# Reader/Writer for Analyze Object Maps for ITK

*Release 1.1*

Jeffrey Hawley, Hans Johnson

March 16, 2009

The University of Iowa

**Abstract**

This document describes an addition to the Image IO Library of the Insight Toolkit (ITK). ITK has been able to read in Analyze image files but not the Analyze Object Maps that correspond to the images. Analyze Object maps define regions of interest that are helpful in designating locations, giving a name to those locations, as well as display characteristics. Also, Analyze Object Maps are useful to tell someone where to look in an image. With the lack of not being able to read/write Analyze Object Maps, a new set of classes have been developed to read/write Analyze Object Maps and manipulate the object maps.

## Contents



## 1 Anaylze Object Maps

An Analyze Object Map is an image where an index from 0-255 correspond to an Analyze Object(or, as will be referred to in this paper as, Analyze Object Entry). Each Analyze Object Entry is a specific Region of Interest that a user has specified. With these Regions of Interest the user can put a name to the Region, a specific color and be able to communicate to other people what areas to look at or know which Regions they are talking about. Regions of Interest could be differentiating a part of the brain, marking a location where someone thinks a tumor is or many other ways to group a specific area to a name and color. The Analyze Object Maps can be shown by themselves or overlaid of another image, such as the original image that was the basis of the Analyze Object Map, to show which areas were highlighted.

## 2 Files

There are two utility classes (itkAnalyzeObjectMap and itkAnalyzeObjectEntry) and the main IO file (itkAnalyzeObjectLabelMapImageIO) and the corresponding IO factory file (itkAnalyzeObjectLabelMapImageIOFactory).

The IO file takes care of reading and writing out an object file. Whenever a file is read or writen an object file will have at the very beginning information about the object map. The very first data is the version of the object map, next there is the x,y,z size of the object map, the number of object entries and finally for version 7 (the newest version) number of volumes. After the beginning information there is the information about each object file, that means that there is going to have to be a vector of all of the entries.

The utility file called itkAnalyzeObjectEntry stores all of the information for one specific entry, if a vector is made of all of the entries of the file then it is easy to pick out the object entries. There are also some other functions in the itkAnalyzeObjectEntry that will read or write the information for the object entry.

Finally, after all of the entries there is a run length encoding of the image. The run length encoding corresponds to an image of the numbers of the entries starting from 0 and going all the way up to the number of object entries.

The itkAnalyzeObjectMap file has functions that take care of doing things to an object map once it is created - such as converting the itkAnalyzeObjectMap to an RGB Image.

The following bullets are the instructions on how to incorprate the files into itk and the functions inside of each file.

- itkAnalyzeObjectLabelMapIO

  Functions:

  - bool CanReadFile(const char* FileNameToRead)

    This checks to see if the filename, that was passed in, ends with an obj extension. If it does then return true that this class can read the file else return false stating that the reader can not read the file.

  - void ReadImageInformation()

    This reads in the header information. It first reads in the verision of the object map, then the x,y,z sizes, then the number of objects and for verision 7 object maps the number of volumes. Then the function reads in each object entry and stores them into a vector of object entries.

  - void Read(void* buffer)

    This will read in and decipher the run length encoded image. The expanded image will be stored in the void pointer that was passed in and which will be used for the call GetOutput().

  - bool CanWriteFile(const char * FileNameToWrite)

    This checks to see if the filename, that was passed in, ends with an obj extension. If it does then return true that this class can write the file else return false stating that the writer can not read the file.

  - void WriteImageInformation()

    This writes out the header information. It first writes out the verision of the object map (which is verisoin 7 right now), the x,y,z sizes of the image, the number of objects and for verision 7 object maps the number of volumes. Then the function writes out each object entry to the file. The

object entry will most likly take up the most room of the file. If an image was created without any object entries then the writer will write out 256 object entries.

- void Write(const void* buffer)

  This will compresse the the void pointer (which is the image and is changed to an unsigned char within the function) that was passed in by run length encoding the void pointer and then write the run length encoding to the specified file.

- bool CanStreamRead()

  This will return false right now because the reader can not stream data. This is something that will be worked on in the future.

- itkAnalyzeObjectEntry

  Functions:

  - void Copy( AnalyzeObjectEntry::Pointer rhs )

    This function will copy all of the ivars except the name of the entry from the Analyze Object Entry that was passed in to the current Analyze Object Entry. The reason why the function does not copy the name, is that each object entry should have a unique name.

  - These next functions are get/set macros that will get/set most of the ivars.

    getName/setName, getDisplayFlag/setDisplayFlag, getCopyFlag/setCopyFlag, getMirrorFlag/setMirrorFlag, getStatusFlag/setStatusFlag, getNeighborsUsedFlag/setNeighborsUsedFlag, getShades/setShades, getStartRed/setStartRed, getStartGreen/setStartGreen, getStartBlue/setStartBlue, getEndRed/setEndRed, getEndGreen/setEndGreen, getEndBlue/setEndBlue, getXRotation/setXRotation, getXRotationIncrement/setXRotationIncrement, getYRotation/setYRotation, getYRotationIncrement/setYRotationIncrement, getZRotation/setZRotation, getZRotationIncrement/setZRotationIncrement, getXTranslation/setXTranslation, getXTranslation/setXTranslation, getYTranslation/setYTranslation, getYTranslationIncrement/setYTranslationIncrement, getZTranslation/setZTranslation, getZTranslation/setZTranslation, getXCenter/setXCenter, getYCenter/setYCenter, getZCenter/setZCenter, getMinimumXValue/setMinimumXValue, getMinimumYValue/setMinimumYValue, getMinimumZValue/setMinimumZValue, getMaximumXValue/setMaximumXValue, getMaximumYValue/setMaximumYValue, getMaximumZValue/setMaximumZValue, getOpacity/setOpacity, GetOpacityThickness/SetOpacityThickness, GetBlendFactor/SetBlendFactor

  - void Print(std::ostream &myfile)

    This function will print out all of the ivars out to any file that the user wants. This is mostly used for debugging purposes.

  - void ReadFromFilePointer(std::ifstream & inputFileStream, const bool NeedByteSwap, const bool NeedBlendFactor)

    This function will read in all of the ivars from a file location that is passed into it. Also, if the computer is little endian then pass in a bool value of true so that the data is swapped from big endian to little endian else pass in false. Finally, it was stated in the documentation from the Mayo Foundation that Analyze Object Entry's don't have Blend Factor in verisions prior to

Verision 7, so that is the reason for const bool NeedBlendFactor, but right now the code does read in Blend Factor no matter what because some verision 6 files were tried and they needed the extra 4 bits.

- void SwapObjectEndedness()
  This function will change the object endedness if the computer is a little endian machine, since the object maps are written in big endian.
- void Write(std::ofstream &outputFileStream)
  This function will write out all of the ivars to a file location that is passed into it.

- itkAnalyzeObjectMap

  Functions:

  - AnalyzeObjectMap operator=
    This function is the assignment operator, which copies a new object map from the right side to the AnalyzeObjectMap variable on the left side.
  - AnalyzeObjectEntryArrayType * GetAnalyzeObjectEntryArrayPointer()
    This will return a pointer to the vector of object entries that an object map has.
  - GetNumberOfObjects/SetNumberOfObjects
    This will get/set the number of object entries that an object map has.
  - itk::AnalyzeObjectMap<TImage>::Pointer PickOneEntry(const int numberOfEntry = -1)
    The user will input the number of the Object entry that they would like to pick. Then the function will create a new object map that will re returned. Then the function will go through the original object map image and get rid of all of the other object numbers and change the number of the object entry the user specified to one. Then the new image will be outputted to the new Object map. After that the object entry from the original object map will be copied over to the new object map's vector of object entries. Finally, the new object map is returned.
  - TRGBImage::Pointer ObjectMapToRGBImage()
    This will convert the object map into an RGB Image based on the end red, end green and end blue specified for each object entry. That means that the function will have to go through the object map image, get the value at each pixel, find the object entry that corresponds to the value of the pixel and then pull out the end red, end green and end blue and set that as the pixel color for the RGB Image. Then that RGB Image will be returned.
  - void AddObjectEntryBasedOnImagePixel(ImageType *Image,const int value = -1,const std::string ObjectName = "",const int Red = 0,const int Green = 0,const int Blue = 0)
    This will go through an image that the user inputs, find the specific pixel value the user inputs and then create an object map at the location that it finds the specific pixel value. The user will also have the option of inputing the red, green and blue they want the object map to be.
  - void AddAnalyzeObjectEntry(const std::string ObjectName = "")
    This will just add an object entry to the end of the vector of object entries that an object map has with the specific name the user passes in.
  - void DeleteAnalyzeObjectEntry(const std::string ObjectName = "")
    This will delete an object entry that a user specifies by inputting the name. The function will go through the image and delete the number that corresponds to the object entry. Then the function will move all of the object entry numbers above the object entry that was deleted down one number. Then the function will move all of the object entries above the object entry that was deleted in the vector one number down.

- int FindObjectEntry(const std::string ObjectName = ””)

  This function will find an object entry based on the name that the user inputs. If the function finds the object entry then it will return the number of the vector of the object entry. If the function does not find the object entry then the function will return -1.

- void PlaceObjectMapEntriesIntoMetaData()

  This function will place the object entries into the meta data so that the object map can be moved around just like a normal image. This function is normally called in the functions that are in this class.

- AnalyzeObjectEntry::Pointer GetObjectEntry( const int index )

  This function will return the smart pointer of the object entry number the user inputs.

- void ImageToObjectMap(ImageType *image)

  This function will take an image and make it into an object map. If there is data for object entries in the meta data then extract that data. Then take the pixel container of the image and place it into the object map’s pixel container.

## 3 How to Compile

There are a couple of ways to compile the libraries, tests and examples. The easiest way is to download the code from the Insight Journal, run CMake on the top level and let CMake create the necessary items. Once the necessary items are made then the libraries will need to be built.

The other way is to place the code inside the ITK source directory. We have come up with some instructions on how to place the code into the ITK source directory.

The way to include all of the files into ITK we have written will go like this:

The folder inside of Source called AnalyzeObjectMapIO will contain these files: itkAnalyzeObjectLabelMapImageIO.h, itkAnalyzeObjectLabelMapImageIO.cxx, itkAnalyzeObjectLabelMapImageIOFactory.h, itkAnalyzeObjectLabelMapImageIOFactory.cxx will need to be placed into the source of ITK->Code->IO. While in the source of ITK->Code->IO the CMakeLists file will need to be changed so that SET... has itkAnalyzeObjectLabelMapImageIOFactory.cxx and itkAnalyzeObjectLabelMapImageIO.cxx in it. Then itkImageIOFactory.cxx will need to be modified so that `#inlcude "itkAnalyzeObjectLabelMapImageIOFactory.h"` is placed with the includes, then add `ObjectFactoryBase::RegisterFactory( AnalyzeObjectLabelMapImageIOFactory::New() );` in the function RegisterBuiltInFactories.

Then the folder inside of Source called AnalyzeObjectMap will need to be placed into the source of ITK->Code->Common. Also the CMakeLists file will need to be changed so that SET... has itkAnalyzeObjectEntry.cxx in it. Then CMake will need to be re-run on ITK and then ITK will have to be re-built.

When done with getting the libraries built, the examples and tests can be run. The next couple of sections will explain the examples and tests.

## 4 Examples

This section provides some examples on how to use the programs.

- Example 1

  This example is for reading in any file, then creating a new object map, then add entries to the object map and finally writing the new object map out to file.

  Command:

  **CreatingObjects.exe [input image] [output Analyze Object Map file]**
  *CreatingObjects.exe 2dtest.nii creatingObjects.obj*

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkAnalyzeObjectMap.h"

int main( int argc, char ** argv )
{
  typedef unsigned char       InputPixelType;

  typedef unsigned char       OutputPixelType;

  const   unsigned int        Dimension = 3;

  typedef itk::Image< InputPixelType,  Dimension >    InputImageType;

  typedef itk::Image< OutputPixelType, Dimension >    OutputImageType;

  typedef itk::ImageFileReader< InputImageType  >  ReaderType;

  typedef itk::ImageFileWriter< OutputImageType >  WriterType;
  ReaderType::Pointer reader = ReaderType::New();
  WriterType::Pointer writer = WriterType::New();
  ...
  reader->SetFileName(NiftiFile);
  ...
  reader->Update();
  ...
  itk::AnalyzeObjectMap<InputImageType>::Pointer CreateObjectMap = itk::AnalyzeObjectMap
  <InputImageType>::New();

  ...

  //Add one entry to the object map named "You Can Delete Me", this entry corresponds to 1
  if you do a pickOneEntry
  CreateObjectMap->AddAnalyzeObject("You Can Delete Me");

  //Add another two entries that will be based on the image that is passed into
  //the function, also, the intensity that you would like searched for, the name of the entry
  and then finally the RGB values
  //you would like the entry to have for the regions that are found.

  //This entry corrsponds to 2 if you do a pickOneEntry
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 200, "Square", 250,
  0, 0);
```

```
  //This entry corrsponds to 3 if you do a pickOneEntry
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 128, "Circle", 0,
  250, 0);

  //This entry corrsponds to 4 if you do a pickOneEntry
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 45,  "SquareTwo",
  0, 0, 250);

  //Then anoter entry is added, this entry corrsponds to 5 if you do a pickOneEntry
  CreateObjectMap->AddAnalyzeObject("Nothing In Here");

  //The entry that was just added is deleted
  CreateObjectMap->DeleteAnalyzeObject("Nothing In Here");

  ...
  writer->SetInput(CreateObjectMapTwo);
  ...

  writer->Update();
  ...
}
```

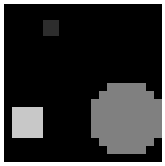

Figure 1: 2dtest.nii

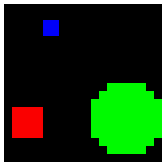

Figure 2: creatingObjects.obj, this is what it should look like when the image is converted it to an RGB image. See next example on how to display object maps.

- Example 2

  This is a simple example for reading in an Analyze object map and then showing how to use vtk to display the object map.

  Command:

  **DisplayingObjectMaps.exe [input Analyze Object Map file]**
  *DisplayingObjectMaps.exe Circle.obj*

```
#include "itkImage.h"
#include "itkRGBPixel.h"
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkAnalyzeObjectMap.h"

//#include "itkImageToVTKImageFilter.h"
//#include "vtkRenderer.h"
//#include "vtkRenderWindowInteractor.h"
//#include <vtkImageViewer2.h>
```

```
typedef unsigned char PixelType;
const unsigned int Dimension = 3;
typedef itk::Image< PixelType, Dimension > ImageType;
typedef itk::RGBPixel<PixelType> RGBPixelType;
typedef itk::Image< RGBPixelType, Dimension > RGBImageType;

typedef itk::ImageFileReader< ImageType > ReaderType;

//typedef itk::ImageToVTKImageFilter<RGBImageType> ConnectorType;

int main(int argc, char * argv [] )
{
  ReaderType::Pointer reader  = ReaderType::New();

  //The input should be an Anaylze Object Map file
  reader->SetFileName( DisplayImage );
  ...

  reader->Update();

  //This will convert the output of the reader into an object map
  itk::AnalyzeObjectMap<ImageType, RGBImageType>::Pointer Objectmap = itk::AnalyzeObjectMap
  <ImageType, RGBImageType>::New();
  Objectmap->ImageToObjectMap(reader->GetOutput());

  //If you have vtk and itkApplications installed then you can uncomment this out to display
  //an object map to the screen.  Otherwise you can see how to display an object map using
  vtk.

  //Set the input, to the connector connecting itk with vtk, with an RGB image of the
  specific
  //colors that corresponds to each entry in the object map.

  //vtkRenderWindowInteractor *windowInteractor = vtkRenderWindowInteractor::New();
  //ConnectorType::Pointer connector= ConnectorType::New();
  //connector->SetInput( Objectmap->ObjectMapToRGBImage() );

  //connector->Update();

  //Display a two dimensional view of the object map that was read in
  //vtkImageViewer2 * twodimage = vtkImageViewer2::New();

  //const int SliceNumber = 0;
  //twodimage->SetInput(connector->GetOutput());
  //twodimage->SetSlice(SliceNumber);
  //twodimage->SetSliceOrientationToXY();

  //Set the background of the renderer to a grayish color so that it is easier to see
  //the outline of the object map since it is usually black
  // twodimage->GetRenderer()->SetBackground(0.4392, 0.5020, 0.5647);
  // twodimage->SetupInteractor(windowInteractor);

  // twodimage->Render();
```

```
  // windowInteractor->Start();
...
}
```

- Example 3

  This example is for reading in any file, then creating a new object map, then add entries to the object map, pick one entry and then write out the new object map with the one entry only.

  Command:

  **PickOneObjectEntry.exe [input image file] [output Analyze Object Map file]**
  *PickOneObjectEntry.exe 2dtest.nii circle.obj*

```
#include "itkImageFileReader.h"
#include "itkImageFileWriter.h"
#include "itkImage.h"
#include "itkAnalyzeObjectMap.h"

int main( int argc, char ** argv )
{
  ...
  //Now we bring in a nifti file that Hans and Jeffrey created,
  //the image as two squares and a circle in it of different intensity values.
  reader->SetFileName(NiftiFile);
  ...
  reader->Update();
  ...

  itk::AnalyzeObjectMap<InputImageType>::Pointer CreateObjectMap = itk::AnalyzeObjectMap
  <InputImageType>::New();

  //Add another two entries that will be based on the image that is passed into
  //the function, also, the intensity that you would like searched for,
  //the name of the entry and then finally the RGB values
  //you would like the entry to have for the regions that are found.
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 200, "Square", 250,
  0, 0);
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 128, "Circle", 0,
  250,0);
  CreateObjectMap->AddObjectEntryBasedOnImagePixel(reader->GetOutput(), 45,  "SquareTwo",
  0, 0, 250);

  //Pick the circle entry and have it put into CreateObjectMap two.  These means
  //that there is only one entry in CreateObjectMapTwo and the image has also
  //been taken care of.
  itk::AnalyzeObjectMap<InputImageType>::Pointer CreateObjectMapTwo = CreateObjectMap->
  PickOneEntry(2);

  //Place all of the entries into the meta data so that the entries can be written
  //out to an object file.
  CreateObjectMapTwo->PlaceObjectMapEntriesIntoMetaData();

  //Now write out an object file
  writer->SetInput(CreateObjectMapTwo);
```

```
  writer->SetFileName(CreatingObject);
  ...
  writer->Update();
  ...
}
```

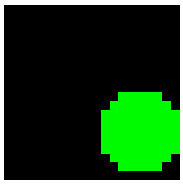

Figure 3: circle.obj, this is what it should look like when it is converted to an RGB image. Also, see the first example to see what 2dtest.nii should look like.

## 5 Testing

Right now we have one test that does all of the testing. The filename for the test is AnalyzeObjectMapTest.cxx. The file takes in seven arguments. Each argument corresponds to a certain name of another file.

For the first argument it should be the name of test.obj and that file is a 3-Dimensional object file that Hans created. The file is read in, then converted into an object map and finally an RGB image is created from the object map. This was just to make sure that everything worked without any errors.

After that the output of the reader is sent to a writer to write out the exact copy of the argument but with a different file name. That is where the second argument comes in. Once the file is written out then both files are opened up and compared to make sure they have the exact same bit by bit data.

After that the third argument that corresponds to the name of a nifti file that Hans and I created which is the same image seen in example 1 is read in. With that nifti file read in we create a new object map with some extra entries and other entries based on the nifti file that was read in. That new object map is written out a file as an object file that corresponds to the fourth argument read in. That file that was just written out is then read back in where the reader output is then converted into an object map and finally an RGB image is created from the object map.

Then from that object map one entry is picked and new object map is created with just the one entry in it. This object map with the one entry is written to file that corresponds with the fifth argument. The file that was just written out is read back in and then converted into an object map and finally an RGB image is created from the object map.

From there, the sitxh argument corresponds to a name for a four dimensional object name that will be written out. A four dimensional image is created and then sent to the writer and then is read back in.

Finally, the seventh argument corresponds to a name for a one dimensional object name that will be written out. A one dimensional image is created and then sent to the writer and then is read back in.

## 6 Future Work

We will make the writer and reader streamable in the future.

## References