Computational Vision, Fall 1999
Programming in TargetJr

From a software viewpoint, TargetJr is organized as a collection of packages and libraries within packages. Libraries are dynamically linked so that changes to add a function and a menu item are isolated. If you have not written event-driven programs before, you should pay extra attention to the examples.

The following discussion will be covered relatively quickly in class. A more thorough understanding will be obtained by examining the directories and source code in light of what's written here.

Packages

Here are the packages (modules) we will be working with at first. Several more will be introduced and used soon.

**Image**: Classes for reading, writing, displaying, and processing images.

**GeneralUtility**: Just as the name says. Our most commonly used library within this package will be **Basics**.

**COOL**: A container class library, plus more.

**Parmesan**: The graphical user interface package, based on *Fresco*. (TargetJr used to be based on *Interviews*. Avoid the package *Viewing*.)

**Numerics**: Numerical algorithms, plus classes to represent mathematical objects such as vectors and matrices.

**SpatialObjects**: Geometric and topological objects and associated algorithms.

**Smallg++**: Utilities to handle compilation issues for a variety of compilers. The version of STL used in TargetJr resides here.

**RPICompVision**: The package, including the menu choices, executable, and software, developed for this class. All code that you write for projects will be stored in your own copy of this package.

Working with images

- The class **Image** stores images and their attributes. These images may or may not be in memory. **Image** is defined in

  `$/SYS_IUPACKAGES/Image/ImageClasses/Image.hC`

- **Image** objects are primarily used for displaying images through the user interface and for input/output. It is possible to manipulate the internals of **Image** objects directly, but not efficiently.
• BufferXY stores memory buffers of byte, short, color, or floating point images. BufferXY is defined in

\$SYS_IUPACKAGES/Image/ImageClasses/bufferxy.[hC]

• The algorithms we write will operate on BufferXY’s, but the callback functions will have to handle Image’s.

User interface managers

• Menu callback functions are part of “user interface” (UI) managers.
• Multiple UI managers may be active at once, but only one of each type.
• Each has (should have) a collection of methods tied to a particular set of algorithms or objects.
• Each must be an immediate subclass of UITaskManager. UITaskManager is defined in

\$SYS_IUPACKAGES/Parmesan/Fresh/UITaskManager.[hC]

COOL and STL

Many TargetJr algorithms use COOL — “Common Object-Oriented Library”.

• Like STL, it is a template library.
• It includes all the types of container classes you might expect and a fair amount more, such as timers and random number generators.
• Several of the container classes have specializations for efficiency, such as lists of pointers.
• On the other hand, computational efficiency can be a problem if you aren’t careful. For example, finding the length of a CoolList is a linear time computation.
• STL is also available, and I find it cleaner and more natural to use.
• COOL is needed, however, to interface with existing TargetJr algorithms.
• Regardless of whether COOL or STL is used, template instantiation can be tricky. It is not done automatically (compiler option), so new template objects must be explicitly instantiated (see below).
Connected components examples

As an illustrative example, I have written a connected components class and callback function. The code is documented to explain features of TargetJr. Here are things to watch for:

- Organizing the code to separate the user interface from the algorithm functionality.
- Working with menus.
- Accessing and working with the image.
- Creating a new image.
- Example uses of STL and of COOL.
- Displaying the results.

Other example code will be available soon!

Organization of the files, directories and libraries

In $SYS_IUPACKAGES/RPICompVision

**BinaryAlgorithms**: The algorithm source code (ConnectedComponents), the user interface manager containing the callbacks (BinaryAlgorithmUI), and the menus (BinaryMenu).

**BinaryAlgorithms/Templates**: These are the template instantiations for the templates needed in the algorithms. Note that template instantiations are not needed for list<int>.

**BinaryAlgorithms/CVDemo**: The main menus and executable. You should only have to change main.c if you need to add a pulldown menu across the top of the TargetJr display. The same goes for menus.h and menus.c

Makefiles and compilation

Makefiles and compilation are well scripted, so that you don’t need to setup include directories, environment variables, and compiler controls. Here’s what you do need to have in your makefiles.

- In a package directory, such as RPICompVision, you need to indicate library (algorithm) directories and top directories, which contain executables.

- In a library directory, such as BinaryAlgorithms, which is mostly where you will be working, you need to indicate the packages you will work with — via the USES statement — and the source files — via the SOURCES statement. (You also need a dll.h file, such as the one in the directory to keep the Microsoft compilers happy, but this is a minor issue.)
You do NOT need to indicate anything about template instantiations. These are done automatically by the compiler by looking in and compiling what’s in the Templates subdirectory.

- Makefiles in the executable directory are somewhat more complicated.
  - They need to have the list of packages, the name of the executable to build, and the name of the source files.
  - What’s more difficult, the libraries to link must be specified.
  - Many libraries are already there, but you may need to add some. The best way to figure out what’s needed is through class_apropos on the names of objects found to be missing when the program links.

**General Comments and Suggestions**

- Don’t try to learn everything! Work from examples and search for understanding of particular issues.
- Build and test programs incrementally, especially until you understand TargetJr better.
- Use class_apropos. Remember, however, that it does not find everything.
- Read the header files and even the source code.

**Homework**

This is due Monday, September 13th at the start of class.

1. cd into your copy of RPICompVision, which I assume you have already checked out and compiled. See the setup instructions handed out in the first class.

2. Type

   ```bash
cvs update -d -P
   ```

   to get any changes I might have made. Recompile if necessary.

3. To see what happens without templates, cd into BinaryAlgorithms and change the name of the Templates directory. Then cd back up into RPICompVision and type

   ```bash
   make clean; make
   ```
Where does the compilation fail? Think about this example when you might need to create your own templates. Restore the Templates directory.

4. Edit the makefile in CVDemo by removing the string -lBasics. Save the changes. Then cd into RPICompVision and type

```
make clean; make
```

Where does the compilation fail? Type

```
class_apropos foo
```

where foo is the name of a class having an “undefined reference”. Note the “Link line;” output. Think about this example when you might need to add new libraries. Restore the makefile.

5. Add a callback function to calculate the center of mass of each component whose size is above a user-specified minimum size. Print the coordinates (via cout), and display each center as an IUPoint (look in SpatialObjects/Geometry). You will need to study the

```
BinaryAlgorithmUI::ComponentBoundingBoxes
```

and

```
ConnectedComponents::BoundingRectangles
```

eexamples carefully. Turn in print outs of your source code, but, to keep my life easier, include only the new methods that you added rather than the complete classes.