

SWT: The Eclipse Standard Widget Toolkit

Carolyn MacLeod
IBM Rational Software
SWT Committer

Grant Gayed
IBM Rational Software
SWT Committer

Getting started

1. Copy “workspace” from appropriate platform directory to your hard disk (i.e. “copy e:\win32\workspace c:\”)
2. Change file permissions
 - On Windows, open a DOS shell and type:
`attrib -R c:\workspace /S /D`
 - On Linux or Macintosh, open a Terminal window and type:
`chmod -R +w /home/user/workspace`
3. Run “eclipse -data c:\workspace”

You are now using Eclipse like an SWT Developer!

What is SWT?

Standard Widget Toolkit: A GUI Toolkit for Java

“The SWT component is designed to provide efficient, portable access to the user-interface facilities of the operating systems on which it is implemented.”

From the “SWT Home Page”, eclipse.org/swt

A bit of history

- Object Technology International (OTI)
 - Virtual Machines, Class Libraries, IDEs, Embedded
 - Configuration Management (ENVY/Manager)
 - Acquired by IBM in 1996
- Smalltalk
 - ParcPlace (ObjectShare), Digitalk, IBM/Smalltalk (OTI)
 - Emulated Widgets vs. Native Widgets
- VisualAge for Java
 - Written in IBM/Smalltalk using CommonWidgets
- VisualAge Micro Edition
 - Written in Java using SWT (port of CommonWidgets)
- Eclipse

The basics: Display, Shell, Composite & Control

- Display
 - Represents “a workstation” (monitor(s), keyboard, mouse)
 - Responsible for event dispatching in the “event loop”
 - Contains a list of top-level Shells, and a list of Monitors
- Shell
 - Represents “a window” on the screen
 - Root of a tree composed of Composites & Controls
- Composite
 - Control that can contain other Composites & Controls
- Control
 - Represents a “heavyweight” operating system control
 - Button, Label, Text, Tree, etc – as well as Shell & Composite

Shell, Composite & Control are subclasses of *Widget*

The event loop

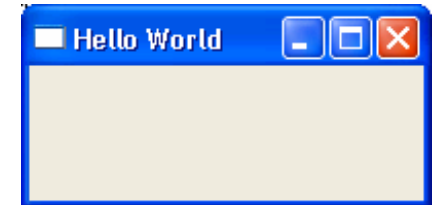
- In SWT, the event loop is explicit: it must be coded by the application
- The loop repeatedly reads and dispatches the next UI event from the OS, and yields the CPU when there are no events to dispatch
- The loop completes when the application finishes – typically when the user closes the application's main window

```
while (!shell.isDisposed ()) {  
    if (!display.readAndDispatch ())  
        display.sleep ();  
}
```

HelloWorld

```
1 import org.eclipse.swt.*;
2 import org.eclipse.swt.graphics.*;
3 import org.eclipse.swt.widgets.*;

4 public class HelloWorld {
5     public static void main(String[] args) {
6         Display display = new Display();
7         Shell shell = new Shell(display);
8         shell.setText("Hello, World!");
9         shell.setSize(200, 100);
10        shell.open ();
11        while (!shell.isDisposed()) {
12            if (!display.readAndDispatch())
13                display.sleep ();
14        }
15        display.dispose ();
16    }
17 }
```




SWT architecture: java packages

- **swt.jar** contains the following Java packages 
 - org.eclipse.swt
 - org.eclipse.swt.widgets
 - org.eclipse.swt.graphics
 - org.eclipse.swt.events
 - org.eclipse.swt.layout
 - org.eclipse.swt.dnd
 - org.eclipse.swt.printing
 - org.eclipse.swt.program
 - org.eclipse.swt.accessibility
 - org.eclipse.swt.custom
 - org.eclipse.swt.browser
 - org.eclipse.swt.awt
 - org.eclipse.swt.internal.*

SWT architecture: platform interface

- Uses Java Native Interface (JNI) to call the C-based API of the underlying platform's UI library
- One-to-one mapping
 - Uses the same function names and arguments as the platform (typically only a subset of the platform API is needed)
 - Uses the same naming conventions for data structures, constants, and variables
- All of the interesting aspects of SWT are coded in Java
- Developers familiar with the API for a platform can develop an SWT port using their full knowledge of the platform code

SWT architecture: CVS source folders

- Code is arranged into component folders 
- Further organized in common and platform-specific sub-folders
- Classpath specifies which folders and sub-folders are included

Folders:

- Eclipse SWT
- Eclipse SWT PI
- Eclipse SWT Custom Widgets
- Eclipse SWT Drag and Drop
- Eclipse SWT Printing
- Eclipse SWT Program
- Eclipse SWT Accessibility
- Eclipse SWT Browser
- Eclipse SWT AWT
- Eclipse SWT Mozilla (Linux only)
- Eclipse SWT OLE Win32 (Windows only)

Sub-folders:

- carbon
- common
- common_j2me
- common_j2se
- emulated
- gtk
- motif
- motif_gtk
- mozilla
- photon
- win32

SWT architecture: jars and shared libraries

- In order to run an SWT application, you need 2 sets of components:
 1. SWT Jars
 - Contain the Java classes that make up the SWT library
 - The main file, **swt.jar**, has its source code in **swtsrc.zip**
 - The set of additional jar files depends on the platform
 - In order to compile your SWT application, these jar files must be available to the java compiler
 2. SWT shared libraries
 - Contain the SWT Platform Interface (PI), which makes the platform's GUI API available in Java
 - Shared library filenames are platform-specific, for example: **swt-XX.dll** on Windows, **libswt-XX.so** on Linux, etc
 - The set of additional shared library files depends on the platform

Hands-on 1: running SWT

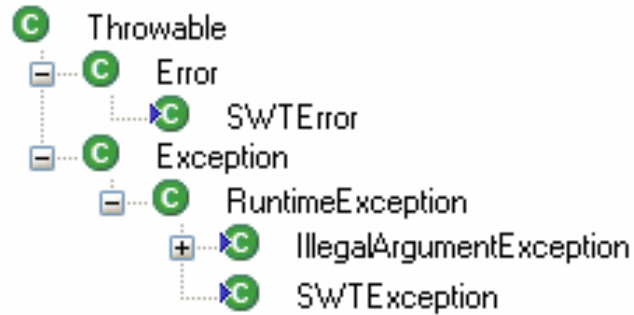
1. Configure the SWT classes for your platform
 - Copy `.classpath_<platform>` to `.classpath`
2. Put SWT shared library on `java.library.path`
 - Window -> Preferences – Java – Installed JREs
 - Select JRE and Edit... “Default VM Arguments” (e.g. on Win):
`-Djava.library.path=C:\workspace\org.eclipse.swt.win32\os\win32\x86`
3. Inside project “SWT OOPSLA Tutorial”
 - Run `examples.template.HelloWorld`



The class SWT

- All constants for SWT
- Examples:
 - SWT.PUSH, SWT.RADIO
 - SWT.Selection
- Interesting methods
 - `getPlatform()`
 - `getVersion()`
 - `error()`
- Notes
 - Platform codes are Strings such as “win32”, “carbon”, “gtk”
 - Version is encoded in an integer (major * 1000 + minor)

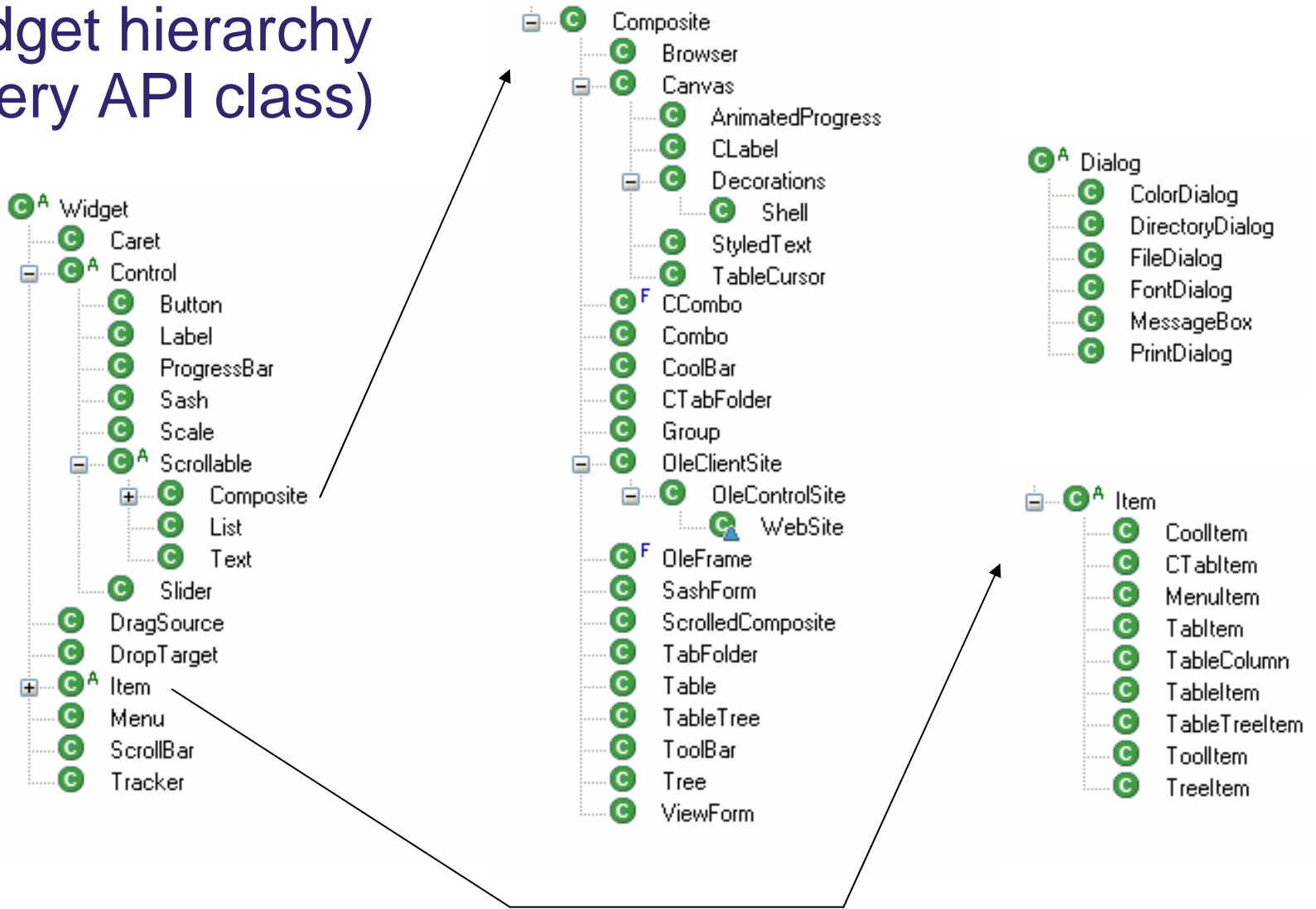
Errors & exceptions



SWT methods throw only three kinds of exceptions and errors:

- **SWTErrors**
 - Thrown when an *unrecoverable* error occurs internally in SWT
 - int code – the SWT error code
 - Throwable throwable – the throwable that caused the problem, or null
 - String getMessage() – text description of the problem
- **SWTErrors**
 - Thrown when a *recoverable* error occurs internally in SWT
 - Same fields and methods as SWTErrors
- **IllegalArgumentException**
 - Thrown with descriptive string when argument is invalid

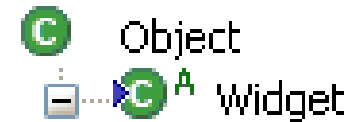
Widget hierarchy (every API class)



Widget constructors & style bits

- Widgets always have a parent
- Typical constructor looks like: `Widget(Composite parent, int style)`
- Styles are the bitwise-or constants from class SWT
- Examples:
 - `new Label(shell, SWT.NONE);`
 - `Button push = new Button(shell, SWT.PUSH);`
 - `Button radio = new Button(parent, SWT.RADIO);`
 - `Text text = new Text(group, SWT.SINGLE | SWT.BORDER);`
- Exceptions: shells have a display or shell parent
 - `Shell shell = new Shell(display, SWT.SHELL_TRIM);`
 - `Shell dialog = new Shell(shell, SWT.DIALOG_TRIM);`

Widget



- Abstract superclass for **all** UI objects
- Created using constructors (no “factories”)
 - OS resources are allocated when a widget is created
- Disposed by the user or programmatically with `dispose()`
 - OS resources are released when a widget is disposed
- Notify listeners when widget-specific events occur
- Allow application-specific data to be stored using
 - `setData(Object)`
 - `setData(String, Object)`
- Event
 - `Dispose`

Disposing widgets & graphics resources

- You must explicitly dispose of resource-based objects
- Resource-based objects: Widget & subclasses, Color, Cursor, Font, GC, Image, Region, Device & subclasses (Display, Printer)
- **Rule 1: “If you created it, you dispose it”**
 - Programmer is responsible for disposing this font:

```
Font font = new Font (display, "Courier", 10, SWT.NORMAL);  
font.dispose ();
```
 - Programmer must not dispose this font:

```
Font font = control.getFont ();
```

Disposing widgets & graphics resources

- **Rule 2: “*Disposing a parent disposes the children*”**
 - `shell.dispose()`; disposes all children of the shell
 - `menu.dispose()`; disposes all menu items in the menu
 - `tree.dispose()`; disposes all items in the tree
 - Also, note that:
 - `control.dispose()`;
 - `menuItem.dispose()`;
 - disposes the menu that was set into the control or menuItem with `setMenu(menu)`;

See “Managing Operating System Resources” on eclipse.org

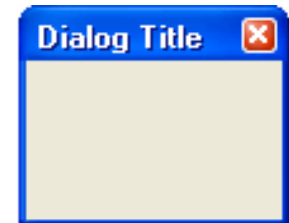
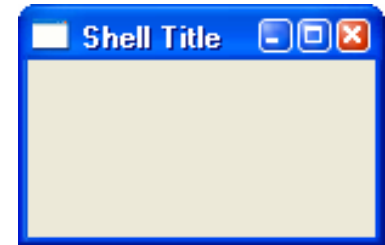
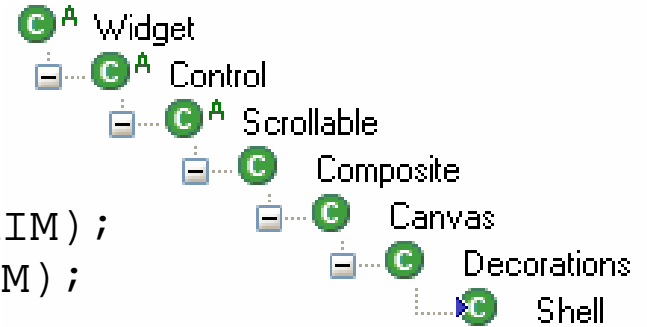
Control



- Abstract superclass of all **heavyweight** UI objects
- Styles
 - BORDER, LEFT_TO_RIGHT, RIGHT_TO_LEFT
- Events
 - FocusIn, FocusOut
 - KeyDown, KeyUp
 - Traverse
 - MouseDown, MouseUp, MouseDoubleClick
 - MouseEnter, MouseExit, MouseMove, MouseHover
 - Move, Resize
 - Paint
 - Help

Shell

- **new** `Shell(display, SWT.SHELL_TRIM);`
- **new** `Shell(shell, SWT.DIALOG_TRIM);`
- Styles (most Shell styles are hints to WM)
 - BORDER, CLOSE, MIN, MAX, NO_TRIM, RESIZE, TITLE
 - APPLICATION_MODAL, MODELESS, PRIMARY_MODAL, SYSTEM_MODAL
- Events
 - Close, Activate, Deactivate, Iconify, Deiconify
- Interesting methods
 - `open()`, `close()`, `setActive()`
- Notes
 - The parent of a top-level shell is the display
 - The parent of a dialog shell is a top-level shell



Label

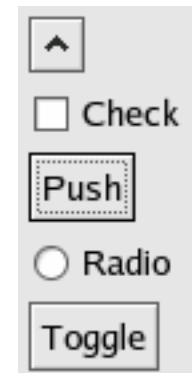


- **new** `Label(parent, SWT.NONE);`
- **Styles**
 - WRAP, LEFT, CENTER, RIGHT, SEPARATOR, HORIZONTAL, VERTICAL, SHADOW_IN, SHADOW_OUT
- **Interesting methods**
 - `setText(String)` – use & in String to set mnemonic
 - `setImage(Image)`
 - `setAlignment(SWT.LEFT or CENTER or RIGHT)`
- **Notes**
 - Static UI element that displays text, or an image, or a separator
 - Does not take focus or participate in tab traversal



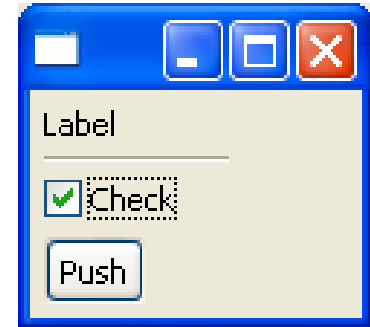
Button

- **new** Button(parent, SWT.PUSH);
- **Styles**
 - ARROW, CHECK, PUSH, RADIO, TOGGLE, FLAT, UP, DOWN, LEFT, CENTER, RIGHT
- **Events**
 - Selection
- **Interesting methods**
 - setText(String) – use & in String to set mnemonic
 - setImage(Image)
 - setAlignment(SWT.LEFT or CENTER or RIGHT)
 - setSelection(boolean) for check, radio, toggle buttons
- **Notes**
 - Radio buttons can be grouped for ‘radio behavior’
 - Arrow buttons have UP, DOWN, LEFT, RIGHT alignment



Hands-on 2: constructors and styles

- Inside project “SWT OOPSLA Tutorial”
 - Edit examples.template.StyleExample
 - Create two Labels and two Buttons
 - Use SWT.SEPARATOR, SWT.CHECK, SWT.PUSH
 - See Javadoc for the class

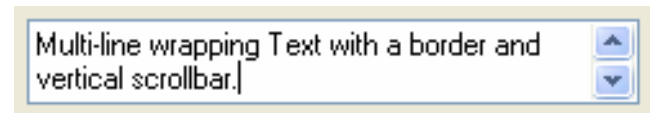
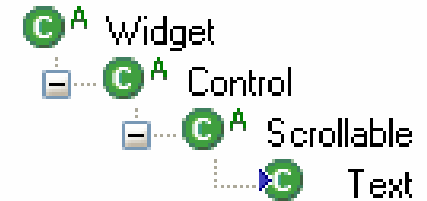


```

Display display = new Display ();
Shell shell = new Shell (display);
shell.setLayout (new GridLayout ());
...your code here ...
shell.pack ();
shell.open ();
while (!shell.isDisposed ()) {
    if (!display.readAndDispatch ()) display.sleep ();
}
display.dispose ();
    
```

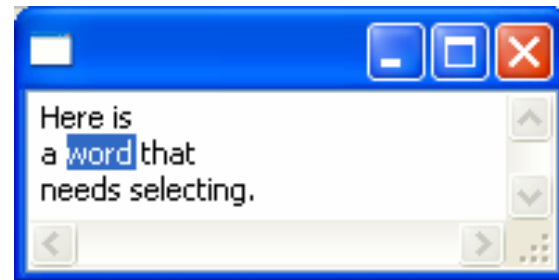
Text

- `new Text(parent, SWT.MULTI | SWT.BORDER | SWT.WRAP | SWT.V_SCROLL);`
- **Styles**
 - SINGLE, MULTI, READ_ONLY, WRAP, LEFT, CENTER, RIGHT, PASSWORD
- **Events**
 - Modify, Verify, DefaultSelection (SINGLE only)
- **Interesting methods**
 - setText(String)
 - setSelection(int, int)
 - cut(), copy(), paste()
 - insert(String), append(String)
 - getLineCount(), getLineHeight()
- **Notes**
 - Caret marks current text entry point
 - Indexing is 0-based, and ranges are inclusive



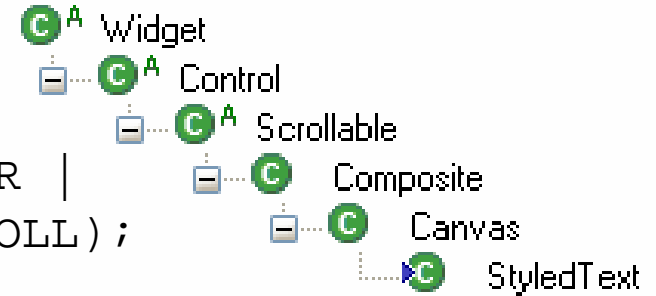
Programmatically select a word in a Text

```
Text text = new Text(shell, SWT.MULTI | SWT.BORDER |
    SWT.V_SCROLL | SWT.H_SCROLL);
text.setText("Here is\na word that\nneeds selecting.");
shell.setSize(200, 100);
shell.open();
String string = "word";
int index = text.getText().indexOf(string);
if (index != -1) {
    text.setSelection(index, index + string.length());
}
```



StyledText

- **new** `StyledText(parent, SWT.BORDER | SWT.MULTI | SWT.WRAP | SWT.V_SCROLL);`
- **Styles**
 - SINGLE, MULTI, READ_ONLY, WRAP, FULL_SELECTION
- **Events**
 - DefaultSelection, Modify, Verify, ExtendedModify
- **Interesting action methods**
 - `invokeAction(int action)` // actions are ST constants
 - `setKeyBinding(int key, int action)`



```

// StyledText with a simple LineStyleListener for Java
Font font = new Font(display, "Courier", 10, SWT.NORMAL);
StyledText styledText = new StyledText(shell, SWT.BORDER
    | SWT.MULTI | SWT.V_SCROLL | SWT.H_SCROLL);
styledText.setFont(font);
styledText.addLineStyleListener(new JavaLineStyler(display));
    
```

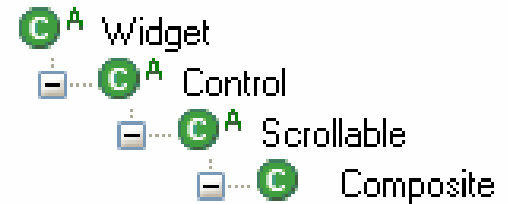
StyledText (continued)

- Methods for setting content and style statically
 - `setText(String)`
 - `setLineBackground(int, int, Color)`
 - `setStyleRanges(StyleRange[])`

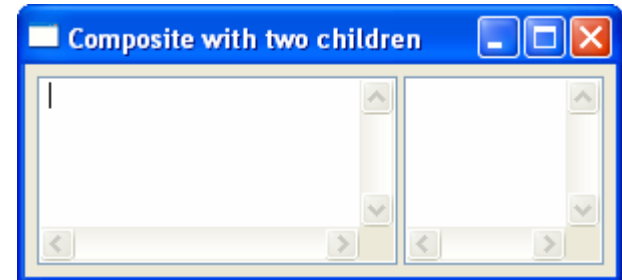
- Methods for setting content and style dynamically
 - `setContent(StyledTextContent)`
 - `addLineBackgroundListener(LineBackgroundListener)`
 - `addLineStyleListener(LineStyleListener)`

- Notes
 - TextEditor example
 - JavaViewer example
 - 2 StyledText articles on eclipse.org

Composite



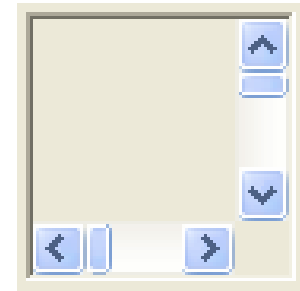
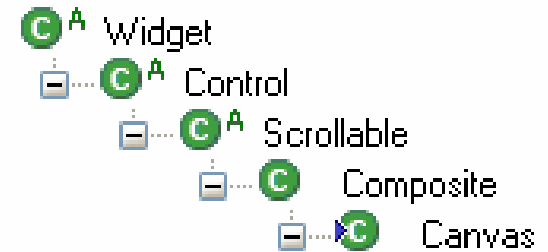
- **new** `Composite(parent, SWT.NONE);`
- **Styles**
 - `NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE, NO_RADIO_GROUP`
- **Interesting methods**
 - `getChildren()`
 - `setLayout(Layout)`
 - `layout(boolean)`
 - `setTabList(Control[])`
- **Notes**
 - Can have child controls
 - Can use a `Layout` to position children, or position on resize
 - Can be subclassed to implement a custom widget



Canvas

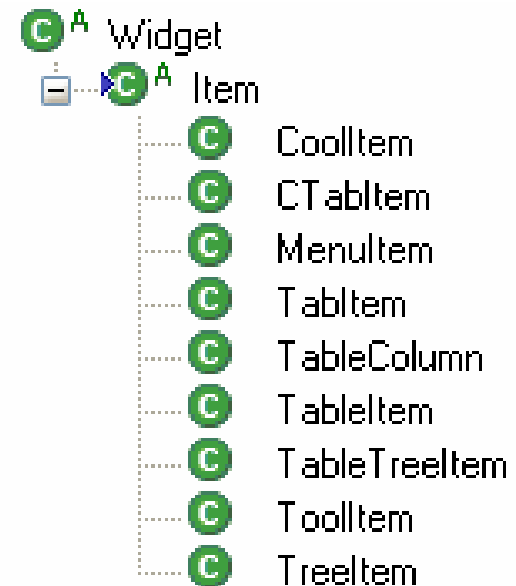
```
new Canvas(parent, SWT.NONE);
```

- Important styles inherited from Composite
 - NO_BACKGROUND, NO_FOCUS, NO_MERGE_PAINTS, NO_REDRAW_RESIZE
- Interesting methods
 - scroll(int, int, int, int, int, int, boolean)
 - setCaret(Caret)
- Notes
 - Typically used as a “blank slate” for drawing with graphics calls
 - Can be subclassed to implement a custom widget

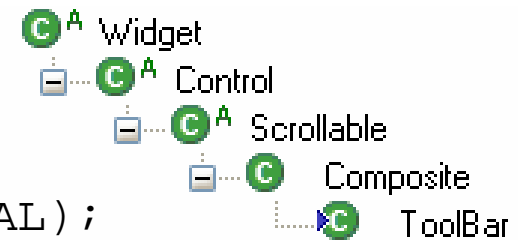


Item

- Abstract superclass of **lightweight** UI objects
- Always occur within a specific widget
 - A Tree contains TreeItems
 - A Menu contains MenuItem
- Typical constructor forms:
 - Item (Widget parent, **int** style)
 - Item (Widget parent, **int** style, **int** index)
- Interesting methods
 - setText(String)
 - setImage(Image)



ToolBar



- **new** `ToolBar(parent, SWT.HORIZONTAL);`
- **Styles**
 - WRAP, FLAT, RIGHT (text to the right of image)
 - HORIZONTAL, VERTICAL
 - SHADOW_OUT
- **Events**
 - None
- **Interesting methods**
 - `indexOf(ToolItem)`
 - `getItems(), getItem(int)`
- **Notes**
 - Display horizontal or vertical rows of ToolItems

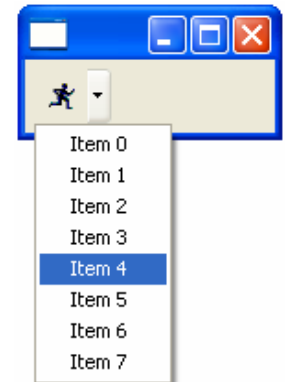


ToolItem

- **new** ToolItem(toolBar, SWT.PUSH);
- Styles
 - PUSH, CHECK, RADIO, SEPARATOR, DROP_DOWN
- Events
 - Selection
- Interesting methods
 - setText(String), setImage(Image), setSelection(boolean), setToolTipText(String), setEnabled(boolean)
 - SEPARATOR style: setWidth(int), setControl(Control)
 - DROP_DOWN style: Selection event with detail == SWT.ARROW, and x, y to position drop-down control (i.e. menu or list or toolbar)
- Notes
 - Similar API to Button

Display a menu in a drop-down tool item

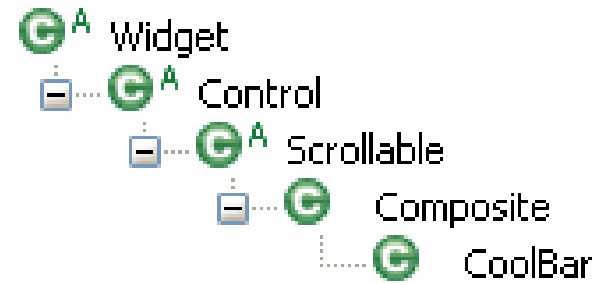
```
final ToolBar toolBar = new ToolBar(shell, SWT.HORIZONTAL);  
final ToolItem item = new ToolItem(toolBar, SWT.DROP_DOWN);  
item.setImage(createToolIcon(display, "run"));  
  
final Menu menu = new Menu(shell, SWT.POP_UP);  
for (int i = 0; i < 8; i++) {  
    new MenuItem(menu, SWT.PUSH).setText("Item " + i);  
}  
item.addSelectionListener(new SelectionAdapter() {  
    public void widgetSelected(SelectionEvent event) {  
        if (event.detail == SWT.ARROW) {  
            Point point = new Point(event.x, event.y);  
            point = display.map(toolBar, null, point);  
            menu.setLocation(point);  
            menu.setVisible(true);  
        }  
    }  
});
```



Display a menu in a drop-down tool item (2)

```
static Image createToolIcon(Display display, String name) {  
    Image result = null;  
    try {  
        InputStream stream =  
            ToolbarTest.class.getResourceAsStream (name+".gif");  
        ImageData source = new ImageData (stream);  
        ImageData mask = source.getTransparencyMask ();  
        result = new Image (display, source, mask);  
        stream.close ();  
    } catch (Exception e) {  
        e.printStackTrace ();  
    }  
    return result;  
}
```

CoolBar & CoolItem



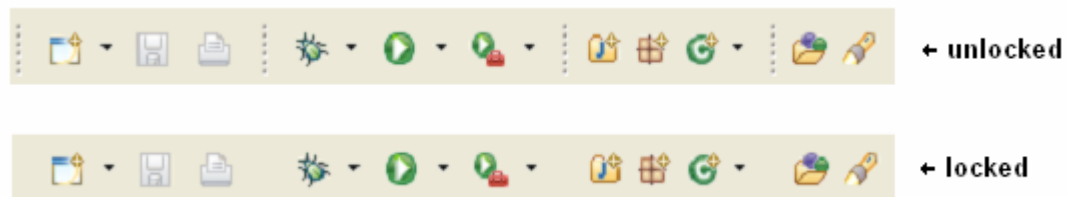
- **new** CoolBar(parent, SWT.FLAT)
- **new** CoolItem(coolBar, SWT.NONE);

- CoolBar Styles

- FLAT

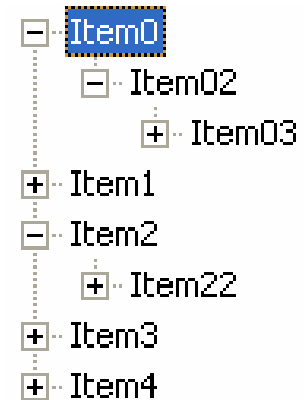
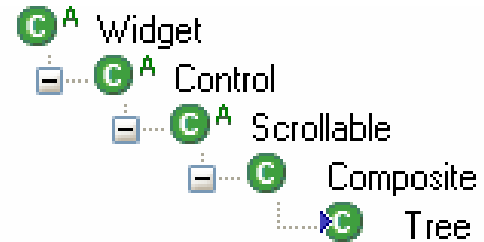
- Interesting methods

- CoolBar.setLocked(boolean)
- CoolItem.setControl(Control)

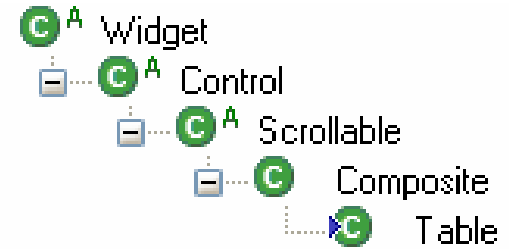


Tree & TreeItem

- **new** `Tree(parent, SWT.MULTI);`
- **new** `TreeItem(tree, SWT.NONE);`
- **new** `TreeItem(treeItem, SWT.NONE);`
- **Styles**
 - SINGLE, MULTI, CHECK
- **Events**
 - Selection, DefaultSelection, Collapse, Expand
- **Interesting methods**
 - `setSelection(TreeItem[])`
 - `setTopItem(TreeItem)`
 - `showItem(TreeItem)`
- **TreeItem**
 - Can have text, image, color, checkbox, gray-state checkbox



Table



- **new** Table(parent, SWT.SINGLE);
- Styles
 - SINGLE, MULTI, CHECK, FULL_SELECTION, HIDE_SELECTION
- Events
 - Selection, DefaultSelection
- Interesting methods
 - setHeaderVisible(boolean), setLinesVisible(boolean)
 - getHeaderHeight(), getItemHeight()
 - indexOf(TableColumn), indexOf(TableItem)
 - getColumn(int), getItem(int)
 - isSelected(int), setSelection(int[]), setSelection(TableItem[])
 - setTopIndex(int)

Col 0	Col 1	Col 2	
C0R0	C1R0	C2R0	
C0R1	C1R1	C2R1	
C0R2	C1R2	C2R2	

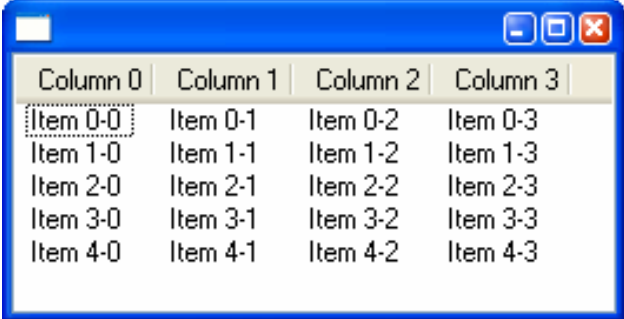
TableColumn & TableItem

- **new** TableColumn(table, SWT.LEFT);
- TableColumn
 - Can have text, image, and alignment
 - Can set column width, and pack columns
 - Generates Move, Resize, and Selection events
- **new** TableItem(table, SWT.NONE);
- TableItem
 - Can have text, image, color, checkbox, gray-state checkbox

Create a table with 4 columns and 5 rows

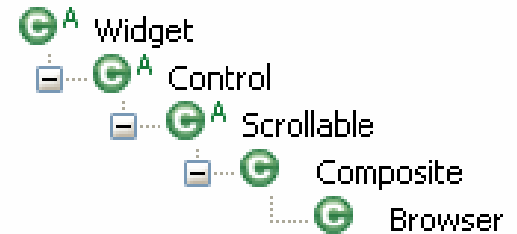
```

Table table = new Table(shell, SWT.BORDER);
table.setHeaderVisible(true);
for (int i = 0; i < 4; i++) {
    TableColumn column = new TableColumn(table, SWT.NONE);
    column.setText("Column " + i);
}
for (int i = 0; i < 5; i++) {
    TableItem item = new TableItem(table, SWT.NULL);
    for (int j = 0; j < 4; j++) {
        item.setText(j, "Item " + i + "-" + j);
    }
}
for (int i = 0; i < 4; i++) {
    table.getColumn(i).pack();
}
table.pack();
    
```

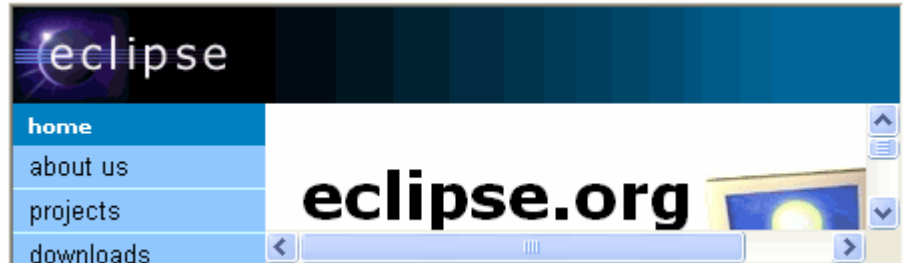


Column 0	Column 1	Column 2	Column 3
Item 0-0	Item 0-1	Item 0-2	Item 0-3
Item 1-0	Item 1-1	Item 1-2	Item 1-3
Item 2-0	Item 2-1	Item 2-2	Item 2-3
Item 3-0	Item 3-1	Item 3-2	Item 3-3
Item 4-0	Item 4-1	Item 4-2	Item 4-3

Browser



- **new** `Browser(parent, SWT.NONE);`
- Events
 - Show, Hide, Open, Close, (URL) Changed, (URL) Changing, (Progress) Changed, (Progress) Completed, (StatusText) Changed, (Title) Changed
- Interesting methods
 - `setURL(String url)`
 - `setText(String html)`
 - `back()`, `forward()`, `stop()`
- Notes
 - Wraps native HTML renderer (IE on Windows, Mozilla on Linux, Safari on OS X, Voyager on QNX)



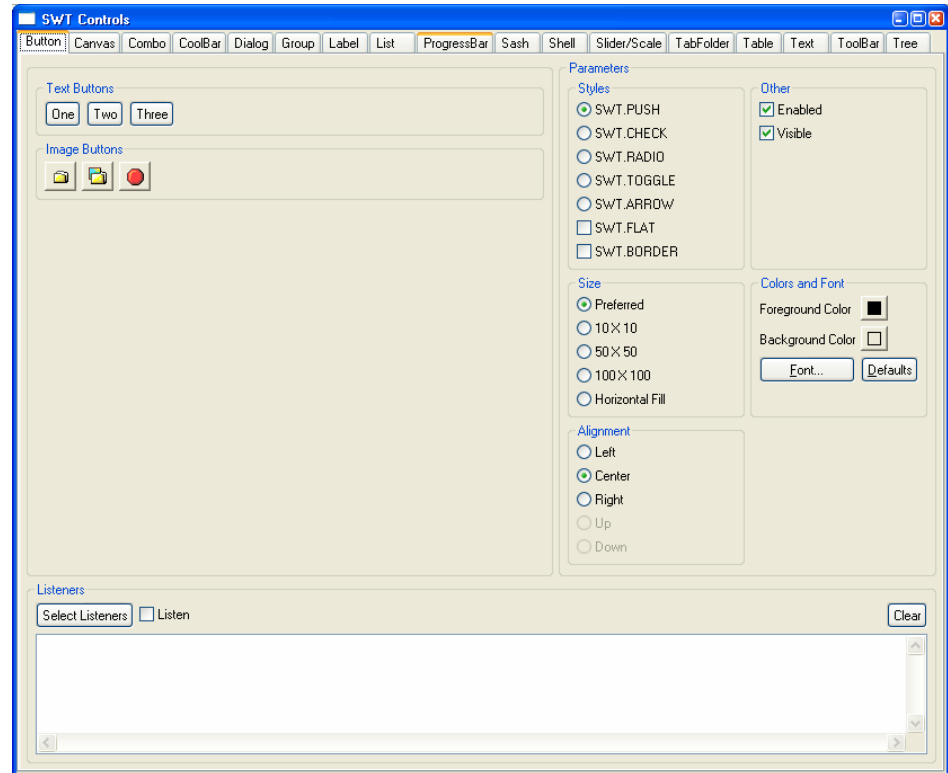
Other interesting widgets

- Combo – text widget with associated drop-down list
- Group – draws a border with an optional title around its children
- List – user selects string(s) from a column of strings
- Sash – moveable separator between two widgets
- ProgressBar – used to show incremental progress
- ScrollBar – lets the user scroll through a scrollable control
- Scale, Slider, Spinner – allow selection of a numeric value
- TabFolder & TabItem – selectable tabbed pages in a ‘notebook’
- TableTree & TableTreeItem – table with a tree in the first column
- Tracker – open() does “rubber banding”

You will see all of these (except Tracker) when running ControlExample

Hands-on 3: controls and items

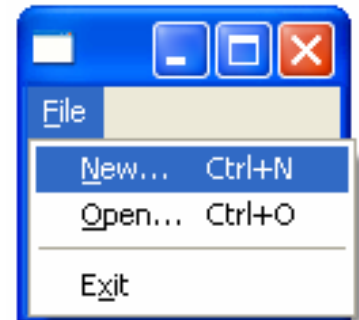
- Open Type (Ctrl+Shift+T)
 - ControlExample
- Run ControlExample (Alt+Shift+X, J)
 - Browse controls
 - Change style bits



Menu



- Menu(Decorations parent, int style)
 - Parent of menus are Decorations or Shell
- Menu(Menu menu), Menu(MenuItem item), Menu(Control parent)
 - Convenience constructors for Menu(Decorations, int)
- Styles
 - BAR, DROP_DOWN, POP_UP, NO_RADIO_GROUP, LEFT_TO_RIGHT, RIGHT_TO_LEFT
- Events
 - Help, Hide, Show
- Interesting Methods
 - setLocation(int x, int y) – tell pop-up menu where to open
 - setVisible(boolean) – to display a pop-up menu
 - Decorations.setMenuBar(Menu), Control.setMenu(Menu)

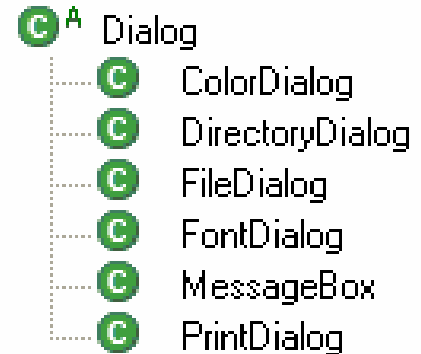


MenuItem

- MenuItem(Menu parent, int style)
- MenuItem(Menu parent, int style, int index)
- Styles
 - CHECK, PUSH, RADIO, SEPARATOR, CASCADE
- Events
 - Arm, Help, Selection
- Interesting methods
 - setText(String) – string may include mnemonic & accelerator text
 - setImage(Image) – feature not available on all platforms
 - setAccelerator(int) – SWT.MOD1 + 'T', SWT.CONTROL + 'T'
 - setEnabled(boolean) – disabled menu item typically drawn 'gray'
 - setSelection(boolean) – for radio or check menu items
 - setMenu(Menu) – for cascade menu items, sets the pull-down menu

Dialogs

- Dialog(Shell parent)
- Dialog(Shell parent, int style)
- Styles
 - PRIMARY_MODAL, APPLICATION_MODAL, SYSTEM_MODAL
- Interesting methods
 - setText(String) – set the dialog title
 - open() – returns the dialog-specific result
- Notes
 - Dialogs are not subclasses of Widget
 - Modality constants the same as Shell
- MessageBox, FileDialog, DirectoryDialog
- ColorDialog, FontDialog, PrintDialog



MessageBox



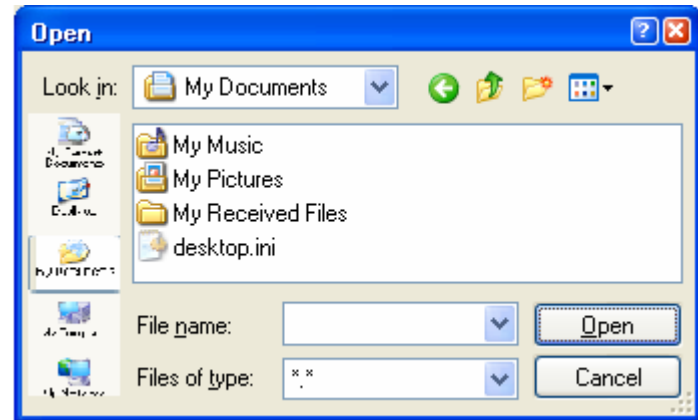
- **new** `MessageBox(shell, SWT.YES | SWT.NO);`
- **Styles**
 - `ICON_ERROR, ICON_INFORMATION, ICON_QUESTION, ICON_WARNING, ICON_WORKING`
 - `OK, OK | CANCEL`
 - `YES | NO, YES | NO | CANCEL`
 - `RETRY | CANCEL, ABORT | RETRY | IGNORE`
- **Interesting methods**
 - `setMessage(String)`
- **Notes**
 - `open()` returns the constant for selected button (`SWT.OK, SWT.CANCEL, etc.`)



FileDialog

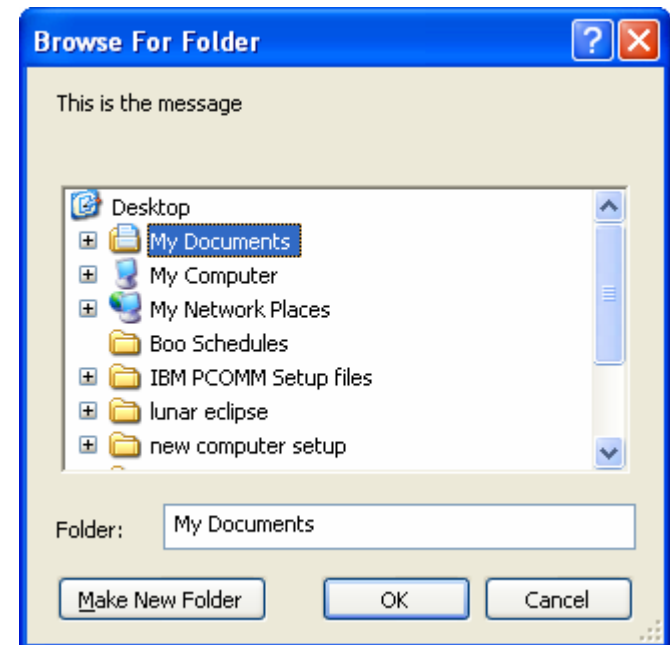


- **new** FileDialog(shell, SWT.OPEN);
- Styles
 - OPEN, SAVE, MULTI
- Interesting methods
 - setFileName(String)
 - setFilterExtensions(String[])
 - setFilterNames(String[])
 - setFilterPath(String)
- Notes
 - open() returns a String which is the absolute path of the first selected file
 - Use getFileNames() for MULTI



DirectoryDialog

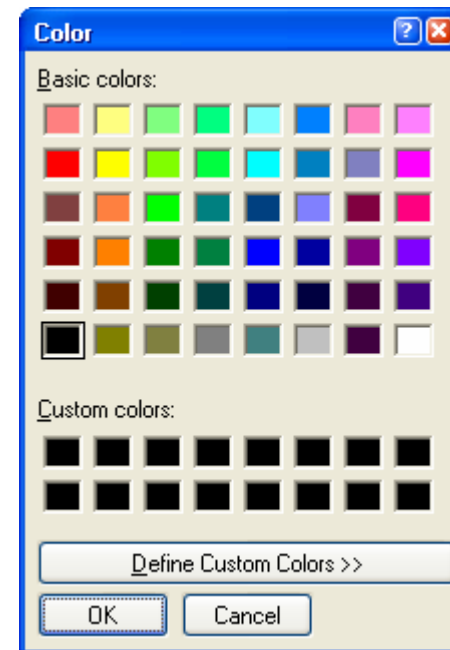
- **new** DirectoryDialog(shell);
- Interesting methods
 - setMessage(String)
 - setFilterPath(String) – initial path
- Notes
 - open() returns a String which is the absolute path of the selected directory



ColorDialog



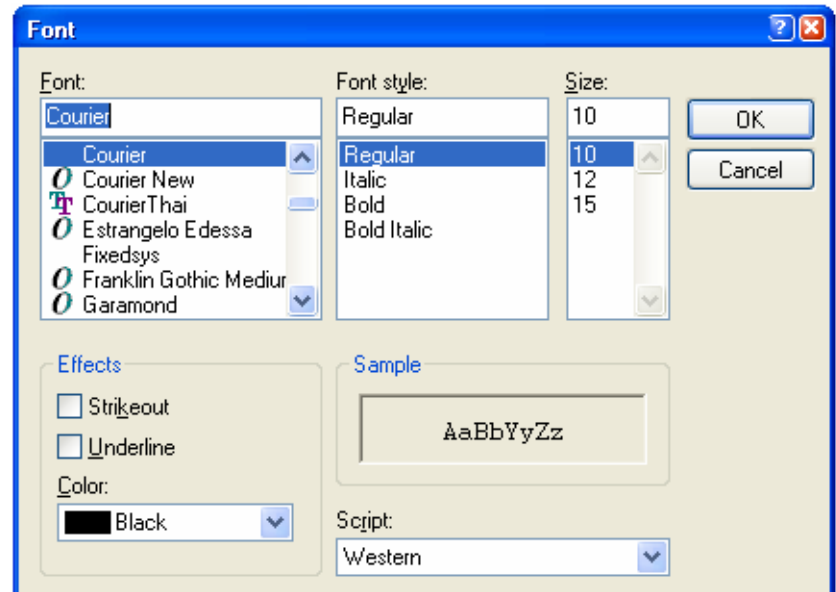
- **new** ColorDialog(shell);
- Interesting methods
 - setRGB(RGB)
- Notes
 - open() returns an RGB



FontDialog



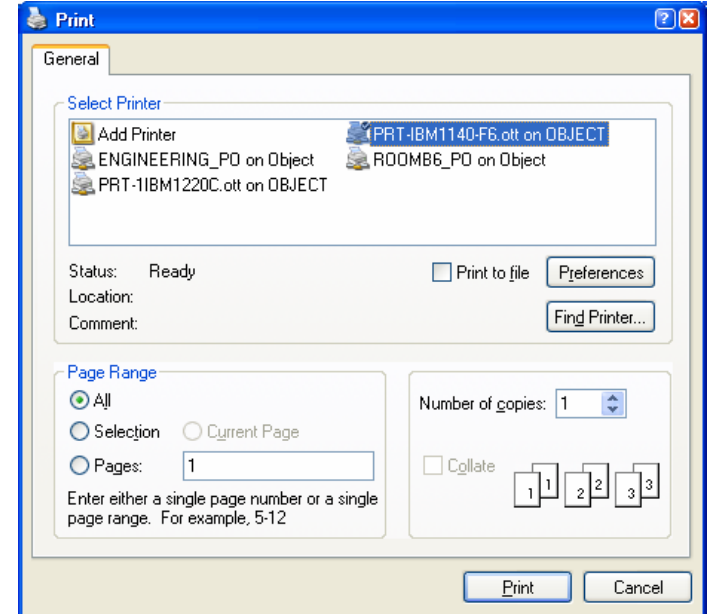
- **new** FontDialog(shell);
- Interesting methods
 - setFontList(FontData[])
 - setRGB(RGB)
- Notes
 - open() returns a FontData
 - Use: FontData[] getFontList() after calling open()



PrintDialog



- **new** PrintDialog(shell);
- Interesting methods
 - setScope(int)
 - setStartPage(int)
 - setEndPage(int)
 - setPrintToFile(boolean)
- Notes
 - open() returns a PrinterData
 - printing currently supported on: Windows, Mac OS X, and Motif



Events & Listeners

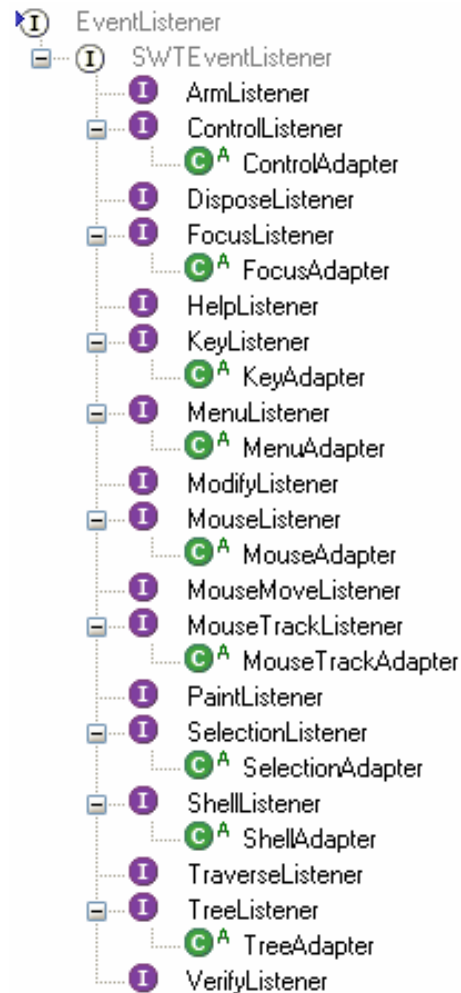
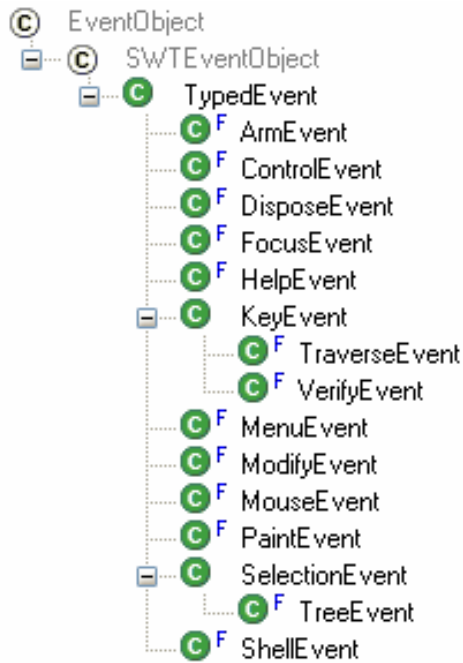
- Events indicate that something interesting has happened
 - mouse, keyboard, paint, focus, move, resize, ...
- Widgets can issue a higher-level event to indicate a state change
 - For example, a Button may track “mouse press”, “mouse move” and “mouse release” in order to issue “selection”
- **Event** classes contain detailed information about what happened
- **Listeners** are instances of classes that implement an interface of event-specific methods

Events & Listeners

- Package org.eclipse.swt.events
- Follows standard Java listener pattern
 - Event classes extend java.util.EventObject
 - Listener interfaces extend java.util.EventListener
 - Adapter classes for multi-method listener interfaces (convenience)
 - Use `addTypeListener(TypeListener)` to hook listener to widget
 - Use `removeTypeListener(TypeListener)` to unhook
 - Multiple listeners are called in the order they were added
- For example, to listen for dispose event on a widget
 - **addDisposeListener(DisposeListener listener)**

```
widget.addDisposeListener(new DisposeListener() {  
    public void widgetDisposed(DisposeEvent event) {  
        // widget was disposed  
    }  
});
```

org.eclipse.swt.events



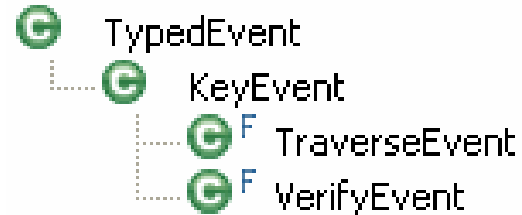
Mouse events



- Events
 - MouseDown, MouseUp
 - MouseMove
 - MouseEnter, MouseExit, MouseHover
 - MouseDoubleClick
 - MenuDetect
 - DragDetect
- Event fields
 - **button** – Button that was pressed or released (1, 2, 3)
 - **x, y** – Location of mouse pointer in widget-relative coordinates
 - **stateMask** – Bit mask representing keyboard modifiers: CONTROL, SHIFT, ALT, COMMAND, MOD1, MOD2, MOD3, MOD4

Keyboard events

- Events
 - KeyDown, KeyUp
 - Traverse, Verify, Modify
- Event fields
 - **character** – The “character” typed – *after* applying modifier keys
 - Examples: (char) 'a', '\u0000', SWT.BS, SWT.CR, SWT.DEL, SWT.ESC, SWT.LF, SWT.TAB
 - **keyCode** – Raw “key code” typed – either Unicode value or keystrokes that are not Unicode values, such as SWT.F4, SWT.PAGE_UP, ...
 - **stateMask** – Bit mask representing keyboard modifiers: CONTROL, SHIFT, ALT, COMMAND, MOD1, MOD2, MOD3, MOD4
 - **doit** (for Traverse and Verify events) – Setting to false cancels the key operation



Selection events

- Events
 - Selection, DefaultSelection
- Event fields
 - **item** – the item that was selected: Tree and Table
 - **detail** – a value representing extra info: CHECK, ARROW, DRAG, HOME, END, ARROW_DOWN, ARROW_UP, PAGE_DOWN, PAGE_UP
 - **stateMask** – Bit mask representing keyboard modifiers: CONTROL, SHIFT, ALT, COMMAND, MOD1, MOD2, MOD3, MOD4
 - **doit** – Setting to false cancels the key operation
- Button, List, Table, Tree, Text, Slider, ScrollBar, Scale, Sash, MenuItem, ToolItem, ...
- `getSelection()`

Adding a selection listener to radio buttons

```
final Button land = new Button(shell, SWT.RADIO);
land.setText("By Land");
final Button sea = new Button(shell, SWT.RADIO);
sea.setText("By Sea");
sea.setSelection(true);
SelectionListener listener = new SelectionAdapter () {
    public void widgetSelected (SelectionEvent e) {
        Button button = (Button) e.widget;
        if (!button.getSelection()) return;
        System.out.println ("Arriving " + button.getText());
    }
};
land.addSelectionListener(listener);
sea.addSelectionListener(listener);
```



Move & resize events

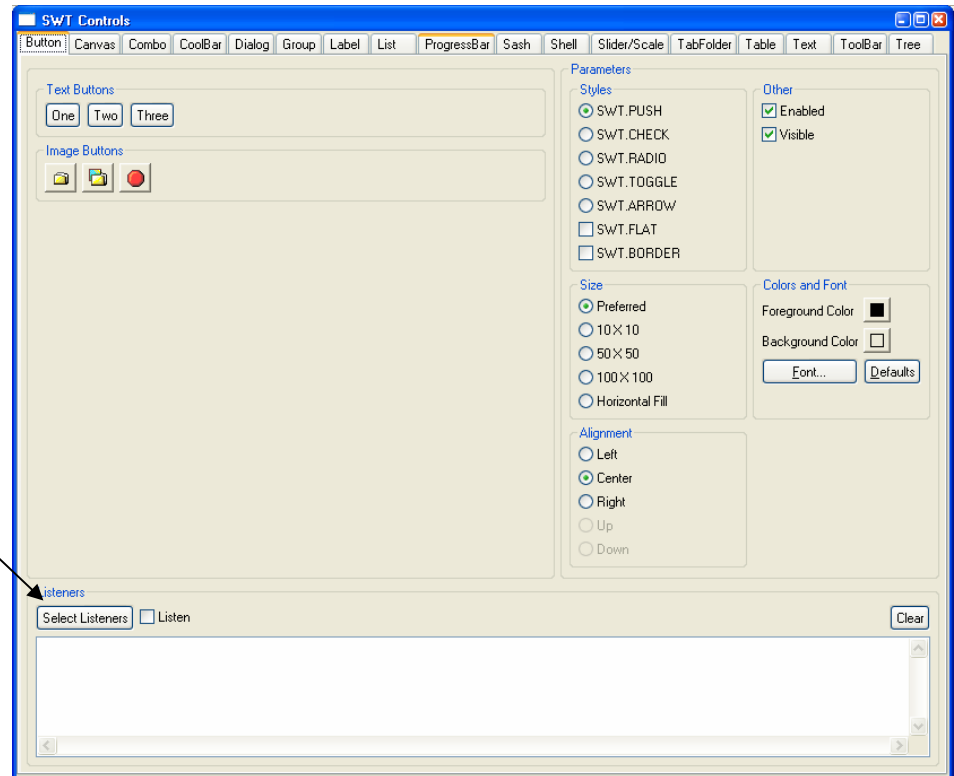
- **SWT.Move event**
 - Sent after a control's location has changed relative to its parent (or, for Shells, relative to the Display)
 - Controls can be moved by calling `setLocation()`, `setBounds()` or, in the case of a Shell, when the user repositions it on the desktop
- **SWT.Resize event**
 - Sent after the size of a control's client area has changed
 - Controls can be resized by calling `setSize()` or `setBounds()`, or because of some operation that the user performs. For example, the user can usually resize a Shell by clicking and dragging one of its corners with the mouse

Paint event

- SWT.Paint event
 - Sent when a Control needs to be drawn
- Event fields
 - **gc** – GC to use to draw the Control, configured with font, foreground and background colors of the Control, and clipped to damaged region
 - **x** – x coordinate of smallest rectangle surrounding damaged region
 - **y** – y coordinate of smallest rectangle surrounding damaged region
 - **width** – width of smallest rectangle surrounding damaged region
 - **height** – height of smallest rectangle surrounding damaged region

Hands-on 4: events

- Open Type (Ctrl+Shift+T)
 - ControlExample
- Run ControlExample
 - Browse events
 - Select events



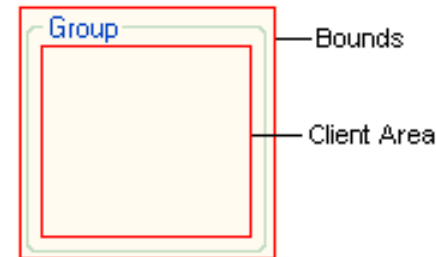
Bounds

- Bounds – the rectangle that describes the location and size, in pixels, of a control within its parent (for Shells, this is the display)
- Expressed in the coordinate system of the parent
 - `getBounds()` – Rectangle describing the size & location of the control
 - `getLocation()` – Point describing the location of the control
 - `getSize()` – Point describing the size of the control

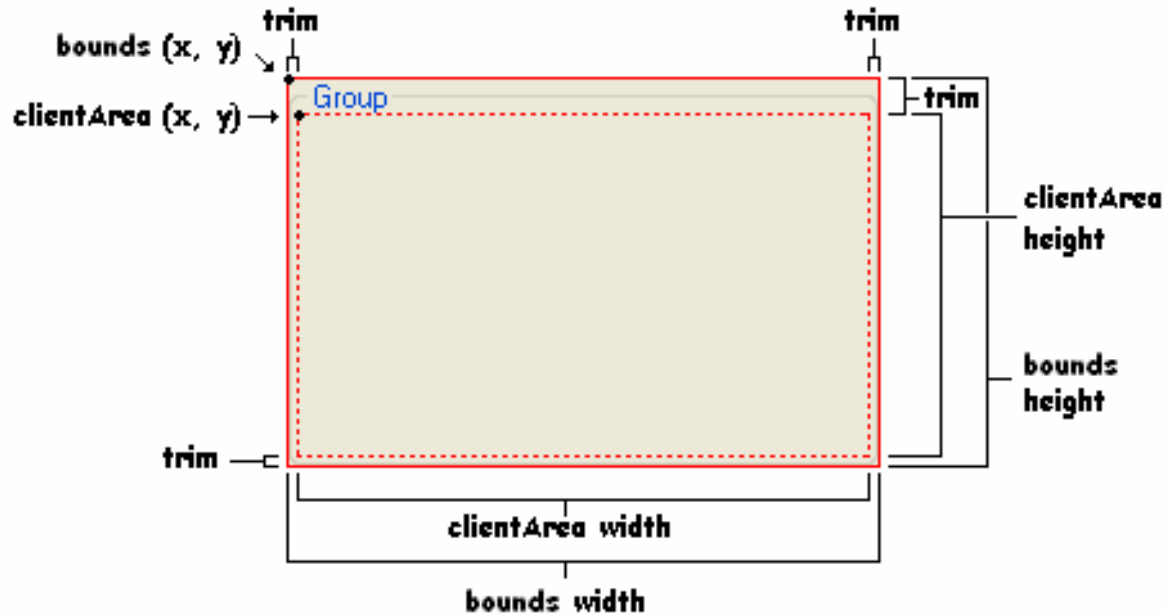


Client area

- Client area – the rectangle within a Composite, List or Text control that describes the area that is available to draw content
- Area not covered by the clientArea is called “trimmings”
 - `getClientArea()` – Rectangle describing the area for displaying data
 - `computeTrim(int x, int y, int width, int height)` – Parent-relative bounding Rectangle needed to produce the specified client area



Bounds & client area of a Group



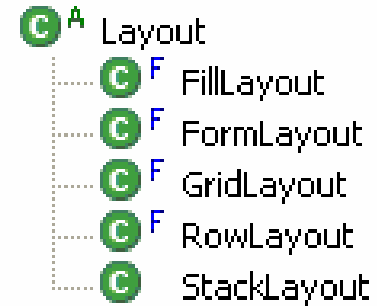
Preferred size

- Layout classes use the preferred size of controls, unless otherwise specified
- `Control.computeSize(int wHint, int hHint)`
- `Control.computeSize(int wHint, int hHint, boolean changed)`
 - Returns a point describing the best size of a control
 - Hints used to force wrapping (`SWT.DEFAULT` or an integer)
 - Changed flag passed to Layout
- `Control.pack()`
- `Control.pack(boolean changed)`
 - `control.setSize(control.computeSize(SWT.DEFAULT, SWT.DEFAULT))`
 - Changed flag passed to Layout

Layouts

- Layout – abstract superclass
- FillLayout – single row or column
- RowLayout – multiple rows or columns
- GridLayout – grid
- FormLayout – attach the edges of each child
- StackLayout – children are stacked and topmost is drawn

- Notes
 - Controls the position and size of the children of a composite
 - Most layouts have a corresponding ‘layout data’ object that can be set into a control using `Control.setLayoutData(Object)`



See “Understanding Layouts” on eclipse.org

FillLayout

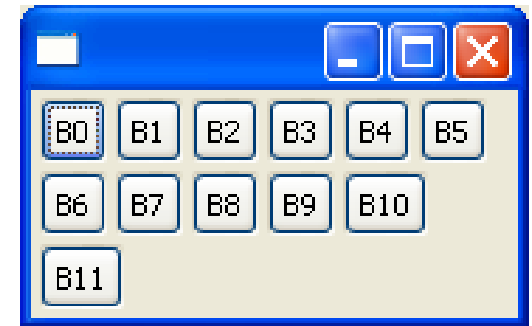
- Lays out the children of a composite in a single row or column
- Controls are forced to be the same size
 - All controls are as tall as the tallest control, as wide as the widest
- Constructors
 - `FillLayout()`
 - `FillLayout(int type)`
- Fields
 - `type` – `HORIZONTAL` or `VERTICAL`
 - `marginWidth`, `marginHeight` – pixels between controls and edge of layout
 - `spacing` – pixels between controls

RowLayout & RowData

- Lays out the children of a composite in rows or columns
- RowData object
 - Specifies the width and height of a specific control
- Constructors
 - RowLayout(), RowLayout(int type)
- Fields
 - type – HORIZONTAL or VERTICAL
 - marginWidth, marginHeight, spacing – same as FillLayout
 - marginLeft, marginTop, marginRight, marginBottom – for fine tuning
 - wrap – wrap control to next row when there is no space in current row
 - pack – if true, controls take their preferred size
 - fill – make controls in the same row all have the same height
 - justify – controls will be ‘spread out’ to occupy all available space

RowLayout example

```
Shell shell = new Shell(display);
RowLayout layout = new RowLayout();
shell.setLayout(layout);
for (int i = 0; i < 12; i++) {
    Button button =
        new Button(shell, SWT.PUSH);
    button.setText("B" + i);
}
Point size = shell.computeSize(SWT.DEFAULT,
    SWT.DEFAULT);
shell.setSize(shell.computeSize(size.x / 2,
    SWT.DEFAULT));
shell.open();
```



GridLayout

- Lays out the children of a composite in a grid
 - Specify the number of columns; the rows are created as needed
- GridData object
 - Specifies the alignment, span, indent, grab, and desired width and/or height properties for each child of the composite
 - Each child must have its own GridData instance (or null)
- Constructors
 - GridLayout()
 - GridLayout(int numColumns, boolean makeColumnsEqualWidth)
- Fields
 - numColumns, makeColumnsEqualWidth
 - marginWidth, marginHeight, horizontalSpacing, verticalSpacing

GridData

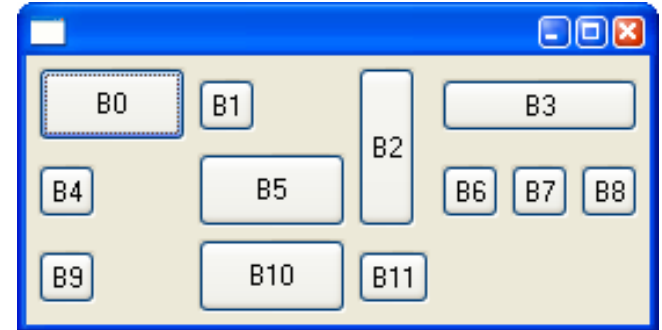
- Constructors
 - GridData()
 - GridData(int style)
 - GridData(int width, int height)
 - .. and other constructors to set alignment, grab and span
- Fields
 - horizontalAlignment, verticalAlignment – align child within its grid cell
 - horizontalIndent, verticalIndent – number of pixels to move the child
 - horizontalSpan, verticalSpan – number of columns to occupy (default 1)
 - grabExcessHorizontalSpace, grabExcessVerticalSpace – occupy any space not taken by other controls in the row or column
 - widthHint, heightHint – attempt to make the child this width and/or height
 - minimumWidth, minimumHeight –child’s minimum width and/or height

GridLayout example

```

Shell shell = new Shell(display);
GridLayout layout=new GridLayout();
layout.numColumns = 6;
shell.setLayout(layout);
for (int i=0; i<12; i++) {
    Button button = new Button(shell, SWT.PUSH);
    button.setText("B" + i);
    if (i % 5 == 0)
        button.setLayoutData (new GridData (64, 32));
    if (i == 2)
        button.setLayoutData (new GridData (
            SWT.LEFT, SWT.FILL, false, false, 1, 2));
    if (i == 3)
        button.setLayoutData (new GridData (
            SWT.FILL, SWT.CENTER, false, false, 3, 1));
}
shell.pack();

```



FormLayout

- Lays out the children of a composite by attaching edges
- FormData object
 - specifies the left, right, top, and bottom FormAttachments
 - specifies the width and height for the control
- FormAttachment object
 - Edges can be attached to a position that is a fraction of the height or width of the parent's client area, or they can be attached to, or centered on, an edge of another control
- Constructors
 - FormLayout()
- Fields
 - marginWidth, marginHeight, spacing

FormAttachment

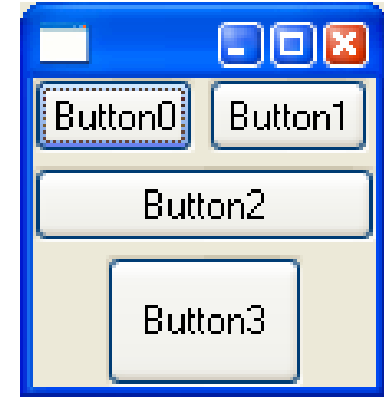
- Constructors
 - FormAttachment(int numerator)
 - FormAttachment(int numerator, int offset)
 - FormAttachment(int numerator, int denominator, int offset)
 - FormAttachment(Control control)
 - FormAttachment(Control control, int offset)
 - FormAttachment(Control control, int offset, int alignment)
- Fields
 - numerator, denominator – fraction specifying how far from the top or left of the parent's client area to attach the edge (default %)
 - offset – # of pixels to move the edge (+/-) from the attachment point
 - control, alignment – align the edge to the TOP, BOTTOM, LEFT, RIGHT, CENTER or DEFAULT edge of another control

FormLayout example (page 1)

```
Shell shell = new Shell (display);  
FormLayout formLayout =  
    new FormLayout ();  
shell.setLayout (formLayout);
```

```
Button button0 = new Button (shell, SWT.PUSH);  
button0.setText ("Button0");  
button0.setLayoutData (new FormData ());
```

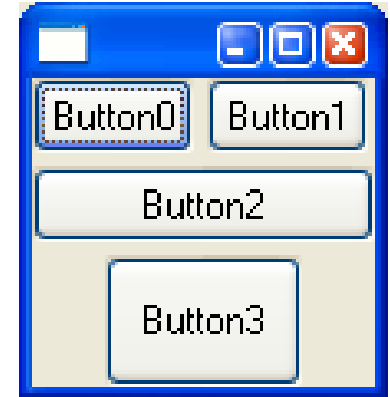
```
Button button1 = new Button (shell, SWT.PUSH);  
button1.setText ("Button1");  
FormData data = new FormData ();  
data.left = new FormAttachment (button0, 4);  
button1.setLayoutData (data);
```



FormLayout example (page 2)

```
Button button2 = new Button (shell, SWT.PUSH);  
button2.setText ("Button2");  
data = new FormData ();  
data.left = new FormAttachment (0, 0);  
data.right = new FormAttachment (100, 0);  
data.top = new FormAttachment (button0, 4);  
button2.setLayoutData (data);
```

```
Button button3 = new Button (shell, SWT.PUSH);  
button3.setText ("Button3");  
data = new FormData (60, 40);  
data.right = new FormAttachment (button2, 0, SWT.CENTER);  
data.top = new FormAttachment (button2, 4);  
button3.setLayoutData (data);  
shell.pack ();
```



Forcing a Layout

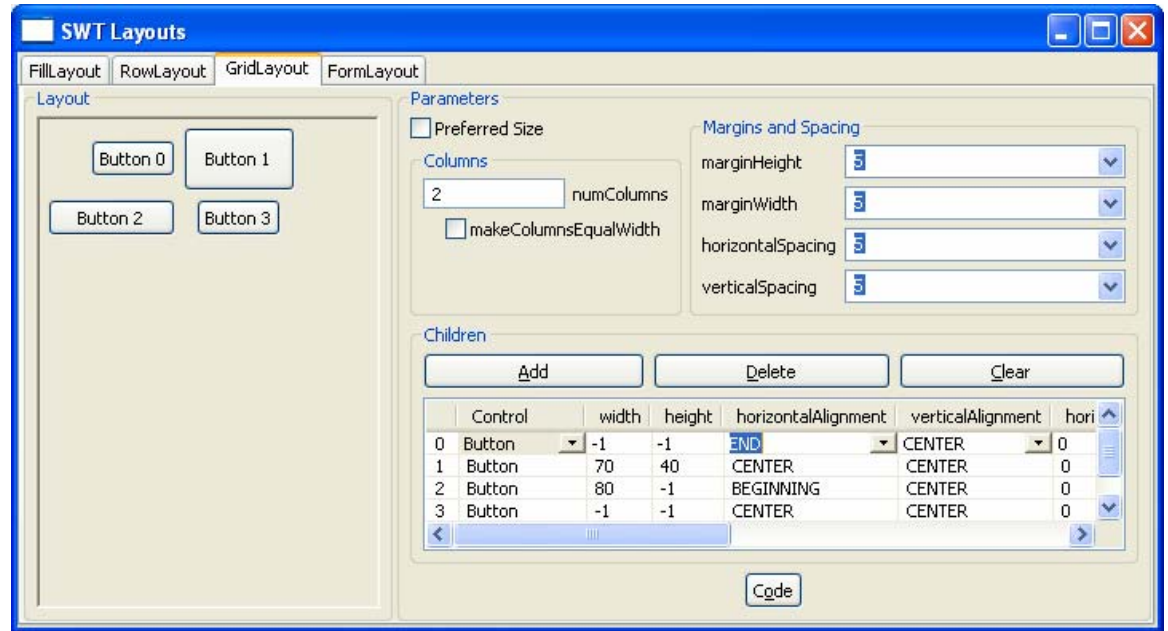
- Composite.layout()
- Composite.layout(boolean changed)
 - Causes the layout algorithm to position and resize children
 - The *changed* flag indicates whether the contents of any children have changed in a way that would affect their preferred size, thereby invalidating the layout's cached data. For example, if you have done any of the following then you should call layout (true):
 - Most setText(...) calls
 - Font changes
 - Adding/removing widgets
 - Adding/removing images
 - Adding/removing items
- Free layout only happens when the Composite is resized
 - Invokes Composite.layout(false)

Layout Optimizations

- `Composite.layout(...)` recursively lays out all children
 - Exception: recursion stops at a child that does not resize
- New layout API in Eclipse 3.1:
 - `Composite.layout(Control[])`
 - Lays out a specific set of controls and their ancestors
 - `Composite.layout(boolean changed, boolean all)`
 - Setting *all* to true forces all children to be laid out
 - `Composite.setLayoutDeferred(boolean defer)`
 - Allows layout calculations to be deferred while *defer* is true
 - Layout will occur automatically once *defer* is set to false

Hands-on 5: layouts

- Open Type (Ctrl+Shift+T)
 - LayoutExample
- Run LayoutExample
 - For each Layout:
 - Add controls
 - Modify data
 - Gen code



Give me a ...

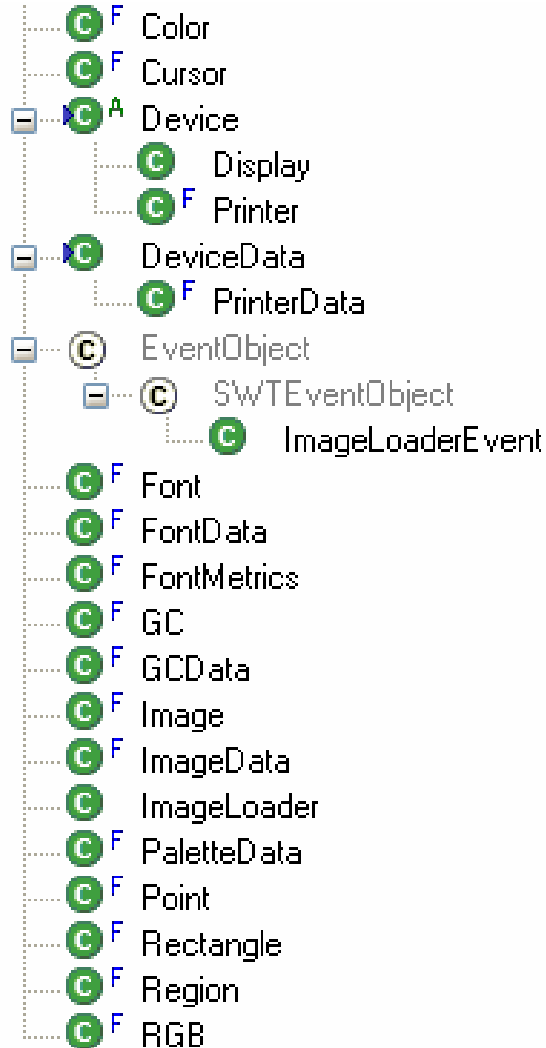
BREAK

Graphics

- Point
 - Data structure (no OS resources): `new Point(int x, int y);`
- Rectangle
 - Data structure (no OS resources): `new Rectangle(int x, int y, int width, int height);`
 - API for `contains(Point)`, `intersects(Rectangle)`, `union(Rectangle)`, etc

- Region
- GC and GCData
- Font and FontData
- Color and RGB
- Image and ImageData
- Cursor and CursorData

Graphics classes



GC (Graphics Context)

- **GC(Drawable)**
- Line, shape, text, and image drawing on a Drawable using color, font, line style, etc
- Three classes implement Drawable:
 - Control
 - Typically draw on a Canvas control
 - Controls typically implement getClientArea() which returns a rectangle to draw on
 - Coordinates are specified relative to the control's client area
 - Device (Display and Printer)
 - Drawing directly on the Display screen or on a Printer page
 - Image
 - Draw on a background image to buffer drawing operations
- GC has OS resources, so if you create a GC, you must dispose it

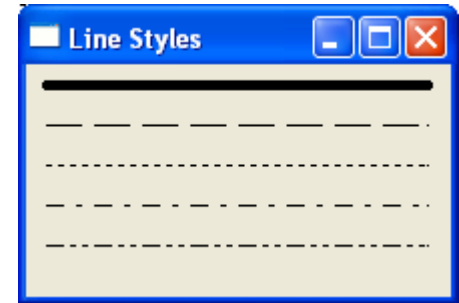
GC line drawing methods

- `drawLine(int x1, int y1, int x2, int y2)`
- `drawPolyline(int[] xyArray)`

- `setLineWidth(int width)`
- `setLineStyle(int style)`
 - `SWT.LINE_SOLID, SWT.LINE_DASH, SWT.LINE_DOT, SWT.LINE_DASHDOT, SWT.LINE_DASHDOTDOT`

Draw lines on a shell in a paint listener

```
shell.addListener(SWT.Paint, new Listener() {  
    public void handleEvent(Event event) {  
        GC gc = event.gc;  
        gc.setLineWidth(5);  
        gc.setLineStyle(SWT.LINE_SOLID);  
        gc.drawLine(10, 10, 200, 10);  
        gc.setLineWidth(1);  
        gc.setLineStyle(SWT.LINE_DASH);  
        gc.drawLine(10, 30, 200, 30);  
        gc.setLineStyle(SWT.LINE_DOT);  
        gc.drawLine(10, 50, 200, 50);  
        gc.setLineStyle(SWT.LINE_DASHDOT);  
        gc.drawLine(10, 70, 200, 70);  
        gc.setLineStyle(SWT.LINE_DASHDOTDOT);  
        gc.drawLine(10, 90, 200, 90);  
    }  
});
```

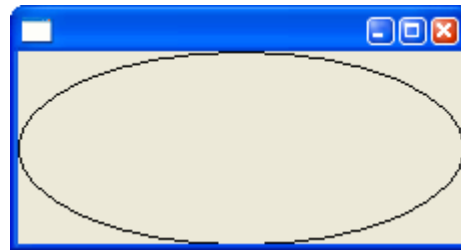


GC shape drawing methods

- `drawRectangle(int x, int y, int width, int height)`
- `drawRectangle(Rect rect)`
- `drawRoundedRectangle(int x, int y, int w, int h, int arcWidth, int arcHeight)`
- `drawOval(int x, int y, int width, int height)`
- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- `drawPolygon(int[] xyArray)`

Draw an oval on a Canvas

```
final Canvas canvas = new Canvas(shell, SWT.NONE);  
canvas.addListener(SWT.Paint, new Listener() {  
    public void handleEvent(Event event) {  
        GC gc = event.gc;  
        Rectangle rect = canvas.getClientArea();  
        gc.drawArc(0,0,rect.width,rect.height,0,360);  
    }  
});
```



GC shape filling methods

- `fillRectangle(int x, int y, int width, int height)`
- `fillRectangle(Rect rect)`
- `fillRoundedRectangle(int x, int y, int w, int h, int arcWidth, int arcHeight)`
- `fillOval(int x, int y, int width, int height)`
- `fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`
- `fillPolygon(int[] xyArray)`

Fill an oval on a Canvas

```
final Canvas canvas = new Canvas(shell, SWT.NONE);  
canvas.addListener(SWT.Paint, new Listener() {  
    public void handleEvent(Event event) {  
        GC gc = event.gc;  
        Rectangle rect = canvas.getClientArea();  
        gc.fillArc(0,0,rect.width,rect.height,0,360);  
    }  
});
```

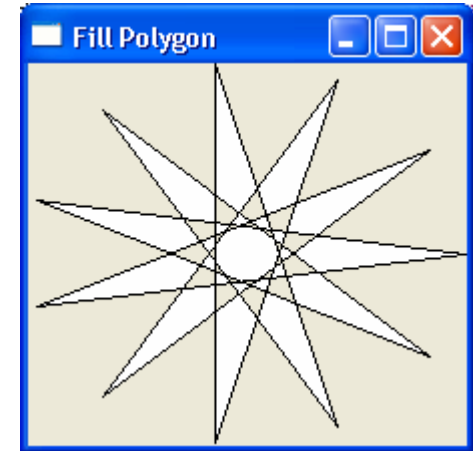


Draw and fill a polygon on a Shell

```

final int points = 11;
final Point center = new Point(0,0);
final int[] radial = new int[points*2];
shell.addListener(SWT.Resize, new Listener() {
    public void handleEvent(Event event) {
        Rectangle bounds = shell.getClientArea();
        center.x = bounds.x + bounds.width/2;
        center.y = bounds.y + bounds.height/2;
        int pos = 0;
        for (int i = 0; i < points; ++i) {
            double r = Math.PI*2 * pos/points;
            radial[i*2] = (int)((1+Math.cos(r))*center.x);
            radial[i*2+1] = (int)((1+Math.sin(r))*center.y);
            pos = (pos + points/2) % points;
        }
    }
});
shell.addListener(SWT.Paint, new Listener() {
    public void handleEvent(Event event) {
        event.gc.setBackground(display.getSystemColor(SWT.COLOR_WHITE));
        event.gc.fillPolygon(radial);
        event.gc.drawPolygon(radial);
    }
});

```



RGB

- **RGB(int red, int green, int blue)**
- Description of an abstract color represented as three 8-bit quantities modeling the red, green, and blue components of the color
- Data only - access red, green, and blue fields directly
- RGB instances can also be created by using the ColorDialog
- No OS storage, therefore don't need to dispose

Color

- **Color(Device device, int red, int green, int blue)**
- **Color(Device device, RGB rgb)**
 - Example: `Color red = new Color(display, 0xFF, 0x00, 0x00);`
- Actual OS color, suitable for display on the specified device which may be a Display or Printer
- May not represent the specified color components exactly, depending on the depth of the device
- `getRed()`, `getGreen()`, `getBlue()`
- `getRGB()`
- OS resources are allocated, therefore must dispose

System colors

- System colors are pre-allocated for each device
- Get a system color using `Device.getSystemColor (int colorConstant)`
 - SWT.COLOR_WHITE
 - SWT.COLOR_RED
 - SWT.COLOR_GREEN
 - SWT.COLOR_YELLOW
 - SWT.COLOR_BLUE
 - SWT.COLOR_MAGENTA
 - SWT.COLOR_CYAN
 - SWT.COLOR_GRAY
 - SWT.COLOR_BLACK
 - SWT.COLOR_DARK_RED
 - SWT.COLOR_DARK_GREEN
 - SWT.COLOR_DARK_YELLOW
 - SWT.COLOR_DARK_BLUE
 - SWT.COLOR_DARK_MAGENTA
 - SWT.COLOR_DARK_CYAN
 - SWT.COLOR_DARK_GRAY
- Do not dispose of system colors

Additional 'widget-related' system colors

- COLOR_WIDGET_DARK_SHADOW
- COLOR_WIDGET_NORMAL_SHADOW
- COLOR_WIDGET_LIGHT_SHADOW
- COLOR_WIDGET_HIGHLIGHT_SHADOW
- COLOR_WIDGET_FOREGROUND
- COLOR_WIDGET_BACKGROUND
- COLOR_WIDGET_BORDER
- COLOR_LIST_FOREGROUND
- COLOR_LIST_BACKGROUND
- COLOR_LIST_SELECTION
- COLOR_LIST_SELECTION_TEXT
- COLOR_INFO_FOREGROUND
- COLOR_INFO_BACKGROUND
- COLOR_TITLE_FOREGROUND
- COLOR_TITLE_BACKGROUND
- COLOR_TITLE_BACKGROUND_GRADIENT
- COLOR_TITLE_INACTIVE_FOREGROUND
- COLOR_TITLE_INACTIVE_BACKGROUND
- COLOR_TITLE_INACTIVE_BACKGROUND_GRADIENT

Color depth and direct/indexed color

- The *depth* of a display or image is the number of bits that are used to represent a single pixel
- Most operating systems support depths of 1, 8, 16, 24, and 32 bits
- Direct Color
 - Almost invariably used when the depth is 16 bits or greater
 - Example: 24-bit depth direct color display:
24-bit pixel value = red << 16 + green << 8 + blue
- Indexed Color
 - Uses pixel value as an index into a table containing color values

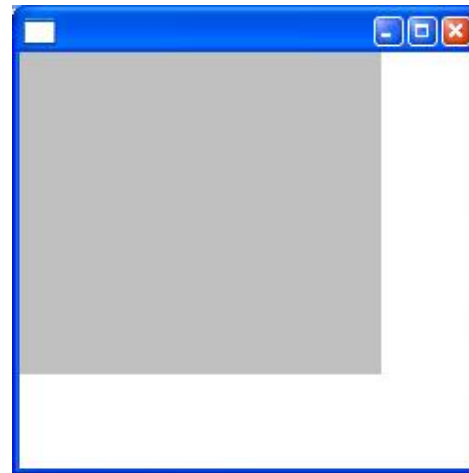
See “SWT Color Model” on eclipse.org

GC color methods

- The graphics context foreground and background are initially the same as the control's or device's foreground and background
- `getBackground()`
- `getForeground()`
- `setForeground(Color)`
- `setBackground(Color)`

Region

- Region(Device)
 - Union of areas covered by a collection of rectangles, polygons & regions
 - Good example is the backwards 'L' that is exposed on resize
 - add(Rectangle rect)
 - add(Region region)
 - add(int[] polygon)
 - subtract(Rectangle rect)
 - subtract(Region region)
 - subtract(int[] polygon)
 - getBounds()
 - contains(int x, int y)
 - intersects(Rectangle rect)
-
- OS resources are allocated, therefore must dispose



GC clipping methods

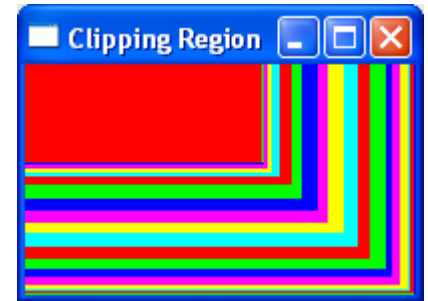
- During SWT.Paint the GC's clipping is the damage area
- `getClipping()`
- `getClipping(Region region)`
- `setClipping(int x, int y, int width, int height)`
- `setClipping(Rect rect)`
- `setClipping(Region region)`

Fill the damage region with color

```

final int colors [] = {
    SWT.COLOR_RED, SWT.COLOR_GREEN, SWT.COLOR_BLUE,
    SWT.COLOR_MAGENTA, SWT.COLOR_YELLOW, SWT.COLOR_CYAN};
final Display display = new Display();
final Shell shell = new Shell(display,
    SWT.SHELL_TRIM | SWT.NO_REDRAW_RESIZE);
shell.addListener(SWT.Paint, new Listener() {
    public void handleEvent(Event event) {
        GC gc = event.gc;
        Color color = display.getSystemColor (
            colors [count++ % colors.length]);
        gc.setBackground(color);
        gc.fillRectangle(shell.getClientArea());
        color.dispose();
    }
});

```



Fill the clipping region

```
final Region region = new Region(display);
region.add(new int[]{0, 0, 200, 0, 200, 200});
region.subtract(new Rectangle(75, 75, 50, 50));
region.add(new int[]{25, 125, 75, 125, 75, 175});
shell.addListener(SWT.Paint, new Listener() {
    public void handleEvent(Event event) {
        GC gc = event.gc;
        gc.setClipping(region);
        Rectangle rect = shell.getClientArea();
        gc.setBackground(display.getSystemColor(SWT.COLOR_CYAN));
        gc.fillRect(0,0,rect.width,rect.height);
    }
});
shell.setSize(shell.computeSize(200, 200));
```



ImageData

- Description of an image
- ImageData(String)
- ImageData(InputStream)
 - Loads an image data from file
 - GIF, JPEG, PNG, BMP, ICO or TIFF
- ImageData(int width, int height, int depth, PaletteData palette)
 - Creates a blank image data

- setPixel(int x, int y, int pixel)
- setPixels(int x, int y, int putWidth, byte[] pixels)
- setPixels(int x, int y, int putWidth, int[] pixels)
- scaleTo(int width, int height)

- No OS storage, therefore no need to dispose

PaletteData

- Describe the color format of an image
 - Depending on the depth of the image it can be *direct* or *indexed*
- PaletteData(RGB[] colors)
- PaletteData(int redMask, int greenMask, int blueMask)
- getPixel(RGB rgb)
- No OS storage, therefore no need to dispose

Image

- Image(Device device, ImageData data)
 - Create image from image data
- Image(Device device, int width, int height)
 - Blank off-screen image used for drawing
- Image(Device device, String filename)
 - Image loaded from file
- Image(Device device, Image image, int flag)
 - Apply some transformation: SWT.DISABLE, SWT.GRAY

- Actual OS image, suitable for display on a Display or Printer
- getBounds()
- getImageData()

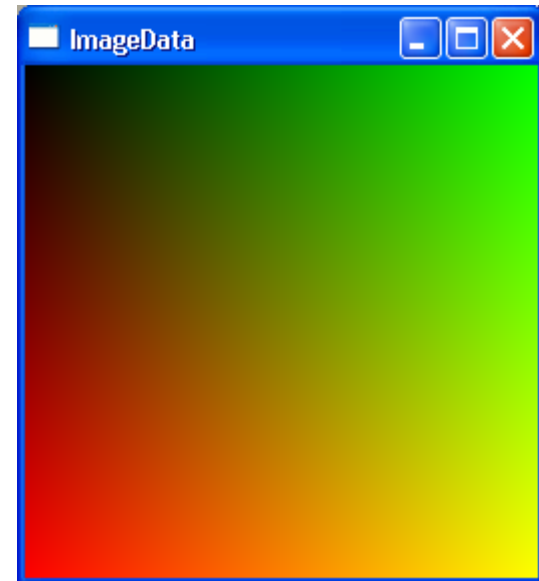
- OS resources are allocated, therefore must dispose

GC image drawing methods

- `drawImage(Image image, int x, int y)`
- `drawImage(Image image,
int srcX, int srcY, int srcWidth, int srcHeight,
int destX, int destY, int destWidth, int destHeight)`

Manipulating ImageData

```
PaletteData palette = new PaletteData(0xFF0000, 0xFF00, 0xFF);
ImageData data = new ImageData(256, 256, 24, palette);
shell.setSize(shell.computeSize(data.width, data.height));
RGB rgb = new RGB(0, 0, 0);
for (int y=0; y<data.height; y++) {
    for (int x=0; x<data.width; x++) {
        rgb.red = y;
        rgb.green = x;
        data.setPixel(x, y, palette.getPixel(rgb));
    }
}
final Image image = new Image(display, data);
shell.addListener(SWT.Paint, new Listener() {
    public void handleEvent(Event event) {
        GC gc = event.gc;
        gc.drawImage(image, 0, 0);
    }
});
```



FontData

- Description of a font
- `FontData(String name, int height, int style)`
 - Name must be the name of a font that is available on the platform
 - Height is specified in points (equivalent to 1/72 of an inch)
 - Style must be one of: NORMAL, ITALIC, BOLD or (BOLD | ITALIC)
 - `setName()`, `setHeight()` and `setStyle()`
- `FontData` instances can also be created by:
 - Using the `FontDialog`
 - Calling `Device.getFontList(String faceName, boolean scalable)`
 - If `faceName` is null, all fonts are returned
 - `scalable` indicates whether to return bitmapped or scalable fonts
- No OS storage, therefore no need to dispose

List all bitmap and scalable fonts on the display

```
Display display = new Display();
Set s = new HashSet();
FontData[] fds = display.getFontList(null, false);
for (int i=0; i<fds.length; ++i)
    s.add(fds[i].getName());
fds = display.getFontList(null, true);
for (int i=0; i<fds.length; ++i)
    s.add(fds[i].getName());
String[] answer = new String[s.size()];
s.toArray(answer);
Arrays.sort(answer);
for (int i=0; i<answer.length; ++i)
    System.out.println(answer[i]);
display.dispose();
```

Font

- **Font(Device device, FontData data)**
- **Font(Device display, String name, int height, int style)**
 - Example: `new Font(display, "Arial", 40, SWT.BOLD);`

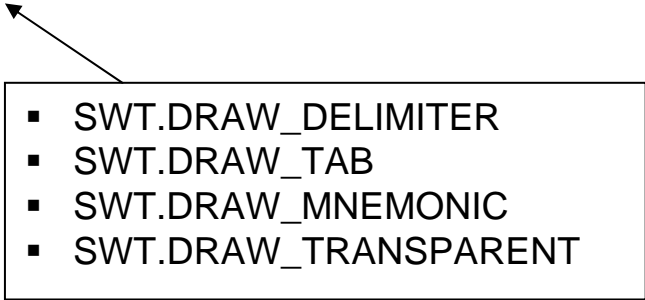
- Actual OS font, suitable for display on a Display or Printer
- `getFontData()`
- `Control.setFont(Font font)`

- OS resources are allocated, therefore must dispose

GC font / text drawing & measuring methods

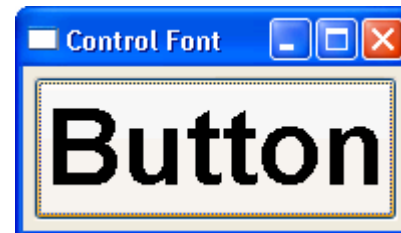
- `setFont(Font font)`
- `drawString(String string, int x, int y)`
- `drawString(String string, int x, int y, boolean isTransparent)`
- `drawText(String string, int x, int y)`
- `drawText(String string, int x, int y, boolean isTransparent)`
- `drawText(String string, int x, int y, int flags)`

- `getAdvanceWidth(char ch)`
- `getCharWidth(char ch)`
- `getFontMetrics()`
- `stringExtent(String string)`
- `textExtent(String string)`
- `textExtent(String string, int flags)`

- 
- `SWT.DRAW_DELIMITER`
 - `SWT.DRAW_TAB`
 - `SWT.DRAW_MNEMONIC`
 - `SWT.DRAW_TRANSPARENT`

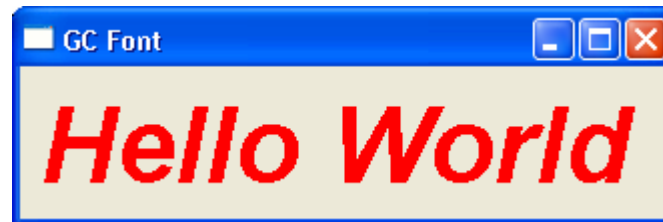
Setting the Font in a Control

```
Button button = new Button(shell, SWT.PUSH);  
button.setText("Button");  
Font font = new Font(display, "Arial", 40, SWT.BOLD);  
button.setFont(font);
```



Setting the Font for drawing

```
final Font font = new Font(display, "Arial", 40, SWT.BOLD|SWT.ITALIC);
shell.addListener(SWT.Paint, new Listener() {
    public void handleEvent(Event event) {
        GC gc = event.gc;
        Rectangle rect = shell.getClientArea();
        gc.setFont(font);
        gc.setForeground(display.getSystemColor(SWT.COLOR_RED));
        gc.drawString("Hello World", rect.x + 10, rect.y + 10, true);
    }
});
```



Cursor

- `Cursor(Device device, int type)`
 - Predefined cursors: `SWT.CURSOR_WAIT`, `SWT.CURSOR_HAND`, ...
- `Cursor(Device device, ImageData data, int hotspotX, int hotspotY)`
 - Custom cursors
 - May be colored on certain platforms

- Specify the appearance of the on-screen pointer
- `Control.setCursor(Cursor cursor)`
- `Display.getCursorSizes()`

- OS resources are allocated, therefore must dispose

System cursors

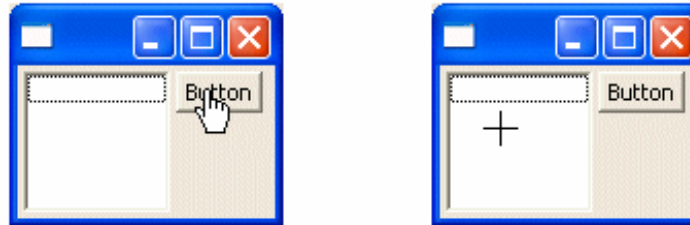
Get a system cursor using `Display.getSystemCursor (int cursorConstant)`

- `CURSOR_ARROW`
- `CURSOR_WAIT`
- `CURSOR_CROSS`
- `CURSOR_APPSTARTING`
- `CURSOR_HELP`
- `CURSOR_UPARROW`
- `CURSOR_IBEAM`
- `CURSOR_NO`
- `CURSOR_HAND`
- `CURSOR_SIZEALL`
- `CURSOR_SIZENESW`
- `CURSOR_SIZENS`
- `CURSOR_SIZENWSE`
- `CURSOR_SIZEWE`
- `CURSOR_SIZEN`
- `CURSOR_SIZES`
- `CURSOR_SIZEE`
- `CURSOR_SIZEW`
- `CURSOR_SIZENE`
- `CURSOR_SIZESE`
- `CURSOR_SIZESW`
- `CURSOR_SIZENW`

Do not dispose of system cursors

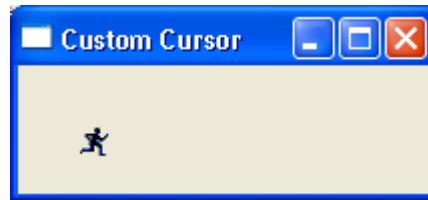
Using system cursors

```
List list = new List(shell, SWT.SINGLE | SWT.BORDER);  
Button button = new Button(shell, SWT.PUSH);  
button.setText("Button");  
Cursor hand = display.getSystemCursor(SWT.CURSOR_HAND);  
Cursor cross = display.getSystemCursor(SWT.CURSOR_CROSS);  
button.setCursor(hand);  
shell.setCursor(cross);
```



Creating a custom cursor

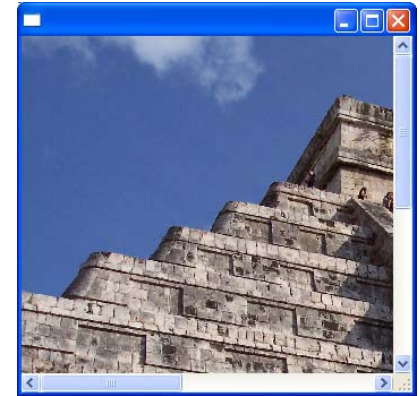
```
ImageData data = new ImageData(  
    CustomCursorExample.class.getResourceAsStream("run.gif"));  
Cursor cursor = new Cursor(display, data, data.width/2, data.height/2);  
shell.setCursor(cursor);
```



Dispose of cursor when done, for example in shell's dispose listener

Hands-on 6.1: Creating the Widgets for an Image Viewer

- In project **“SWT EclipseCon2005 Tutorial”**
- Start with **ImageViewerApp.java**
- **Hints**
 - Create a Shell that contains a Canvas that will be drawn on
 - Create a drop-down menu on the Shell with **“Open...”** and **“Exit”** menu items
 - Open a FileDialog when the user invokes the **“Open”** item, which allows them to select a graphics file to draw



Hands-on 6.2: Displaying the Image

- **Start with ImageViewerApp1.java**

- **Hints**
 - Load the Image with Image (Device, String filename)
 - Create the Canvas with SWT.NO_BACKGROUND style so that your image will not flicker on repaints
 - Add a paint callback to the Canvas, which displays the image
 - Use GC.drawImage(...) to draw the image
 - Use GC.fillRect(...) to fill parts of Canvas background not covered by the image (needed because of NO_BACKGROUND)
 - Use Canvas.redraw() to redraw whenever your image changes
 - Remember to dispose your image when you will no longer need it

Hands-on 6.3: Implementing Scrolling Support

- **Start with ImageViewerApp2.java**
- **Hints**
 - Canvas will need styles `SWT.H_SCROLL | SWT.V_SCROLL`
 - Add listeners to scrollbars
 - Implement `scrollHorizontal` and `scrollVertical` methods:
 - use `Canvas.scroll(...)`
 - update `xOffset` and `yOffset` according to scrollbar selections
 - Update scrollbar attributes (e.g. `thumb`, `pageIncrement`, `maximum`) whenever image is set or when canvas resizes

Writing your own widget

- Eclipse uses non-native controls (“Custom Controls”)
 - StyledText
 - CTabFolder
 - CLabel
 - CCombo

- Subclass Composite or Canvas
 - If your widget needs children, subclass Composite
 - If your widget is drawn using graphics, subclass Canvas

See “Creating Your Own Widgets” on eclipse.org

Threads and Timers

- Single UI thread
 - Apartment threaded vs. Free threaded
- Background threads
 - Use Display methods `asyncExec` and `syncExec`
- Widget calls are **not** allowed in a non-UI thread
- Graphics drawing calls **are** allowed in a non-UI thread
 - Because platforms do not prevent graphics calls from other threads
 - Extra care is required when drawing from background threads, but it can be useful
 - For example, updating a graphical progress indicator that is monitoring work done in a background thread
- Timers run in the UI thread

asyncExec, syncExec, and timerExec

- `Display.asyncExec(Runnable runnable)`
 - Causes the argument to be run by the UI thread of the Display
- `Display.syncExec(Runnable runnable)`
 - Causes the argument to be run by the UI thread of the Display, and causes the current thread to wait for the Runnable to finish
- `Display.timerExec(int milliseconds, Runnable runnable)`
 - Causes the Runnable argument to be evaluated after the specified number of milliseconds have elapsed
- Notes
 - These methods do **not** create a thread; they simply execute the Runnable argument in the UI thread

Using asyncExec to update a ProgressBar

```
final ProgressBar bar = new ProgressBar(shell, SWT.NONE);
bar.setSize(200, 32);
shell.pack();
shell.open();
final int maximum = bar.getMaximum();
new Thread() {
    public void run() {
        for (int i = 0; i <= maximum; i++) {
            try {Thread.sleep(100);} catch (Throwable th) {}
            final int index = i;
            display.asyncExec(new Runnable() {
                public void run() {
                    bar.setSelection(index);
                }
            });
        }
    }
}.start();
```

Using timerExec

- **Execute a Runnable after waiting for two seconds:**

```
display.timerExec(2000, new Runnable() {  
    public void run() {  
        System.out.println("Once, after 2 seconds.");  
    }  
});
```

- **Timers run only once. To run a timer repeatedly, queue the same Runnable again using the same time interval. Execute a Runnable every two seconds:**

```
display.timerExec(2000, new Runnable() {  
    public void run() {  
        System.out.println("Every 2 seconds.");  
        display.timerExec(2000, this);  
    }  
});
```

Drag & drop and Clipboard

- **DragSource**
 - Create on a widget that is eligible as a drag source
 - Provides content and format information from the source widget to the Drag and Drop operation
- **DropTarget**
 - Create on a widget that is eligible as a drop target
 - Receives content and format information from a Drag and Drop operation, and updates the target widget accordingly
- **Clipboard**
 - Allows deferred transfer of data within and between applications
- **Transfer & subclasses**
 - Converts between Java representation and platform specific representation of data and vice versa
 - Provided by default: Text, RTF, File

See “Drag and Drop” article on eclipse.org

Drag and Drop Example

```
1  DragSource source = new DragSource(text1, DND.DROP_COPY | DND.DROP_MOVE);
2  source.setTransfer(new Transfer[] {TextTransfer.getInstance()});
3  source.addDragListener(new DragSourceAdapter() {
4      public void dragStart(DragSourceEvent event) {
5          event.doit = !(text1.getText().equals(""));
6      }
7      public void dragSetData(DragSourceEvent event) {
8          event.data = text1.getText();
9      }
10     public void dragFinished(DragSourceEvent event) {
11         if (event.detail == DND.DROP_MOVE) text1.setText("");
12     }
13 });
14 DropTarget target = new DropTarget(text2, DND.DROP_COPY | DND.DROP_MOVE);
15 target.setTransfer(new Transfer[] {TextTransfer.getInstance()});
16 target.addDropListener(new DropTargetAdapter() {
17     public void drop(DropTargetEvent event) {
18         text2.setText((String)event.data);
19     }
20 });
```

Printing

- Get a PrinterData using either:
 - Printer.getPrinterList()
 - Printer.getDefaultPrinterData()
 - PrintDialog.open()
- Create a Printer from the PrinterData
- Start/end a print job using:
 - startJob(String title) ... endJob()
- Allocate graphics resources for the Printer device
 - Make sure to do this step, and dispose resources when done
- Create a GC for drawing to the Printer
- Start/end page(s) using:
 - startPage() ... endPage()
- Draw using GC calls



Printing example: draw a red rectangle around text

```
PrinterData data = Printer.getDefaultPrinterData(); // check null
Printer printer = new Printer(data);
if (printer.startJob("SWT Printing Example")) {
    Color red = printer.getSystemColor(SWT.COLOR_RED);
    Rectangle trim = printer.computeTrim(0, 0, 0, 0);
    Point dpi = printer.getDPI();
    int left = dpi.x + trim.x; // 1" from left side of paper
    int top = dpi.y / 2 + trim.y; // ½" from top edge of paper
    GC gc = new GC(printer);
    if (printer.startPage()) {
        String testString = "Hello World!";
        Point extent = gc.stringExtent(testString);
        gc.drawString(testString, left,
            top + gc.getFont().getFontData()[0].getHeight());
        gc.setForeground(red);
        gc.drawRect(left, top, extent.x, extent.y);
        printer.endPage();
    }
    gc.dispose(); printer.endJob();
}
printer.dispose();
```

Program

- Provides support for accessing external programs through the OS
- `Program.getPrograms()`
- `Program.findProgram(String extension)`
 - Find a program for the specified extension
- `getName()`
 - Retrieves the program's name
- `getImageData()`
 - Retrieves the program's icon
- `execute(String fileName)`
 - Launches the program with the single argument

Program example: find icon for “.bmp” program

```
Label label = new Label (shell, SWT.NONE);
label.setText ("Can't find icon for .bmp");
Image image = null;
Program p = Program.findProgram (".bmp");
if (p != null) {
    ImageData data = p.getImageData ();
    if (data != null) {
        image = new Image (display, data);
        label.setImage (image);
    }
}
label.pack ();
...
if (image != null) image.dispose ();
```

Accessibility

- Provides a bridge between “assistive technology” clients (programs like screen readers) and the controls in your application
- `Control.getAccessible()`
 - Allows default platform accessible behavior to be overridden by implementing the desired methods of the following listeners
 - `AccessibleListener`
 - `getName(AccessibleEvent)`
 - `getHelp(AccessibleEvent)`
 - `getKeyboardShortcut(AccessibleEvent)`
 - `getDescription(AccessibleEvent)`
 - `AccessibleControlListener`
 - 11 methods – only needed if implementing a custom control
 - `AccessibleTextListener`
 - `getCaretOffset(AccessibleTextEvent)`
 - `getSelectionRange(AccessibleTextEvent)`

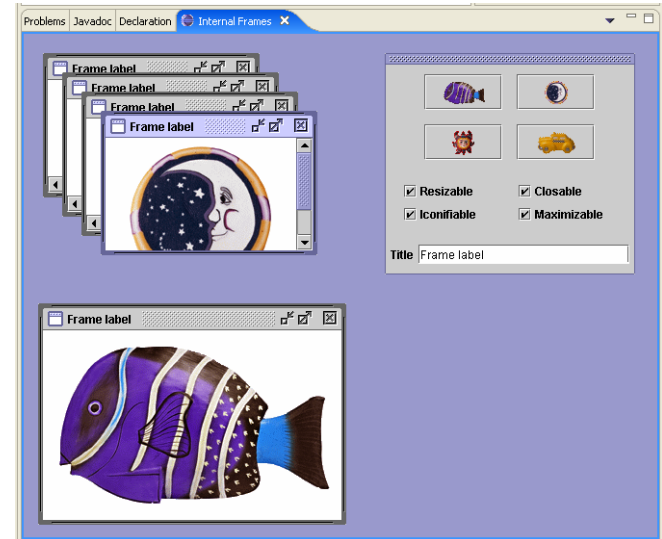
Accessibility example: override a button's name

```
Button button = new Button(shell, SWT.PUSH);
button.setImage(horseImage);
button.getAccessible().addAccessibleListener(
    new AccessibleAdapter() {
        public void getName(AccessibleEvent e) {
            e.result = "Horse";
        }
    });
```

AWT/Swing Interoperability

Use static methods on SWT_AWT

- To embed SWT within AWT:
 - `org.eclipse.swt.widgets.Shell new_Shell(display, java.awt.Canvas parent)`
- To embed AWT within SWT:
 - `java.awt.Frame new_Frame(org.eclipse.swt.widgets.Composite parent)`
 - parent Composite must have style `SWT.EMBEDDED`



See example code snippets on eclipse.org

OLE & ActiveX Support (win32 only)

- OleFrame
 - Handles sizing, menu management and window placement
 - Create on an SWT Composite
 - API useful for merging OLE-provided menus with application menus
- OleClientSite (OLE) / OleControlSite (ActiveX)
 - Handles interactions with a specific OLE Object
 - Create on an OleFrame
 - Interesting methods: doVerb(int verb), deactivateInPlaceClient()
- OleAutomation
 - Create on an OleClientSite
 - Used to access specialized info and call methods on an OLE site
 - Interesting methods: getProperty(...), invoke(..)

See “ActiveX Support In SWT” on eclipse.org

Useful SWT resources

- eclipse.org
 - Online documentation and articles
 - Eclipse Bugzilla (product: platform, component: SWT)
 - SWT user and developer newsgroup (eclipse.platform.swt)
 - SWT developer mailing list (platform-swt-dev@eclipse.org)
 - org.eclipse.swt.examples from CVS
- eclipse.org/swt
 - Example code snippets, organized by SWT class
 - SWT FAQ, downloads
- Book:
 - *SWT: The Standard Widget Toolkit, Volume 1*, Steve Northover and Mike Wilson

For links to all of the above and more, see the SWT “Getting Started” page:

http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-swt-home/SWT_Resources.html