Image Model Refactoring

Adam Gu, 2015/8/14

Orientation Info

***ImageProcessor* keeps two extra copies of the orientation info.**

```
class ImageProcessor {
private:
  OrientationInfos orientation_;
  OrientationInfos virtual_orientation_;
};
```

Struct

```
struct OrientationInfos {
  bool bMirror;
  bool bRotate90;
  int nRotate90;
  bool bRotate180;
  int nRotate180;
};
```

Orientation Info (cont.)

***DicomImage* provides two getters for these orientation info.**

Client usage

```
int ImageBoxBase::GetViewRotation() {
  ...
  imageprocessor::OrientationInfos oi;
  image_model_->GetDicomImage()->GetVirtualOrientationInfos(oi);
  rotation = 0;
  if (oi.bRotate180) {
    rotation = 180 * oi.nRotate180;
  }
  if (oi.bRotate90) {
    rotation += oi.nRotate90 * 90;
  }
  ...
}
```

Orientation Info (cont.)

In order to synchronize with SDK, *ImageProcessor* has to do really a lot!

Init

```
bool ImageProcessor::InitProcessingParameters() {
  ...
  orientation_.bMirror = IsAlgoEnabled(PROC_MIRROR);
  orientation_.bRotate90 = IsAlgoEnabled(PROC_ROTATE90);
  orientation_.bRotate180 = IsAlgoEnabled(PROC_ROTATE180);
  orientation_.nRotate90 = orientation_.bRotate90?1:0;
  orientation_.nRotate180 = orientation_.bRotate180?1:0;

  virtual_orientation_ = orientation_;

  return true;
}
```

Orientation Info (cont.)

Rotate180

```
bool ImageProcessor::Rotate180Image(bool enabled) {
  ...
  orientation_.nRotate180 = (++orientation_.nRotate180) % 2;
  orientation_.bRotate180 = orientation_.nRotate180 > 0;

  virtual_orientation_.nRotate180 =
      (++virtual_orientation_.nRotate180) % 2;
  virtual_orientation_.bRotate180 =
      virtual_orientation_.nRotate180 > 0;
}
```

Orientation Info (cont.)

Rotate90

```
bool ImageProcessor::Rotate90Image(bool enabled) {
  ...
  orientation_.nRotate90 = (++orientation_.nRotate90)%4;
  orientation_.bRotate90 = orientation_.nRotate90>0;

  virtual_orientation_.nRotate90 =
      (++virtual_orientation_.nRotate90) % 4;
  virtual_orientation_.bRotate90 =
      virtual_orientation_.nRotate90>0;
}
```

Orientation Info (cont.)

Mirror

```
bool ImageProcessor::MirrorImage(bool enabled) {
  ...
  orientation_.bMirror = enabled;
  virtual_orientation_.bMirror = enabled;

  switch ((orientation_.nRotate90+2*orientation_.nRotate180)%4) {
    case 1:
      if (enabled) {
        virtual_orientation_.nRotate180 =
           (++virtual_orientation_.nRotate180)%2;
      } else {
        virtual_orientation_.nRotate180 = (--
virtual_orientation_.nRotate180)%2;

        if (virtual_orientation_.nRotate180<0) {
```

```
        virtual_orientation_.nRotate180 =
-virtual_orientation_.nRotate180;
       }
     }

     virtual_orientation_.bRotate180 =
virtual_orientation_.nRotate180!=0;
     break;

   case 3:
     if (enabled) {
       virtual_orientation_.nRotate180 = (--
virtual_orientation_.nRotate180)%2;

       if (virtual_orientation_.nRotate180<0) {
         virtual_orientation_.nRotate180 =
-virtual_orientation_.nRotate180;
       }
     } else {
```

```
      virtual_orientation_.nRotate180 = (+
+virtual_orientation_.nRotate180)%2;
    }

    virtual_orientation_.bRotate180 =
virtual_orientation_.nRotate180!=0;
    break;

  default:
    break;
 }
```

Orientation Info (cont.)

My refactoring removes all these redundant codes!

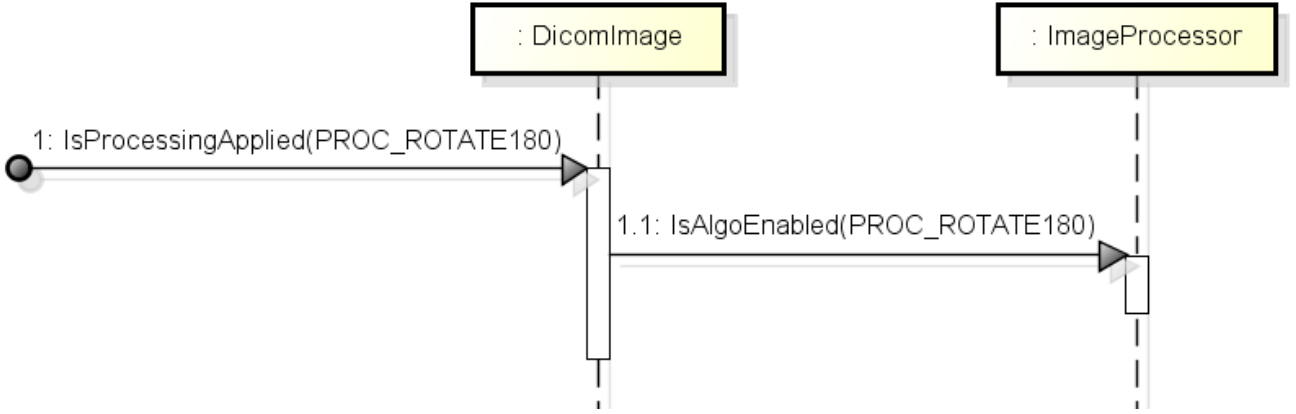
Client usage

```
int ImageBoxBase::GetViewRotation() const {
  ...
  int rotation = 0;
  if (image_model_->IsRotate180Applied()) {
    rotation += 180;
  }
  if (image_model_->IsRotate90rApplied()) {
    rotation += 90;
  }
  ...
}
```

Similarily, *IsMirrorApplied()* is provided.

Is <Algo> Applied

The old style



Is <Algo> Applied (cont.)

The new style

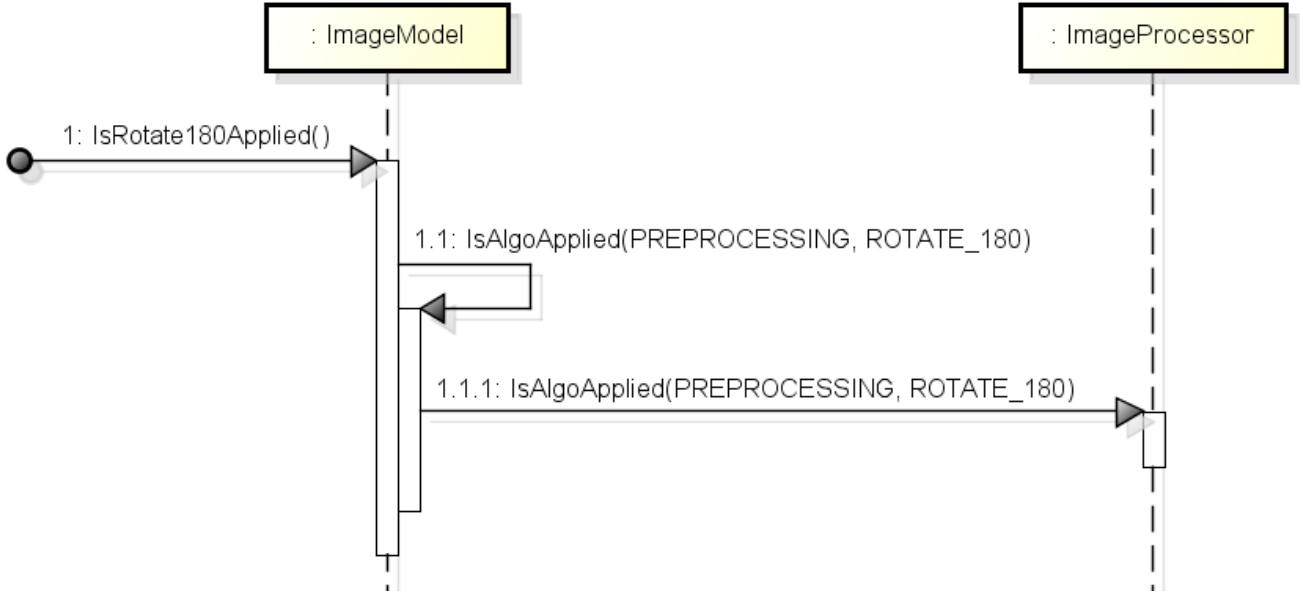


Image Model Methods

The class providing processing methods to the client should be *ImageModel* instead of *DicomImage*.

And the methods should be easy to use.

General Idea

```
bool Is<Algo>Available() const;
bool Is<Algo>Applied() const;
bool Apply<Algo>(bool apply[, int param], bool notify);
```

Image Model Methods (cont.)

Mirror

```
bool IsMirrorAvailable() const;
bool IsMirrorApplied() const;
bool ApplyMirror(bool apply, bool notify);
```

Emboss filter

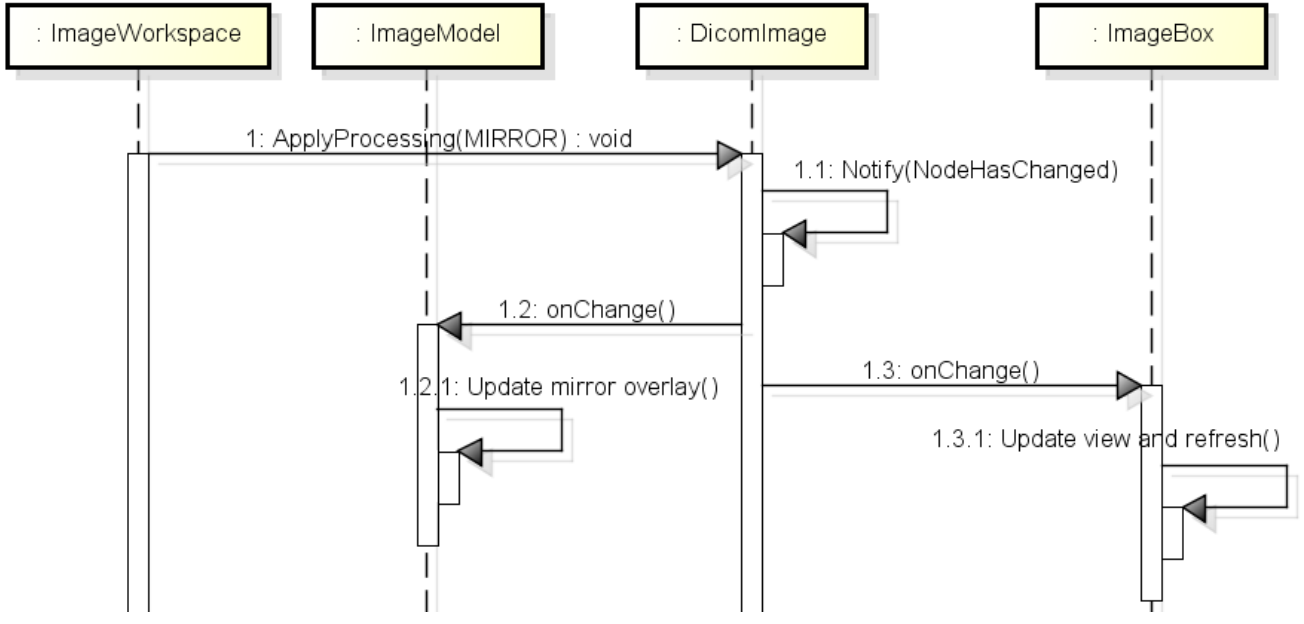
```
bool IsEmbossFilterAvailable() const;
bool IsEmbossFilterApplied() const;
bool ApplyEmbossFilter(bool apply, bool notify);
```

Color

```
bool IsColorsAlgoAvailable(AlgoId algo_id) const;
bool IsColorsAlgoApplied(AlgoId algo_id) const;
bool ApplyColorsAlgo(AlgoId algo_id, bool apply, int param, bool
notify);
```

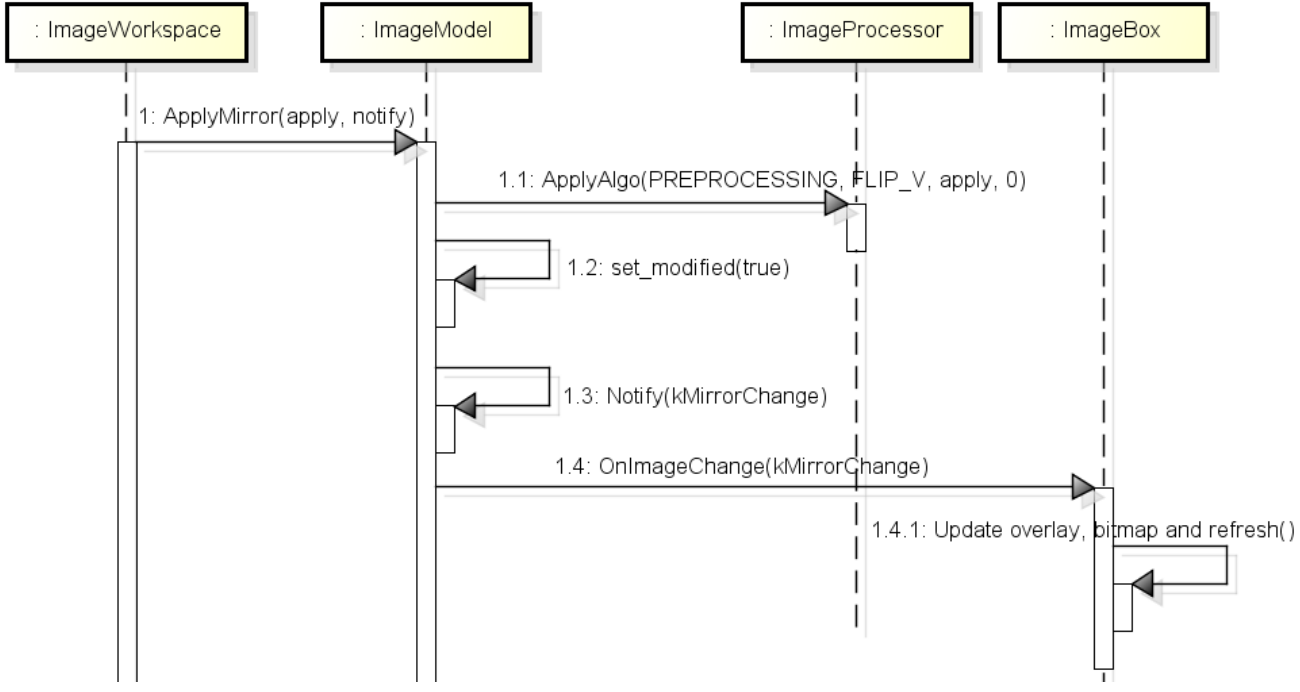
Model View

The old style



Model View (cont.)

The new style



The Annotation Listeners

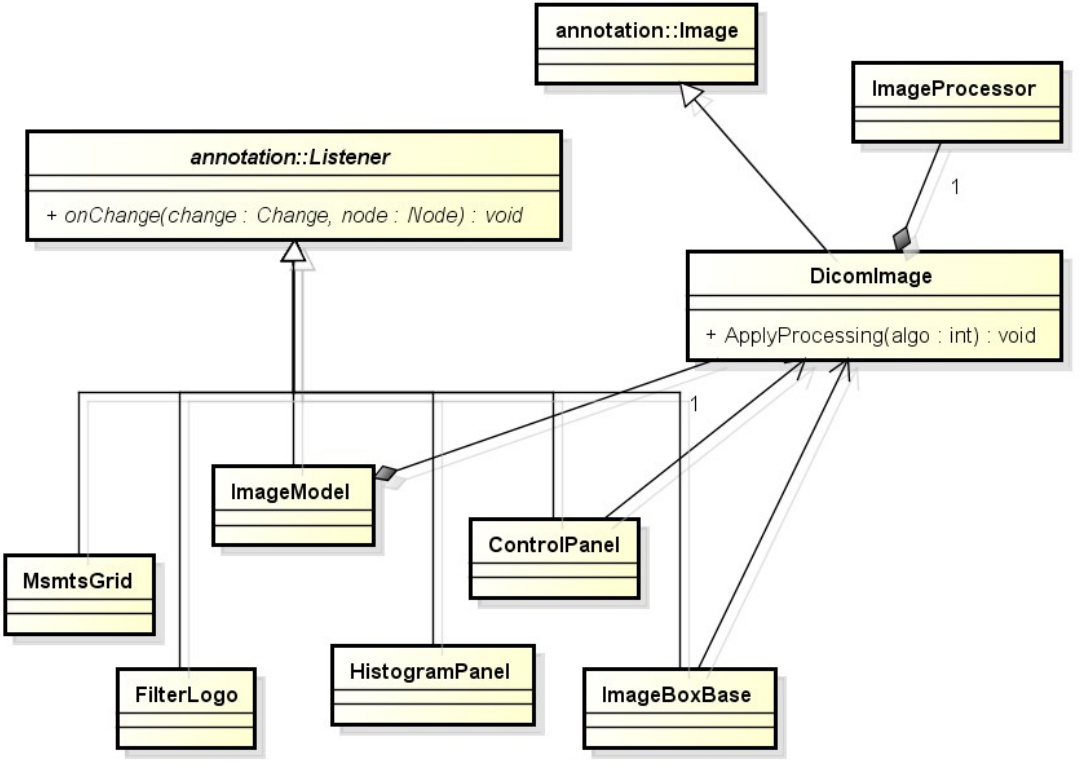


Image Listener

Derive from this class to listen to image changes from image model.

```
class ImageListener {
public:
  virtual ~ImageListener();

  virtual void OnImageChange(int change) = 0;

  void AddImage(ImageModel* image_model);
  void RemoveImage(ImageModel* image_model);

private:
  // Listening image models.
  std::set<ImageModel*> image_models_;
};
```

Image Listener (cont.)

Image changes

```
class ImageModel {
  enum ChangeType {
    kLoaded = 0,  // 举 loading animation 的例子
    kSaved,
    kInfoChange,      // Tooth numbers, acq date, etc.
    kCalibrationChange,
    kMirrorChange,
    kRotate90rChange,  // 为什么区分两种 rotate?
    kRotate180Change,
    kCropChange,
    kProcessingChange,  // 还可以再细分
    kPseudo3DChange,
    kRefreshChange,  // Let views to refresh.
  };
};
```

Image Processor

***ImageProcessor* provides a minimal wrapper for Processing 2D SDK.**

No *GetImageType*, *GetFamilyString* or *GetSubFamilyString*.

```
std::string ImageProcessor::GetImageType() const {
  if (!IsDllOk()) {
    return "";
  }
  const char* const image_type = GetStringValue(IMAGEINFO_TYPE);
  return image_type == NULL ? "" : image_type;
}
```

No *IsMirrorAvailable*, *IsMirrorApplied* or *ApplyMirror*.

Change int to bool for return value, combine width/height to wxSize. Use wxString for input file path instead of const char*.

Image Show & Move

***Show* and *Move* shouldn't be methods of *ImageBox*.**

Show

```
bool ImageBoxBase::Show(bool show = true) {
  if (image_window_ != NULL) {
    return image_window_->Show(show);
  } else {
    return wxWindow::Show(show);
  }
}
```

```
bool ImageBoxBase::ShowSelf(bool show = true) {
  return wxWindow::Show(show);
}
```

Image Show & Move (cont.)

Move

```
void ImageBoxBase::Move(const wxRect& rect, bool show) {
  ImageWindow* image_window = dynamic_cast<ImageWindow*>(image_window_);
  if (image_window == NULL) {
    return;
  }
  ...
}
```

```
void ImageBoxBase::Move(const wxPoint& pt) {
  ImageWindow* imgWnd = dynamic_cast<ImageWindow*>(image_window());
  if (NULL != imgWnd) {
    return imgWnd->Move(pt);
  } else {
    return wxWindow::Move(pt);
  }
}
```

Image Show & Move (cont.)

The old style

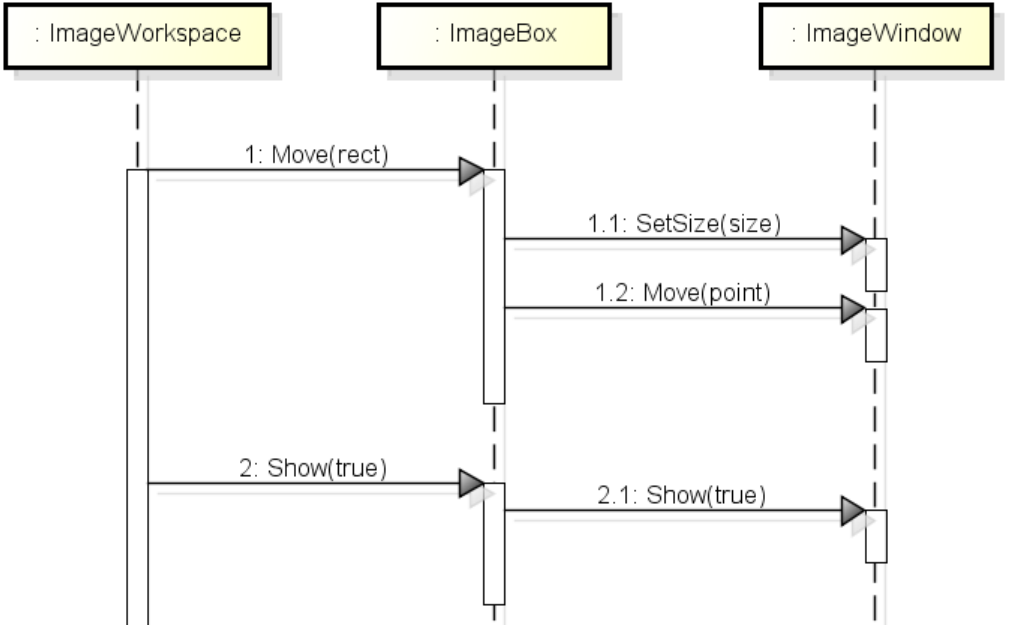
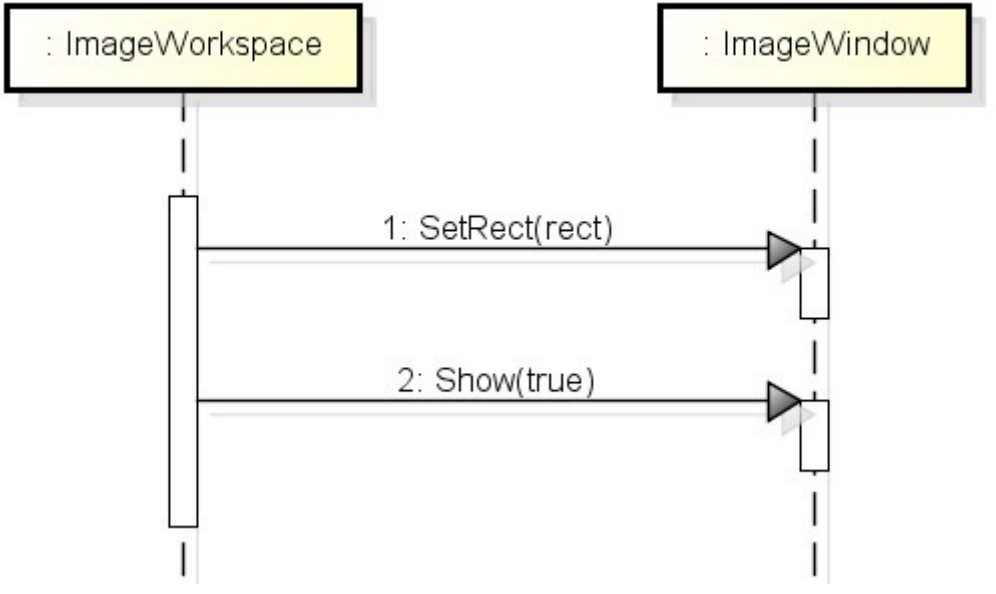


Image Show & Move (cont.)

The new style



Inside Looking Out

Inside looking out is NOT a model property.

And general XML shouldn't care about "mirrortoolstatus".

The old style

```
class ImageModel {
private:
  bool inside_looking_out_;
  bool mirror_tool_status_;
};
```

```
bool ImageModel::GetMirrorStatus() {
  bool bInsideLookingOut = inside_looking_out_ &&
CanApplyInsideLookingOut();

  return mirror_tool_status_^bInsideLookingOut;
}
```

Inside Looking Out (cont.)

The new style

```
class ImageBoxBase {
private:
  bool inside_looking_out_;
};
```

```
void ImageBoxBase::UpdateCanvasMirror() {
  m_canvas->setMirror(image_model_->IsMirrorApplied() ^
GetInsideLookingOut());
}
```

```
void ImageBoxBase::HandlePaint(...) {
  if (GetInsideLookingOut()) {
    wxImage image = bitmap_.ConvertToImage().Mirror();
    dc.DrawBitmap(wxBitmap(image), x, y, false);
  }
```