4/8/2009

MITK data management

Introduction to basic concepts of the Medical Imaging Interaction Toolkit

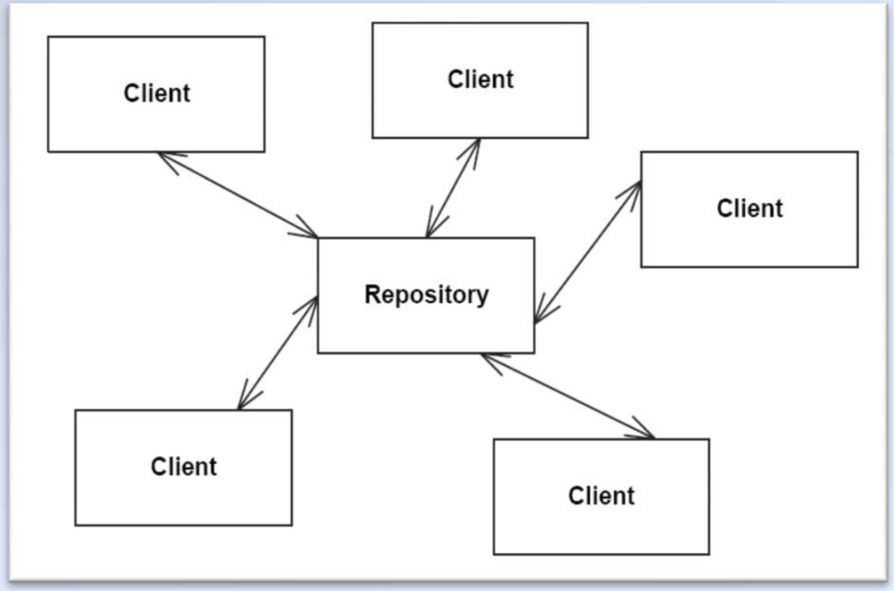
GERMAN
CANCER RESEARCH CENTER
IN THE HELMHOLTZ ASSOCIATION

Data management – shared repository

Pattern: Shared Repository

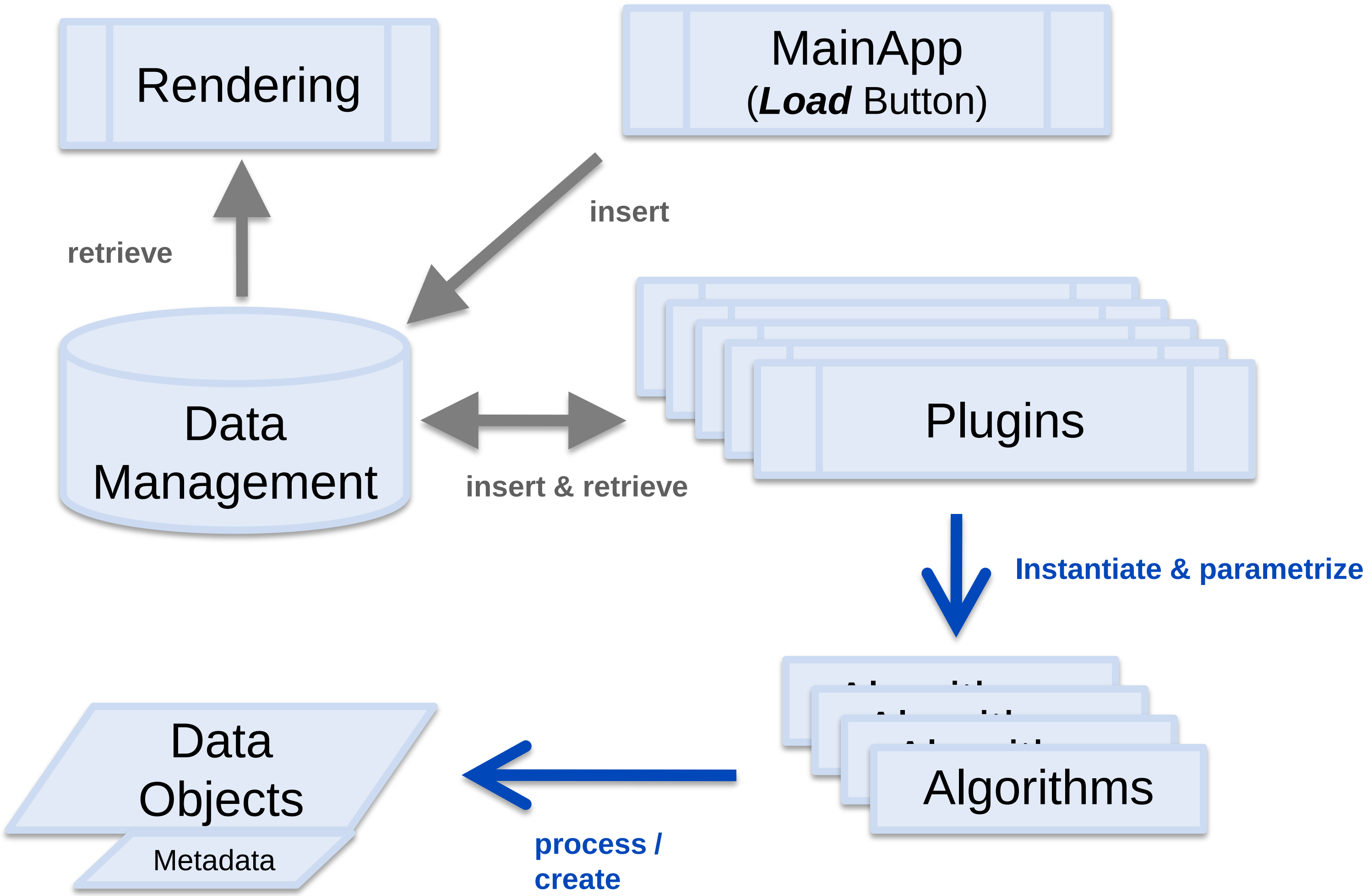
Data needs to be shared between components. In sequential architectures like LAYERS or PIPES AND FILTERS the only way to share data between the components (layers or filters) is to pass the information along with the invocation, which might be inefficient for large data sets. Also it might be inefficient, if the shared information varies from invocation to invocation because the components' interfaces must be prepared to transmit various kinds of data. Finally the long-term persistence of the data requires a **centralized data management**.

In the **SHARED REPOSITORY pattern** one component of the system is used as a **central data store**, accessed by all other **independent** components. This SHARED REPOSITORY offers suitable means for accessing the data, for instance, a query API or language. The SHARED REPOSITORY must be scalable to meet the clients' requirements, and it must ensure data consistency. It must handle problems of resource contention, for example by locking accessed data. The SHARED REPOSITORY might also introduce transaction mechanisms.



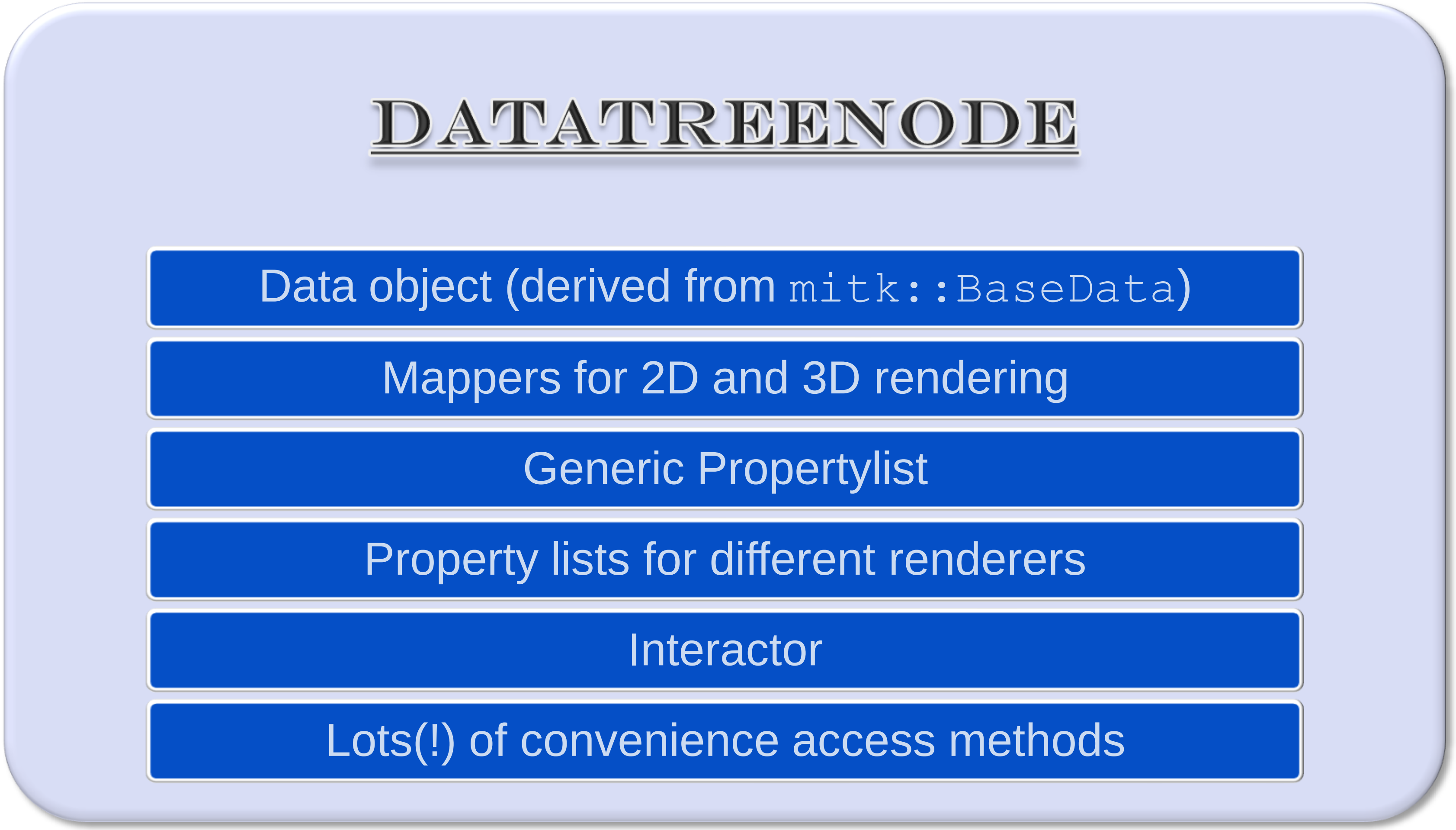
From http://www.infosys.tuwien.ac.at/staff/zdun/publications/ArchPatterns.pdf

Shared repository in MITK MainApp



Data objects and meta data

- Data Object is encapsulated in DataTreeNode


DATATREENODE
Data object (derived from `mitk::BaseData`)
Mappers for 2D and 3D rendering
Generic Propertylist
Property lists for different renderers
Interactor
Lots(!) of convenience access methods

DataTreeNode

```
mitk::Surface::Pointer myData = mitk::Surface::New();

mitk::DataTreeNode::Pointer myNode = mitk::DataTreeNode::New();
myNode->SetData(myData);

myNode->SetName(“My surface node”);
mitk::Color color;  color.Set(1.0f, 0.0f, 0.0f);
myNode->SetColor(color);

mitk::VtkRepresentationProperty::Pointer repProp =
                             mitk::VtkRepresentationProperty::New();
repProp->SetRepresentationToWireframe();
myNode->SetProperty( "representation", repProp,
                      myWireFrameRenderWindow->GetRenderer());

mitk::AffineInteractor::Pointer interactor =
    mitk::AffineInteractor::New( "AffineInteractions ctrl-drag", myNode);

// Now add myNode to a data repository...
```

Metadata in MITK

- Describes how data should be rendered
- Arbitrary metadata
- Identifier, Flags
- mitk::DataTreeNode stores Metadata as named Property objects in a mitk::PropertyList (*name* ➔ value)
- Generic PropertyList & renderer specific PropertyLists
- Available data types:
    - Bool, float, int, string, enumeration
    - Point3D, Vector3D
    - Color, Material, Lookup table, Transfer function
    - Tags

Properties

- Rendering specific properties

Generic

- visible
- layer
- name

Image

- opacity
- color
- use color
- binary
- outline binary
- texture interpolation
- reslice interpolation
- volumerendering
- levelwindow
- LookupTable
- TransferFunction

PointSet

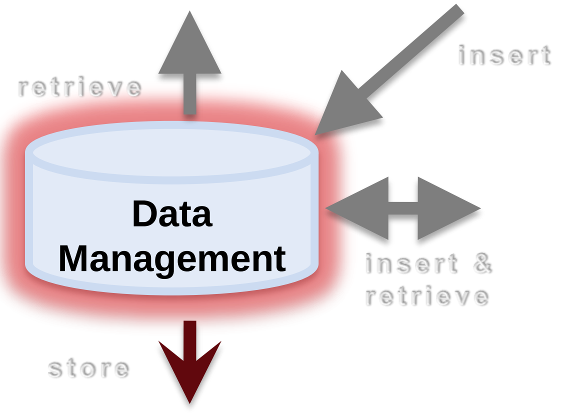
- line width
- pointsize
- selectedcolor
- color
- contour
- contourcolor
- close
- show points
- show distances
- distance decimal digits
- show angles
- show distant lines

Surface

- line width
- scalar mode
- wireframe line width
- material
- scalar visibility
- color mode
- representation
- interpolation

- Many more are used by individual functionalities…
  - e.g. *volume*, *helper object, active navigation image, tip offset,…*

central data repository classes in MITK



- `mitk::DataTree`
  - Tree structure
  - Use iterators to add & retrieve data
- `mitk::DataStorage`
  - Graph structure
  - *„SQL like“* add & retrieve of data
- `mitk::DataTreeStorage`
  - Encapsulation of DataTree
  - Provides DataStorage‘s „Database like“ interface

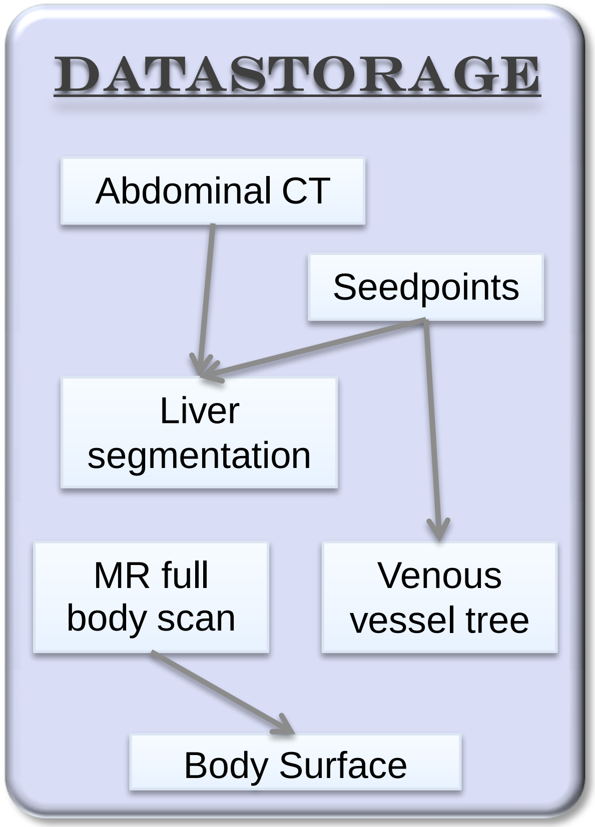
Used in Qt3 MainApp

Used in Qt4 OpenCherry MainApp

Data management with the `mitk::DataStorage`

DataStorage

- Database like queries for objects
  - Similar to ***SELECT * FROM repository WHERE DataType == mitk::Image***
  - Predicate objects to build WHERE statement
- Stores relation of objects in a ***directed acyclic graph***
  - object can be derived from multiple source objects
  - object can have multiple children objects
- Events:
  - *AddNodeEvent*
  - *RemoveNodeEvent*



Adding elements

```
/* create some DataTreeNodes */
mitk::DataTreeNode::Pointer n1 = mitk::DataTreeNode::New();
mitk::Image::Pointer image = mitk::Image::New();
n1->SetData(image);

mitk::DataTreeNode::Pointer n2 = mitk::DataTreeNode::New();
mitk::Surface::Pointer surface = mitk::Surface::New();
n2->SetData(surface);
mitk::Color color;  color.Set(1.0f, 0.0f, 0.0f);
n2->SetColor(color);

mitk::DataTreeNode::Pointer n3 = mitk::DataTreeNode::New();
n3->SetColor(color);

/* Create Data Storage */
mitk::DataStorage::Pointer ds = mitk::DataStorage::GetInstance();
/* Fill DataStorage */
ds->Add(n1);
ds->Add(n2);
mitk::DataStorage::SetOfObjects::Pointer parents =
                         mitk::DataStorage::SetOfObjects::New();
parents->InsertElement(0, n1);  // n3 is source of n1
ds->Add(n3, parents);
```

$$\begin{tikzcd} \boxed{\mathbf{n1}} \arrow[r, "\text{is source of}"] & \boxed{\mathbf{n3}} \end{tikzcd}$$

Retrieving elements

```
/* retrieve all objects */
mitk::DataStorage::SetOfObjects::ConstPointer all = ds->GetAll();
for (SetOfObjects::ConstIterator it = all->Begin(); it != all->End();
  ++it)
{
  mitk::DataTreeNode::Pointer node = it.Value();
}

/* retrieve objects with specific criteria */
mitk::NodePredicateDataType predicate("Image");
SetOfObjects::ConstPointer rs = ds->GetSubset(predicate); // == n1

mitk::NodePredicateProperty p("color",
                              mitk::ColorProperty::New(color));
SetOfObjects::ConstPointer rs = ds->GetSubset(p); // == n2 & n3
```

Retrieving elements II

- Existing NodePredicates:
  - `mitk::NodePredicateData` – Check for specific data object
  - `mitk::NodePredicateDataType` – Check for data type (Image, Surface,…)
  - `mitk::NodePredicateDimension` – Check for dimension of data object
  - `mitk::NodePredicateProperty` – Check for existence of property with specific **name** or for existance of property with specific **name and value**
  - `mitk::NodePredicateAND` – combine multiple predicates
  - `mitk::NodePredicateOR` – combine multiple predicates
  - `mitk::NodePredicateNOT` – negate predicate

- Example:

```
mitk::NodePredicateDataType p1("Image");
mitk::NodePredicateProperty p2("color", new ColorProperty(color));
mitk::NodePredicateOR pOR(p1, p2);
mitk::NodePredicateDataType p4("Surface");
mitk::NodePredicateNOT pNOT(p4);
mitk::NodePredicateAND pAND(pOR, pNOT);
SetOfObjects::ConstPointer rs = ds->GetSubset(pAND); // == ???
```

Retrieving related elements

```
/* retrieve source & derived objects */
SetOfObjects::ConstPointer sources = ds->GetSources(n3); // == n1
SetOfObjects::ConstPointer child = ds->GetDerivations(n1); // == n3

/* retrieve source & derived objects with specific criteria */
mitk::NodePredicateDataType p("Image");
SetOfObjects::ConstPointer sources = ds->GetSources(n3, &p); // == n1
SetOfObjects::ConstPointer child = ds->GetDerivations(n1, &p); // == ?
```

```
GetSources(const mitk::DataTreeNode* node, const NodePredicateBase*
  condition = NULL, bool onlyDirectSources = true)
```

➔ Retrieve

- **direct** sources/derivations
- **all** sources/derivations

Convenience DataStorage methods

```
/* Is a node already in the DataStorage? */
mitk::DataTreeNode* n = […]
bool nodeInDataStorage = ds->Exists(n);

/* Retrieve single Nodes/objects */
mitk::NodePredicateDataType p("Image");
mitk::DataTreeNode* n = ds->GetNode(&p);
mitk::DataTreeNode* n = ds->GetNamedNode("MyNode");
mitk::Image* image = ds->GetNamedObject<mitk::Image>("data");
```

Thank you

More information:

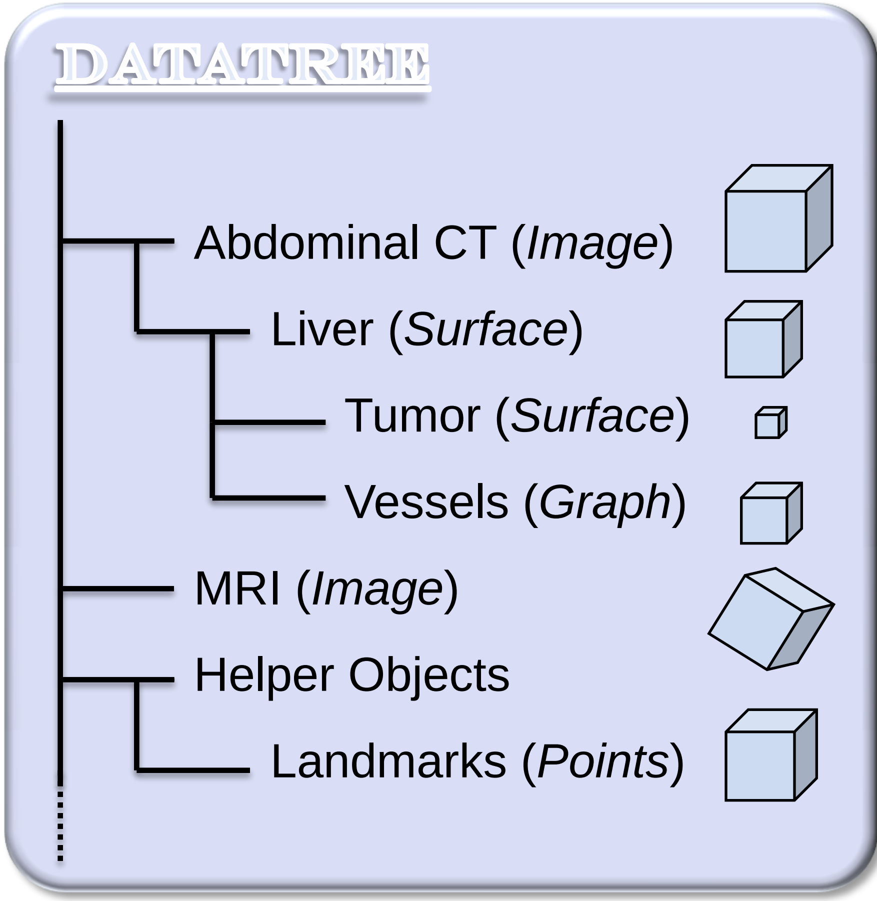
http://docs.mitk.org/nightly/group__DataManagement.html

www.mitk.org

Data management with the `mitk::DataTree`

DataTree

- Hierarchical organization of data objects in a tree structure
- Nodes can have arbitrary number of child nodes
- Events:
    - *TreeNodeChangeEvent*
    - *TreeAddEvent*
    - *TreeRemoveEvent*
    - *TreePruneEvent*
- *Data Retrival with iterator objects*
    - *InOrderIterator*
    - *PostOrderIterator*
    - *PreOrderIterator*
    - *LeafIterator*
    - *RootIterator*
    - *ChildIterator*



Using the DataTree

```
mitk::DataTreeNode::Pointer n1 = mitk::DataTreeNode::New();
mitk::DataTreeNode::Pointer n2 = mitk::DataTreeNode::New();
n1->[…]; n2->[…];

mitk::DataTree::Pointer tree = mitk::DataTree::New();
mitk::DataTreePreOrderIterator it(tree);
it.Add(n1);
it.GoToChild();
it.Add(n2);

mitk::DataTreePreOrderIterator it2(tree);
it2.GoToBegin();
while (!it->IsAtEnd())
{
  mitk::DataTreeNode* node = it->Get();
  node->[…]
  ++it;
}
```