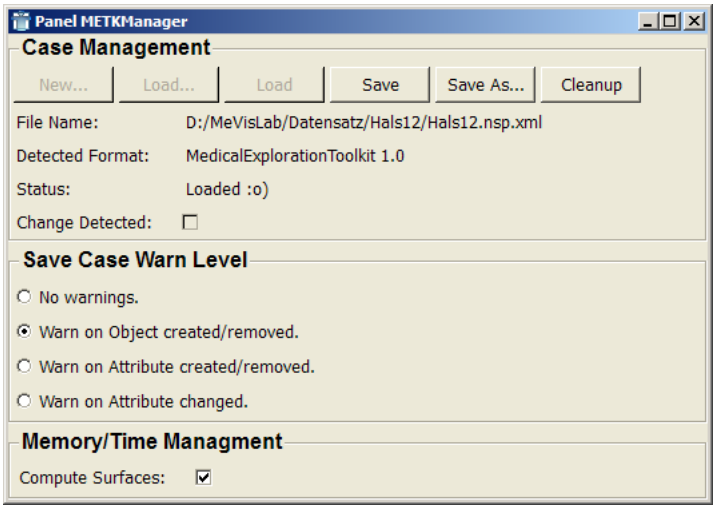
The METKManager is always needed in an METK network. It takes care about the consistency, creates Open Inventor Surfaces out of segmentation masks if necessary and converts different XML files into the METK readable format.

To load a case, e.g. a neck or liver data set, enter the filename and load the case. All METK modules, you want to use, must be connected to the METKManager before you press "'Load"'! While loading the case, the manager prompts several status informations. Loading has been finished when "'Loaded :o)"' is prompted in the status message. After that, the manager sends the message "'Loaded"' to all connected METK modules. If you want to load a new case, you have to cleanup the old case first. This will cause the manager to tell all other METK modules to clean up their internal data structures.

METKManager comes up with a warning message and a question to save the case, if something noticable has changed in the data. The case is stored in a XML-file (see File format). When load a case a first time, converted from another application, there will be a lot of changes, so you will get a warning without changing anything on your own. The saved XML file must be in the same directory as the original XML file.

You may enable different warning levels:

- **No warnings:** What you expect.
- **Warn on Object created/removed:** If an object in the ObjMgr data was created or added, the module will warn you.
- **Warn on Attribute created/removed:** If an info in the ObjMgr data was created or added, the module will warn you.
- **Warn on Attribute changed:** Or in other words, warn almost always.



Developer Information

Generating a Case When a case will be loaded, the METKManager proceeds in the following order:

1. Prompts for location and name.
2. Checks whether the case already exists. If so, do nothing.
3. If not, set the main ObjMgr's event logging to false.
4. Set path and file name in the Case object.
5. Set the main ObjMgr's event logging to true.
6. Send "'Loaded"' event.

Loading a Case When a case will be loaded, the METKManager proceeds in the following order:

1. Check whether there is an already loaded case. If so, do nothing and print out a warning.
2. If not, load the XML file into the ObjMgr "'source"' and tell `ConvertXML` to try to convert the data into METK data format.
3. If successful, set the main ObjMgr's event logging to false.
4. The file and database integrity will be checked. For example, whether there are all referenced image files and entries.
5. If a found structure object has an Open Inventor file entry, check for existence.
6. For layer Appearance, set the obligatory info entries if they do not exist (Color, Visible etc.)

7. For layer Image, set the obligatory info entries if they do not exist (ObjectValue, VoxelSize etc.)
8. After the integrity check, generate the missing IV files if necessary.
9. Create all containers `METKInventorObject` for providing the Open Inventor files.
10. If successful, set the main ObjMgr's event logging to true.
11. Send "'Loaded"' event.

Clean up a Case

1. Check whether there is a loaded case. If not, do nothing and print out a warning.
2. Clear ML cache.
3. Send "'Save"' event if the user confirms the dialog. This will cause all other METK modules to save their files.
4. Send "'Cleanup"' event. This will cause all other METK modules to clean up their stuff.
5. Remove all `METKInventorObject` containers.
6. Clear all internal ObjMgr's object containers.

Known problems Warnings also appear when a non-persistent entry was changed.

The saved XML file must be in the same directory as the original XML file.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID ⇒Appearance ⇒Visible`	TRUE	Create a surface container if not exist
	FALSE	do nothing

Send Global Messages

Message	Data	Description
`Loaded`	—	Tells everyone that case and database are loaded.
`Cleanup`	—	Tells everyone to clean up their data.

Received METKMsgMessages

Message	Data	Description
createIVFile	ObjectID	Generate the surface of the given structure.
getMeasures	ObjectID	Compute the measures of the given structure.
loadFile	Path and file name	Another way to load an XML case file.

Fields

new:	generate a new dataset. Prompts for location and name.
load:	will cause the manager to load the current case
browse:	will cause the manager to load a case, selected by a file dialog
save:	Trigger to save the case with the given filename.
saveAs:	Save the case with the user specified filename via dialog.
cleanup:	will cause the manager to clean up the case
fileName:	path/filename of the desired case
detectedFormat:	format of the XML file
status:	current status of the manager
changeDetected:	Is true when changes are recognized.
warnLevel:	If a case is going to close, and changes were made, the warn level defines, if when a message appear. Possible values: 0...3
computeSurfaces:	Compute surfaces on load event

ToDo List

Include METKSaveCase into the manager and establish a new message "‘Save"’ to tell all modules, that their temporary generated data needs to be saved, or to be discarded when the message is not sent.

Related modules

ConvertXML, NeckVisionXMLConverter, HepaXMLConverter, StandardXMLConverter, **??**

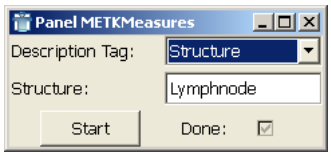
9.22 METKMeasures

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	This module computes the main axis and volume of a structure as exact as possible.

Usage

This module computes the main axis and volume of a structure as exact as possible. For computing the main axis as fast as possible, the segmentation mask is opened and converted into a point set. The main axis is computed using this point set.

You can specify a description tag the ObjMgr is searched for.



Send ObjMgr-Events

ObjMgr-Key	Value	Action
ObjID ⇒Measures ⇒ExactMeasures	TRUE	Is set to TRUE after computation
ObjID ⇒Measures ⇒BaryCenter	Vec3f	bary center
ObjID ⇒Measures ⇒MidPoint	Vec3f	midpoint
ObjID ⇒Measures ⇒xAxis	Vec3f	main x-axis
ObjID ⇒Measures ⇒yAxis	Vec3f	main y-axis
ObjID ⇒Measures ⇒zAxis	Vec3f	main z-axis
ObjID ⇒Measures ⇒xDiameter	Double	length of the x-axis
ObjID ⇒Measures ⇒yDiameter	Double	length of the y-axis
ObjID ⇒Measures ⇒zDiameter	Double	length of the z-axis
ObjID ⇒Measures ⇒maxDiameter	Double	largest main axis
ObjID ⇒Measures ⇒Volume	Double	volume in cmm
ObjID ⇒Measures ⇒VolumeUnit	cmm	volume unit

Fields

descriptionTag:	Description tag to search for
structure:	Value of the description tag
start:	Start computing
done:	Is true when computing is finished, false otherwise

Related modules

UMDSoMainAxis

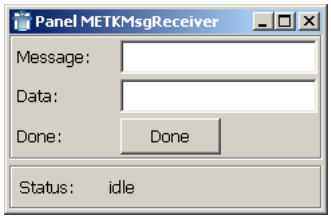
9.23 METKMsgReceiver

Genre:	METKMessaging
Authors:	Christian Tietjen
Short description:	Module to receive data from `METKMsgSender` for synchronized message handling

Usage

Module to receive data from `METKMsgSender`. There must be only one receiver listening to the same message! For more general information, have a look into Section 6.

On receiving a message, `METKMsgReceiver` automatically informs the sender about the reception of the message. After handling the message/data, handling may be confirmed by pressing "done". No further information can be sent back to the sender.



Known problems The behavior is of the messaging modules is undefined when there is more than one `METKMsgReceiver` for one message!

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`Communication ⇒StatusEvents ⇒Message`	Data channel	Must fit to the field **`message`**
`Communication ⇒StatusEvents ⇒Data`	Received data	Received data
`Communication ⇒StatusEvents ⇒Status`	initialize	Senders handling query

Send ObjMgr-Events

ObjMgr-Key	Value	Action
`Communication ⇒StatusEvents ⇒Status`	processing or done	Current handling information

Fields

`message:`	Defines the communication channel
`data:`	Received message
`done:`	Button to confirm handling (after handling is finished)
`status:`	handling status: **idle/done**: nothing to do **processing**: message is being processed

Related modules

`METKGlobalMessages`, `METKMsgSender`

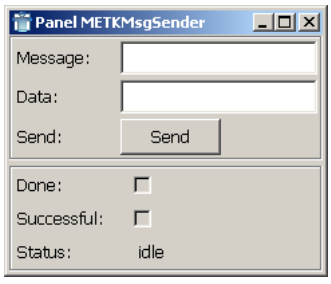
9.24 METKMsgSender

Genre:	METKMessaging
Authors:	Christian Tietjen
Short description:	Module to send data to `METKMsgReceiver` for synchronized message handling

Usage

Module to send data to `METKMsgReceiver`. There must be only one receiver listening to the same message! For more general information, have a look into Section 6.

If there is no responsible receiver for a message, the transmission is canceled. Otherwise, the message will be send. After processing, `METKMsgSender` will be informed whether the data was handled. No further information can be sent back to the sender.



Known problems The behavior is of the messaging modules is undefined when there is more than one `METKMsgReceiver` for one message!

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
textttCommunication ⇒StatusEvents ⇒Status	done	Senders handling query

Send ObjMgr-Events

ObjMgr-Key	Value	Action
`Communication ⇒StatusEvents ⇒Message`	Data channel	Same as the field **`message`**
`Communication ⇒StatusEvents ⇒Data`	Sent data	Data to send to the receiver
textttCommunication ⇒StatusEvents ⇒Status	initialize	Current handling information

Fields

`message:`	Defines the communication channel
`data:`	Message to be send
`send:`	Button to begin handling
`done:`	False if no confirmation was received. True on reception or when no receiver was found.
`successful:`	Only true when the message was handled successfully
`status:`	handling status: **idle/done**: nothing to do **processing**: message is being processed **Nobody's listening.**: no appropriate receiver was found

Related modules

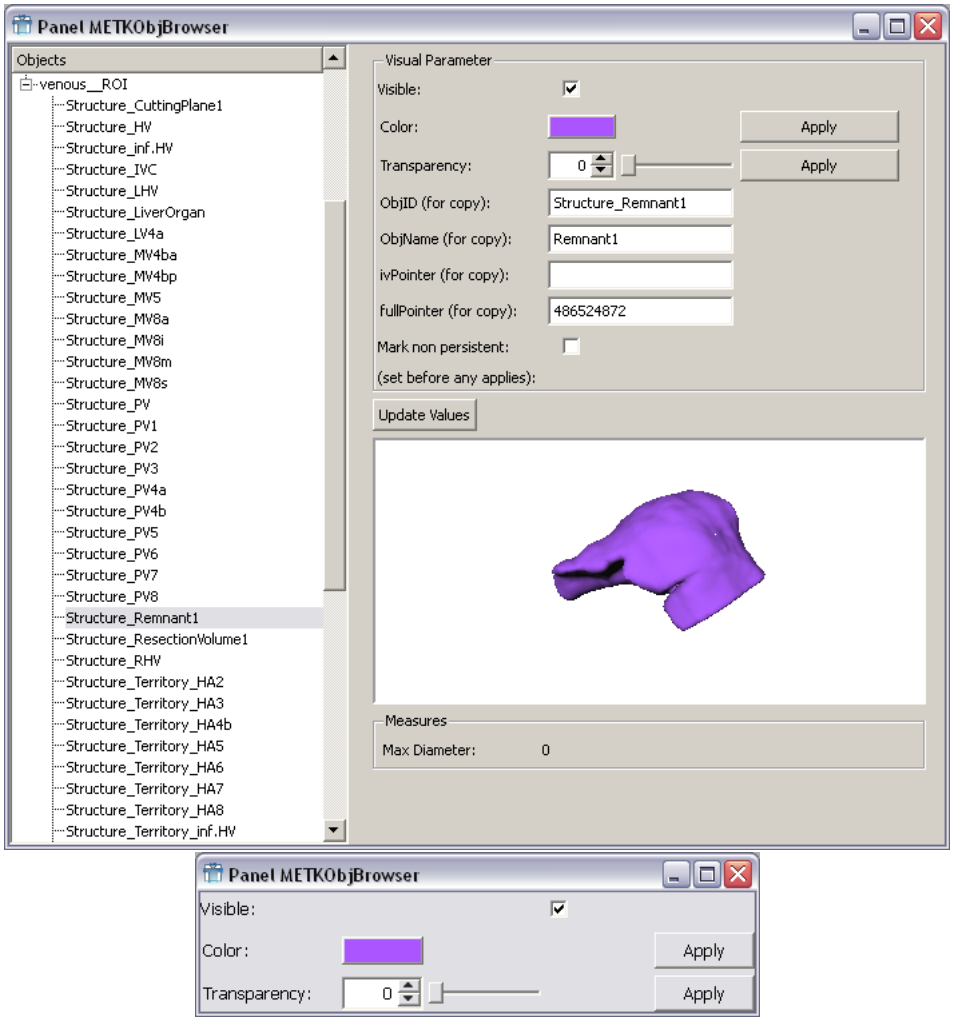
`METKGlobalMessages`, `METKMsgReceiver`

9.25 METKObjBrowser

Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	Provides a little GUI to browse objects and change their visual parameter

Usage

This module provides a little GUI to browse objects and change their visual parameter. Besides the main window it provides a window named `shortVisuals` with some important visual controls on it.



Fields

objID:	object id
updateList:	refresh the METKHierarchyBrowser
applyColor:	writes color into objMgr
applyTransp:	writes transparency into objMgr
updateObject:	refresh object values in user interface
visible:	visible value of object
visibleGUI:	some visible value for a nice gui workaround
color:	color value of object
transparency:	transparency value of object
objName:	name of object
ivPointer:	inventor pointer of plain object
fullPointer:	pointer to inventor object full colored etc.
markNonPersistent:	if this flag is set before any value is applied, the values are marked as non persistent
maxDiameter:	maximum diameter of object

Related modules

METKExplorer, METKHierarchyBrowser

9.26 METKOverlay2D

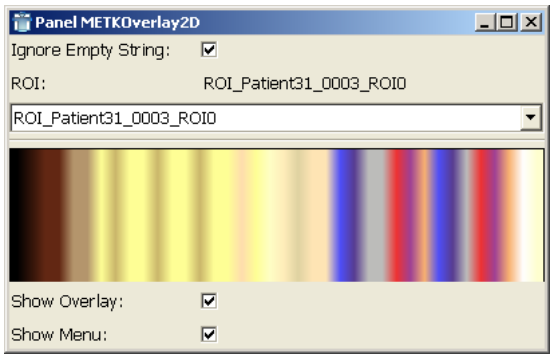
Genre:	METKImageDisplay
Authors:	Christian Tietjen
Short description:	This module generates an overlay displaying the visible segmented structures in a 2D viewer.

Usage

This module generates an overlay displaying the visible segmented structures in a 2D viewer. The input must be a METKCodedSegmentation. By default, a small button is displayed in the top left corner of the METKViewer2D to hide/show the overlay. This button and the overlay itself may be blend out by the settings. The panel also shows the generated LUT for the overlay. You can not interact with it, it is just for debugging purposes.

If the user selects a structure, the values of the currently selected object (CSO) are updated in the ObjMgr (selection and objectID).

METKOverlay2D listens to the Appearance Layer, strictly speaking to Visible, Color and Transparency.



Listen ObjMgr-Events

Listen to Appearance Values of all objects and change the overlay values that way.

Inputs and Outputs

Input:	image (MLImage)	Multi coded segmentation image loaded via METKCodedSegmentation
Output:	overlay (Inventor)	Overlay to show in METKViewer2D

Fields

ignoreEmptyString:	If ROI is set to [None], the module maintains the previous ROI as input. On NO, the output contains no data.
ROI:	List of the available ROIs in the case. In combination with METKCSO you can always choose the right one.
listenToRoi: (string)	Tag to synchronize ROI selection. All modules with the same tag in listenToRoi updating their rois if one of them change its selection.
showOverlay:	Toggles the display of the overlay.
showMenu:	Toggles the display of the menu button in the connected METKViewer2D.

Related modules

METKCodedSegmentation, METKViewer2D, METKCSO

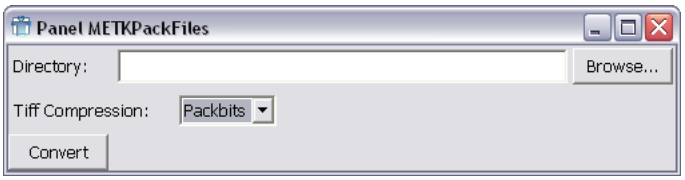
9.27 METKPackFiles

Genre:	METKOpen
Authors:	Konrad Mühler
Short description:	Compress all segmentation masks in folder.

Usage

This standalone application compresses all segmentation mask in a folder with the given compression level.

Packed files are loaded slower but need less disk space. For ROIs it is not recommended to compress them because the loading time will be explode. For single segmentation mask it is ok, because they will be only used once, when the inventor files are generated.



Fields

directory:	Directory for tiff files
convert:	Starts conversion process

Related modules

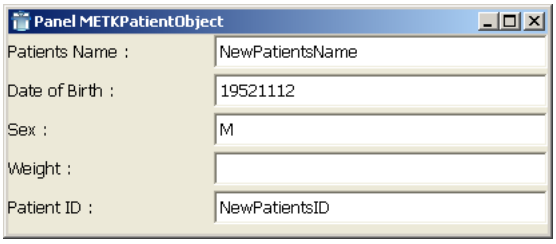
METKCaseOptimizer

9.28 METKPatientObject

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Simple module for setting and getting patient info data in an METK case.

Usage

Simple module for setting and getting patient info data in an METK case.



Listen and send ObjMgr-Events

ObjMgr-Key	Value	Action
`PatientObject ⇒Patient ⇒*`		Listen to changes
`PatientObject ⇒Patient ⇒*`		Write changes

Fields

11cm **`name:`**	Name of the patient
`DOB:`	Day of birth
`PID:`	Patients identification number
`weight:`	One of the patients top secrets
`sex:`	Male or female

Related modules

`METKPatientObject`

9.29 METKPicking

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Enables picking of objects in a 3D scene. Is already included in METKViewer3D.

Usage

This module is already included in METKViewer3D, but it may also be used separately.

It enables picking of objects in a 3D scene. The main problem of the most scenes is the existence of transparent objects. Thus, in the most cases, you may want to pick objects located behind other transparent objects. Also, the desired object itself may be transparent, too. Another requirement is, that toggling between Open Inventor's "'viewing"' and "'picking"' modi should be unnecessary. For that purpose, the module METKPicking is provided. In use with the SoCustomExaminerViewer, this viewer can be set up to send the viewing events to the scene, too. METKPicking grabs these events and acquires the selected objects dependent on some parameters.

Only objects which are registered by the METK can be selected. The registration is needed to provide information about importance (transparency). It is a bit hard to find out the transparency of an object in Open Inventor during a rendering pass. The METK is also used to check the ignorePickable flag. This flag can easily be set by the METKCSO module. For setting this parameter by hand, have a look into the METKCSO documentation.

Developer Information

Known problems The selection coordinate is not always the assumed one, especially when a transparent object is selected and several consecutive triangles were hit.

Inputs and Outputs

Output:	outPickedObject (Inventor)	Returns an Open Inventor pointer to the picked object.
Output:	outScene (Inventor)	Connection to the viewer containing the whole scene. This field is used to grab the mouse events and is an input field only.

Fields

scrapLight:	When the desired object is located behind some other transparent objects, the object must be visible with scrapLight percent. Or, in other words, scraplight defines the early ray termination known from volume rendering or ray tracing.
pickPixelTolerance:	To distinguish between rotating a scene and picking into a scene, pickPixelTolerance defines how many pixels the mouse press and mouse release events may differ.
importanceWeighting:	You can define `LAY_DESCRIPTION` / `INF_IMPORTANCE`, otherwise, the inverse of `INF_TRANSPARENCY` is used. More important objects are preferred when selecting, weighted by this parameter.
boundingBoxSizeWeighting:	The other weighting parameter is the size of the bounding box of an object. It is assumed that the user wants to select a smaller object when he take care about the placement of the mouse over a very small object laying behind a large one.
objName:	Returns the name of the selected object.
selection:	Returns the coordinate of the selection.

Related modules

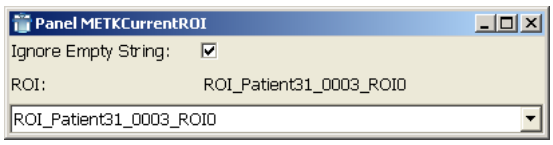
`METKViewer3D`, `METKCSO`

9.30 METKROISelect

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Selects one of the available ROIs

Usage

If a segmented structure is selected, you may want to use the corresponding ROI. In combination with METKCSO you can always choose the right one. The module checks whether there is a ROI in the current case. If so, the first one is loaded.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID ⇒Image ⇒ImageType`	Original	
`ObjID ⇒Global ⇒ObjectType`	Result	If both fields have these entries, it is identified as a ROI
`Communication ⇒ROIS ⇒roiTag`	selected Roi as string	Can be changed by all modules having this tag as `listenToRoi`

Fields

`ignoreEmptyString` `(bool)`	(default yes) If ROI is set to [None], the module maintains the previous ROI as input. On NO, the output contains no data.
`rois:` `(string)`	List of the available ROIs in the case.
`roiSelect:` `(string)`	Currently selected ROI.
`refresh:` `(trigger)`	Reload list of available ROIs.
`listenToRoi:` `(string)`	Tag to synchronize ROI selection. All modules with the same tag in `listenToRoi` updating their rois if one of them change its selection.

Related modules

`METKOverlay2D`, `METKCodedSegmentation`, `METKViewer2D`

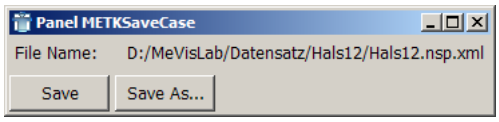
9.31 METKSaveCase

Genre:	METKOpen
Authors:	Konrad Mühler, Christian Tietjen
Short description:	Saves a case to a XML file

Usage

You should use the functionality of `METKManager` instead!

Using this module you can save a loaded case either under the given filename of the case (`Save`) or a new filename (`Save As`).



Developer Information

Known problems The saved XML file must be in the same directory as the original XML file.

Fields

`fileName:`	Filename of case including path. This name is automatically set to filename of a case on loaded event.
`save:`	Trigger to save the case with the given filename.
`saveAs:`	Save the case after the user specified a filename via dialog.

Related modules

`METKManager`

9.32 METKScriptBuilder

Genre:	METKAnimation
Authors:	Konrad Mühler
Short description:	Builds animation scripts from different ObjMgr states

Usage

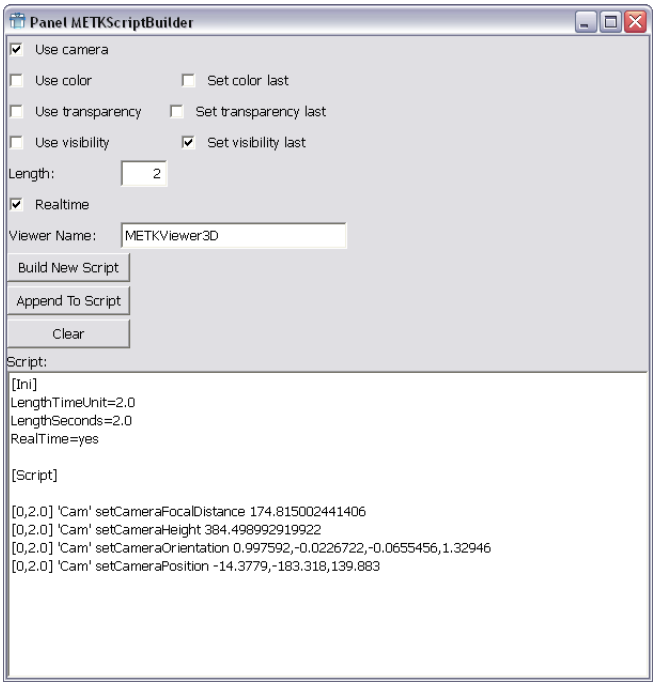
This module is included in the `METKCollections` but can also be used separately.

Using the `buildNewScript` field a animation script is generated that changes the scene from any state to the current state of the ObjMgr the `METKScriptBuilder` is connected to. In `METKCollections` an extra ObjMgr is used to load different scene states that can be used to generate a script.

Using the field `appendToScript` the current state of the ObjMgr is added as scene change at the end of the script. So, loading different scene state in the ObjMgr (i.e. different collections) scripts can be generated that animate these states one after the other.

The length of each transition is set in the `length` field.

The different "'use"' and "'set"' fields can be used to define, what parameter should be animated.



Developer Information

Ask Konrad for more information.

Fields

buildNewScript:	Create a new script of the length given by the `length` field
appendToScript:	Append ObjMGr state to the current script
useCamera:	Animate the camera parameter like position over time
useColor:	Animate colors of objects
useTransparency:	Animate transparency of objects
useVisibility:	Animate the changes of visibility of objects using the transparancy parameter (soft fading)
setColorLast:	Set the color of objects in the last frame ad hoc and do not interpolate the value
setTransparencyLast:	Set the transparency value of objects in the last frame ad hoc and do not interpolate the value
setVisibilityLast:	Set the visibility of objects in the last frame and do not use the transparency parameter to fade
length:	Length in seconds for the current script section to generate
realtime:	TRUE, if the animation should be executed in realtime in the viewer and FALSE if a video should be generated
viewerName:	Name of the viewer the camera parameters are taken from
script:	Generated script, that can be used to send it to the UMDAnimation2 module via the METKManager
clear:	Clear the script

Related modules

UMDAnimation2, METKCollections

9.33 METKScriptBunch

Genre:	METKAnimation
Authors:	Konrad Mühler
Short description:	Provides links to some standard animation scripts

Usage

The module provides some scripts for ENT and Liver cases. The scripts can be selected from a list (and only from a list, not via module fields!). The script file will be send via the METKManager to the UMDAnimation2 where it is executed.

Developer Information

To extend the list of scripts, create script file in the ENT or Liver folder of the Animation project's script folder and add them to the list in the `METKScriptBunch.script` file.

Send ObjMgr-Events

ObjMgr-Key	Value	Action
`OBJ_ANIMATION` ⇒`LAY_ANIMATION_SCRIPT` ⇒`INF_SCRIPT_SCRIPTFILE`	script file	Communicates a script file to the UMDAnimation2 module

Fields

`baseScriptPath:` `(string)`	Path relative to the animation projects path (default: Scripts/)
`ENTScripts:` `(string)`	List of scripts for ENT cases
`LiverScripts:` `(string)`	List of scripts for Liver cases

Related modules

`UMDAnimation2`

9.34 METKSilhouette

Genre:	METKRenderModes
Authors:	Christian Tietjen
Short description:	Loads the silhouettes of the segmented structures.

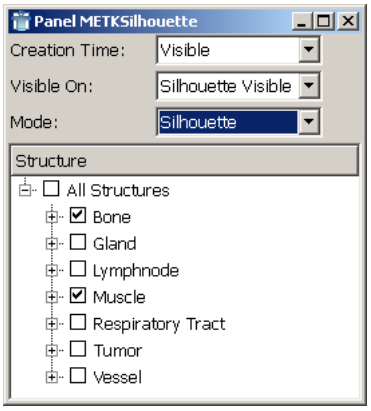
Usage

Loads the silhouettes of the segmented structures. The silhouette of a structure is black and two pixels wide by default. The silhouettes are created when the Loaded event was sent. It is also possible to load the silhouettes when they are treated as visible.

All structures can be switched to (in)visible with the internal `METKStructureGroupBrowser`. The same behavior is reached by setting `ObjectID ⇒Appearance ⇒Visible` and `Appearance ⇒Silhouette`.

The width of a silhouette may be set by `ObjectID ⇒Appearance ⇒SilhouetteWidth`.

The color of a silhouette may be set by `ObjectID ⇒Appearance ⇒SilhouetteColor`.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`* ⇒Appearance ⇒Visible or Silhouette`		Create a **??** if not exist

Inputs and Outputs

Output:	inventorScene (Inventor)	Visible objects with silhouette

Fields

creationTime:	**Load:** creation after Loaded event. Takes some time at the beginning. **Visible:** creation on general visible. Takes some time while exploring the data.
visibleOn:	Silhouette is turned on at **SilhouetteVisible** (Silhouette) or **GeneralVisible** (Visible)
mode:	Two modules for line rendering are available in MeVisLab: SoSilhouette and SoEdgeEmphasize (Verweis auf Module!)

Related modules

METKStippling, METKTexturing, **??**

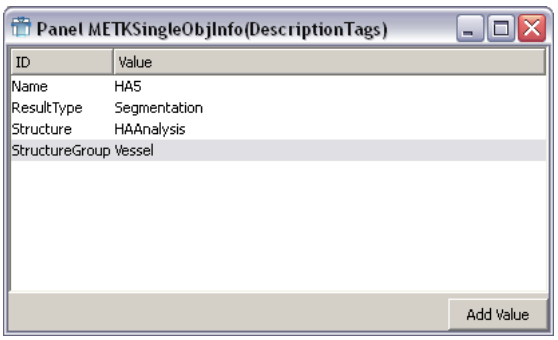
9.35 METKSingleObjInfo

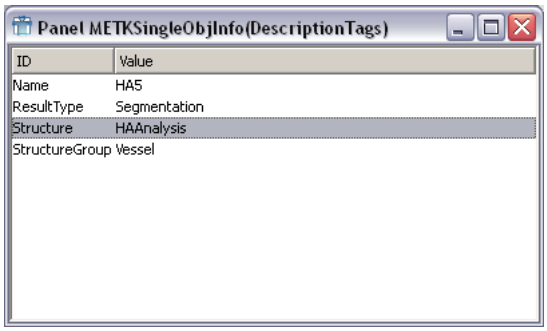
Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	This module provides several windows to view and edit the data of single objects or layers.

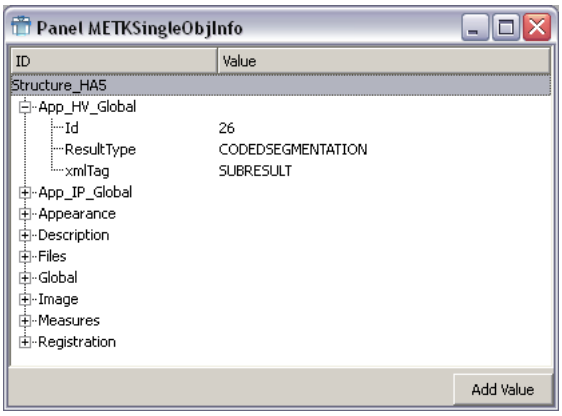
Usage

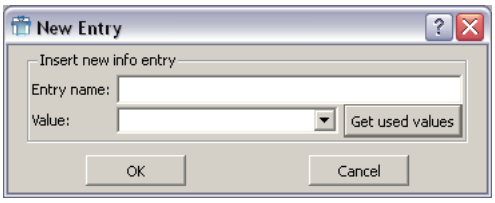
This module provides several windows to view and edit the data of single objects or layers. Depending on the chosen window, values can be added or edited. The user/developer needs to be set the object name and the layer name. If the layer name is empty, all layers of the object are shown in a tree view. In the main window an "'Add Value"' button is visible. Clicking this button new values can be added in the current layer. Double clicking an existing entry open a window, where the value can be changed.

If a new value is added, a window appears. If the entry name (=info layer) was typed, via "'Get used values"' all values can be obtained, that are used for this entry in other objects. This helps to save time and prevents typing errors.









Send ObjMgr-Events

Creates new info entries given by the user.

Fields

objID:	ID of object, whose values are shown
layerID:	ID of layer. If its empty, a tree view is shown with all layers and its infos.
update:	Refresh the view
newEntry:	Opens a window to add new entry
comboNewEntryEnum:	Values obtained for new info entries used by other objects having this info entry.

ToDo List

- In the tree view no values can be added or edited.

Related modules

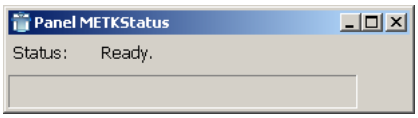
METKExplorer

9.36 METKStatus

Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Prompts the current status of the METK.

Usage

Prompts the current status of the METK. For example, the METKManager sends it current status to this module. The status is sent via the ObjMgr. After one second of idle time (where neither the status message nor the progress value is updated), the module is set back to "'Ready."'



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
OBJ_COMMUNICATION ⇒LAY_STATUS ⇒INF_STATUSMSG		sets the status message
OBJ_COMMUNICATION ⇒LAY_STATUS ⇒INF_PROGRESS		sets the progress of a process (must be between 0..1)

Fields

progress:	Current progress value, received via ObjMgr
statusMsg:	Current status message received from ObjMgr

Related modules

METKInfoWin

9.37 METKStippling

Genre:	METKRenderModes
Authors:	Alexandra Baer, Christian Tietjen
Short description:	Loads the stipple rendering of the segmented structures

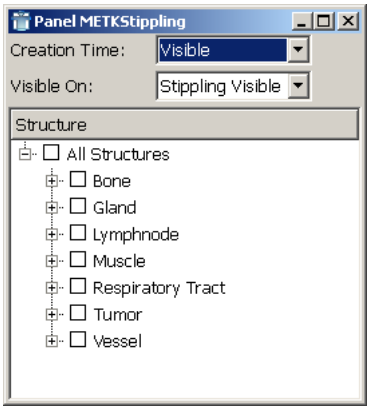
Usage

Loads the stipple rendering of the segmented structures. The stipple rendering of a structure is black a by default. The stipple rendering is applied when the Loaded event was sent. It is also possible to load the stipple rendering when it is treated as visible.

All structures can be switched to (in)visible with the internal METKStructureGroupBrowser. The same behavior is reached by setting objectID ⇒LAY_APPEARANCE ⇒INF_VISIBLE and LAY_APPEARANCE ⇒INF_STIPPLING.

The density of a stipples may be set by objectID ⇒LAY_APPEARANCE ⇒INF_STIPPLINGGAMMA.

The color of a stipples may be set by objectID ⇒LAY_APPEARANCE ⇒INF_STIPPLINGCOLOR.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
* ⇒Appearance ⇒Visible or Stippling		Create a **??** if not exist

Inputs and Outputs

Output:	inventorScene (Inventor)	Visible objects with stippling

Fields

creationTime:	**Load:** creation after Loaded event. Takes some time at the beginning. **Visible:** creation on general visible. Takes some time while exploring the data.
visibleOn:	silhouette is turned on at **StipplingVisible** (INF_STIPPLING) or **GeneralVisible** (INF_VISIBLE)

Related modules

METKSilhouette, METKTexturing

9.38 METKStructureDetails

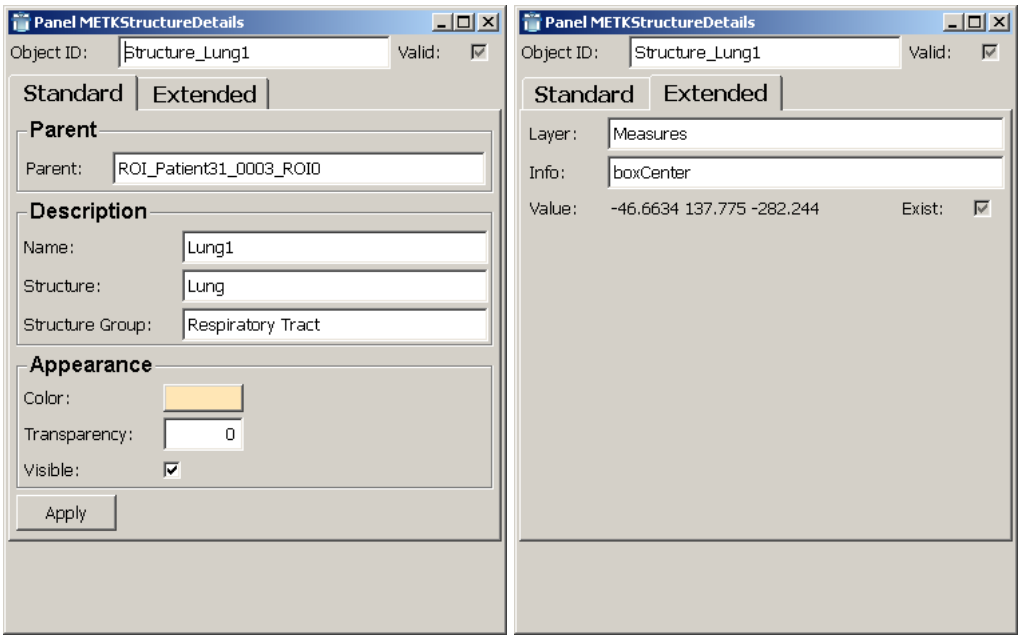
Genre:	METKMisc
Authors:	Christian Tietjen
Short description:	Displays the details of an existing structure.

Usage

Displays the details of an existing structure. A valid structure has to be two entries in the ObjMgr:

1. ObjectID ⇒LAY_GLOBAL ⇒INF_OBJTYPE == "'Result"'
2. ObjectID ⇒LAY_IMAGE ⇒INF_IMAGETYPE == "'Segmentation"'

The standard settings of a structure can be changed in the standard TabView. Changes are must be applied explicitly. In the extended TabView, other entries can be read out.



Fields

The names of the boxes and the fields correspond to the LAYER and INFO tags in the Obj-Mgr.

Related modules

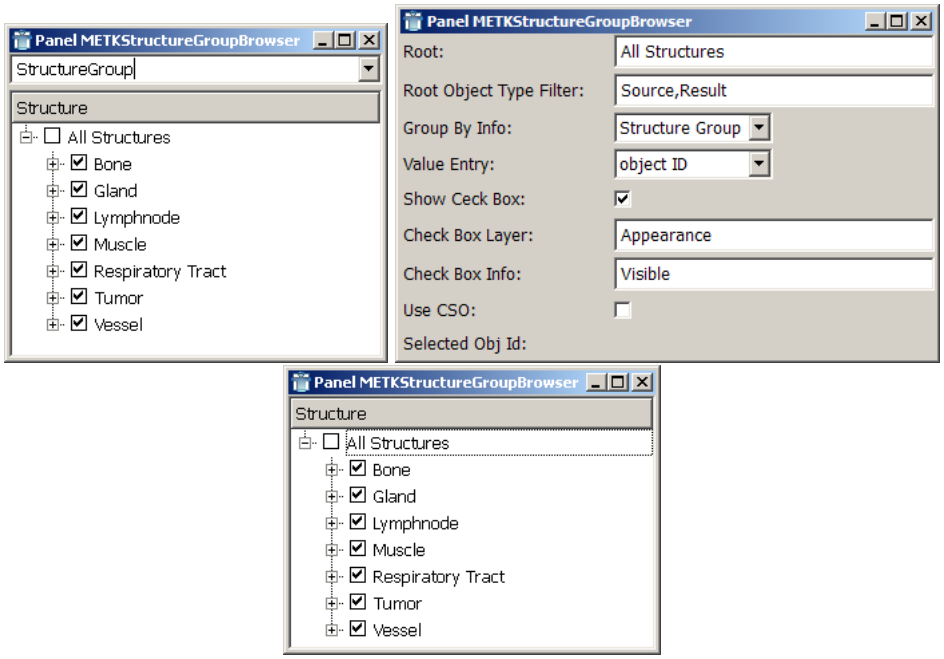
METKObjBrowser

9.39 METKStructureGroupBrowser

Genre:	METKGUI
Authors:	Christian Tietjen, Konrad Mühler
Short description:	This module enables the user to browse through the case data by using the entries ⇒Structure and ⇒StructureGroup.

Usage

This module enables the user to browse through the case data by using the entries Structure and StructureGroup, which are provided by every segmented structure in a METK case. With the check boxes, it is possible to toggle the flag objectID ⇒Appearance ⇒Visible by default. But it is possible to toggle every other bool flag of a structure. It is possible to turn off the check boxes. In that case, no entries will be modified, except the CSO if you want.



Known problems The METKStructureGroupBrowser uses the standard ListView from the MeVisLab MDL. The checkboxes of the ListView have only two states: on or off. Thus, if one item has several subitems with different states, there is no possibility to indicate the subordinate state properly.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`CSO ⇒SelectedObject ⇒ObjectID`	*	Check for changes
`* ⇒Selected Layer ⇒Selected Info`	*	Check for changes

Send ObjMgr-Events

ObjMgr-Key	Value		Action
CSO ⇒SelectedObject ⇒ObjectID	TRUE FALSE	or	only when useCSO is true

Fields

root:	String used for the root list item in the list view
rootObjectTypeFilter:	discards all objectIDs who do not match with their objectType
valueEntry:	the list view can display the objectID or the name (objectID ⇒Description ⇒Name) of the structures
showCheckBox:	decide whether the check box is shown
checkBoxLayer:	layer for check box toggling
checkBoxInfo:	info for check box toggling
useCSO:	if true, the selected structure is applied as CSO
checkedObjIds:	object IDs of all checked items, devided by comma and with a leading comma (important!)
writeToObjMgr:	if true (standard), the checked values are written to ObjMgrin the given layer and info of the objects
setValuesBy CheckedObjIdString:	if true (false is standard), the checkedObjIds string is used the set the checked state of all items. Only object IDs in checkedObjIds will be checked. If it false, the values of given layer and info of each object are used.

Related modules

See also METKHierarchyBrowser

This module is included by METKIsoSurface, for example.

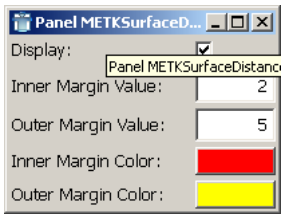
9.40 METKSurfaceDistance2D

Genre:	METKSurfaceDistance
Authors:	Christian Tietjen
Short description:	This module displays two isobars of a distance transformation. Color and value may be specified.

Usage

This module displays two isobars of a distance transformation. Color and value may be specified.

This module is not really an METK module at the moment, but an extension is planned to select single structures.



Inputs and Outputs

Input:	mask (MLImage)
Output:	Margins (Inventor)

Fields

`display:`	toggle the display of the isobars
`innerMarginValue:`	the density of the inner isobar in mm
`outerMarginValue:`	the density of the outer isobar in mm
`innerMarginColor:`	the color of the inner isobar
`outerMarginColor:`	the color of the outer isobar

Related modules

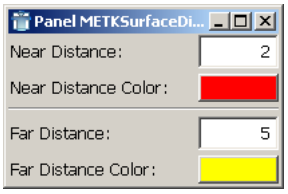
`METKDistanceTransform`, `METKSurfaceDistance3D`

9.41 METKSurfaceDistance3D

Genre:	METKSurfaceDistance
Authors:	Christian Tietjen
Short description:	This module displays two regions of a distance transformation on the segmented surfaces.

Usage

This module displays two regions of a distance transformation on the segmented surfaces. Color and value may be specified. The regions are displaced a bit to avoid unwanted occlusions with the original surface. To turn on the region for a single structure, the value `objectID ⇒Appearance ⇒SurfaceDistance` has to be set to `true`.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`ObjID ⇒Appearance ⇒Visible`	TRUE	
`ObjID ⇒Appearance ⇒SurfaceDistance`	TRUE	Both values must be true for displaying

Inputs and Outputs

Input:	input0 (MLImage)
Output:	outInventor (Inventor)

Fields

`nearDistance:`	the density of the inner region in mm
`nearDistanceColor:`	the color of the inner region
`farDistance:`	the density of the outer region in mm
`farDistanceColor:`	the color of the outer region

Related modules

`METKDistanceTransform`, `METKSurfaceDistance2D`

9.42 METKTexturing

Genre:	METKRenderModes
Authors:	Rocco Gasteiger, Christian Tietjen
Short description:	Loads textures coordinates for segmented structures.

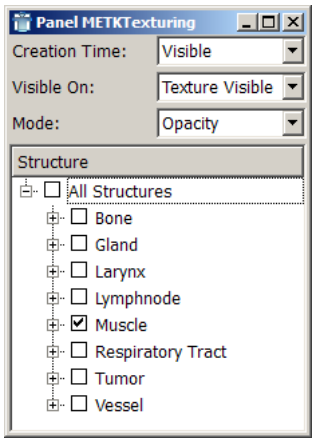
Usage

Loads textures coordinates for segmented structures. Opacity mapping is enabled by default. The textures are applied when the Loaded event was sent. It is also possible to apply the textures when they are treated as visible.

All structures can be switched to (in)visible with the internal `METKStructureGroupBrowser`. The same behavior is reached by setting `ObjectID ⇒Appearance ⇒Visible` and `Appearance ⇒Texturize`.

The texture type (opacity or TAM) may be set by `ObjectID ⇒Appearance ⇒TextureType`.

The scale factor of a texture may be set by `ObjectID ⇒Appearance ⇒TextureScale`.



Listen ObjMgr-Events

ObjMgr-Key	Value	Action
`* ⇒Appearance ⇒Visible or Texturize`		Create a **??** if not exist
`* ⇒Appearance ⇒TextureType`	HATCHING_OPACITY or HATCHING_TAM	Choose texture type
`* ⇒Appearance ⇒TextureScale`		Scale factor of the texture

Inputs and Outputs

Output:	inventorScene (Inventor)	Visible objects with texture

Fields

creationTime:	**Load:** creation after Loaded event. Takes some time at the beginning. **Visible:** creation on general visible. Takes some time while exploring the data.
visibleOn:	Texturing is turned on at **SilhouetteVisible** (Texturize) or **GeneralVisible** (Visible)
mode:	Two texture types are available at the moment: HATCHING_OPACITY and HATCHING_TAM

Related modules

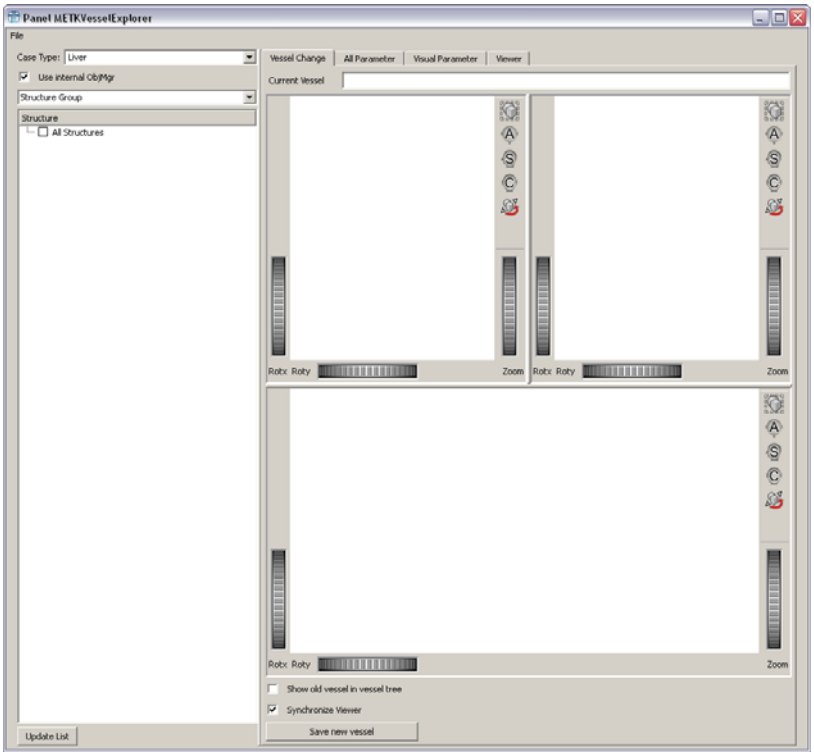
METKStippling, METKSilhouette, **??**

9.43 METKVesselExplorer

Genre:	METKGUI
Authors:	Konrad Mühler
Short description:	Standalone application to convert single vessel parts to convolution surfaces

Usage

This standalone was build to convert single vessel branches to convolution surfaces. I've no idea, how to use it or if it ever really worked fine. Sorry.



Developer Information

Known problems I don't know, how does it work???

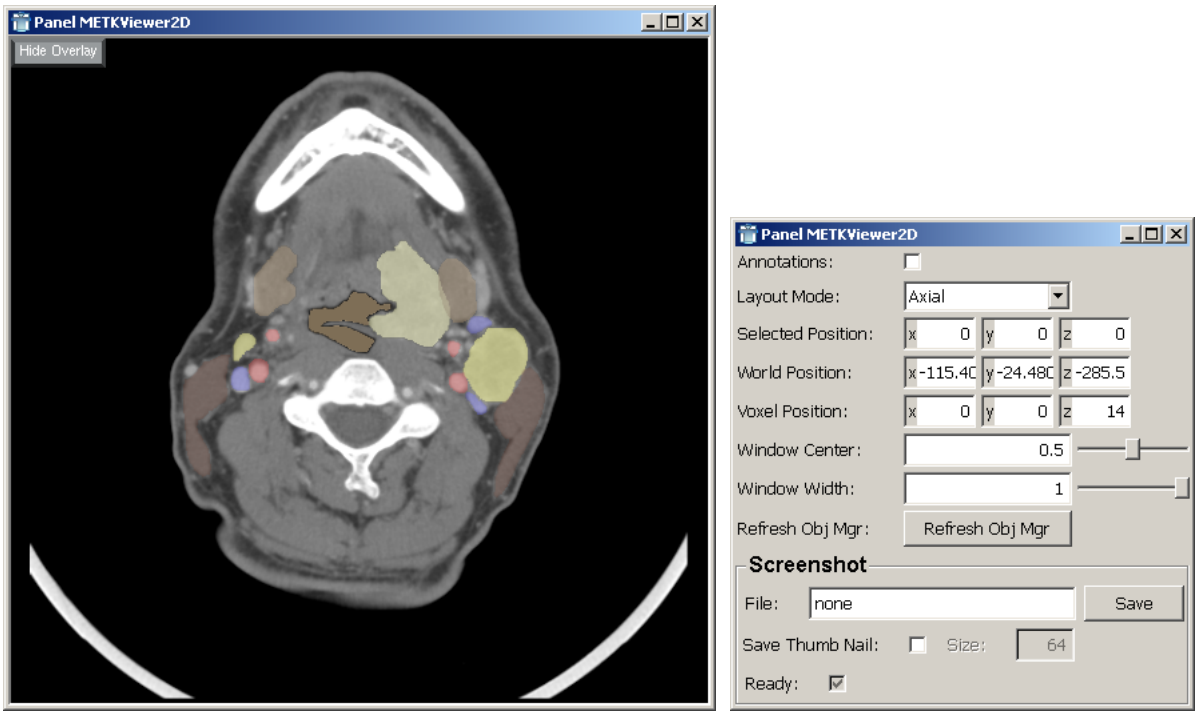
9.44 METKViewer2D

Genre:	METKViewer
Authors:	Christian Tietjen, Konrad Mühler
Short description:	The METKViewer2D simply displays a ML images, but it provides some additional features.

Usage

The METKViewer2D simply displays a ML images, but it provides some additional features. When retrieving the loaded event, the viewer sets up the stored viewing parameters. This is also done when a Collection is loaded. When there is no viewing information stored in the case, the viewer adds some own, non-persistent entries in the ObjMgr. The name of the `METKViewer2D` module is used as `objectID` in the ObjMgr. Thus, the information will not be saved, because it is possible to load such a case in different applications with different

names. The METKCollections in contrast stores the information, but checks for availability of a viewer name on loading a collection.



Developer Information

If you add a menu button (SoView2DMenuItem) in your connected module, it will be displayed automatically. METKOverlay2D uses the functionality.

Listen ObjMgr-Events

ObjMgr-Key	**Value**	**Action**
`objectID ⇒Global ⇒Refresh`	TRUE	If set to `true`, the current viewing parameters are stored in the ObjMgr by the viewer. Afterwards, the flag is again set to `false`.

Bidirectional ObjMgr-Connections

ObjMgr-Key	Action
`objectID ⇒Properties ⇒Annotations`	corresponds to the annotation field
`objectID ⇒Properties ⇒LayoutMode`	corresponds to the layoutMode field
`objectID ⇒Properties ⇒WindowCenter`	corresponds to the windowCenter field
`objectID ⇒Properties ⇒WindowWidth`	corresponds to the windowWidth field
`objectID ⇒Properties ⇒WorldPosition`	corresponds to the worldPosition field
`objectID ⇒Screenshot ⇒ScreenshotFilename`	Path/name of the screenshot file
`objectID ⇒Properties ⇒ScreenshotThumb`	generate thumbnail from screenshot
`objectID ⇒Properties ⇒ScreenshotFlag`	`True` = make screenshot `False` = Done

Inputs and Outputs

Input:	intput (Inventor)	Overlay
Input:	soview2d (MLImage)	ROI
Output:	view2d	

Fields

annotations:	Display the standard DICOM annotations or not
layoutMode:	Switches to the layout modes provided by `SoOrthoView2D`
selectedPosition:	The last selected position (with left mouse button only) in world coordinates
worldPosition:	The current world position. Differs from `selectedPosition` when you scroll through the data without selecting a voxel.
voxelPosition:	The `worldPosition` in voxel coordinates.
windowCenter/windowWidth:	Specifies the center and width of the intensity window.
refreshObjMgr:	The current viewing parameters are stored in the ObjMgr by the viewer.
Screenshot panel:	Saves a screenshot of the current content. This can also be realized by the ObjMgr (used by METKCollections).

Related modules

`METKCollections`, `METKOverlay2D`, `SoOrthoView2D`

9.45 METKViewer3D

9.46 UMDAnimation2

10 Local macros (for developers only)

These modules are not published in the module database of MeVisLab. They are local macros, that are used in other METK macro modules. They are not for daily use so we comment them only for developers.

10.1 ConvertXML

Genre:	Local/Developer
Authors:	Christian Tietjen
Short description:	This module converts XML files to the METK format

Location

Used in METKManager.

Developer Information

This module contains a HepaXMLConverter, a NeckVisionXMLConverter and a StandardXMLConverter. It force these three modules to convert a xml file to load and only one or none of them convert the file. Depending on the module, that converted the file, the `detectedFormat` field is set.

Fields

`convertFile:` `(trigger)`	Starts conversion
`detectedFormat:` `(string)`	a string that shows, which converter was used
`successful:` `(bool)`	indicates, if the convertion process was successful

Related modules

`StandardXMLConverter`, `HepaXMLConverter`, `NeckVisionXMLConverter`

10.2 HepaXMLConverter

Genre:	Local/Developer
Authors:	Konrad Mühler, Christian Tietjen
Short description:	This module converts XML files from HepaVision to the METK format

Location

Used in ConvertXML in METKManager.

Developer Information

Important: If there is no ObjectValue in the Image-Layer of an object, the TraceColor-Code is taken, if it exists!

What's a HepaVision case?:

- `PatientObject⇒Description⇒Description == "'MeVisLiverCase"'`
- or `PatientObject⇒Description⇒Description == "'HepaVision Import"'`
- or `PatientObject⇒Description⇒Description == "'Liver Analysis"'`

Fields

`convert:` `(trigger)`	Starts conversion
`ready:` `(bool)`	TRUE if ready
`readableFormat:` `(bool)`	TRUE if format is HepaVision

Related modules

`ConvertXML`, `NeckVisionXMLConverter`, `StandardXMLConverter`

10.3 NeckVisionXMLConverter

Genre:	Local/Developer
Authors:	Christian Tietjen, Konrad Mühler
Short description:	This module converts XML files from NeckVision (old version) to the METK format

Location

Used in ConvertXML in METKManager.

Developer Information

What's a NeckVision case?:

- `Global⇒NeckVision⇒Version == "'1.0"'`

Fields

`convert:`	Starts conversion
`ready:`	TRUE if ready
`readableFormat:`	TRUE if format is NeckVision

Related modules

`ConvertXML`, `HepaXMLConverter`, `StandardXMLConverter`

10.4 StandardXMLConverter

Genre:	Local/Developer
Authors:	Christian Tietjen, Konrad Mühler
Short description:	This module converts XML files from METK to the METK format

Location

Used in ConvertXML in METKManager.

Developer Information

What's a METK case?:

- `Application⇒Application⇒ApplicationName == "'MedicalExplorationTool`
- or `Application⇒Application⇒ApplicationName == "'NeckSegmenter"'`
- and `Application⇒Application⇒ApplicationVersion == "'1.0"'`

Fields

convert:	Starts conversion
ready:	TRUE if ready
readableFormat:	TRUE if format is METK

Related modules

ConvertXML, HepaXMLConverter, NeckVisionXMLConverter

10.5 METKInventorObject

Genre:	METKMisc
Authors:	Christian Tietjen, Konrad Mühler
Short description:	Internal object container of METKManager

Usage

This module holds the Open Inventor file of a segmented structure, stores the InventorPointer of it in the ObjMgr and calculates the bounding box.

Listen ObjMgr-Events

ObjMgr-Key	Value	Action
own ObjectID ⇒Files ⇒InventorFile	Filename	Listen to changes

Send ObjMgr-Events

ObjMgr-Key	Value	Action
own ObjectID ⇒Files ⇒InventorPointer	Pointer	Pointer to Open Inventor file

Fields

objectID:	Own ObjectID
initialize:	Trigger to load the file and set up the internal ObjInfo
filename:	File name of the Open Inventor file
measuresValid:	Bounding box is computed (or not)

Related modules

METKManager, **??**

A ObjMgr Entries (for developers only)

In the python source code, the use of constants instead of the final string is recommended. Firstly, conflicts in nomenclature could be avoided, by simply replacing the string. Secondly, spelling mistakes will be detected at compiling time, because the compiler needs a valid defined constant, which is substituted by the correct string.

A.1 Fixed ObjectIDs

Names of fixed ObjectIDs are reserved. All entries are used and generated by the corresponding modules.

Constant	Substituted String
OBJ_ANIMATION	Animation
OBJ_APPLICATION	ApplicationObject
OBJ_CASE	CaseObject
OBJ_COLLECTIONS	Collections
OBJ_COMMUNICATION	Communication
OBJ_CSO	CSO
OBJ_EVENTS	Events
OBJ_PATIENT	PatientObject

A.2 All Layers and Infos for Fixed ObjectIDs

A.2.1 Animated Rendering – OBJ_ANIMATION

These entries are used by `UMDAnimation2` `METKCollections` **?? ??**

Constant	Substituted String	Type
LAY_ANIMATION_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String TODO!!!
INF_ANIMATION_RESET	Reset	
LAY_ANIMATION_SCRIPT	Script	
INF_SCRIPT_FULLSCRIPT	FullScript	
INF_SCRIPT_STATUS	Status	
INF_SCRIPT_SCRIPTFILE	ScriptFile	
INF_SCRIPT_OBJNAMESPATH	ObjNamesPath	

A.2.2 Application Version Information – OBJ_APPLICATION

General information about the application generated the opened XML file. These information is read out by the METKManager at the beginning of loading a case to select the right converter.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Application
LAY_APPLICATION	Application	
INF_APPLICATIONNAME	ApplicationName	String
INF_APPLICATIONVERSION	ApplicationVersion	String Version Number
INF_MEVISLAB	MeVisLab	String Version Number

A.2.3 Setting Bookmarks in 3D – OBJ_BOOKMARKS

OBJ_BOOKMARKS = Bookmarks LAY_GLOBAL = Global INF_OBJTYPE = ObjectType String TODO!!!

INF_COUNT3D = Count3D INF_CAMPOSITION = CamPosition INF_CAMORIENTATION = CamOrientation INF_CAMHEIGHT = CamHeight INF_CAMFOCALDISTANCE = CamFocalDistance

A.2.4 General Case Information – OBJ_CASE

General information about the case, used by METKCaseObject.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Case
LAY_CASE	Case	String
INF_COMMENT	Comment	String
INF_FINDING	Finding	String
INF_ORDER	Order	String
INF_CASEDIR	Directory	String must be nonpersistent!!!
INF_XMLFILE	XMLFile	String must be nonpersistent!!!

A.2.5 Clipping in 3D – OBJ_CLIPPING

LAY_GLOBAL = Global INF_OBJTYPE = ObjectType String TODO!!!

OBJ_CLIPPING = Clipping INF_CLIPPING_SCALE = Scale LAY_CLIPPING_LEFT = PlaneLeft LAY_CLIPPING_RIGHT = PlaneRight LAY_CLIPPING_TOP = PlaneTop LAY_CLIPPING_BOTTOM

= PlaneBottom LAY_CLIPPING_REAR = PlaneRear LAY_CLIPPING_FRONT = PlaneFront INF_CLIPPING_TRANSLATION = Translation INF_CLIPPING_ROTATION = Rotation INF_CLIPPING_ON = ClippingOn

A.2.6 Coded Segmentation Handling – OBJ_CODEDSEGMENTATION

OBJ_CODEDSEGMENTATION = CodedSegmentation LAY_GLOBAL = Global INF_OBJTYPE = ObjectType String not set

A.2.7 Collection Handling – OBJ_COLLECTIONS

In contrast to all other fixed ObjectIDs, OBJ_COLLECTIONS has dynamic LayerIDs. The layers are added when a new collection is added. Used by `METKCollections`.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	not set
INF_FILENAME	Filename	String
INF_COMMENT	Comment	String
INF_THUMBNAME	ThumbFilename	String
INF_SCREENSHOT	Screenshot	String
INF_TITLE	Title	String

A.2.8 Message Exchange – OBJ_COMMUNICATION

This Object handles the communication. Used by `METKMsgSender`, `METKMsgReceiver`, `METKGlobalMessages` and `METKStatus`.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	not set
LAY_STATUSEVENTS	StatusEvents	
INF_MESSAGE	Message	Message
INF_DATA	Data	String
INF_STATUS	Status	Message
LAY_GLOBALEVENTS	GlobalEvents	
INF_CASELOADED	CaseLoaded	Message Loaded Cleanup
INF_GETALLVIEWER	GetAllViewer	Message
LAY_STATUS	Status	
INF_STATUSMSG	StatusMessage	Message
INF_PROGRESS	Progress	Double 0..1

There are some predefined messages used by the METK. The global messages are received by all modules.

Constant	Substituted String	Comment
MSG_LOADED	Loaded	Is sent from METKManager
MSG_TOTALLOADED	TotalLoaded	Is sent from a METKScriptInit after initial animation is ready
MSG_CLEANUP	Cleanup	Sent for telling all modules to cleanup their stuff
MSG_GET	Get	Triggers all viewer to write their initial data into the ObjMgr

The METKMessaging messages are only received by `METKMsgSender` and `METKMsgReceiver`.

Constant	Substituted String	Comment
MSG_INIT	initialize	Start with a new message
MSG_PROCESSING	processing	Message is processed
MSG_DONE	done	Done

A.2.9 Currently Selected Object – OBJ_CSO

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String CSO
LAY_GLOBAL	Global	
INF_IGNOREPICK	IgnorePickable	String Set to ignore Appearance ⇒PickStyle
LAY_SELECTEDOBJ	SelectedObject	
INF_OBJID	ObjectID	String
INF_MULTIOBJID	MuliObjectID	String
INF_SELECTION	Selection	Vec3f Selection point in world coordinates

A.2.10 Time for a cup of coffee – OBJ_GUI

Used for `METKInfoWin` communication. OBJ_GUI = GUI LAY_GLOBAL = Global INF_OBJTYPE = ObjectType String TODO!!! LAY_INFOWIN = InfoWin INF_ISON = isOn INF_INFOHEADER = InfoHeader INF_INFOTEXT = InfoText

A.2.11 Measurement Tools – OBJ_MEASTOOL

It is only possible to include only one measurement tool at a time.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Tool
LAY_DETAILS	Details	
INF_MEASTOOL	SelectedTool	String Off,MainAxis,MinimalDistance, UserDefDistance,Angle
INF_MEASOBJ1	MeasuredObject_1	String Object ID of the selected object 1
INF_MEASOBJ2	MeasuredObject_2	String Object ID of the selected object 2
INF_MEASOBJ3	MeasuredObject_3	String Object ID of the selected object 3
INF_MEASPOS1	MeasuredPosition_1	Vec3f Selected Position 1
INF_MEASPOS2	MeasuredPosition_2	Vec3f Selected Position 2
INF_MEASPOS3	MeasuredPosition_3	Vec3f Selected Position 3

A.2.12 General Patient Information – OBJ_PATIENT

General information about the patient, used by `METKPatientObject`.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Patient
LAY_PATIENT	Patient	
INF_DOB	DOB	String
INF_NAME	Name	String
INF_PID	PID	String
INF_SEX	Sex	String
INF_WEIGHT	Weight	String

A.3 All Layers and Infos for Dynamic ObjectIDs

Most of the objects in an METK case are generated dynamically. For every image, ROI or segmented structure, one ObjectID is created. Also, for some relevant GUI elements, such as viewers, ObjectIDs are created.

A.3.1 Images, ROIs and Segmented Structures

The following table includes all tags used by images, ROIs and segmented structures. LAY_IMAGE is optional for structures. In general, structures will inherit the image layer from the parent ROI or image it is derived from.

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Result
INF_CHILDS	ChildIds	String
INF_PARENT	ParentId	String
INF_VALID	Valid	Bool
LAY_DESCRIPTION	Description	
INF_NAME	Name	String
INF_STRUCTURE	Structure	String
INF_STRUCTUREGROUP	StructureGroup	String
INF_COMMENT	Comment	String
INF_SIDE	Side	String Left, Right
INF_RESULTTYPE	ResultType	String
INF_IMPORTANCE	Importance	Double 0..1
LAY_IMAGE	Image	
INF_FILENAME	Filename	String
INF_IMAGETYPE	ImageType	String Segmentation (Structure) Original (Original Image, ROI)
INF_OBJVALUE	ObjectValue	Int32
INF_WMATRIX	ToWorldMatrix	Mat4
INF_MODALITY	Modality	String
INF_PROTOCOL	Protocol	String
INF_VOXELSIZEX	VoxelSizeX	Double
INF_VOXELSIZEY	VoxelSizeY	Double
INF_VOXELSIZEZ	VoxelSizeZ	Double
INF_SIZEX	SizeX	Int32
INF_SIZEY	SizeY	Int32
INF_SIZEZ	SizeZ	Int32
INF_STUDYDATE	StudyDate	String
INF_STUDYTIME	StudyTime	Double
INF_INSTITUTION	Institution	String
INF_MANUFACTURER	Manufacturer	String
INF_MMODELNAME	ManufacturersModelName	String
INF_BPEX	BodyPartExamined	String
INF_CODSEGIMAGE	CodedSegFilename	String is always a CodedSegmentation
INF_CODSEGVALUE	CodedSegObjectValue	String e.g. 1,23,45
INF_STARTVOXEL	StartVoxel	Vec3f (for ROI Objects)
INF_ENDVOXEL	EndVoxel	Vec3f (for ROI Objects)

A.3.2 Layers for structures only

Constant	Substituted String	Type
LAY_APPEARANCE	Appearance	
INF_COLOR	Color	Vec3f
INF_DRAWSTYLE	DrawStyle	String FILLED, LINES, POINTS, INVISIBLE
INF_PICKSTYLE	PickStyle	String SHAPE, BOUNDING_BOX, UNPICKABLE
INF_TRANSPARENCY	Transparency	Double 0..1
INF_SILHOUETTE	Silhouette	Bool
INF_SILHOUETTECOLOR	SilhouetteColor	Vec3f
INF_SILHOUETTEWIDTH	SilhouetteWidth	Double
INF_STIPPLING	Stippling	Bool
INF_STIPPLINGGAMMA	StipplingGamma	Double > 0
INF_STIPPLINGCOLOR	StipplingColor	Vec3f
INF_VISIBLE	Visible	Bool
INF_BB	BoundingBoxOn	Bool
INF_CLIPPING	Clipping	Bool
INF_SURFACEDIST	SurfaceDistance	Bool
LAY_FILES	Files	
INF_IVFILE	InventorFile	String
INF_IVPOINTER	InventorPointer	Int32 must be nonpersistent!!! Pointer to original inventor file
INF_IVFULLPOINTER	InventorFullPointer	Int32 must be nonpersistent!!! Output from METKIsoSurface, ignores visibility
INF_ISOPOINTER	IsoSurfacePointer	Int32 must be nonpersistent!!! Output from METKIsoSurface, including visibility
INF_STIPPLEPOINTER	StipplePointer	Int32 must be nonpersistent!!! Output from METKStippling
INF_SILHOUETTEPOINTER	SilhouettePointer	Int32 must be nonpersistent!!! Output from METKSilhouette

Constant	Substituted String	Type
LAY_MEASURES	Measures	
INF_BARYCENTER	BaryCenter	Vec3f
INF_MAX_DIAMETER	maxDiameter	Double
INF_EXACT	ExactMeasures	Bool
INF_MIDPOINT	MidPoint	Vec3f
INF_X_AXIS	xAxis	Vec3f
INF_X_DIAMETER	xDiameter	Double
INF_Y_AXIS	yAxis	Vec3f
INF_Y_DIAMETER	yDiameter	Double
INF_Z_AXIS	zAxis	Vec3f
INF_Z_DIAMETER	zDiameter	Double
INF_VOLUME	Volume	Vec3f
INF_VOLUMEDIGITS	VolumeDigits	
INF_VOLUMEUNIT	VolumeUnit	String (cmm)
INF_BBMAX	BoundingBix_max	Vec3f
INF_BBMIN	BoundingBix_min	Vec3f
INF_OBJCENTER	objectCenter	Vec3f
INF_BBCENTER	BoxCenter	Vec3f
INF_WORLDBB	MidPoint	String (2x Vec3: min, max)
LAY_SEGMENTATION	Segmentation	
INF_CONTOURFILE	ContourFile	String
INF_MARKERFILE	MarkerListFile	String
INF_INTERVALTHRESHLOW	LowerIntervalThreshold	Double
INF_INTERVALTHRESHHIGH	UpperIntervalThreshold	Double
INF_THRESH	Threshold	Double

A.3.3 3D and 2D Viewers

Constant	Substituted String	Type
LAY_GLOBAL	Global	
INF_OBJTYPE	ObjectType	String Viewer2D or Viewer3D
INF_GLOBAL_REFRESH	Refresh	Bool
LAY_VIEWER_CAMERA	Camera	
INF_VIEWER_CAMERA_POSITION	Position	Vec3f
INF_..._ORIENTATION	Orientation	Vec4f
INF_..._HEIGHT	Height	Double
INF_..._FOCALDISTANCE	FocalDistance	Double
LAY_VIEWER_PROPERTIES	Properties	
INF_..._FRAMERATE	Framerate	
INF_..._CAPTUREFRAME	CaptureFrame	
INF_..._CREATEAVI	CreateAVI	
INF_..._CANCELRECORDING	CancelRecording	
INF_..._UPDATEMLOUTPUT	UpdateMLOutput	
INF_..._VIEWERSTATUS	ViewerStatus	
INF_..._PROTOCOLVIEWERSTATUS	ProtocolViewerStatus	
INF_..._BGCOLOR	BGColor	Vec3f
INF_..._BGGREYCENTER	BGGreyCenter	Double
INF_..._BGGREYWIDTH	BGGreyWidth	Double
LAY_VIEWER_SCREENSHOT	Screenshot	
INF_SCREENSHOTFILE	ScreenshotFilename	String
INF_SCREENSHOTTHUMB	ScreenshotThumb	Bool
INF_SCREENSHOTFLAG	ScreenshotFlag	Bool True = make screenshot False = Done

Additional Tags for Viewer2D:

Constant	Substituted String	Type
INF_WINDOWCENTER	WindowCenter	Double 0..1
INF_WINDOWWIDTH	WindowWidth	Double 0..1
INF_WPOSITION	WorldPosition	Vec3f Current world position
INF_LAYOUT	LayoutMode	String LAYOUT_AXIAL,LAYOUT_SAGITTAL, LAYOUT_CORONAL,LAYOUT_CUBE, LAYOUT_CUBE_EQUAL, LAYOUT_CUBE_CUSTOMIZED, LAYOUT_ROW, LAYOUT_ROW_EQUAL, LAYOUT_ROW_AXIALEXTRA, LAYOUT_COLUMN
INF_ANNOTATIONS	Annotations	Bool