|  |
| --- |
| **8장: Processing** |
| 로보이드 표준 GUI |
| Robomation Co., Ltd. |

# Processing - 로보이드 표준 GUI

Processing은 Java을 기본으로 만들어진 디자이너/ 아티스트를 위한 편리한 그래픽 작성 도구인 동시에 그래픽 함수의 모음입니다. 이전 장에서 보았듯이 플래시 또는 브라우저를 사용하여 사용자와 상호 작용하는 방법이 있으나 이는 미리 제작된 플래시나 홈페이지를 상황에 맞추어 불러서 재생하는 경우 입니다. 로보이드가 실행 중에 동적으로 새로운 GUI컴포넌트를 만들고 싶거나 자바스크립트로 자신만의 GUI을 프로그래밍하기 위해 로보이드 스튜디오에는 Processing을 내장하여 표준 GUI기능을 구축하였습니다.

## Processing 이란?

Processing은 MIT Media Lab에서 프로그래밍교육용으로 개발된 “Design By Number”를 그 기원으로 하는 공개 소프트웨어 입니다.

* Processing의 역사

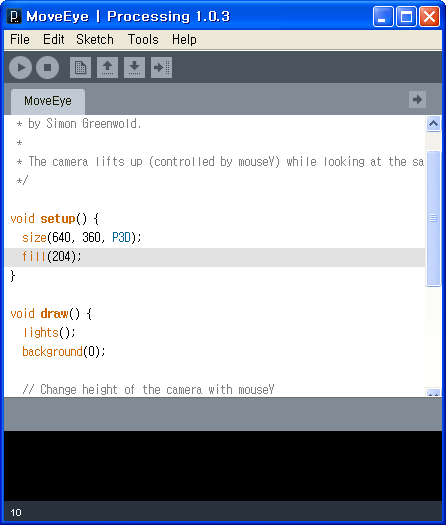
Processing은 2001년 가을 MIT Media Lab의 수업의 하나인, interaction design, interactive art을 연구하기 위한 도구로 처음 사용되었습니다. 이 과목의 이름은 Computational Media Design이라고 하는데 당시 MIT Media Lab에 소속되어 있던 Ben Fry, Tome White, John Maeda 세 사람이 진행 했습니다. 초기의 Processing은 MIT내의 연구자와 학생들 사이에서만 사용 되었으며 사용자들이 기능의 추가나 수정을 했습니다. 이와 같은 과정을 거쳐, 2002년 8월에 alpha버전이 공개 되었습니다. 이때 매우 큰 반응을 얻었고 이 공개에 대해 게시판에 “Processing의 공개를 감사 드립니다”라는 많은 사람들의 메시지가 쇄도 했다고 한다. 이 이후, 주로 미술관련 대학, 공업계 학교에서 프로그래밍 교육을 위한 간편한 환경으로 널리 사용되게 되었습니다.

* Processing과 open source

Processing은 open source입니다. Open source란 소프트웨어의 설계도에 해당하는 소스코드를 누구나 볼 수 있도록 공개 하고, 개량이나 재 배포 등을 권장하는 방식입니다. 로보이드 스튜디오 에 내장된 Processing역시 기본은 동일하지만 사용이 편리하도록 개량한 것입니다.

* Processing과 Flash

로보이드 스튜디오는 플래시를 실행할 수 있지만 플래시콘텐츠를 개발하지는 못합니다. 또한 플래시를 개발하려면 플래시나 Flex, Action Script등의 개발환경이 따로 필요 합니다. 두 가지 개발환경에 익숙해 지기 위해 많은 노력을 해야 하지만 그 외에도 두 프로그램을 긴밀한 통신을 하고 동기화 되어 실행시키려면 상당한 프로그래밍 지식이 요구 됩니다. Processing은 로보이드 스튜디오에 내장이 가능하므로 따로 개발환경을 요구 하지 않습니다. 그래픽을 취급하는 기능 면을 보면, Processing과 Flash는 닮았습니다만, 사용 목적을 보면 별개 입니다. 즉 Flash가 상업용으로 구상한 작품을 완성도 높게 구현하는 도구인 반면, Processing은 아이디어를 확인하는 쪽에 중점이 있습니다. 간단히 몇 줄의 프로그램으로 자신의 생각대로 동작하는 것을 확인한다는 과정이 processing(가공 또는 처리한다)이 가지는 의미 입니다.



< Processing version 1.0.3 >

### 스크립트언어의 장점

로보이드 스튜디오에서는 Processing기능 중 일부 만 사용합니다. 그래픽 처리 기능 외, 편집 또는 배포 기능은 로보이드 스튜디오의 기능을 그대로 이용하므로 따로 공부할 필요가 없습니다. 가장 중요한 차이점은 Processing의 실행을 자바스크립트로 행한다는 점입니다. 자바스크립트 편집기로 편집된 코드를 바로 수행할 수 있도록 하여 별도의 컴파일 과정 없이 가능 합니다. 따라서 기존의 로보이드 제어, 플래시, 브라우저 제어뿐만 아니라 그래픽 UI도 통합적으로 연동 할 수 있게 됨으로 로보이드 스크립트의 기능이 크게 확장 되게 되었습니다.

### LiveConnect 기술

자바 객체가 자바스크립트로 전달되면 자바 스크립트는 이 객체가 마치 자바스크립트의 객체인 것처럼 조작할 수 있다. 자바 객체의 모든 public field와 메서드가 노출 된다. 예를 들면 로보이드와 디바이스는 자바 객체이지만 bindings 과정을 통해 자바 스크립트로 전달 되어 있다. 마찬가지로 Processing을 하나의 자바 객체로 만들어 전달하면 자바스크립트에서는 Processing의 내부 변수를 읽거나 메서드를 자유롭게 호출 할 수 있다. 또한 전달인자를 받고 결과값을 반환하는 메서드도 호출 할 수 있다. 메서드에 전달인자를 건네고 결과값을 돌려 받는 과정에서 필요하면 타입의 변환이 일어난다. 끝으로 자바 메서드는 자바 객체를 결과값으로 반환할 수도 있는데 자바스크립트는 이 반환된 자바 객체의 public필드를 읽고 쓰거나 public 메서드를 호출 할 수 있다. LiveConnect를 이용하면 자바스크립트 코드는 자신만의 자바 객체를 생성 할 수 있다.

## 로보이드 스크립트에 내장된 Processing의 특징

### 그래픽 요소를 사용한 인간-로봇 상호 작용

로봇을 제어하다 보면 어떤 순간의 상태를 사용자에게 전달하거니 사용자의 판단을 기다릴 필요가 있다. 하지만 로보이드 스크립트는 자바 스크립트와 같이 직접 그래픽 요소를 만들거나 제어 하지 못합니다. 브라우저에 사용되는 자바스크립트 역시 브라우저의 DOM(Document Object Model)을 조작하여 그래픽 요소를 제어 합니다. 같은 방법으로 로보이드 스크립트는 그래픽요소를 생성할 수 있습니다. 이때 여러 가지 그래픽 라이브러리가 활용 가능하지만, 바로 사용 가능한 방법은 자바나 Eclipse에 내장된 그래픽 라이브러리인 AWT, Swing, SWT가 있습니다.

* AWT library을 이용한 경우, 다음과 같은 자바스크립트 코드를 start.js에 내장 하시기 바랍니다. 컨텐츠를 실행시키면 화면의 100, 100위치에 아래와 같은 윈도우가 나타납니다.

**importPackage**(java.awt)

f = Frame("Swing/Button")

f.setBounds(100, 100, 200, 150);

b = Button("Hello!");

f.add(b);

f.show();



* Swing library을 이용한 경우,

**importPackage**(javax.swing);

f = JFrame("Swing/Button")

f.setBounds(100, 100, 200, 150);

b = JButton("Hello!");

f.add(b);

f.show();



* SWT library을 이용한 경우,

이 경우는 Eclipse표준 GUI입니다만, 자바스크립트가 사용하기엔 손이 많이 가므로 권장하지 않습니다.

위와 같은 방법으로 사용자와 간단한 대화는 가능합니다. 하지만 버튼이나 레이블 객체 같은 미리 만들어진 제어 객체가 아닌 일반적인 그래픽요소를 프로그램 하기에는 상당한 노력이 필요 합니다. 여기에서 Processing이 왜 필요 한지 이유가 됩니다. 콘덴서에 전하가 차오르는 모양을 형상화 한다든지, 불꽃의 온도를 칼라로 표현하는 등 다양한 표현기법을 제공하는 덕분에 로보이드 프레임워크의 기본 GUI로 채택되었던 것입니다. 여기에서 Processing의 모든 기능과 그 가능성을 논하기에는 어려우므로 직접 자료를 찾아서 공부하기를 바라면서 로보이드 스크립트에 내장된 Processing[[1]](#footnote-2)의 특징과 사용법에 준해 설명하기로 한다.

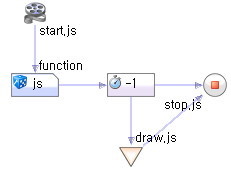
## Visualizing Data with Processing

### Basic Programming

* Sketch을 생성하기 위해 Processing 객체를 제어한다.

Processing을 이용하여 그래픽 작업을 하기 위해 먼저 자바 스크립트에 임베딩되어 있는 processing 이라는 자바 객체를 통해 그래픽 작업을 행할 sketch화면을 만들어야 한다.

이제부터 예제는 다음 로보이드 콘텐츠를 기본으로 작성 된다.



다음 과정을 따라 하면 가장 간단한 GUI화면을 만들 수 있다.

시작 객체인 start.js에 다음과 같은 자바스크립트 코드를 내장한다.

//--- start.js

p = **processing**.create("Hello", 240, 160);

p.background(255);

실행 버튼을 눌러 실행하면 다음과 같은 화면이 왼쪽 상단에 나타납니다.



짐작하겠지만, “Hello”는 창의 타이틀이고 (240,160)은 창의 크기를 지정합니다. 다음 줄의 255는 배경의 밝기를 백색으로 지정합니다. 창이 나타나는 위치는 초기에는 왼쪽 상단이지만 이동한 후에는 그 자리를 계속 유지 합니다. 만약 초기 위치를 항상 원하는 위치에 나타나기를 바란다면, 다음과 같이 창의 위치(100, 100)를 직접 지정합니다.

//--- start.js

p = **processing**.create("Hello", 100, 100, 240, 160);

p.background(255);

Sketch 창을 복수로 생성하는 예를 살펴 봅니다.

//--- start.js

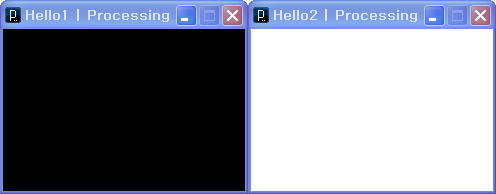
p1 = **processing**.create("Hello1", 100, 100, 240, 160);

p1.background(0);

p2 = **processing**.create("Hello2", 348, 100, 240, 160);

p2.background(255);

다음과 같이 두 개의 sketch창이 나란히 나타납니다.



여기서 기억해야 할 사항은 처음 한 줄밖에 없습니다. Processing객체를 감싸고 있는 자바스크립트 객체 p1 또는 p2가 생성되면 이 이후는 Processing의 모든 public field 와 method을 호출 할 수 있습니다.

* 2D 그래픽 요소를 시간에 맞추어 출력한다.

다음 코드는 각 창에 사각형을 그리고 각각의 내부를 프레임의 증가에 따라 밝기를 변화시키는 예입니다.

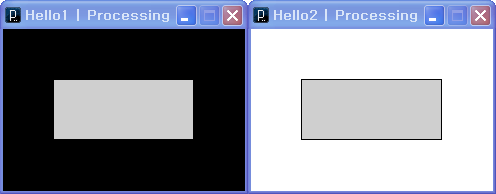
//--- draw.js

p1.fill(frame%256);

p1.rect(50, 50, 140, 60);

p2.fill(frame%256);

p2.rect(50, 50, 140, 60);



다음 코드는 프레임의 증가에 텍스트로 표시하는 예입니다. 시작 객체에 창을 열고, 폰트를 준비합니다. 트리거 객체에서 각 프레임마다 프레임 값을 화면에 출력합니다.

//--- start.js

p = **processing**.create("Hello1", 100, 100, 240, 160);

p.background(255);

f = p.createFont("Arial", 48, **true**);

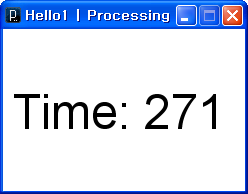
p.textFont(f);

//--- draw.js

p.background(255);

p.fill(0);

p.text("Time: " + frame,10,100)

****

* 3D 그래픽 요소를 시간에 맞추어 출력한다.

Processing은 2D 그래픽뿐 아니라 3D그래픽 기능을 지원합니다. 3D 사용하기 위해 창을 생성할 때 다음과 같은 생성방법을 사용합니다.

p = **processing**.create("Hello1", "P3D", 240, 160);

또는

p = **processing**.create("Hello1", "P3D", 100, 100, 240, 160);

두 번째 argument는 Processing의 그래픽 엔진의 동작 모드를 지정합니다. 지정하지 않으면 “JAVA2D”가 되므로 기본 이차원 그래픽 모드가 됩니다. 그 외 Processing에서는 “P2D”와 “OPENGL”이 가능합니다만, 로보이드 스튜디오에서는 지원하지 않습니다.

이상과 같이 상당히 간단히 GUI를 구현 할 수 가 있습니다만, Processing에 관한 보다 자세한 정보는 <http://processing.org/> 을 참고하시기 바랍니다.

### Advanced Programming

* 애니메이션 만들기

시간 표시 예에서 보면 각 프레임 마다 시간이 하나씩 증가 함을 알 수 있다. 이와 같은 애니메이션을 로보이드 프레임워크에서 실행할 경우 다음 과 같은 문제 가 발생한다. 즉, processing이 화면을 계산하고 그리는 시간과 로보이드 프레임워크에서 데이터를 넘겨 주는 순간이 동시에 일어날 경우 두 thread 간에 충돌이 발생하여 화면의 flickering이 나타날 수 있다. 만약 예와 같이 트리거 객체에서 직접 sketch화면을 조작할 경우는 이 현상이 좀더 심각해 진다.

현재 자바스크립트는 이러한 병렬처리 문제에 대한 대비가 완전하지 못하므로 고 품질의 애니메이션을 위해 다음과 같은 방법을 따라야 한다. Processing에서는 draw()라는 method내의 코드는 화면을 갱신하기 위한 안전한 시간에 수행되므로 이 함수 내에서 기술된 모든 코드는 화면의 번쩍임을 야기하지 않습니다. 로보이드 스크립트에는 이 draw()와 같은 기능을 하는 function draw()를 정의 할 수 있도록 되어 있습니다. 이 function 내에 정의된 자바스크립트 코드는 안전한 시간에 불러지고 실행됩니다. draw()함수의 정의는 어디에 위치해도 상관없지만 정의가 된 순간부터 애니메이션 루프에 의해 일정한 시간 간격으로 실행됩니다. 이 draw()함수는 재 정의가 가능하므로 정해진 시간 후에 알고리즘을 바꾸는 일도 할 수 있지요.

다음 예는 다이스 객체에 내장된 draw()함수를 살펴 봅니다.

우선 시작 객체는 처음으로 돌아가서 다음과 같은 코드입니다.

//--- start.js

p = **processing**.create("Hello1", 240, 160);

p.background(255);

다이스 객체에 내장된 함수 draw()는 흐름에 따라 시작 객체내의 스크립트가 실행된 후 계속해서 불러지게 됩니다. 따라서 여기의 변수 p는 시작객체의 p와 동일합니다.

//--- function

**function** draw()

{

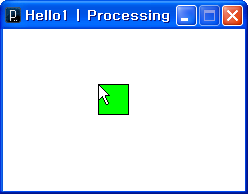
p.background(255);

p.fill(0,255,0);

p.rect(p.mouseX, p.mouseY, 30, 30);

}

draw()함수의 첫 줄은 애니메이션을 위해 화면을 완전히 지웁니다. 둘째 줄은 사각형을 그리되 마우스 커서가 위치한 자리에 그리게 됩니다. 실행 시켜 보면 마우스를 따라 다니는 녹색 사각형을 볼 수 있습니다. 마우스 값 대신에 센서 값을 대치 하면 센서의 변화에 따른 움직임을 만들 수 있겠지요.



* 복수의 창에서 애니메이션 만들기

Draw함수를 사용할 경우, Processing창을 두 개 이상 만들면 곤란한 문제가 발생합니다. 각 창에 할당된 thread는 모두 한번씩 draw()함수를 호출하게 됩니다. 만약 창이 두 개 라면 draw()함수를 한 프레임당 두 번을 호출하게 됩니다. 이때 자신을 부른 창에만 화면을 조작 할 수 있으나 함수 내에서는 구별이 되지 않으므로 원하는 동작이 아닌 이상한 현상이 발생합니다. 창마다 다른 draw()함수를 할당하기 위해 draw()함수의 지정이 가능한 다음과 같은 생성자를 사용 합니다.

//--- start.js

p = **processing**.create("Hello1", 240, 160);

p.background(255);

첫 줄은 p 객체는 draw()를 호출하지만 두 번째 줄의 p2객체는 paint()라는 함수를 호출합니다. 즉, 마지막 argument는 호출할 함수의 이름을 지정합니다.

//--- functions

**function** draw()

{

p.background(255);

p.fill(0, 255, 0) //fill green

p.rect(p.mouseX, p.mouseY, 30, 30);

}

**function** paint()

{

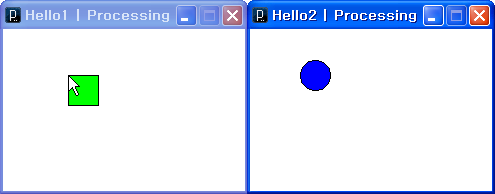
p2.background(255);

p2.fill(0, 0, 255) //fill green

p2.ellipse(p.mouseX, p.mouseY, 30, 30);

}

실행을 시켜보면, 두 개의 창에 각각 사각형과 원이 나타납니다. Hello1창에 마우스를 놓고 움직이면 사각형과 원이 동시에 움직입니다만, Hello2에 마우스를 올려놓으면 아무런 반응이 없습니다. 그 이유는 함수내의 마우스 정보가 p.mouseX와 같이 p객체에서만 참조 하기 때문입니다. paint함수내의 p.mouseX, p.mouseY를 p2.mouseX, p2.mouseY로 바꾸면 각각 독립적으로 동작합니다.



변수를 읽는 것은 이와 같이 구별 없이 해도 동작합니다만, draw()에서 p2의 그래픽요소를 조작하거나 paint()에서 p1을 조작하는 것은 규칙위반이 되니 특히 조심하여야 합니다. 즉, p1과 p2는 서로 다른 그래픽 thread에서 실행 중이므로 충돌현상을 일으키게 됩니다

* 인터랙티브 애니메이션 만들기

마우스와 키보드 입력, 그리고 로봇의 센서정보를 종합적으로 이용하므로 로보이드 스튜디오의 장점이 발휘됩니다. Processing은 draw() 함수 내에서 마우스나 키보드의 입력 정보를 정확히 전달하는 기능을 가지고 있어 안심하고 사용자의 반응에 대응 할 수 있습니다. 다음 자바스크립트 코드를 가지고 마우스와 키 입력에 대응하는 법을 알아봅니다.

//--- functions

**function** draw()

{

**if**(p.mousePressed == **true**)

{

p.background(255);

}

**else** p.background(0);

}

마우스의 어떤 버튼을 눌러도 배경이 white, 놓으면 black이 된다. 만약 왼쪽 버튼을 누를 경우에만 반응 하려면, 다음과 같이 된다.

//--- functions

**function** draw()

{

**if**(p.mousePressed == **true**)

{

**if**(p.mouseButton == p.LEFT)

p.background(255);

}

**else** p.background(0);

}

키보드 입력에 대응하는 코드는 아래와 같다.

//--- functions

**function** draw()

{

**if**(p.keyPressed == **true**)

{

p.background(255);

}

**else** p.background(0);

}

스페이스 키에만 반응하게 하는 경우는 다음과 같다.

//--- functions

**function** draw()

{

**if**(p.keyPressed == **true**)

{

**if**(p.key == 32) p.background(255);

}

**else** p.background(0);

}

* 이벤트 기반의 인터랙티브 애니메이션 만들기

마우스의 위치나 클릭 정보, 키 입력 등은 사용자가 임의로 행하는 행동이므로 정확한 시간을 예측 할 수는 없지요. PC에서는 이러한 정보를 event로 처리 하여, 이벤트 발생시 이 정보를 원하는 곳으로 전달하는 메커니즘이 있습니다. mouseX나 mouseY는 Processing이 제공하는 정보이긴 하지만 이벤트 방식은 아니며, 언제라도 읽을 수 있습니다. 하지만 키 입력은 이런 식으로 확인은 불가 합니다. 확인 주기가 느리면 휙 지나가 버릴 수가 있으니까요. 로보이드 스크립트에는 draw() 함수와 유사하게 정확하게 이벤트를 받을 수 있고 동시에 안전하게 그래픽 정보를 갱신하는 함수를 제공합니다.

Mouse Event: mousePressed(), mouseReleased(), mouseMoved(), mouseDragged()

Keyboard Event: keyPressed(), keyReleased();

모두 6 종류가 있습니다. 또한 이 방식은 draw()함수를 사용하지 않고 어떤 장소에서도 그래픽 요소를 제어 할 수 있는 장점이 있습니다. 이제 draw()함수를 제거하고 다음과 같이 수정합니다.

//--- functions

**function** mousePressed()

{

p.background(255);

}

**function** mouseReleased()

{

p.background(0);

}

마우스를 드래깅하면 선을 긋는 코드는 아래처럼 간단히 작성 됩니다. 이때 pmouseX, pmouseY는 이전 마우스 커서의 좌표를 의미합니다.

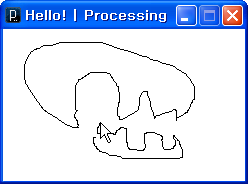
//--- functions

**function** mouseDragged()

{

p.line(p.pmouseX, p.pmouseY, p.mouseX, p.mouseY);

}



* 애니메이션 루프 제어하기

Processing의 애니메이션 루프는 초당 60회의 속도로 새로운 화면을 만들어 냅니다. 만약 표현하고자 하는 데이터의 변화가 그렇게 빠르지 않다면 화면 갱신속도를 줄여 CPU의 부담을 들어 줄 수 있습니다.

//--- start.js

p = **processing**.create("Hello1", 240, 160);

p.frameRate(5);

위 코드는 초당 5 프레임으로 화면을 갱신하므로 마우스의 움직임 에 대해 사각형의 움직임에 좀 둔해 보입니다. 즉 draw()함수를 일초당 5회 속도로 호출하게 됩니다.

//--- functions

**function** draw()

{

p.background(255);

p.fill(0, 255, 0) //fill green

p.rect(p.mouseX, p.mouseY, 30, 30);

}

또 다른 방법은 화면의 갱신을 로보이드 콘텐츠 재생 속도로 동기화 하는 방법이 있습니다. 우선 다음과 같이 noLoop()명령을 사용하여 자체 루프를 중단시킵니다.

//--- start.js

p = **processing**.create("Hello1", 240, 160);

p.noLoop();

이때 draw()함수는 다음과 같습니다. posX의 값에 따라 원이 오른쪽으로 이동하게 됩니다.

//--- functions

**function** draw()

{

p.background(255);

p.fill(0, 255, 0) //fill green

p.ellipse(posX, 75, 30, 30);

}

트리거 객체에서 20msec 간격으로 redraw()함수를 호출합니다.

//--- draw.js

posX = frame%120;

p.redraw();

**false**;

실행하면 초당 50 pixel의 속도로 오른쪽으로 이동하고 120pixel 부근에서 다시 원점으로 되돌아가는 동작을 반복합니다.

다음은 5 프레임속도로 redraw()함수를 호출합니다.

//--- draw.js

**if**(frame%10 == 0)

{

posX = frame%120;

p.redraw();

}

**false**;

실행하면 10 pixel씩 건너 뛰면서 이동하는 것을 알 수 있습니다. 물론 Basic Programming에서와 같은 방법으로 화면을 직접 제어 할 수 있습니다. 우선 다이스 객체의 draw()함수를 삭제 하고, 트리거 객체에서 직접 화면을 조작 합니다.

//--- draw.js

**if**(frame%10 == 0)

{

posX = frame%120;

p.background(255);

p.fill(0, 255, 0) //fill green

p.ellipse(posX, 75, 30, 30);

p.redraw();

}

마찬가지로 똑같이 동작하게 보이지만 프레임 속도를 높이면 flickering현상이 좀 보입니다.

이상의 예를 보았듯이 애니메이션 방법은 표현 하고자 하는 상황에 맞추어 선택해야 합니다.

로보이드로부터 가끔 이벤트가 전달될 경우에는 직접 화면을 조작하는 편이 간편한 반면, 복잡한 고 품질의 그래픽 표현을 하는 경우에는 draw() 함수를 사용하는 쪽이 이상적입니다.

* 공통 변수 사용시 주의 할 점

로보이드의 센서 값을 화면상에 나타내고자 할 때 대개 하나의 변수를 매개로 하게 된다. 변수 x를 마이크의 출력이라고 하고 processing의 애니메이션 loop에서 이 변수 값을 읽어서 도형을 그리게 되는 경우를 생각해 봅니다.

시작 객체에는 보편적으로 변수의 정의를 담게 된다. 두 번째 줄의 디바이스 정의는 로보이드의 마이크 장치를 지정한다.

//--- start.js

p = **processing**.create("Hello1", 240, 160);

mic = **robot**.findDevice("NetBrain.Microphone");

Draw() 함수의 자바 스크립트 코드는 다음과 같습니다

//--- functions

**function** draw()

{

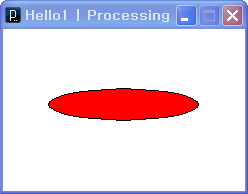
p.background(255);

p.fill(255, 0, 0);

p.ellipse(120, 75, 150, v);

}

v변수의 값에 따라 아래와 같은 입술모양이 벌어지거나 닫히게 됩니다.



이때 v 변수를 트리거 객체에서 하게 된다면 하나의 변수를 양쪽 thread에서 공유 한 모양이 되겠지요. 각 thread는 서로 다른 타이밍으로 돌아가므로 v 변수가 계산되는 동안에 draw()함수내의 v값을 일정하지 않습니다

//--- draw.js

v = 0;

**for**(i = 0; i < 320; i++)

{

**if**(mic.read(i) > 0)

v += mic.read(i);

}

v = v/3000 + 1;

**false**;

목소리의 크기에 따라 v가 커집니다. 위의 코드를 실행해보면 화면의 입술이 매우 불안정해 짐을 느낄 수 있습니다. 그 이유는 각 코드 조각이 실행되는 thread가 달라서 입니다. CPU에서는 짧은 시간에 수없이 thread간의 스위칭이 일어나게 됩니다. 변수 v의 계산이 미처 끝나기 전에 그래픽 도형을 그리면 당연히 불안정해 지겠지요. 이 현상을 최소한으로 억제하는 방안은 버퍼 변수를 도입하는 것입니다. 즉, 계산에 사용되는 변수와 공통으로 사용하는 변수를 분리하여, 계산이 완결된 후 공통변수를 업데이트 하면 됩니다.

//--- functions

**function** draw()

{

p.background(255);

p.fill(255, 0, 0);

p.ellipse(120, 75, 150, s);

}

//--- draw.js

v = 0;

**for**(i = 0; i < 320; i++)

{

**if**(mic.read(i) > 0)

v += mic.read(i);

}

s = v/3000 + 1;

**false**;

위와 같이 공통 변수 s을 사용하시기 바랍니다. 이 경우 변수가 정수 값인 경우에는 thread스위칭에 대해 안전하므로 충돌현상을 막을 수 있습니다. 변수가 정수가 아니거나 배열인 경우는 100% 보장은 안 되는 점을 상기하시기 바랍니다.

### Embedding in a Swing Application

때에 따라서 Processing만으로 정보를 나타내기에 부족한 경우가 있다. 부가적인 인터페이스, 예를 들면 익숙한 버튼, 슬라이더 바 등을 같은 화면에 구상 한다면 더 효과적일 것이다. 실제 Processing의 Sketch화면은 커스텀 그래픽 컴포넌트와 동일합니다. 따라서 어떠한 호스트 컴포넌트에도 연결되므로 TV같이 애니메이션 화면과 선택 버튼이 동시에 포함된 인터페이스를 만들 수 있습니다. 다음 코드를 살펴 봅니다.

//--- start.js

**importPackage**(java.awt, javax.swing);

f = JFrame("Swing+Sketch")

f.setBounds(100, 100, 250, 150);

f.setLayout(**new** BorderLayout());

p = **processing**.createSketch(250, 50);

f.add(p, BorderLayout.CENTER);

b = JButton("Hello!");

f.add(b, BorderLayout.SOUTH);

f.show();

p.background(255, 0 , 0);

객체 p는 Processing창이 아니라 sketch객체입니다. 따라서 다른 호스트 컴포넌트, 즉 Swing의 JFrame이나 AWT의 Frame 에 연결 시킬 수 있습니다.



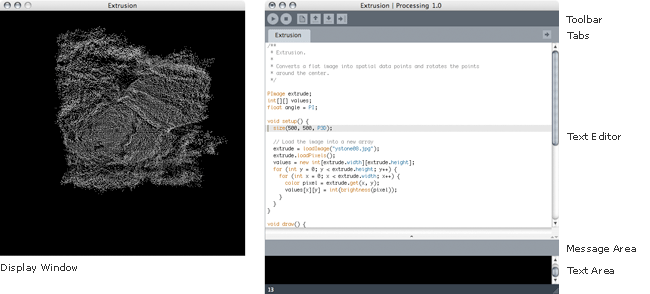
상단의 sketch는 이전과 같이 제어가 가능하며, 하단의 버튼은 Swing의 버튼 객체 이므로 자세한 프로그래밍 방법은 Java Swing의 기술 문서를 참조하기 바랍니다.

로보이드 스크립트는 자바스크립트의 LiveConnect기술을 기반으로 하고 있습니다. 따라서 속도 면에서 일반 프로그램언어를 사용하는 경우에 비해 매우 느리다는 점을 염두에 두시고 너무 많은 연산을 하지 않는 간단한 그래픽에만 적용하기를 바랍니다.

## Processing IDE 사용법

### ENVIRONMENT (IDE). The Processing environment includes a text editor, a compiler, and a display window. It enables the creation of software within a carefully designed set of constraints.

##### **[Top](http://processing.org/reference/environment/#top)**Processing Development Environment (PDE)

The Processing Development Environment (PDE) consists of a simple text editor for writing code, a message area, a text console, tabs for managing files, a toolbar with buttons for common actions, and a series of menus. When programs are run, they open in a new window called the display window.  
  
  


Software written using Processing are called sketches. These sketches are written in the text editor. It has features for cutting/pasting and for searching/replacing text. The message area gives feedback while saving and exporting and also displays errors. The console displays text output by Processing programs including complete error messages and text output from programs with the print() and println() functions. The toolbar buttons allow you to run and stop programs, create a new sketch, open, save, and export:

|  |  |  |
| --- | --- | --- |
| http://processing.org/reference/environment/images/play.gif |  | Run Compiles the code, opens a display window, and runs the program inside. Hold down shift to Present instead of Run. |
| http://processing.org/reference/environment/images/stop.gif |  | Stop Terminates a running program. |
| http://processing.org/reference/environment/images/new.gif |  | New Creates a new sketch (project) in the current window. To create a new sketch in its own window, use File → New. |
| http://processing.org/reference/environment/images/open.gif |  | Open Provides a menu with options to open files from the sketchbook, open an example, or open a sketch from anywhere on your computer. Opening a sketch from the toolbar will replace the sketch in the current window. To open a sketch in a new window, use File → Open. |
| http://processing.org/reference/environment/images/save.gif |  | Save Saves the current sketch to its current location. If you want to give the sketch a different name, select “Save As” from the File menu. |
| http://processing.org/reference/environment/images/export.gif |  | Export Exports the current sketch as a Java Applet embedded in an HTML file. The folder containing the files is opened. Click on the index.html file to load the software in the computer's default web browser. Hold down shift to export an application instead of an applet. Note that exporting a sketch will delete the previous contents of the “applet” or “application.xxxx” folders. |

Additional commands are found within the five menus: File, Edit, Sketch, Tools, Help. The menus are context sensitive which means only those items relevant to the work currently being carried out are available.

###### File

* New (Ctrl+N)  
  Creates a new sketch in a new window, named as the current date is the format "sketch\_YYMMDDa".
* Open (Ctrl+O)  
  Open a sketch in a new window.
* Sketchbook  
  Open a sketch from the sketchbook folder.
* Examples  
  Open one of the examples included with Processing.
* Close (Ctrl+W)  
  Close the sketch in the frontmost window. If this is the last sketch that's open, you will be prompted whether you would like to quit. To avoid the prompt, use Quit instead of Close when you want to exit the application.
* Save (Ctrl+S)  
  Saves the open sketch in it's current state.
* Save as... (Ctrl+Shift+S)  
  Saves the currently open sketch, with the option of giving it a different name. Does not replace the previous version of the sketch.
* Export (Ctrl+E)  
  Exports a Java Applet and creates and embeds it into an HTML file. After the files are exported, the directory containing the exported files is opened. There is more information about exporting [here](http://processing.org/reference/environment/export.html). Note that exporting a sketch will delete the previous contents of the “applet” folder.
* Export Application (Ctrl+Shift+E)  
  Exports as a Java application as an executable file. Opens the directory containing the exported files. Note that exporting a sketch will delete the previous contents of the “application.xxxxx” folders.
* Page Setup (Ctrl+Shift+P)  
  (Not working yet)
* Print (Ctrl+P)  
  (Not working yet)
* Preferences (Ctrl+,)  
  Allows you to change some of the ways Processing works.
* Quit (Ctrl+Q)  
  Exits the Processing Environment and closes all Processing windows.

###### Edit

* Undo (Ctrl+Z)  
  Reverses the last command or the last entry typed. Cancel the Undo command by choosing Edit » Redo.
* Redo (Ctrl+Y)  
  Reverses the action of the last Undo command. This option is only available, if there has already been an Undo action.
* Cut (Ctrl+X)  
  Removes and copies selected text to the clipboard (an off-screen text buffer)
* Copy (Ctrl+C)  
  Copies selected text to the clipboard.
* Copy for Discourse (Shift+Ctrl+C)  
  Formats code so that it will appear in the [Processing Discourse](http://www.processing.org/discourse/) the same way it appears in the Processing environment and copies it to the clipboard so it can be pasted somewhere else.
* Paste (Ctrl+V)  
  Inserts the contents of the clipboard at the location of the cursor, and replaces any selected text.
* Select All (Ctrl+A)  
  Selects all of the text in the file which is currently open in the text editor.
* Comment/Uncomment (Ctrl+/)  
  Comments the selected text. If the selected text is already commented, it uncomments it.
* Increase Indent (Ctrl+])  
  Indents the selected text two spaces.
* Decrease Indent (Ctrl+[)  
  If the text is indented, removes two spaces from the indent.
* Find (Ctrl+F)  
  Finds an occurance of a text string within the file open in the text editor and gives the option to replace it with a different text.
* Find Next (Ctrl+G)  
  Finds the next occurance of a text string within the file open in the text editor.

###### Sketch

* Run (Ctrl+R)  
  Runs the code (compiles the code, opens the display window, and runs the program inside)
* Present (Ctrl+Shift+R)  
  Runs the code in the center of the screen with a neutral background. Click the "stop" button in the lower left to exit the presentation.
* Stop  
  If the code is running, stops the execution. Programs written with the Basic Mode or using the draw() structure are stopped automatically after they draw.
* Import Library  
  Adds the necessary import statements to the top of the current sketch. For example, selecting Sketch » Import Library » video adds the statement "import processing.video.\*;" to the top of the file. These import statements are necessary for using the Libraries.
* Show Sketch Folder  
  Opens the directory for the current sketch.
* Add File  
  Opens a file navigator. Select an image, font, or other media files to add it to the sketches "data" directory.

###### Tools

* Auto Format (Ctrl-T)  
  Attempts to format the code into a more human-readable layout. Auto Format was previously called Beautify.
* Create Font...  
  Converts fonts into the Processing font format and adds to the current sketch. Opens a dialog box which give options for setting the font, it's size, if it is anti-aliased, and if all characters should be generated. If the "All Characters" options is selected, non-English characters such as ü and Å are generated, but the font file is larger in size. The amount of memory required for the font is also determined by the size selected. Processing fonts are textures, so larger fonts require more image data.
* Color Selector  
  Interface for selecting colors.
* Archive Sketch  
  Archives a copy of the current sketch in .zip format. The archive is placed in the same directory as the sketch.
* Fix Encoding and Reload  
  Sketches that contain non-ASCII characters and were saved with Processing 0140 and earlier may look strange when opened. Garbled text and odd characters may appear where umlauts, cedillas, and Japanese formerly lived. This will reload your sketch using the same method as previous versions of Processing, at which point you can re-save it which will write a proper UTF-8 version.

###### Help

* Getting Started  
  Opens the reference for the Processing Environment in the default Web browser.
* Troubleshooting  
  Opens the troubleshooting information in the default Web browser.
* Reference   
  Opens the reference in the default Web browser. Includes reference for the language, programming environment, libraries, and a language comparison.
* Find in Reference (Ctrl+Shift+F)  
  Select a word in your program and select "Find in Reference" to open that reference HTML page.
* Frequently Asked Questions  
  Answers to some basic question about the Processing project.
* Visit Processing.org (Ctrl+5)  
  Opens default Web browser to the Processing.org homepage.
* About Processing  
  Opens a concise information panel about the software.

##### **[Top](http://processing.org/reference/environment/#top)**Sketchbook

All Processing projects are called sketches. Each sketch has its own folder. The main program file for each sketch has the same name as the folder and is found inside. For example, if the sketch is named "Sketch\_123", the folder for the sketch will be called "Sketch\_123" and the main file will be called "Sketch\_123.pde". The PDE file extension is an acronym for the Processing Development Environment.

A sketch folder sometimes contains other folders for media files and code libraries. When a font or image is added to a sketch by selecting "Add File..." from the Sketch menu, a "data" folder is created. Files may also be added to your Processing sketch by dragging them into the text editor. Image and sound files dragged into the application window will automatically be added to the current sketch's "data" folder. All images, fonts, sounds, and other data files loaded in the sketch must be in this folder. Sketches are stored in the Processing folder, which will be in different places on your computer or network depending if you use PC, Mac, or Linux and how the preferences are set. To locate this folder, select the "Preferences" option from the "File" menu (or from the “Processing” menu on the Mac) and look for the "Sketchbook location".

It is possible to have multiple files in a single sketch. These can be Processing text files (the extension .pde) or Java files (the extension .java). To create a new file, click on the arrow button to the right of the file tabs. This button gives access to creating, deleting, and renaming the files that comprise the current sketch. You can write functions and classes in new PDE files and you can write any Java code in files with the JAVA extension. Working with multiple files makes it easier to re-use code and to separate programs into small sub-programs.

##### **[Top](http://processing.org/reference/environment/#top)**Tabs, Multiple Files, and Classes

It can be inconvenient to write a long program within a single file. When programs grow to hundreds or thousands of lines, breaking them into modular units helps manage the different parts. Processing manages files with the Sketchbook and each sketch can have multiple files that are managed with tabs. The arrow button in the upper-right corner of the Processing Development Environment is used to manage these files. Click this button to reveal options to create a new tab, rename the current tab, and delete the current tab. If a project has more than one tab, they can also be hidden and revealed. Hiding a tab temporarily removes that code from the sketch (it will not be compiled with the program when you press Run).

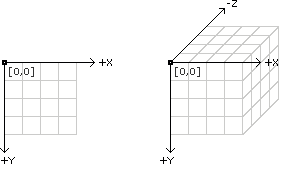
Tabs are intended for more advanced users, and for this reason, the menu that controls the tabs is intentionally made less prominent.

For programmers familiar with Java. When a program with multiple tabs is run, the code is grouped together and the classes in other tabs become inner classes. Because they're inner classes, they cannot have static variables. Simply place the "static" variable outside the class itself to do the same thing (it need not be explicitly named "static" once you list it in this manner). If you don't want code to be an inner class, you can also create a tab with a .java suffix, which means it will be interpreted as straight java code. It is also not possible to use static classes in separate tabs. If you do this, however, you'll need to pass the PApplet object to that object in that tab in order to get PApplet functions like line(), loadStrings() or saveFrame() to work.

Currently, the tabs get truncated when there are too many ([Bug 54](http://dev.processing.org/bugs/show_bug.cgi?id=54)).

##### **[Top](http://processing.org/reference/environment/#top)**Coordinates

Processing uses a Cartesian coordinate system with the origin in the upper-left corner. If your program is 320 pixels wide and 240 pixels high, coordinate [0, 0] is the upper-left pixel and coordinate [320, 240] is in the lower-right. The last visible pixel in the lower-right corner of the screen is at position [319, 239] because pixels are drawn to the right and below the coordinate.



Processing can also simulate drawing in three dimensions. At the surface of the image, the z-coordinate is zero, with negative z-values moving back in space. When drawing in simulated 3D, the "camera" is positioned in the center of the screen.

##### **[Top](http://processing.org/reference/environment/#top)**Programming Modes

Processing allows people to program at three levels of complexity: Basic Mode, Continuous Mode, and Java Mode. People new to programming should begin with the Basic Mode to learn about coordinates, variables, and loops before moving to Continuous and Java modes.

###### Basic

This mode is used drawing static images and learning fundamentals of programming. Simple lines of code have a direct representation on the screen. The following example draws a yellow rectangle on the screen:

size(200, 200);

background(255);

noStroke();

fill(255, 204, 0);

rect(30, 20, 50, 50);

###### Continuous

This mode provides a setup() structure that is run once when the program begins and a draw() structure which by default continually loops through the code inside. This additional structure allows writing custom functions and classes and using keyboard and mouse events.  
  
This example draws four circles on the screen and utilizes a custom function called circles(). The circles() function is not a part of the Processing language, but was written for this example. The code in draw() only runs once because noLoop() is called in setup().

void setup() {

size(200, 200);

noStroke();

background(255);

fill(0, 102, 153, 204);

smooth();

noLoop();

}

void draw() {

circles(40, 80);

circles(90, 70);

}

void circles(int x, int y) {

ellipse(x, y, 50, 50);

ellipse(x+20, y+20, 60, 60);

}

This example draws rectangles that follow the mouse position (stored in the system variables mouseX and mouseY). The **draw()** block runs forever until the program is stopped, thus creating the potential for motion and interaction.

void setup() {

size(200, 200);

rectMode(CENTER);

noStroke();

fill(0, 102, 153, 204);

}

void draw() {

background(255);

rect(width-mouseX, height-mouseY, 50, 50);

rect(mouseX, mouseY, 50, 50);

}

###### Java

This mode is the most flexible, allowing complete Java programs to be written from inside the Processing Environment (as long as they're still subclasses of PApplet). This mode is for advanced users only and is not really recommended. Using this mode means that any additional tabs will no longer be inner classes, meaning that you'll have to do extra work to make them communicate properly with the host PApplet. It is not necessary to use this mode just to get features of the Java language.

public class MyDemo extends PApplet {

void setup() {

size(200, 200);

rectMode(CENTER);

noStroke();

fill(0, 102, 153, 204);

}

void draw() {

background(255);

rect(width-mouseX, height-mouseY, 50, 50);

rect(mouseX, mouseY, 50, 50);

}

}

##### **[Top](http://processing.org/reference/environment/#top)**Rendering Modes

Processing currently has four rendering modes. The programs written with Processing can be rendered using the Java 2D drawing libraries, a custom 3D engine called P3D, and through OpenGL using the JOGL interface, and a custom 2D engine called P2D. The rendering mode is specified through the size() function. A large effort has been made to make the Processing language behave similarly across the different rendering modes, but there are currently some inconsistencies.

For more information, see the [size()](http://processing.org/reference/size_.html) reference entry.

##### **[Top](http://processing.org/reference/environment/#top)**Applet Export

The export feature packages a sketch to run within a Web browser. When code is exported from Processing it is converted into Java code and then compiled as a Java Applet. When a project is exported, a series of files are written to a folder named "applet" that is created within the sketch folder. All files from the sketch folder are exported into a single JAR file with the same name as the sketch. For example, if the sketch is named "Sketch\_123", the exported file will be called "Sketch\_123.jar".

The applet folder contains the following:

* **index.html**  
  HTML file with the applet embedded and a link to the source code and the Processing homepage. Doubleclick this file to open it in the default Web browser.
* **Sketch\_123.jar**  
  Java Archive containing all necessary files for the sketch to run. Includes the Processing core classes, those written for the sketch, and all included media files from the data folder such as images, fonts, and sounds.
* **Sketch\_123.java**  
  The JAVA file generated by the pre-processor from the PDE file. This is the actual file that is compiled into the Applet by the Java Compiler used in Processing.
* **Sketch\_123.pde**  
  The original program file. It is linked from the index.html file.
* **loading.gif**  
  An image file displayed while the program is loading in a Web browser.

Every time a sketch is exported, the contents of the "applet" folder are deleted and the files are written from scratch. Any changes previously made to the "index.html" file are lost. To customize the HTML that's included when an applet is exported, copy the applet.html file from Processing → lib → export to the root of your sketch folder.

Media files not needed for the applet should be deleted from the data folder before exporting to keep the file size small. For example, if there are unused images in the "data" folder, they will be added to the JAR file, thus needlessly increasing its size.

When exporting an applet, a description can be included on the default HTML page that's created. Processing will include everything from the first instance of /\*\* to a closing \*/ as a javadoc comment. So you can add this to your sketch:

/\*\*

\* You can't rock Processing like I can rock Processing.

\*/

and that text would automatically be embedded in the exported html page. Like a javadoc comment, the text can (and should) be any valid html since it will be embedded directly in the page. Asterisks at the left of each line will be removed.

##### **[Top](http://processing.org/reference/environment/#top)**Application Export

In addition to exporting Java Applets for the Web, Processing can also export Java Applications for the Linux, Macintosh, and Windows platforms. When "Export Application" is selected from the "File" menu, a dialog box opens and you can select which platforms you want to export to. You may also select if you want the application to run full screen (in present mode). A folder will be created for each of the operating systems selected; each folder contains the application, the source code for the sketch, and all required libraries for a specific platform.

Making applications can be trickier than applets. Some hints and notes follow below. If you find problems, [file a bug](http://dev.processing.org/bugs/).

* Just like when exporting as an applet, the "application.xxxx" folders will be removed completely on export.
* It is important that you don't have a method named main() in your sketch, unless you know what you're doing (writing your own main). Otherwise this will fool the preprocessor into thinking you have a clue, when in fact you don't.
* If running in "Java" mode, where your code starts "public class blah extends PApplet", you'll need to write your own main() method in order for Export to Application to work. It should look something like this:
* static public void main(String args[]) {
* PApplet.main(new String[] { "YourClassName" });
* }

Not doing this, or using your own main can cause problems internally with variables not being properly set up. If you opt not to use PApplet.main(), make sure you read the source code for it so that you understand how it works, and don't whine if it breaks.

* The "Movie" and "Capture" examples do not yet work properly, this will be fixed shortly. ([Bug 230](http://dev.processing.org/bugs/show_bug.cgi?id=230))
* Exporting sketches that use the video library may be broken, it's not clear the extent of this issue ([Bug 231](http://dev.processing.org/bugs/show_bug.cgi?id=231)), but it will be repaired for a future release.
* When exporting for Mac OS X from Windows, be sure to read through the readme.txt file in the application.macosx folder—your application likely won't work on Mac OS X without a minor modification.
* The Mac OS X export is a nice .app bundle like a regular OS X application. You can change the icon or edit its settings by using "Show Package Contents" and editing Info.plist or replacing sketch.icns with something more exciting.
* Windows is a shell .exe that calls the code and requires the "lib" folder along with it be kept intact. Someday we might make Windows be a nice single .exe file, but this requires more time to implement, and more specifically, make work consistently with the libraries. If you're finicky about this, use a Java EXE generator such as [JSmooth](http://jsmooth.sourceforge.net/) or [launch4j](http://launch4j.sourceforge.net/).
* To set the icon used on the title bar for Windows, use the following code: (submitted by dxtx)
* ImageIcon titlebaricon = new ImageIcon(loadBytes("myicon.gif"));
* frame.setIconImage(titlebaricon.getImage());

Note that within the PDE, the icon will only show up when a code folder is in use or a library has been imported. However, it should show up on all occasions when exported to an application.

* Linux is just a shell script, which can probably be used on most Unix platforms (there's almost nothing to it).
* When distributing your application, the "source" folder can be removed from the export if you'd like, but other files (such as the lib folder and any .dll files or whatever) should be left intact otherwise the application may not work properly.
* If you just want an executable jar file, export as an applet. Executable jar files will most likely not work with libraries because the path information will not be properly set, and the additional library jars won't be referenced.
* Library writers can now specify what files to export for each platform, see the updated libraries/howto.txt for more information.
* Your current memory settings will be exported with the application. If you've set outrageous memory requirements, you might want to undo that before exporting for others, or edit the exported files by hand (Contents/Resources/Info.plist on Mac OS X and lib/args.txt on Windows).
* If you want to replace (or add) titlebar text, just do this in setup():

frame.setTitle("This is in the titlebar!");

(thanks dxtx for pointing this out)

* Holding down shift while clicking "Export" on the toolbar will export to application instead of an applet.
* Except for Mac OS X, users will need to install Java to use your exported application. To avoid the installation requirement, you can also copy the "java" folder from Processing to the same folder as your application. This also ensures that a certain version of the Java VM is used. At least for now, the error message shown when Java is missing may be nonexistant or cryptic. Similar to Processing itself, the folder can also by a symbolic link (but not a Mac OS X or Windows shortcut) to the Java runtime installation.
* Someday we'll also add features like icon files to the applications produced by Processing. Or at least a default icon for the exported stuff. Or maybe even a splash screen while Java is loading. Again, if you want more control, use your own application generator (see above for Windows, or use the Java package maker whatchamacallit on Mac OS X).

##### **[Top](http://processing.org/reference/environment/#top)**Applet Security Restrictions

Applets running inside a web browser have several security restrictions, in order to protect web users. For instance, an applet cannot read files from the local disk or connect to servers other than the server from which it came. To avoid these restrictions, it's possible to "sign" an applet. In a signed applet, a dialog box will appear to ask the user whether they trust you as the provider of the applet, and if it's ok for the applet to perform actions that might be a security risk. More information can be found in Sun's documentation.

##### **[Top](http://processing.org/reference/environment/#top)**"Present" Features

* The ESC key will quit a sketch, even in Present mode. To prevent this from happening, intercept the ESC on keyPressed() so that it isn't passed through to PApplet. Use the following code to prevent ESC from quitting the application:
* void keyPressed() {
* if (key == ESC) {
* key = 0; // Fools! don't let them escape!
* }
* }
* You can hide the stop button with the --hide-stop command line option to PApplet. More details about command line options are above in the "Export to Application" section. From inside the Processing environment, you can't hide the stop button (unless your sketch window obscures it anyway) easily, so better to export as an application.
* To make a sketch run with a window the size of the screen, use size(screen.width, screen.height). The "screen" variable will be initialized with the width and height of your main display (your default monitor).

****

**Shape** -------------------------  
  
[PShape](http://processing.org/reference/PShape.html)

*2D Primitives*

[arc()](http://processing.org/reference/arc_.html)  
[ellipse()](http://processing.org/reference/ellipse_.html)  
[line()](http://processing.org/reference/line_.html)  
[point()](http://processing.org/reference/point_.html)  
[quad()](http://processing.org/reference/quad_.html)  
[rect()](http://processing.org/reference/rect_.html)  
[triangle()](http://processing.org/reference/triangle_.html)

*Curves*

[bezier()](http://processing.org/reference/bezier_.html)  
[bezierDetail()](http://processing.org/reference/bezierDetail_.html)  
[bezierPoint()](http://processing.org/reference/bezierPoint_.html)  
[bezierTangent()](http://processing.org/reference/bezierTangent_.html)  
[curve()](http://processing.org/reference/curve_.html)  
[curveDetail()](http://processing.org/reference/curveDetail_.html)  
[curvePoint()](http://processing.org/reference/curvePoint_.html)  
[curveTangent()](http://processing.org/reference/curveTangent_.html)  
[curveTightness()](http://processing.org/reference/curveTightness_.html)

*3D Primitives*

[box()](http://processing.org/reference/box_.html)  
[sphere()](http://processing.org/reference/sphere_.html)  
[sphereDetail()](http://processing.org/reference/sphereDetail_.html)

*Attributes*

[ellipseMode()](http://processing.org/reference/ellipseMode_.html)  
[noSmooth()](http://processing.org/reference/noSmooth_.html)  
[rectMode()](http://processing.org/reference/rectMode_.html)  
[smooth()](http://processing.org/reference/smooth_.html)  
[strokeCap()](http://processing.org/reference/strokeCap_.html)  
[strokeJoin()](http://processing.org/reference/strokeJoin_.html)  
[strokeWeight()](http://processing.org/reference/strokeWeight_.html)

*Vertex*

[beginShape()](http://processing.org/reference/beginShape_.html)  
[bezierVertex()](http://processing.org/reference/bezierVertex_.html)  
[curveVertex()](http://processing.org/reference/curveVertex_.html)  
[endShape()](http://processing.org/reference/endShape_.html)  
[texture()](http://processing.org/reference/texture_.html)  
[textureMode()](http://processing.org/reference/textureMode_.html)  
[vertex()](http://processing.org/reference/vertex_.html)

*Loading & Displaying*

[loadShape()](http://processing.org/reference/loadShape_.html)  
[shape()](http://processing.org/reference/shape_.html)  
[shapeMode()](http://processing.org/reference/shapeMode_.html)

**Input ---------------------**

*Mouse*

[mouseButton](http://processing.org/reference/mouseButton.html)  
[mouseClicked()](http://processing.org/reference/mouseClicked_.html)  
[mouseDragged()](http://processing.org/reference/mouseDragged_.html)  
[mouseMoved()](http://processing.org/reference/mouseMoved_.html)  
[mousePressed()](http://processing.org/reference/mousePressed_.html)  
[mousePressed](http://processing.org/reference/mousePressed.html)  
[mouseReleased()](http://processing.org/reference/mouseReleased_.html)  
[mouseX](http://processing.org/reference/mouseX.html)  
[mouseY](http://processing.org/reference/mouseY.html)  
[pmouseX](http://processing.org/reference/pmouseX.html)  
[pmouseY](http://processing.org/reference/pmouseY.html)

*Keyboard*

[key](http://processing.org/reference/key.html)  
[keyCode](http://processing.org/reference/keyCode.html)  
[keyPressed()](http://processing.org/reference/keyPressed_.html)  
[keyPressed](http://processing.org/reference/keyPressed.html)  
[keyReleased()](http://processing.org/reference/keyReleased_.html)  
[keyTyped()](http://processing.org/reference/keyTyped_.html)

*Files*

[createInput()](http://processing.org/reference/createInput_.html)  
[loadBytes()](http://processing.org/reference/loadBytes_.html)  
[loadStrings()](http://processing.org/reference/loadStrings_.html)  
[open()](http://processing.org/reference/open_.html)  
[selectFolder()](http://processing.org/reference/selectFolder_.html)  
[selectInput()](http://processing.org/reference/selectInput_.html)

*Web*

[link()](http://processing.org/reference/link_.html)  
[param()](http://processing.org/reference/param_.html)  
[status()](http://processing.org/reference/status_.html)

*Time & Date*

[day()](http://processing.org/reference/day_.html)  
[hour()](http://processing.org/reference/hour_.html)  
[millis()](http://processing.org/reference/millis_.html)  
[minute()](http://processing.org/reference/minute_.html)  
[month()](http://processing.org/reference/month_.html)  
[second()](http://processing.org/reference/second_.html)  
[year()](http://processing.org/reference/year_.html)

**Output -------------------------**

*Text Area*

[print()](http://processing.org/reference/print_.html)  
[println()](http://processing.org/reference/println_.html)

*Image*

[save()](http://processing.org/reference/save_.html)  
[saveFrame()](http://processing.org/reference/saveFrame_.html)

*Files*

[PrintWriter](http://processing.org/reference/PrintWriter.html)  
[beginRaw()](http://processing.org/reference/beginRaw_.html)  
[beginRecord()](http://processing.org/reference/beginRecord_.html)  
[createOutput()](http://processing.org/reference/createOutput_.html)  
[createReader()](http://processing.org/reference/createReader_.html)  
[createWriter()](http://processing.org/reference/createWriter_.html)  
[endRaw()](http://processing.org/reference/endRaw_.html)  
[endRecord()](http://processing.org/reference/endRecord_.html)  
[saveBytes()](http://processing.org/reference/saveBytes_.html)  
[saveStream()](http://processing.org/reference/saveStream_.html)  
[saveStrings()](http://processing.org/reference/saveStrings_.html)  
[selectOutput()](http://processing.org/reference/selectOutput_.html)

**Transform** -------------------  
  
[applyMatrix()](http://processing.org/reference/applyMatrix_.html)  
[popMatrix()](http://processing.org/reference/popMatrix_.html)  
[printMatrix()](http://processing.org/reference/printMatrix_.html)  
[pushMatrix()](http://processing.org/reference/pushMatrix_.html)  
[resetMatrix()](http://processing.org/reference/resetMatrix_.html)  
[rotate()](http://processing.org/reference/rotate_.html)  
[rotateX()](http://processing.org/reference/rotateX_.html)  
[rotateY()](http://processing.org/reference/rotateY_.html)  
[rotateZ()](http://processing.org/reference/rotateZ_.html)  
[scale()](http://processing.org/reference/scale_.html)  
[translate()](http://processing.org/reference/translate_.html)

**Lights, Camera** ---------------

*Lights*

[ambientLight()](http://processing.org/reference/ambientLight_.html)  
[directionalLight()](http://processing.org/reference/directionalLight_.html)  
[lightFalloff()](http://processing.org/reference/lightFalloff_.html)  
[lightSpecular()](http://processing.org/reference/lightSpecular_.html)  
[lights()](http://processing.org/reference/lights_.html)  
[noLights()](http://processing.org/reference/noLights_.html)  
[normal()](http://processing.org/reference/normal_.html)  
[pointLight()](http://processing.org/reference/pointLight_.html)  
[spotLight()](http://processing.org/reference/spotLight_.html)

*Camera*

[beginCamera()](http://processing.org/reference/beginCamera_.html)  
[camera()](http://processing.org/reference/camera_.html)  
[endCamera()](http://processing.org/reference/endCamera_.html)  
[frustum()](http://processing.org/reference/frustum_.html)  
[ortho()](http://processing.org/reference/ortho_.html)  
[perspective()](http://processing.org/reference/perspective_.html)  
[printCamera()](http://processing.org/reference/printCamera_.html)  
[printProjection()](http://processing.org/reference/printProjection_.html)

*Coordinates*

[modelX()](http://processing.org/reference/modelX_.html)  
[modelY()](http://processing.org/reference/modelY_.html)  
[modelZ()](http://processing.org/reference/modelZ_.html)  
[screenX()](http://processing.org/reference/screenX_.html)  
[screenY()](http://processing.org/reference/screenY_.html)  
[screenZ()](http://processing.org/reference/screenZ_.html)

*Material Properties*

[ambient()](http://processing.org/reference/ambient_.html)  
[emissive()](http://processing.org/reference/emissive_.html)  
[shininess()](http://processing.org/reference/shininess_.html)  
[specular()](http://processing.org/reference/specular_.html)

**Structure** -------------------------  
  
[[] (array access)](http://processing.org/reference/arrayaccess.html)  
[= (assign)](http://processing.org/reference/assign.html)  
[class](http://processing.org/reference/class.html)  
[, (comma)](http://processing.org/reference/comma.html)  
[// (comment)](http://processing.org/reference/comment.html)  
[{} (curly braces)](http://processing.org/reference/curlybraces.html)  
[delay()](http://processing.org/reference/delay_.html)  
[/\*\* \*/ (doc comment)](http://processing.org/reference/doccomment.html)  
[. (dot)](http://processing.org/reference/dot.html)  
[draw()](http://processing.org/reference/draw_.html)  
[exit()](http://processing.org/reference/exit_.html)  
[extends](http://processing.org/reference/extends.html)  
[false](http://processing.org/reference/false.html)  
[final](http://processing.org/reference/final.html)  
[implements](http://processing.org/reference/implements.html)  
[import](http://processing.org/reference/import.html)  
[loop()](http://processing.org/reference/loop_.html)  
[/\* \*/ (multiline comment)](http://processing.org/reference/multilinecomment.html)  
[new](http://processing.org/reference/new.html)  
[noLoop()](http://processing.org/reference/noLoop_.html)  
[null](http://processing.org/reference/null.html)  
[() (parentheses)](http://processing.org/reference/parentheses.html)  
[popStyle()](http://processing.org/reference/popStyle_.html)  
[private](http://processing.org/reference/private.html)  
[public](http://processing.org/reference/public.html)  
[pushStyle()](http://processing.org/reference/pushStyle_.html)  
[redraw()](http://processing.org/reference/redraw_.html)  
[return](http://processing.org/reference/return.html)  
[; (semicolon)](http://processing.org/reference/semicolon.html)  
[setup()](http://processing.org/reference/setup_.html)  
[size()](http://processing.org/reference/size_.html)  
[static](http://processing.org/reference/static.html)  
[super](http://processing.org/reference/super.html)  
[this](http://processing.org/reference/this.html)  
[true](http://processing.org/reference/true.html)  
[void](http://processing.org/reference/void.html)

**Environment** ---------------------  
  
[cursor()](http://processing.org/reference/cursor_.html)  
[focused](http://processing.org/reference/focused.html)  
[frameCount](http://processing.org/reference/frameCount.html)  
[frameRate()](http://processing.org/reference/frameRate_.html)  
[frameRate](http://processing.org/reference/frameRate.html)  
[height](http://processing.org/reference/height.html)  
[noCursor()](http://processing.org/reference/noCursor_.html)  
[online](http://processing.org/reference/online.html)  
[screen](http://processing.org/reference/screen.html)  
[width](http://processing.org/reference/width.html)

**Data** ------------------------------

*Primitive*

[boolean](http://processing.org/reference/boolean.html)  
[byte](http://processing.org/reference/byte.html)  
[char](http://processing.org/reference/char.html)  
[color](http://processing.org/reference/color_datatype.html)  
[double](http://processing.org/reference/double.html)  
[float](http://processing.org/reference/float.html)  
[int](http://processing.org/reference/int.html)  
[long](http://processing.org/reference/long.html)

*Composite*

[Array](http://processing.org/reference/Array.html)  
[ArrayList](http://processing.org/reference/ArrayList.html)  
[HashMap](http://processing.org/reference/HashMap.html)  
[Object](http://processing.org/reference/Object.html)  
[String](http://processing.org/reference/String.html)  
[XMLElement](http://processing.org/reference/XMLElement.html)

*Conversion*

[binary()](http://processing.org/reference/binary_.html)  
[boolean()](http://processing.org/reference/boolean_.html)  
[byte()](http://processing.org/reference/byte_.html)  
[char()](http://processing.org/reference/char_.html)  
[float()](http://processing.org/reference/float_.html)  
[hex()](http://processing.org/reference/hex_.html)  
[int()](http://processing.org/reference/int_.html)  
[str()](http://processing.org/reference/str_.html)  
[unbinary()](http://processing.org/reference/unbinary_.html)  
[unhex()](http://processing.org/reference/unhex_.html)

*String Functions*

[join()](http://processing.org/reference/join_.html)  
[match()](http://processing.org/reference/match_.html)  
[matchAll()](http://processing.org/reference/matchAll_.html)  
[nf()](http://processing.org/reference/nf_.html)  
[nfc()](http://processing.org/reference/nfc_.html)  
[nfp()](http://processing.org/reference/nfp_.html)  
[nfs()](http://processing.org/reference/nfs_.html)  
[split()](http://processing.org/reference/split_.html)  
[splitTokens()](http://processing.org/reference/splitTokens_.html)  
[trim()](http://processing.org/reference/trim_.html)

*Array Functions*

[append()](http://processing.org/reference/append_.html)  
[arrayCopy()](http://processing.org/reference/arrayCopy_.html)  
[concat()](http://processing.org/reference/concat_.html)  
[expand()](http://processing.org/reference/expand_.html)  
[reverse()](http://processing.org/reference/reverse_.html)  
[shorten()](http://processing.org/reference/shorten_.html)  
[sort()](http://processing.org/reference/sort_.html)  
[splice()](http://processing.org/reference/splice_.html)  
[subset()](http://processing.org/reference/subset_.html)

**Control** ---------------------------------

*Relational Operators*

[== (equality)](http://processing.org/reference/equality.html)  
[> (greater than)](http://processing.org/reference/greaterthan.html)  
[>= (greater than or equal to)](http://processing.org/reference/greaterthanorequalto.html)  
[!= (inequality)](http://processing.org/reference/inequality.html)  
[< (less than)](http://processing.org/reference/lessthan.html)  
[<= (less than or equal to)](http://processing.org/reference/lessthanorequalto.html)

*Iteration*

[for](http://processing.org/reference/for.html)  
[while](http://processing.org/reference/while.html)

*Conditionals*

[break](http://processing.org/reference/break.html)  
[case](http://processing.org/reference/case.html)  
[?: (conditional)](http://processing.org/reference/conditional.html)  
[continue](http://processing.org/reference/continue.html)  
[default](http://processing.org/reference/default.html)  
[else](http://processing.org/reference/else.html)  
[if](http://processing.org/reference/if.html)  
[switch()](http://processing.org/reference/switch_.html)

*Logical Operators*

[&& (logical AND)](http://processing.org/reference/logicalAND.html)  
[! (logical NOT)](http://processing.org/reference/logicalNOT.html)  
[|| (logical OR)](http://processing.org/reference/logicalOR.html)

**Color** -----------------------------

*Setting*

[background()](http://processing.org/reference/background_.html)  
[colorMode()](http://processing.org/reference/colorMode_.html)  
[fill()](http://processing.org/reference/fill_.html)  
[noFill()](http://processing.org/reference/noFill_.html)  
[noStroke()](http://processing.org/reference/noStroke_.html)  
[stroke()](http://processing.org/reference/stroke_.html)

*Creating & Reading*

[alpha()](http://processing.org/reference/alpha_.html)  
[blendColor()](http://processing.org/reference/blendColor_.html)  
[blue()](http://processing.org/reference/blue_.html)  
[brightness()](http://processing.org/reference/brightness_.html)  
[color()](http://processing.org/reference/color_.html)  
[green()](http://processing.org/reference/green_.html)  
[hue()](http://processing.org/reference/hue_.html)  
[lerpColor()](http://processing.org/reference/lerpColor_.html)  
[red()](http://processing.org/reference/red_.html)  
[saturation()](http://processing.org/reference/saturation_.html)

**Image** --------------------------------  
  
[PImage](http://processing.org/reference/PImage.html)  
[createImage()](http://processing.org/reference/createImage_.html)

*Loading & Displaying*

[image()](http://processing.org/reference/image_.html)  
[imageMode()](http://processing.org/reference/imageMode_.html)  
[loadImage()](http://processing.org/reference/loadImage_.html)  
[noTint()](http://processing.org/reference/noTint_.html)  
[requestImage()](http://processing.org/reference/requestImage_.html)  
[tint()](http://processing.org/reference/tint_.html)

*Pixels*

[blend()](http://processing.org/reference/blend_.html)  
[copy()](http://processing.org/reference/copy_.html)  
[filter()](http://processing.org/reference/filter_.html)  
[get()](http://processing.org/reference/get_.html)  
[loadPixels()](http://processing.org/reference/loadPixels_.html)  
[pixels[]](http://processing.org/reference/pixels.html)  
[set()](http://processing.org/reference/set_.html)  
[updatePixels()](http://processing.org/reference/updatePixels_.html)

**Rendering** ---------------------------------  
  
[PGraphics](http://processing.org/reference/PGraphics.html)  
[createGraphics()](http://processing.org/reference/createGraphics_.html)  
[hint()](http://processing.org/reference/hint_.html)

**Typography** --------------------------------  
  
[PFont](http://processing.org/reference/PFont.html)

*Loading & Displaying*

[createFont()](http://processing.org/reference/createFont_.html)  
[loadFont()](http://processing.org/reference/loadFont_.html)  
[text()](http://processing.org/reference/text_.html)  
[textFont()](http://processing.org/reference/textFont_.html)

*Attributes*

[textAlign()](http://processing.org/reference/textAlign_.html)  
[textLeading()](http://processing.org/reference/textLeading_.html)  
[textMode()](http://processing.org/reference/textMode_.html)  
[textSize()](http://processing.org/reference/textSize_.html)  
[textWidth()](http://processing.org/reference/textWidth_.html)

*Metrics*

[textAscent()](http://processing.org/reference/textAscent_.html)  
[textDescent()](http://processing.org/reference/textDescent_.html)

**Math** --------------------------------------  
  
[PVector](http://processing.org/reference/PVector.html)

*Operators*

[+= (add assign)](http://processing.org/reference/addassign.html)  
[+ (addition)](http://processing.org/reference/addition.html)  
[-- (decrement)](http://processing.org/reference/decrement.html)  
[/ (divide)](http://processing.org/reference/divide.html)  
[/= (divide assign)](http://processing.org/reference/divideassign.html)  
[++ (increment)](http://processing.org/reference/increment.html)  
[- (minus)](http://processing.org/reference/minus.html)  
[% (modulo)](http://processing.org/reference/modulo.html)  
[\* (multiply)](http://processing.org/reference/multiply.html)  
[\*= (multiply assign)](http://processing.org/reference/multiplyassign.html)  
[-= (subtract assign)](http://processing.org/reference/subtractassign.html)

*Bitwise Operators*

[& (bitwise AND)](http://processing.org/reference/bitwiseAND.html)  
[| (bitwise OR)](http://processing.org/reference/bitwiseOR.html)  
[<< (left shift)](http://processing.org/reference/leftshift.html)  
[>> (right shift)](http://processing.org/reference/rightshift.html)

*Calculation*

[abs()](http://processing.org/reference/abs_.html)  
[ceil()](http://processing.org/reference/ceil_.html)  
[constrain()](http://processing.org/reference/constrain_.html)  
[dist()](http://processing.org/reference/dist_.html)  
[exp()](http://processing.org/reference/exp_.html)  
[floor()](http://processing.org/reference/floor_.html)  
[lerp()](http://processing.org/reference/lerp_.html)  
[log()](http://processing.org/reference/log_.html)  
[mag()](http://processing.org/reference/mag_.html)  
[map()](http://processing.org/reference/map_.html)  
[max()](http://processing.org/reference/max_.html)  
[min()](http://processing.org/reference/min_.html)  
[norm()](http://processing.org/reference/norm_.html)  
[pow()](http://processing.org/reference/pow_.html)  
[round()](http://processing.org/reference/round_.html)  
[sq()](http://processing.org/reference/sq_.html)  
[sqrt()](http://processing.org/reference/sqrt_.html)

*Trigonometry*

[acos()](http://processing.org/reference/acos_.html)  
[asin()](http://processing.org/reference/asin_.html)  
[atan()](http://processing.org/reference/atan_.html)  
[atan2()](http://processing.org/reference/atan2_.html)  
[cos()](http://processing.org/reference/cos_.html)  
[degrees()](http://processing.org/reference/degrees_.html)  
[radians()](http://processing.org/reference/radians_.html)  
[sin()](http://processing.org/reference/sin_.html)  
[tan()](http://processing.org/reference/tan_.html)

*Random*

[noise()](http://processing.org/reference/noise_.html)  
[noiseDetail()](http://processing.org/reference/noiseDetail_.html)  
[noiseSeed()](http://processing.org/reference/noiseSeed_.html)  
[random()](http://processing.org/reference/random_.html)  
[randomSeed()](http://processing.org/reference/randomSeed_.html)

**Constants** ---------------------------  
  
[HALF\_PI (1.57079...)](http://processing.org/reference/HALF_PI.html)  
[PI (3.14159...)](http://processing.org/reference/PI.html)  
[QUARTER\_PI (0.78539...)](http://processing.org/reference/QUARTER_PI.html)  
[TWO\_PI (6.28318...)](http://processing.org/reference/TWO_PI.html)

1. 참고 서적:

   “Processing: A Programming Handbook for Visual Designers and Artists” [Casey Reas](http://www.amazon.co.jp/exec/obidos/search-handle-url?%5Fencoding=UTF8&search-type=ss&index=books-us&field-author=Casey%20Reas) , [Ben Fry](http://www.amazon.co.jp/exec/obidos/search-handle-url?%5Fencoding=UTF8&search-type=ss&index=books-us&field-author=Ben%20Fry)

   # “Processing: Creative Coding and Computational Art” [Ira Greenberg](http://www.amazon.co.jp/exec/obidos/search-handle-url?%5Fencoding=UTF8&search-type=ss&index=books-us&field-author=Ira%20Greenberg)

   # “Learning Processing: A Beginner's Guide to Programming Images, Animation, and Interaction” [Daniel Shiffman](http://www.amazon.co.jp/exec/obidos/search-handle-url?%5Fencoding=UTF8&search-type=ss&index=books-us&field-author=Daniel%20Shiffman)

   [↑](#footnote-ref-2)