ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE
TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Trabajo Fin de Grado

Lenguaje visual para la especificación de sistemas Teleo-Reactivos utilizando herramientas tipo Scratch

AUTOR: Pablo Rubio Ibáñez

DIRECTOR: Dra. Dña. Mª Francisca Rosique Contreras

Septiembre 2015

Autor	Pablo Rubio Ibáñez
Email	paruiba@gmail.com
Directores	Dr. D. Juan Ángel Pastor Blanco Dra. Dña. Mª Francisca Rosique Contreras
Email de los directores	juanangel.pastor@upct.es paqui.rosique@upct.es
Título del TFG	*Lenguaje visual para la especificación de sistemas Teleo-Reactivos utilizando herramientas tipo Scratch*

Resumen

En este trabajo crearemos una herramienta que empleará un lenguaje visual mediante bloques para la programación de drones que permitirá definir de manera sencilla su correcto comportamiento así como la manera en que se comportarán cada uno de sus componentes e interactuarán entre sí.

Titulación	Grado en Ingeniería Telemática
Departamento	Tecnologías de la Información y las Comunicaciones (TIC)

Contenido

Índice de figuras

1. Introducción

Los agentes autónomos, tales como los drones con los que se trabajará en este proyecto, operan en entornos sujetos a todo tipo de cambios (por ejemplo, el clima u otros objetos móviles). Estos entornos pueden dar lugar a todo tipo de cambios inesperados los cuales pueden ser difíciles de evitar para el agente, siendo la inteligencia artificial desarrollada hasta el momento incapaz de solventar este problema de manera adecuada.

A raíz de este problema, el científico Nils J. Nilsson propuso un formalismo denominado formalismo Teleo-Reactivo [] (1992) el cuál presenta un funcionamiento diferente a los métodos convencionales y que permite al agente operar en entornos dinámicos y reaccionar de manera eficiente a los cambios que se puedan producir en dicho entorno mientras persigue un determinado objetivo.

Para este proyecto se utilizará una extensión de este lenguaje, llamada TeleoR, la cual se explicará con mayor detenimiento en el siguiente capítulo.

A pesar de las ventajas del uso del lenguaje TeleoR, este lenguaje presenta una sintaxis con cierta complejidad la cual requiere su conocimiento previo en gran medida. Esto se traduce en una mayor dificultad para un programador a la hora de realizar una especificación textual mediante TeleoR y por lo tanto en un aumento considerable en el tiempo necesario para realizar dicha especificación con respecto a otros lenguajes.

Debido a lo expuesto en los párrafos anteriores se puede apreciar la necesidad de construir una herramienta visual que haga más intuitiva la elaboración de las especificaciones TeleoR, reduzca el tiempo empleado en ellas y haga más atractiva la idea de utilización del formalismo TeleoR para los programadores.

1.1 Motivación y objetivos

En este proyecto se presenta un lenguaje visual sencillo e intuitivo junto con un entorno gráfico web que facilite la programación lógica de drones mediante el formalismo TR.

Para ello, el usuario dispondrá de los distintos componentes (sensores y actuadores) necesarios para realizar la especificación de un modelo de dron y que éstos puedan relacionarse para crear una lógica específica empleando el lenguaje visual desarrollado. Una vez se construya la lógica, el entorno gráfico web tendrá la capacidad de transformar dicha lógica al lenguaje TeleoR de manera transparente al programador.

Para realizar dicha tarea, se han planteado los siguientes objetivos principales:

1. **Estudio metodologías desarrollo de software:** Se estudiará brevemente sobre conceptos básicos de los distintos tipos de lenguajes de programación y entornos de desarrollo software existentes.

2. **Estudio del formalismo TR:** Se estudiará brevemente el funcionamiento del formalismo TR para conocer sus características principales, centrándose

concretamente en una extensión del mismo, llamada lenguaje **TeleoR**, ya que será el utilizado en este proyecto.

3. **Estudio de los lenguajes visuales y entornos gráficos existentes:** Se investigarán las diferentes alternativas de lenguajes visuales y entornos de programación de código abierto existentes para elegir aquellos más adecuados para utilizar como base en este proyecto.

4. **Creación de un lenguaje visual y entorno gráfico web específico para TeleoR:** Se crearán un lenguaje visual y un entorno gráfico web, a partir de otros ya existentes, modificándolos de manera que permitan generar código en lenguaje TeleoR.

2. Estado del arte

En este capítulo se explicarán brevemente una serie de conceptos teóricos necesarios para entender el trabajo realizado en este proyecto. Se empezará explicando en qué consiste el lenguaje de programación lógica de drones **TeleoR** con el objetivo de entender su funcionamiento para facilitar la posterior transformación del lenguaje visual desarrollado a TeleoR. También se explicará en qué consisten los **lenguajes de programación visual**, ya que serán el principal foco de estudio junto con aquellas aplicaciones que les dan soporte.

2.1 Lenguaje TeleoR

En los últimos años, Keith L. Clark y Peter J Robinson han desarrollado una extensión del lenguaje TR creado por Nilsson (ya comentado en el capítulo 1), denominado **TeleoR** [1], y un lenguaje de programación lógica denominado **Qulog** para implementar programas Teleo-Reactivos.

Podemos definir el lenguaje TeleoR como un lenguaje para programar agentes robóticos que se comunican entre sí y pueden colaborar en la misma tarea. Algunas de las nuevas características que incorpora el lenguaje TeleoR son la posibilidad de controlar múltiples componentes robóticos con un solo agente mediante el intercambio de mensajes entre agentes que colaboran para alcanzar un objetivo, la implementación de agentes mediante múltiples hilos y un amplio conjunto de extensiones para las reglas.

Las especificaciones TeleoR, al igual que en el lenguaje TR, consisten en una serie de reglas agrupadas en procesos de la forma:

$$K_1 \rightarrow a_1$$

$$K_2 \rightarrow a_2$$

$$\dots$$

$$K_m \rightarrow a_m$$

Las K_i son condiciones que serán evaluadas en función de determinados valores de los sensores, y las a_i son una o más acciones de determinados componentes del agente para ser ejecutadas en paralelo.

Nilsson también definió la propiedad de regresión entre dos reglas, la cual es satisfecha cuando una regla K_i hace que la condición de una regla superior K_j $(j < i)$ sea inalcanzable tras ejecutar la acción a_i. Si esto es cierto, se dice que la condición K_i es una regresión de la condición K_j $(j < i)$ mediante la acción a_i. Un programa TR satisface la propiedad de

[1] Especificación de misiones para drones utilizando el enfoque Teleo-Reactivo – Francisco Miguel Moreno Olivo
Teleo-Reactive Programs for Agent Control – Nils J. Nilsson [http://arxiv.org/pdf/cs/9401101.pdf]

regresión si cada condición K_i es una regresión de cualquier regla superior K_j mediante a_i $(1 < i \leq m,\ 1 \leq j < i)$. Esta propiedad es necesaria para asegurar que las acciones del agente lo guiarán hacia el objetivo.

Las acciones pueden ser primitivas, como girar a la derecha, subir o esquivar un obstáculo; o llamadas a otros programas TR con el objetivo de alcanzar un objetivo secundario (subgoal), las cuales pueden ser llamadas recursivas.

A su vez, las acciones primitivas pueden se pueden catalogar en dos tipos diferentes: **discretas** o **durativas**. A continuación se explican brevemente:

- **Discretas**: Son ejecutadas una vez, como un beep de sonido, sin embargo, una vez que son ejecutadas, no pueden ser detenidas.
- **Durativas:** Son ejecutadas indefinidamente hasta que ocurre hasta que se dispare otra regla. Por ejemplo, la acción "mover hacia delante" será ejecutada cuando una regla se dispare y permanecerá activa hasta que sea detenida o modificada por la ejecución de otra regla.

Por lo general, un proceso TR se organiza de manera que cada condición dispara una acción que a su vez dispara una condición de mayor prioridad.

Tras esta breve introducción al formalismo TR, es preciso estudiar el método por el cuál simplificaremos la programación en TeleoR. Dicho método será mediante el empleo de la programación visual, la cual se estudiará en el siguiente apartado.

2.2 Lenguajes de programación visual

Existe la creencia de que la programación visual es aquella interfaz gráfica que permite visualizar lo que se está desarrollando por medio de un lenguaje de programación textual, sin embargo este concepto es totalmente erróneo, y dada la naturaleza de este proyecto, es necesario tener muy claro lo que es la programación visual antes de proseguir.

La **programación visual** [2] se refiere al desarrollo de software que emplea y manipula dinámicamente elementos visuales (gráficos, iconos, etc.) con el objetivo de definir programas. Pueden ser utilizados para formar la sintaxis de nuevos lenguajes de programación visuales.

Un **lenguaje de programación visual** puede ser definido de diferentes maneras:

- Un lenguaje de programación que elimina el código textual mediante una serie de elementos visuales (como gráficos, dibujos, iconos, animaciones, parcial o completamente).
- Un lenguaje visual que manipula información visual, permite interacción visual o permite programar con expresiones visuales.
- Cualquier lenguaje de programación que permite al usuario escribir usando dos o más dimensiones.

[2] El paradigma de la programación visual – Eugenio Jacobo Hernández Valdelamar & Humberto Manuel Uribe León [http://es.scribd.com/doc/2469975/El-paradigma-de-la-programacion-visual#scribd]

- Un lenguaje visual es un conjunto de arreglos espaciales de símbolos de texto y gráficos con una interpretación semántica que es usada para comunicar acciones en el mundo.

La representación visual puede servir como entrada, conexiones y/o salida de información [3].

Por ejemplo, un bloque que represente una operación booleana que realice una comparación de dos números enteros ***A*** y ***B*** y devuelva ***true*** si son iguales consistiría en dos entradas (A y B) y una salida de información.

En los últimos años, han ganado importancia entre la comunidad de desarrolladores hardware y software debido a la creación de lenguajes visuales que facilitan en gran medida la labor de desarrollo de los programadores profesionales, reduciendo considerablemente el tiempo de desarrollo y permitiendo a los usuarios menos experimentados un aprendizaje con mayor fluidez.

Existen gran cantidad de lenguajes de programación, por lo que resulta apropiado agruparlos para poder distinguir el tipo más adecuado a utilizar en función de las necesidades del programador. En el siguiente apartado se explicarán algunas de estas posibles clasificaciones junto con algunos ejemplos de lenguajes visuales conocidos en la actualidad.

2.2.1 Clasificación de los lenguajes visuales

En este apartado se realizará la clasificación de los lenguajes de programación visuales en función de dos tipos de clasificación diferentes: según su **sintaxis concreta** y según su **sintaxis abstracta**.

Por su **sintaxis concreta**, podemos distinguir dos tipos:

- **Híbridos de texto y elementos visuales**: Aquellos que emplean en su mayoría la manipulación de elementos visuales para la programación pero que aún emplean texto para realizar determinadas tareas.
- **Puramente visuales**: Sólo emplean elementos visuales para programar. Ejemplo: Google Blockly.

Por su **sintaxis abstracta**, podemos distinguir dos tipos:

- **Genéricos** [4]**:** Pueden ser usados para varios propósitos, acceso a bases de datos, comunicación entre ordenadores, comunicación entre dispositivos, captura de datos, cálculos matemáticos, diseño de imágenes o páginas, etc. Ejemplos: Pascal, C, C++ y Java.

[3] Visual Programming Language (VPL) – Techopedia
[https://www.techopedia.com/definition/22855/visual-programming-language-vpl]
[4] Lenguaje de programación de propósito general – Wikipedia
[https://es.wikipedia.org/wiki/Lenguaje_de_programación_de_propósito_general]

- **Específicos del dominio** [5]**:** Dedicado a resolver un problema en particular, representar un problema específico y proveer una técnica para solucionar una situación particular. Ejemplos: Logo para niños, VHDL y Mathematica.

Figura 1: Ejemplos de lenguajes de programación

Se ha explicado qué es exactamente un lenguaje visual y los diversos tipos que existen, sin embargo, ¿qué los hace atractivos frente a los lenguajes textuales? En el siguiente apartado se enumerarán una serie de ventajas del por qué los lenguajes visuales están ganando poco a poco importancia en el mundo de la programación y están convirtiéndose en preferencia entre un gran número de desarrolladores.

2.2.2 Ventajas de los lenguajes visuales frente a los textuales

Analizando las diferentes características de los lenguajes visuales, se pueden distinguir una serie de ventajas frente a los lenguajes textuales [6]:

- Los lenguajes visuales son muy amigables e intuitivos, los cuales permiten crear aplicaciones con una complejidad media de manera mucho más sencilla que utilizando lenguajes textuales.
- En los lenguajes de programación textuales, cualquier pequeño error en el código (la ausencia de una coma, por ejemplo), puede suponer que el programa realizado no funcione correctamente, sin embargo, no tendremos problemas de este tipo con los lenguajes de programación visuales.
- Existen algunos entornos gráficos de programación que utilizan lenguajes visuales creados especialmente para niños, con el objetivo de facilitar su aprendizaje o para usuarios sin conocimientos previos de lenguajes de programación.
- No requiere conocimiento previo de la sintaxis específica del lenguaje de programación a utilizar.

[5] Lenguaje específico del dominio – Wikipedia
[https://es.wikipedia.org/wiki/Lenguaje_específico_del_dominio]
[6] Lenguaje de Programación – Wikipedia
[https://es.wikipedia.org/wiki/Lenguaje_de_programaci%C3%B3n]
Visual Programming Language – Wikipedia
[https://en.wikipedia.org/wiki/Visual_programming_language]
Programación Visual – Wikipedia [https://es.wikipedia.org/wiki/Programación _visual]

- Multitud de lenguajes visuales usados para todo tipo de plataformas hardware (Arduino, FPGA, etc.) y software (Android, Windows, MacOS X).

A pesar de lo expuesto anteriormente, es cierto que en muchos casos los lenguajes de programación visuales no son suficientes para determinadas aplicaciones con un elevado nivel de complejidad, teniendo que optar por lenguajes de programación textuales (los cuáles suelen ser más potentes). Por lo tanto, se puede concluir que el lenguaje utilizado será elección del usuario en función de las necesidades de la aplicación.

Hasta aquí se ha explicado en qué consiste la programación visual y, más concretamente, los lenguajes de programación visual. Sin embargo, para hacer uso de estos lenguajes, serán necesarias aplicaciones que les den soporte. Estas aplicaciones son **editores** y **entornos gráficos de programación** y serán explicadas más a fondo en el siguiente apartado.

2.3 Entornos de programación visual

Un **entorno de programación visual** [7] es una aplicación que emplea un lenguaje de programación visual y una interfaz gráfica para permitir al usuario diseñar aplicaciones de manera sencilla y eficiente. Se hace uso de objetos, los cuales tienen la capacidad de interactuar con el entorno y con otros objetos para lograr una funcionalidad específica. Cada uno de dichos objetos tendrá asociadas una serie de propiedades u otros objetos.

La mayor parte de los entornos de programación visual cuentan con editores basados en estructuras [8], donde lo esencial es la estructura lógica del programa. Estos editores manipulan directamente la estructura lógica del código y no su representación como texto. La edición de la estructura se hace sobre elementos sintácticos tales como expresiones, sentencias o elementos gráficos en el caso de los lenguajes visuales.

Los editores basados en estructura son representados habitualmente por su árbol de sintaxis abstracta (AST). A continuación se muestra un ejemplo:

[7] Ambiente de desarrollo integrado – Wikipedia
[https://es.wikipedia.org/wiki/Ambiente_de_desarrollo_integrado]
[8] Entornos de programación - Dep. de Lenguajes y Sistemas Informáticos e Ingeniería de Software Universidad Politécnica de Madrid [http://lml.ls.fi.upm.es/ep/0910/e-o-estructura.pdf]

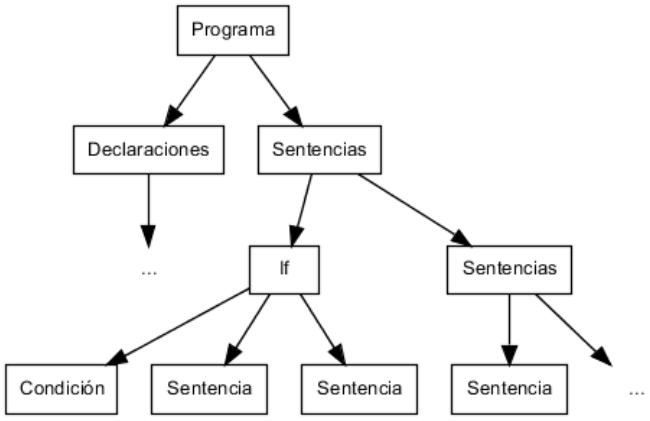


Figura 2: Ejemplo de editor basado en estructura (Árbol)

En general, los entornos orientados a estructura suelen ser específicos para un lenguaje de programación. Por lo tanto son un caso particular de entornos centrados en un lenguaje y comparten sus características generales, pero están concebidos de manera diferente:

- Normalmente soportan un único lenguaje de programación.
- Garantizan que el código es sintácticamente correcto.
- La compilación se realiza de manera incremental, a medida que se edita el código.
- Permite la ejecución inmediata del código editado, incluso aunque esté incompleto.
- Soportan el desarrollo de software a nivel individual, pero no el desarrollo en equipo a gran escala.

Por otro lado, los editores visuales que permiten trabajar directamente a nivel de lógica presentan unas ventajas considerables con respecto a los editores textuales:

- Evitan los errores sintácticos.
- Evitan tener que escribir los elementos fijos del código: palabras clave, puntuación, etc.
- Presentan el código con un estilo uniforme, bien encolumnado (formato automático).
- Guían al programador indicando qué elementos pueden insertarse en cada punto y recordándole la sintaxis de cada sentencia estructurada.
- Facilitan la reorganización del código al permitir la selección directa de secciones de código: funciones, bucles, etc.

2.3.1 Ventajas de los entornos de programación web frente a los de escritorio

Los entornos de programación web ofrecen una serie de ventajas interesantes que no podemos encontrar en los entornos de escritorio, tal y como se muestra en la siguiente tabla:

	Entorno web	Entorno de escritorio
Dependencias	X	✓
Requerimientos técnicos: instalación de software, etc.	X	✓
Multiplataforma	✓	X
Accesible desde cualquier ubicación	✓	X
Requiere conexión a Internet	✓	X
Almacenamiento de proyectos en la nube	✓	X
Compatibilidad con dispositivos móviles	✓	X

Tabla 1: Ventajas de los entornos web frente a los de escritorio

Si se observa detenidamente la tabla adjunta, se puede observar que a excepción del requisito de conexión a Internet (el cuál hoy en día rara vez suele ser un impedimento), los entornos de desarrollo web ofrecen mayor comodidad frente a los textuales.

Por nombrar algunos ejemplos, orientado hacia el aprendizaje de niños existe **Scratch**, para Android encontramos un entorno de programación web llamado **MIT App Inventor**, para Arduino encontramos multitud de entornos como **S4A**, **Ardublock** y **Modkit**. También se pueden encontrar otros entornos como **LabVIEW** [9]**,** el cual es recomendado para sistemas hardware y software de pruebas, control y diseño, simulado real y embebido.

Figura 3: Ejemplos de entornos de programación web

En el siguiente subapartado se analizarán una serie de entornos de programación web ya existentes para elegir el más adecuado.

2.3.2 Entornos de programación web

Como ya se comentó en el apartado anterior, se ha elegido partir de un entorno de programación web ya existente para este proyecto. En este apartado se analizarán algunos de ellos a fondo.

[9] LabVIEW – Wikipedia [https://es.wikipedia.org/wiki/LabVIEW]

2.3.2.1 Scratch

Figura 4: Logo de Scratch

El primero de esta lista de entornos web es Scratch, el cuál es utilizado usualmente para labores docentes. Este entorno web es utilizado como base por muchos entornos web actualmente, por lo que es conveniente que sea uno de los primeros en ser analizado.

Scratch [10] es un entorno de programación gráfico y gratuito que facilita crear historias interactivas, juegos y animaciones, además de compartir las creaciones elaboradas con otros en la Web. Scratch se lanzó oficialmente en Mayo de 2007 e inicialmente tuvo amplia acogida entre quienes venían trabajando con alguna de las versiones de Logo. Pero, en muy corto tiempo, su audiencia se amplió y consiguió cautivar a docentes de todo el planeta que comenzaron a usarlo en sus clases.

En la siguiente figura se muestra la interfaz gráfica de Scratch, destacando los componentes y zonas más importantes:

Figura 5: Entorno de programación web Scratch

[10] Guía de referencia de Scratch 2.0 – Eduteka
[http://www.eduteka.org/pdfdir/ScratchGuiaReferencia.pdf]

1. Escenario

El escenario de Scratch se encuentra situado en la parte izquierda del entorno web y es donde se pueden visualizar las historias, juegos y animaciones que se van creando. Este componente solo visualizará el resultado del programa desarrollado en el editor de bloques, por lo que no requiere de mayor explicación.

2. Paleta de bloques

En la parte central del entorno web se encuentra la **paleta de bloques**. Scratch se compone de las 10 categorías mostradas:

Figura 6: Categorías de bloques de Scratch

A continuación se explica brevemente la función de cada categoría [11]:

- **Movimiento**: Permite mover al objeto en x-y, girar tanto en sentido de reloj como sentido contrario, cambiar la dirección del objeto derecha-izquierda, arriba, abajo, posicionar al objeto en el lugar deseado, rebotar al objeto si se toca algún borde, etc.
- **Apariencia**: Permite cambiar de disfraz al objeto, decir algún comentario, aplicar algún efecto digital a la imagen de disfraz, cambiar tamaño, mostrar, esconder, enviar al frente, enviar hacia atrás N capas.
- **Sonido**: Permite tocar algún sonido desde archivo, una nota musical en específico, cambiar el volumen, cambiar el tempo de la nota musical.
- **Lápiz**: Permite dibujar en el escenario conforme se va moviendo el objeto, se puede cambiar el color, intensidad y tamaño del lápiz, así mismo se puede bajar, subir o sellar el lápiz.
- **Datos**: Permite crear variables, las cuales solo pueden almacenar un valor, y listas que son variables que almacenan un conjunto de variables.
- **Eventos**: Permite detectar eventos o acciones realizados por otros objetos y reaccionar a ellos. También permite detectar el teclado y reaccionar a alguna tecla presionada.
- **Control**: Permite crear ciclos iterativos y condicionales, dentro de los cuales se realizarán instrucciones de otros bloques.
- **Sensores**: Permite detectar si el objeto está tocando algún color, puede detectar alguna tecla presionada del teclado, leer las posiciones x-y del ratón, detectar la distancia al apuntador del ratón. En este grupo de instrucciones se obtienen los valores de los dispositivos externos o kits robóticos (acciones

[11] Paleta de Bloques – Programación Scratch [http://www.programacionscratch.com]

para robots). Podrás saber si el volumen esta fuerte. Se puede hacer una pregunta y leer el valor tecleado para almacenarlo en una variable.

- **Operadores**: Permite realizar operaciones lógicas como aritméticas básicas. Entre las operaciones lógicas encontramos or, and y not, y operaciones matemáticas como suma, resta, multiplicación, división, raíz cuadrada, operaciones logarítmicas y trigonométricas básicas, mayor, menor e igual.
- **Más Bloques**: Esta categoría no incluye bloques por defecto, sino que incluye opciones para crear bloques personalizados.

3. Área de programas

El área de programas situada en la parte derecha de Scratch es el corazón del editor de bloques de Scratch.

Este tipo de entornos de programación visuales proporcionan una novedosa manera de programar que hace sencillo el construir programas sin tener ninguna experiencia previa en la programación. Basta con interconectar una serie de bloques como si fuera un puzle para crear la aplicación, pudiéndose adquirir la capacidad de crear aplicaciones realmente complejas en muy poco tiempo. Además existe multitud de documentación y vídeos explicativos que ayudarán a familiarizarse rápidamente con el entorno.

En el caso de Scratch, se pueden relacionar determinadas acciones a realizar por los objetos del escenario. Para intentar ilustrar la explicación anterior, se muestra el siguiente ejemplo de un programa sencillo en Scratch:

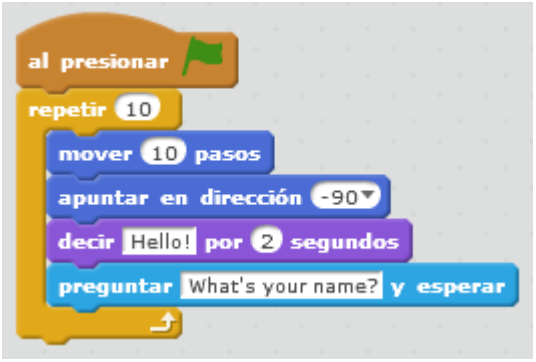


Figura 7: Ejemplo de programa en Scratch

El programa anterior hará que cada vez que se presione el símbolo de la bandera situado en el escenario, el personaje de Scratch se mueva diez pasos, se gire -90 grados, diga “Hello!”, pregunte al usuario “What´s your name?” y espere a su respuesta. Posteriormente se repetirá el proceso 10 veces.

2.3.2.2 S4A (Scratch for Arduino)

Figura 8: Logo de S4A

S4A [12] es una modificación de Scratch que permite programar la plataforma de hardware libre Arduino de una forma sencilla. Al igual que Scratch, la finalidad principal de S4A es atraer a gente al mundo de la programación, además de proporcionar una interfaz de alto nivel para programadores de Arduino.

Es necesario aclarar que S4A no es un entorno de programación web, ya que aunque está basado en Scratch, sólo está disponible en versión escritorio. Aun así, S4A presenta características interesantes que no poseía Scratch y que podrían ser de utilidad en este proyecto, por lo que a continuación se estudiará su funcionamiento y, sobretodo, en aquellas características importantes en las que difiere de Scratch.

En la siguiente captura se muestra el aspecto de la interfaz de S4A, delimitando las áreas de mayor interés:

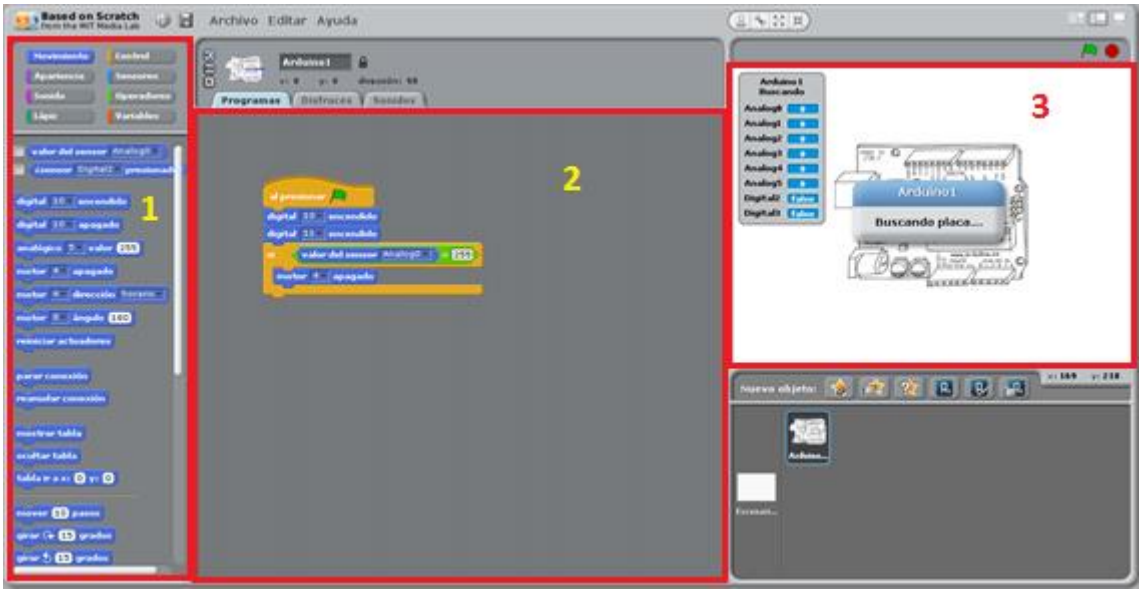


Figura 9: Entorno de programación de escritorio S4A

Se puede observar claramente que sigue teniendo la mayoría de herramientas de las que dispone Scratch, aunque en S4A el escenario está situado a la derecha (3), mientras que la paleta de categorías se encuentra a la izquierda (1) y el área de programas en el centro (2), tal y como se ilustra.

[12] Sitio web oficial S4A [http://s4a.cat/index_es.html]

Una de las nuevas características más importantes es que se han incorporado bloques para las funcionalidades básicas del microcontrolador, escrituras y lecturas digitales y analógicas, y otras funcionalidades de más alto nivel de Arduino. También se pueden encontrar bloques para tratar con motores estándar y servomotores de rotación continua. Estos bloques pueden ser encontrados en la categoría **Movimiento**:

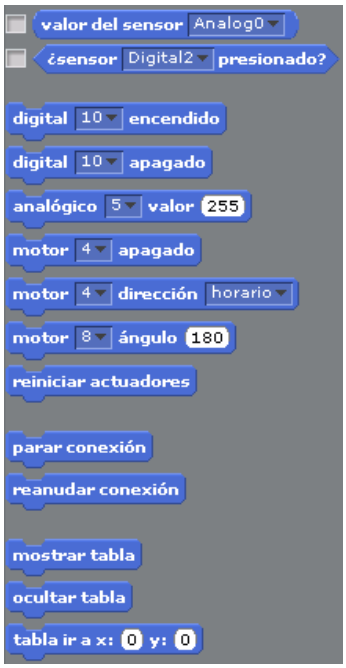


Figura 10: Nuevos bloques para Arduino en S4A

El entorno S4A cuenta también con una tabla de sensores (situada en el escenario) en la cual se puede observar el estado, tanto de las entradas digitales como de las analógicas. Esta tabla aparece en el momento en que se abre el programa y se conecta la tarjeta Arduino al puerto USB del ordenador:

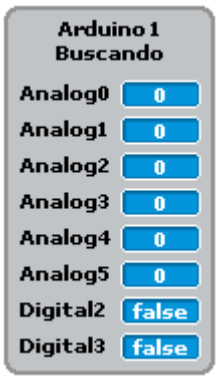


Figura 11: Estado de sensores en S4A

Este es un ejemplo simple mostrando como activar un LED mediante un pulsador:

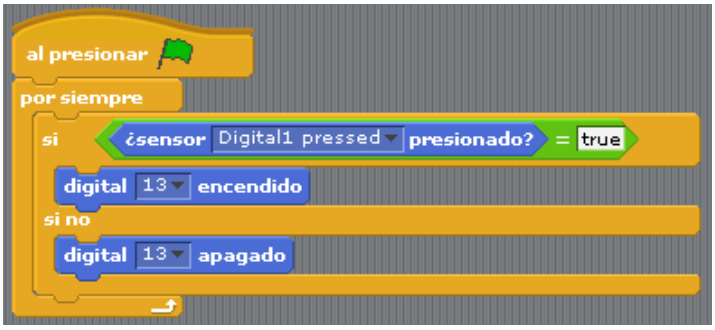
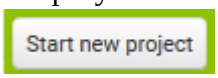


Figura 12: Ejemplo de activación de LED mediante pulsador en S4A

2.3.2.3 MIT App Inventor

Figura 13: Logo de MIT App Inventor

MIT App Inventor [13] es un entorno gráfico de programación que utiliza un lenguaje visual basado en bloques el cual permite a los usuarios, incluso a principiantes, empezar a programar y crear aplicaciones totalmente funcionales para dispositivos Android.

Al abrir App Inventor, la primera pantalla que se mostrará será la de **My projects**. En esta pantalla se mostrarán los proyectos creados hasta ahora. Para crear un proyecto, se debe presionar el botón 

. Tras ingresar un nombre para el nuevo proyecto, aparecerá la ventana del **modo diseño** del proyecto de App Inventor.

En MIT App Inventor, a diferencia de los dos entornos de programación visual vistos hasta el momento, dispondremos de dos modos: el **modo de diseño** y el **modo del editor de bloques**. Estos dos modos son el corazón de este entorno y se complementan entre ellos para proporcionar al usuario las herramientas necesarias para definir las aplicaciones. A continuación se explicará su funcionamiento detalladamente.

2.3.2.3.1 Modo de diseño

A la hora de programar aplicaciones Android en App Inventor, es necesario seleccionar los componentes necesarios, así como definir valores para sus propiedades para el

[13] App Inventor en Español – Primeros pasos [https://sites.google.com/site/appinventormegusta/primeros-pasos]
App Inventor – Wikipedia [https://es.wikipedia.org/wiki/App_Inventor]

correcto funcionamiento de la aplicación que se quiere desarrollar. El **modo de diseño** proporciona una interfaz con herramientas para realizar esta tarea de manera sencilla.

A continuación se muestra la interfaz del modo de diseño destacando las áreas más importantes:

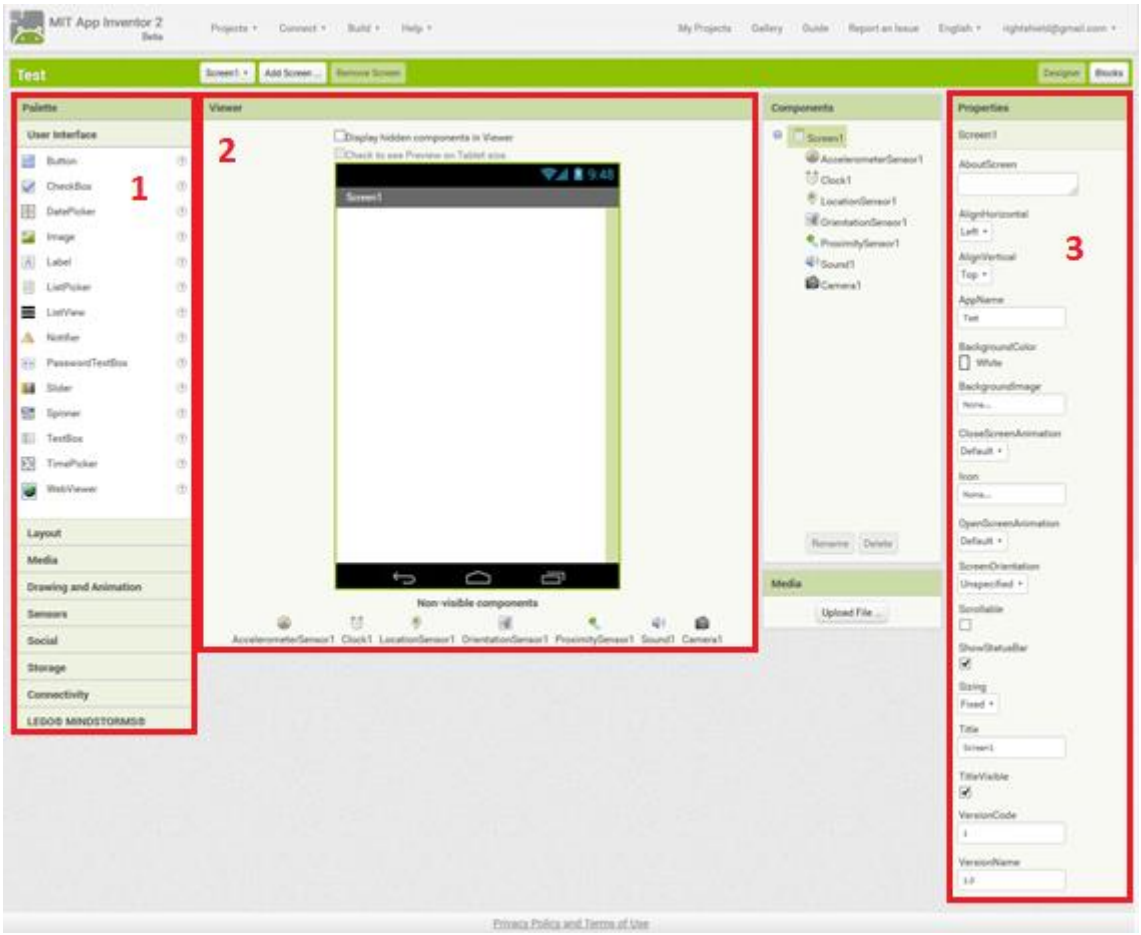


Figura 14: Modo de diseño del entorno de programación web MIT App Inventor

La captura anterior muestra MIT App Inventor en su **modo de diseño**, sin embargo existe otro modo mediante el cual se realizará la programación lógica mediante bloques, el cual es denominado **editor de bloques**. Este editor de bloques se mostrará y explicará posteriormente.

Primero se analizarán los distintos componentes de la parte de diseño [14]:

1. **Paleta de componentes**

   En esta parte del modo de diseño se podrán seleccionar los componentes deseados y añadirlos a la aplicación.

   Los componentes son los elementos básicos que utilizamos para hacer aplicaciones Android. Son como los ingredientes de una receta. Algunos componentes son muy simples, como una etiqueta (Label), que sólo muestra el texto en pantalla, o un botón

[14] App Inventor en Español – Antonio Ricoy Riego
[https://sites.google.com/site/appinventormegusta/conceptos]

(Button) que se pulsa para iniciar una acción. Otros componentes son más elaborados: un lienzo (Canvas) que puede almacenar imágenes fijas o animaciones, un sensor de movimiento (AccelerometorSensor) que funciona como un mando de Wii y detecta cuando se mueve o agita el teléfono.

En App Inventor existen multitud de componentes para diversas funcionalidades. Dichos componentes se encuentran englobados en 9 categorías siguientes:

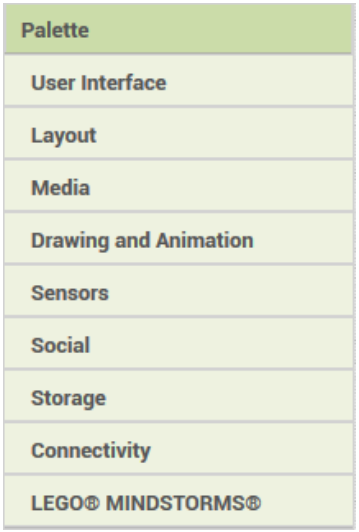


Figura 15: Paleta de categorías de MIT App Inventor

A continuación se explica la función de cada categoría:

- **User Interface**: Proporciona componentes para construir la interfaz de usuario de la aplicación, tales como botones, imágenes, cajas de texto, etc.
- **Layout**: Definen áreas donde van a insertarse otros elementos.
- **Media**: Permite insertar componentes multimedia, ya sean cámaras, reproductores o grabadores audio y vídeo, reconocimiento de voz, etc.
- **Drawing and Animation**: Permite insertar dibujos y animaciones mediante objetos que se mueven por la aplicación.
- **Sensors**: Permite añadir sensores a la aplicación, los cuales detectarán determinados valores como el movimiento, la aceleración, la ubicación, la orientación física del dispositivo, la proximidad de un objeto, etc.
- **Social**: Permite interactuar con diferentes componentes que permiten la comunicación o la compartición, como la lista de contactos del teléfono, el correo electrónico, twitter, etc.
- **Storage**: Permite almacenar datos, ya sea ficheros o determinados valores en una base de datos para evitar la pérdida de los mismos, etc.
- **Connectivity**: Mediante determinados componentes, permite realizar conectividad con otros dispositivos mediante Bluetooth, gestionar solicitudes HTTP, etc.
- **LEGO® MINDSTORMS®**: Proporciona componentes para gestionar los diferentes sensores disponibles en un robot LEGO MINDSTORMS NXT.

Al añadir los componentes a la aplicación, estos serán mostrados en el **Visor** del modo de diseño. Algunos de los componentes son visibles, mientras que otros no. Dichos componentes no visibles añadidos serán mostrados en la **barra de componentes no visibles** del **Visor**.

2. Visor

El visor de la pantalla, simula la apariencia visual que tendrá la aplicación en el dispositivo Android.

El **Visor** se compone de dos partes: la **Screen** y la **barra de componentes no visibles**.

Cada componente **visible** que sea añadido a través de la **paleta de componentes** aparecerá en la **Screen,** como por ejemplo botones y cajas de texto, mientras que los componentes no visibles se situarán debajo de la **Screen**, en la **barra de componentes no visibles**.

Por ejemplo, si se añaden una caja de texto y un botón, la **Screen** se verá de la siguiente manera:

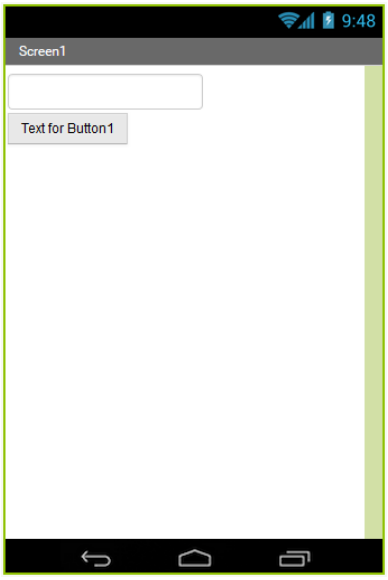


Figura 16: Ejemplo de componentes en Screen de MIT App Inventor

En cambio, si se añade un acelerómetro y un reloj, estos componentes serán visibles en la **barra de componentes no visibles**:

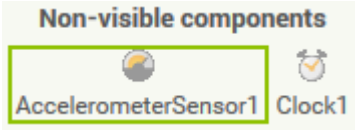


Figura 17: Barra de componentes añadidos no visibles de MIT App Inventor

3. Propiedades

Los componentes tienen propiedades que se pueden ajustar para cambiar la forma en que el componente aparece o actúa dentro de la aplicación. Para ver y cambiar las propiedades de un componente, primero debe seleccionar el componente deseado en la lista de componentes. Los valores de algunas propiedades de ciertos componentes no son modificables, aun así, si se podrán consultar.

Cada vez que en el **Visor** se seleccione un componente, en **Propiedades** aparecerán todos los detalles que se puedan cambiar de ese componente, aunque dichos valores podrán ser modificados posteriormente de manera dinámica en el **editor de bloques**.

Por ejemplo, si se selecciona el componente **AccelerometerSensor**, el menú de propiedades se verá de la siguiente manera:

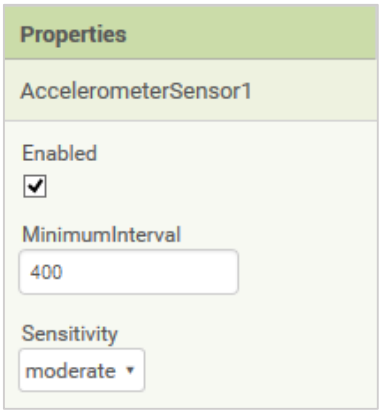


Figura 18: Menú de propiedades para AccelerometerSensor en MIT App Inventor

2.3.2.3.2 Modo del editor de bloques

Ahora se va a explicar detenidamente el funcionamiento del **editor de bloques**. A diferencia de Scratch y S4A, App Inventor hace uso del editor de bloques **Google Blockly** para construir la lógica de la aplicación. Para acceder al editor de bloques, sólo será necesario pulsar el botón **Blocks** situado encima del menú **Propiedades** de App Inventor.

Con el **editor de bloques** vamos a definir cómo se comportará la aplicación. Estableceremos lo que los componentes deben hacer y cuándo hacerlo. Por ejemplo, que debe ocurrir cuando el usuario pulsa un botón.

A continuación se muestra la interfaz del **editor de bloques**, destacando las áreas de mayor interés:

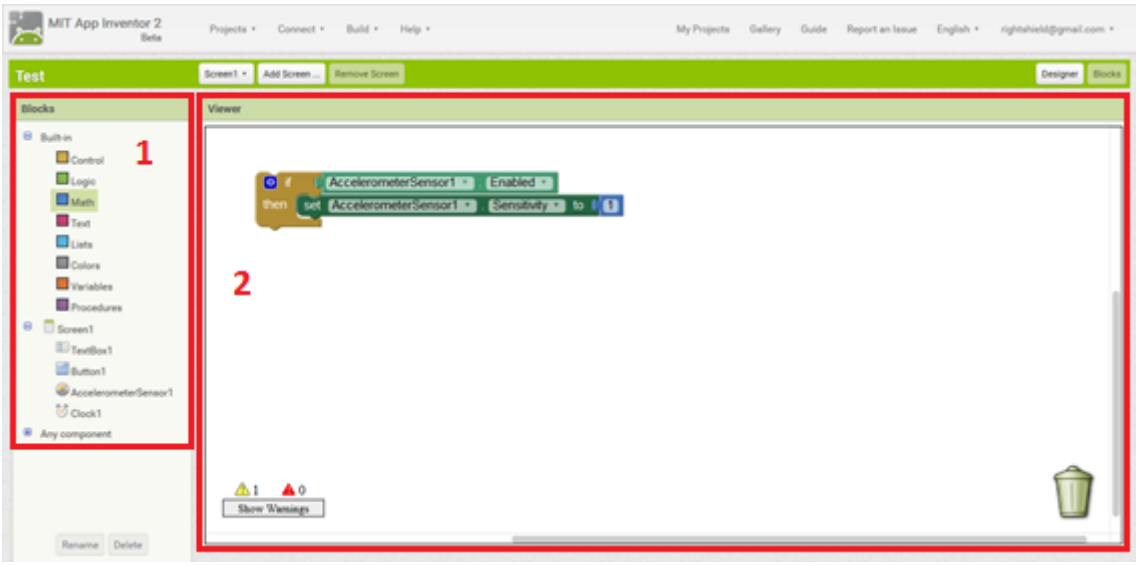


Figura 19: Modo del editor de bloques de MIT App Inventor

1. Paleta de bloques

En el editor de bloques de App Inventor podemos distinguir dos tipos de bloques diferentes:

- **Built-in**: Bloques originales de Google Blockly, que sirven para implementar un amplio número de bucles, comparaciones, operaciones, etc. Dichos bloques están agrupados en las siguientes categorías:

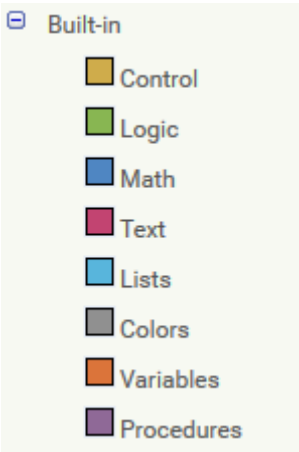


Figura 20: Bloques Built-in de MIT App Inventor

Cada categoría agrupa bloques con diferentes finalidades. A continuación se explica brevemente los tipos de bloques contenidos en cada una:

- **Control**: Contiene bloques que permitirán realizar determinadas acciones si se cumple una condición o realizar acciones de manera repetida.
- **Logic**: Permite realizar operaciones lógicas elementales como OR, AND y NOT.
- **Math**: Permite realizar operaciones aritméticas básicas y generar números aleatorios.

- **Text**: Proporciona bloques que permiten manipular cadena de texto.
- **Lists**: Permite crear y manipular listas, las cuales podrán almacenar varios valores.
- **Colors**: Permite modificar el color del elemento al que está asociado.
- **Variables**: Permite crear variables para almacenar un valor.
- **Procedures**: Proporciona bloques para crear procesos que contengan una lógica determinada y los bloques necesarios para ejecutar dichos procesos dados unos parámetros definidos por el usuario.

- **Bloques asociados a componentes**: Cada componente localizado en la parte de diseño tiene asociados una serie de bloques los cuales, si se ha añadido dicho bloque en el modo de diseño, aparecerán disponibles. Por ejemplo, al añadir los componentes **TextBox**, **Button**, **AccelerometerSensor** y **Clock** en el modo diseño, se podrá observar la siguiente paleta de categorías de este tipo de bloques:

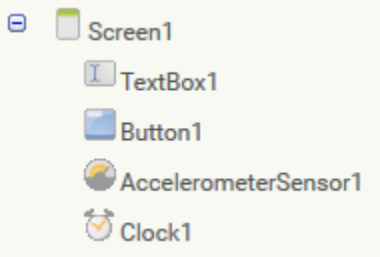


Figura 21: Ejemplo de categorías de bloques de componentes de MIT App Inventor

2. Área de trabajo

En el área de trabajo se podrán interconectar los distintos tipos de bloques para crear la lógica de la aplicación.

Se observa un editor de bloques con una gran variedad de bloques que permiten desarrollar aplicaciones de manera intuitiva gracias a su interfaz.

Por ejemplo, a continuación se muestra un bloque **if…then** sencillo el cuál modifica el valor de la sensibilidad del acelerómetro si éste está activado:

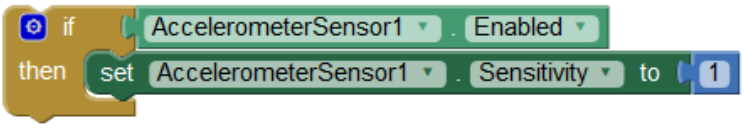


Figura 22: Ejemplo del funcionamiento del editor de MIT App Inventor

Una novedad interesante es que App Inventor avisará de cualquier problema o error presente en la lógica de bloques. Por ejemplo, en el ejemplo anterior se notificaba de un problema en los bloques implementados. Si se pulsa el botón ***Show warnings***

situado en la **esquina inferior izquierda** del área de trabajo, aparecerá un icono de warning en el bloque afectado:

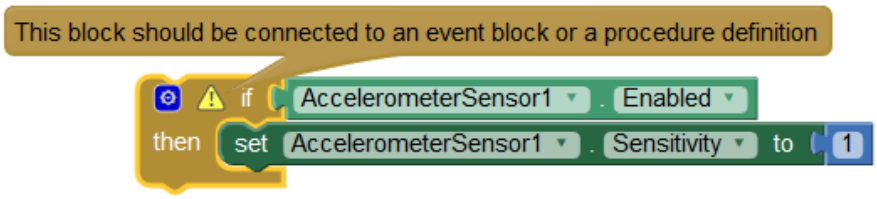


Figura 23: Ejemplo de notificación de avisos y errores en MIT App Inventor

3. Lenguaje visual para sistemas Teleo-Reactivos

Tal y como se comentó en el capítulo anterior, los lenguajes visuales facilitan y mejoran el proceso de desarrollo gracias a la utilización de representaciones visuales para programar la lógica de la aplicación sin requerir previo conocimiento de lenguajes de programación textuales y, por lo tanto, tal y como se fijó en los objetivos del proyecto, se ha desarrollado un lenguaje visual para facilitar la especificación de sistemas Teleo-Reactivos. Para ello, y siguiendo con uno de los principios básicos de la ingeniería del software, se optó por reutilizar los recursos ya existentes y desarrollar un lenguaje visual a partir de uno de los lenguajes estudiados en el capítulo anterior. Para ello en primer lugar se ha realizado un estudio detallado de las necesidades del nuevo lenguaje para sistemas TR [15].

3.1 Estudio de los requisitos del lenguaje TeleoR

Antes de proceder a la creación del lenguaje visual, es conveniente conocer los **conceptos**, los **tipos de datos primitivos** usados, la **estructura** de los programas TeleoR y los **conjuntos de extensiones** para las reglas TeleoR. Dichos puntos de interés se estudian y explican en los siguientes apartados.

3.1.1 Conceptos básicos

Con el objetivo de entender el funcionamiento del lenguaje TeleoR, se deben conocer una serie de conceptos fundamentales:

- **Percept**: Representan la información que el entorno posee, y son almacenados en la BeliefStore.
- **Beliefs**: Representan información obtenida del mismo agente o de otros agentes (pero no del entorno).
- **Predicados**: Datos más complejos deducidos a partir de los percepts.
- **BeliefStore**: Es la base de datos dinámica donde se almacena información. Contiene los percepts más básicos que vienen de los sensores y los predicados abstractos derivados de esos percepts así como las reglas para crear dichos predicados.

Será necesario conocer también los tipos primitivos de datos utilizados por TeleoR para implementarlos en el lenguaje visual a desarrollar. En el siguiente apartado se analizan cada uno de ellos.

[15] Especificación de misiones para drones utilizando el enfoque Teleo-Reactivo – Francisco Miguel Moreno Olivo

3.1.2 Tipos de datos primitivos

La siguiente tabla muestra los diferentes tipos de datos primitivos de QuLog con su identificador asociado:

Identificador	Tipo de dato
num	Número en coma flotante
int	Número entero
nat	Número entero sin signo
string	Cadena de texto
atom	Cadena *atómica* de texto

Tabla 2: Tipos de datos primitivos de QuLog

De los tipos de datos descritos anteriormente, el más complicado de entender es ***atom***. Un **átomo** puede ser visto como un identificador, como una cadena de texto *atómica* que no puede ser compuesta o dividida.

Una vez vistos los tipos de datos primitivos, es necesario conocer la estructura básica de un programa TeleoR. Esta estructura tendrá su equivalente en el lenguaje visual que se va a desarrollar.

3.1.3 Estructura y elementos que componen un programa TeleoR

Los programas TeleoR pueden ser divididos en diferentes partes o bloques de código. Algunas de estas partes estarán siempre implementadas mientras que otras son opcionales, a elección del programador dependiendo de la funcionalidad a implementar en el agente.

Están definidos en ficheros **QuLog**, y sabiendo que QuLog es un lenguaje declarativo, el orden en que las diferentes partes del código son escritas no importa. Sin embargo, es recomendable seguir siempre la misma estructura para facilitar la lectura y comprensión para otros programadores.

Un programa TeleoR puede estar segmentados en las siguientes partes:

1. **Inicialización**: En esta parte se inicializará la *BeliefStore* y se declarará todo lo que se vaya a usar:
   - **Definición de tipos**: Se tendrán que definir tipos predefinidos para extender los primitivos. Esta parte es opcional, ya que a veces es suficiente con los tipos primitivos.
   - **Declaraciones de Percepts/Beliefs**.
   - **Declaración e inicialización de variables globales**.
   - **Declaración de acciones**: Tanto las acciones discretas como las durativas deben ser declaradas.
   - **Declaración de recursos**: Diversos recursos disponibles para el agente.

- **Definición de relaciones**: Se declararán y definirán las relaciones para crear complejos predicados de los simples percepts. Esta parte es opcional, ya que no será necesario definir una relación si no va a usarse.

2. **Procesos TeleoR**: El proceso principal de TeleoR y otras subrutinas para los diferentes goals y sub-goals. Estos procesos son escritos en TeleoR.
3. **Procesos de acciones de QuLog**: Definición de los procesos de acciones de QuLog, usados para modificar la *BeliefStore* y enviar mensajes a otros agentes. Esta parte es opcional, y es posible integrar estos procesos en la parte de acciones de las reglas de TeleoR.

4. **Manejadores básicos**: Procesos usados para recibir mensajes en sus hilos correspondientes. Ambos opcionales.
   - **Manejador de percepts**: Este es el proceso que procesa los mensajes de percepts recibidos del entorno. Está implementado por defecto en QuLog.
   - **Manejador de mensajes**: Este manejador no está implementado por defecto porque solo será necesario cuando se esté trabajando con agentes que se comunican entre sí, porque los mensajes serán diferentes en cada instante, por lo que no puede existir una implementación genérica.

5. **Función de ejecución**: Una proceso de acción típica de QuLog el cual es llamado desde el intérprete y cuyo propósito es iniciar el agente y sus tareas principales. Este proceso es opcional, pero simplifica mucho el proceso de inicio.

3.1.4 Extensiones de reglas TeleoR

Una de las características más potentes de TeleoR es el conjunto de extensiones para las reglas TR. Estas extensiones son mecanismos que alteran la condición y/o la acción de una regla. Aparecen de la experiencia de programar agentes robóticos mediante TR y ayudan a mantener el código simple cuando se implementa un comportamiento complejo.

Para la parte de **acciones** de las reglas se encuentran las siguientes extensiones:

- **Secuencias de acciones**: Permiten ejecutar repetidamente una secuencia de acciones durativas de tiempo limitado o llamadas a procesos TeleoR como si fuera una sola acción durativa.
- **Wait/Repeat**: Mecanismo usado en las acciones discretas que pueden fallar en su ejecución. Cuando se usa la extensión **Wait/Repeat** en una acción, si la acción falla, se vuelve a ejecutar tras un periodo de tiempo específico.

e igual manera, existen una serie de extensiones para la parte de **condiciones** de las reglas. Son las siguientes:

- **While**: Son condiciones secundarias usadas para mantener una regla activa, incluso si la condición que ejecuta la regla se vuelve inalcanzable.
- **Until**: Es también una condición secundaria, sin embargo su propósito es evitar la ejecución de reglas con mayor prioridad hasta que esta condición es satisfecha.

3.2 Elección del lenguaje y entorno visual

Tras haber estudiado diferentes características importantes de TeleoR, se debe elegir el lenguaje y entorno visual más adecuado de entre los estudiados en el apartado anterior. Para ello se deben comparar los lenguajes visuales y entornos de programación visual con los conceptos analizados anteriormente y encontrar aquel que presente un mayor nivel de afinidad.

Tras realizar un estudio, se ha seleccionado el entorno de programación web **MIT App Inventor** junto con su lenguaje visual **Google Blockly**. A continuación se muestra el estudio realizado para ello.

3.2.1 Tipos de datos

En cuanto a los tipos de datos empleados por TeleoR, los tres entornos estudiados presentan lenguajes visuales que emplean los tipos de datos siguientes:

Entorno visual	Int	Float	Boolean	String	Atom
Scratch	✓	✓	✓	✓	X
S4A	✓	✓	✓	✓	X
MIT App Inventor	✓	✓	✓	✓	X

Tabla 3: Tipos de datos de Scratch, S4A y MIT App Inventor

Vemos que además de añadir un nuevo tipo de dato (*boolean*), todas las alternativas carecen del tipo primitivo *atom*, pero que son capaces de manejar el resto de tipos. Sin embargo, el lenguaje visual **Google Blockly** empleado por **MIT App Inventor** presenta un mayor número de bloques para manipular dichos tipos de datos primitivos. En la siguiente tabla se muestra qué categorías de MIT App Inventor permiten la creación y manipulación de cada tipo de dato primitivo:

Tipo de dato	Categoría
Int	Math
Float	Math
Boolean	Logic
String	Text

Tabla 4: Tipos de datos primitivos en MIT App Inventor

3.2.2 Componentes del programa

Al estudiar la estructura de un programa TeleoR, se han estudiado algunos de los componentes esenciales que componen un programa TeleoR, y uno de los entornos estudiados lleva incorporados algunos de ellos por defecto: **MIT App Inventor**.

En el apartado anterior se han estudiado los conceptos de **Percepts** y **Beliefs**. Como se vio al estudiar los entornos visuales, **MIT App Inventor** tiene un rasgo muy distintivo con respecto a Scratch y S4A: el **modo diseño**.

Este **modo de diseño** resulta muy interesante debido a que permite añadir componentes al proyecto, los que a su vez proporcionan propiedades y nuevos bloques. Ya que dependiendo del modelo de dron utilizado se disponen de unos sensores u otros, esta característica especial de MIT App Inventor resultaría muy adecuada. Además, como ya se vio, MIT App Inventor lleva incorporados ya una serie de **sensores** por lo que éstos serían reutilizables en el entorno visual desarrollado. Los bloques que están disponibles en estos componentes permitirían obtener en el editor el equivalente a los **Percepts** y **Beliefs**.

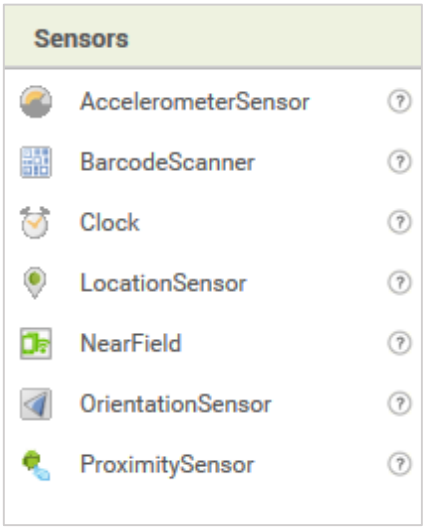


Figura 24: Sensores por defecto de MIT App Inventor

Por otro lado, otros de los componentes de los programas TeleoR son los **procesos TeleoR**. Según se vio al estudiar los distintos editores de bloques, el **modo del editor de bloques** de MIT App Inventor posee unos tipos de bloques los cuales ejecutan **procesos**, y están agrupados en la categoría **Procedures**. Estos bloques pueden utilizarse como equivalentes de los **procesos TeleoR**, por lo que podrán reutilizarse.

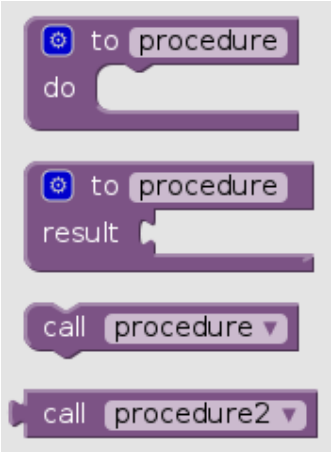


Figura 25: Bloques de procesos de MIT App Inventor

3.2.3 Extensiones de reglas

Además de las razones vistas en los apartados anteriores, hay que tener en cuenta las extensiones de las condiciones y las acciones que se vieron anteriormente.

Todos los lenguajes visuales estudiados tienen bloques de **condicionales** y **bucles** que pueden servir para esta tarea (if, while, etc.). Sin embargo, en el lenguaje visual **Google Blockly** utilizado en el editor de bloques **MIT App Inventor** existen un mayor número de bloques de este tipo y además algunos de ellos pueden ser modificados dinámicamente (como en el caso del bloque condicional **if**). Esta característica será estudiada más detalladamente en capítulos posteriores.

3.3 Estudio de elementos reutilizables y nuevos elementos a añadir

Una vez decidido el lenguaje visual y entorno web a utilizar, lo primero es realizar un estudio sobre aquellas características de MIT App Inventor que se deben conservar para su reutilización junto con aquellas nuevas características que deben ser implementadas. Dicho estudio se realizará para cada modo de MIT App Inventor.

3.3.1 Modo de diseño

Primero se estudiarán los elementos correspondientes al **modo de diseño**. Las características que se deberán mantener son las siguientes:

- Aspecto general de la interfaz (localización de sus elementos, etc).
- Categoría de componentes **Sensors**, conservando la mayoría de sus componentes.
- Componentes **Camera** y **Sound** pertenecientes a la categoría **Media**, aunque estos serán cambiados a otra categoría.

En cuanto a las modificaciones y nuevos elementos a añadir, destacamos los siguientes (ver **Capítulo 4**):

- Modificar el **Visor** del **modo diseño** para eliminar la pantalla de dispositivo Android y sustituirla por una imagen de dron.
- Eliminar todas las categorías excepto **Sensors** y añadir una categoría que contenga los actuadores (**Actuators**).
- Añadir los componentes **Camera** y **Sound** a la categoría **Actuators**.
- Añadir nuevos componentes, tanto sensores como actuadores.
- Modificar, eliminar y añadir ciertas propiedades de la mayoría de los componentes.

3.3.2 Modo del editor de bloques

En cuanto a los elementos correspondientes al editor de bloques, se deberán mantener los siguientes elementos:

- Tipos de datos: **entero**, **en coma flotante**, **booleano** y **cadena de texto.**
- Categoría de bloques **Procedures**, junto con todos sus bloques, aunque se realizarán algunas modificaciones menores.
- Diversos bloques de las diversas categorías de bloques (ver **Capítulo 5**).

Se necesitarán realizar las siguientes modificaciones (ver **Capítulo 5**):

- Añadir nuevos bloques, correspondientes a determinadas funciones del lenguaje TeleoR, al editor de bloques.
- Añadir nuevo tipo de dato primitivo: ***atom***.
- Añadir bloque para **declaración de tipo personalizado**.
- Añadir nueva categoría que albergue los **procesos QuLog** (vistos en el **apartado 3.1**) y añadir dichos procesos.

Para poder implementar los cambios listados anteriormente, es necesario conocer cómo está organizado el código de MIT App Inventor. En el siguiente apartado se estudiará la estructura de dicho código.

3.4 Estructura del código fuente de MIT App Inventor

Como se ha comentado en el apartado anterior, se va a estudiar la distribución del **código fuente** de **MIT App Inventor**. En la siguiente figura se muestra la estructura general del código fuente, marcando en rojo aquellos directorios que serán de interés a la hora de realizar las modificaciones mencionadas en el apartado anterior, cuya función se explicará a continuación:

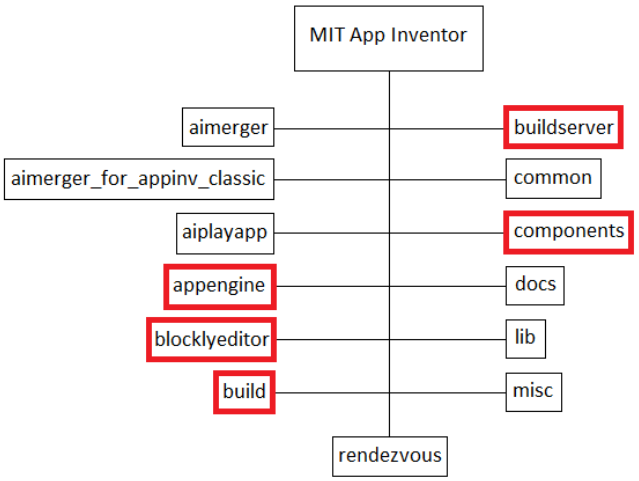


Figura 26: Estructura del código de MIT App Inventor

- **appengine**: En este directorio se almacenarán los ficheros que permitirán modificar el aspecto de la interfaz, tanto en el **modo de diseño** como en el **modo del editor de bloques**. Por ejemplo, se podrá modificar el aspecto de la **Screen**, añadir el **icono** representativo de un **componente** (creado previamente) o añadir una **categoría** y su **icono** en el **modo del editor de bloques**.
- **blocklyeditor**: Aquí se almacenará todo el código relacionado con los **bloques** del **editor de bloques** (no de las categorías que los contienen, excepto el color asociado a las mismas). Por ejemplo, en un fichero se podrá encontrar el código correspondiente a todos los bloques pertenecientes a la categoría **Control** (directorio **blocks**), mientras que en otro se encontrará el **código que generará** cada uno de los bloques de dicha categoría (directorio **generators**).
- **build**: Tras compilar el código fuente, los ficheros generados, los cuales se ejecutarán para arrancar el entorno web, serán generados en este directorio.
- **buildserver**: Contiene los ficheros de código necesarios para arrancar el *BuildServer*, el cual se ocupará de la **compilación del proyecto** de MIT App Inventor en curso y de generar el fichero **APK** correspondiente.
- **components**: Contiene los ficheros referentes a la **lógica de los componentes del modo de diseño** y las **categorías** que los contienen. Por ejemplo, si se quisiera añadir o eliminar una categoría o un componente, se podría hacer creando, modificando o eliminando ficheros contenidos en este directorio l

Ahora que se conoce la estructura general del código fuente de **MIT App Inventor**, se puede proceder a su modificación como se ha descrito en este capítulo. En el siguiente capítulo se comenzará la modificación de **MIT App Inventor** comenzando por el **modo de diseño** (**Capítulo 4**) y siguiendo con el **modo del editor de bloques** (**Capítulo 5**).

4. Construcción de la parte de diseño

En este capítulo se parte del supuesto de que el código ya ha sido descargado y se conoce como recompilar dicho código para poder observar los cambios realizados. Para no alargar innecesariamente la extensión de la memoria, dicha explicación se adjunta en el **Anexo 1**.

El directorio ***<RAIZ_APPINVENTOR>*** al que se hace referencia en esta memoria se corresponde con el directorio raíz donde se ha almacenado el código fuente de App Inventor. En el **Anexo 1** se indica más detalladamente la localización de este directorio.

También cabe destacar que los ficheros de código originales más importantes que se vayan a modificar aparecen en el **Anexo 2**. El aspecto de dichos ficheros tras su modificación los podremos localizar en el **Anexo 3**.

La primera modificación que se realizará en el código de App Inventor será la de adaptar la parte de diseño, la cual se abrirá automáticamente cada vez que se abra un proyecto de App Inventor. A continuación se comentará paso a paso cada uno de los cambios realizados.

4.1 Modificación de la Screen

En el centro de la parte de diseño, se puede ver una imagen que simula un teléfono móvil como se muestra en la siguiente captura:

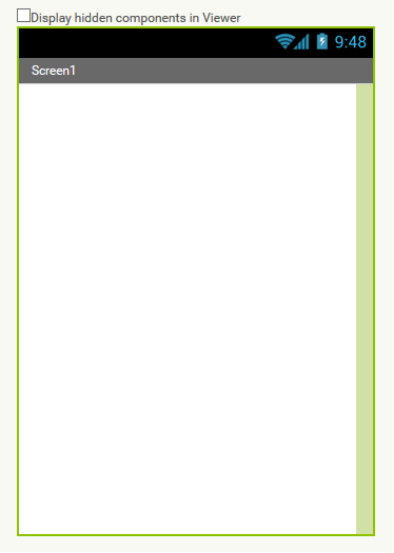


Figura 27: Screen Inicial

Esta imagen resulta inadecuada para la aplicación web que se quiere construir, por lo que el objetivo es sustituirla por la imagen de un dron (en principio fija). Para ello, se siguen los siguientes pasos:

1. Se debe buscar el fichero de estilos CSS **Ya.css** localizado en el directorio ***<RAÍZ_APPINVENTOR>\appengine\war***, abrirlo con un editor (por ejemplo gedit), y buscar el siguiente fragmento de código:

```
862   .ode-SimpleMockForm {
863     background-color: #d2e0a6;
864     border: 2px solid black;
865   }
```

Figura 28: Eliminando el color de fondo de la Screen

A continuación, se debe sustituir la línea marcada en rojo de manera que el código quede de la siguiente forma:

```
862   .ode-SimpleMockForm {
863     background: url('images/dron.jpeg');
864     border: 2px solid black;
865   }
```

Figura 29: Añadiendo la imagen de fondo

Con esto se consigue que se muestre la imagen, sin embargo, habrá otros elementos que bloquearán la visibilidad de la imagen, además de que las dimensiones del cuadro que contiene la imagen pueden no ajustarse a las dimensiones de la imagen elegida. Esto será corregido en los siguientes puntos.

2. Ahora se debe abrir el fichero **MockForm.java** situado en el directorio ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client\editor\simple\components***.

   En primer lugar, se debe eliminar una amplia porción de código al principio de la clase ***MockForm***. Para no alargar excesivamente el tamaño de la memoria, en el **Anexo 2** se puede encontrar el fichero **MockForm.java** antes de su modificación, mientras que en el **Anexo 3** se encontrará el fichero tras su modificación.

   Tras eliminar dicho código, el comienzo de la clase ***MockForm*** lucirá de la siguiente manera:

```
35  /**
36   * Mock Form component.
37   *
38   */
39  public final class MockForm extends MockContainer {
40
41    /**
42     * Component type name.
43     */
44    public static final String TYPE = "Form";
45
46    private static final String VISIBLE_TYPE = "Screen";
47
48    // TODO(lizlooney) 320x480 is the resolution of the G1. Do we want to change this to the
49    // resolution of the Nexus One?
50    private static final int PORTRAIT_WIDTH = 320;
51    private static final int PORTRAIT_HEIGHT = 480;
52    private static final int LANDSCAPE_WIDTH = 480;
53    private static final int LANDSCAPE_HEIGHT = 320;
54
55    // Property names
56    private static final String PROPERTY_NAME_TITLE = "Title";
57    private static final String PROPERTY_NAME_SCREEN_ORIENTATION = "ScreenOrientation";
58    private static final String PROPERTY_NAME_SCROLLABLE = "Scrollable";
59    private static final String PROPERTY_NAME_ICON = "Icon";
60    private static final String PROPERTY_NAME_VCODE = "VersionCode";
61    private static final String PROPERTY_NAME_VNAME = "VersionName";
62    private static final String PROPERTY_NAME_ANAME = "AppName";
```

Figura 30: Modificando dimensiones del cuadro de imagen

En la captura anterior se ha marcado en rojo una porción del código. Este código debe ser sustituido por el siguiente:

```
50    private static final int PORTRAIT_WIDTH = 160;
51    private static final int PORTRAIT_HEIGHT = 160;
52    private static final int LANDSCAPE_WIDTH = PORTRAIT_WIDTH;
53    private static final int LANDSCAPE_HEIGHT = PORTRAIT_HEIGHT;
```

Figura 31: Dimensiones del cuadro de imagen modificadas

En este caso, se ha elegido el valor 160 para todas las variables. Estos valores deben de corresponder con las dimensiones de la imagen que se desea mostrar (en este caso 160x160).

3. En el mismo fichero que en el paso anterior, se debe localizar el siguiente fragmento de código y eliminar el código marcado en rojo:

```
64    // Form UI components
65    AbsolutePanel formWidget;
66    ScrollPanel scrollPanel;
67    private TitleBar titleBar;
68    private MockComponent selectedComponent;
```

Figura 32: Eliminando variables en desuso

Con estas modificaciones se está eliminando el ScrollPanel que se muestra en la imagen del dispositivo móvil junto con la barra de título de la Screen. A continuación, se modificarán las partes del código que hacían uso de las variables que han sido eliminadas.

4. En este paso se debe eliminar el código marcado en **rojo** en la siguiente captura del código del constructor ***MockForm***:

```
108     formWidget = new AbsolutePanel();
109     formWidget.setStylePrimaryName("ode-SimpleMockForm");
110
111     // Initialize mock form UI by adding the phone bar and title bar.
112     formWidget.add(new PhoneBar());
113     titleBar = new TitleBar();
114     formWidget.add(titleBar);
115
116     // Put a ScrollPanel around the rootPanel.
117     scrollPanel = new ScrollPanel(rootPanel);
118     formWidget.add(scrollPanel);
119
120     screenWidth = PORTRAIT_WIDTH;
121     screenHeight = PORTRAIT_HEIGHT;
122     usableScreenHeight = screenHeight - PhoneBar.HEIGHT - TitleBar.HEIGHT;
```

Figura 33: Eliminando variables en desuso y modificando las dimensiones

Se puede observar también que se ha marcado una línea en **azul**. Esta línea debe ser modificada eliminando los dos últimos componentes de la operación resta. El fragmento de código quedará de la siguiente manera:

```
108     formWidget = new AbsolutePanel();
109     formWidget.setStylePrimaryName("ode-SimpleMockForm");
110
111     screenWidth = PORTRAIT_WIDTH;
112     screenHeight = PORTRAIT_HEIGHT;
113     usableScreenHeight = screenHeight;
```

Figura 34: Modificando las dimensiones 1

5. Ahora se modificará el método ***resizePanels()*** eliminando las líneas marcadas en **rojo**:

```
138   private void resizePanels() {
139     // Set the scrollPanel's width to account for the width of the vertical scrollbar.
140     int vertScrollbarWidth = getVerticalScrollbarWidth();
141     scrollPanel.setPixelSize(screenWidth + vertScrollbarWidth, usableScreenHeight);
142     formWidget.setPixelSize(screenWidth + vertScrollbarWidth, screenHeight);
143   }
```

Figura 35: Modificando el método resizePanels()

También se ha de modificar el código marcado en **azul** para que quede finalmente como el mostrado a continuación:

```
138   private void resizePanels() {
139     formWidget.setPixelSize(screenWidth, screenHeight);
140   }
```

Figura 36: Método resizePanels() resultante

6. Al igual que hicimos en el paso 4, se debe modificar la siguiente línea, eliminando los valores marcados en rojo

```
306        usableScreenHeight = screenHeight - PhoneBar.HEIGHT - TitleBar.HEIGHT;
```

Figura 37: Modificando las dimensiones 2

7. Por último, se debe eliminar el código marcado en rojo en la siguiente línea del método ***onPropertyChange***:

```
539    @Override
540    public void onPropertyChange(String propertyName, String newValue) {
541      super.onPropertyChange(propertyName, newValue);
542
543      // Apply changed properties to the mock component
544      if (propertyName.equals(PROPERTY_NAME_BACKGROUNDCOLOR)) {
545        setBackgroundColorProperty(newValue);
546      } else if (propertyName.equals(PROPERTY_NAME_BACKGROUNDIMAGE)) {
547        setBackgroundImageProperty(newValue);
548      } else if (propertyName.equals(PROPERTY_NAME_SCREEN_ORIENTATION)) {
549        setScreenOrientationProperty(newValue);
550      } else if (propertyName.equals(PROPERTY_NAME_SCROLLABLE)) {
551        setScrollableProperty(newValue);
552        adjustAlignmentDropdowns();
553      } else if (propertyName.equals(PROPERTY_NAME_TITLE)) {
554        titleBar.changeTitle(newValue);
555      } else if (propertyName.equals(PROPERTY_NAME_ICON)) {
556        setIconProperty(newValue);
```

Figura 38: Eliminando referencia a variable en desuso

Si se han seguido todos los pasos correctamente, podremos ver el siguiente resultado en la pantalla de diseño:

Figura 39: Screen Final

4.2 Modificación de la paleta de categorías

En la parte izquierda de la parte de diseño, se encuentra la paleta de categorías. En ella se distinguen 9 categorías:

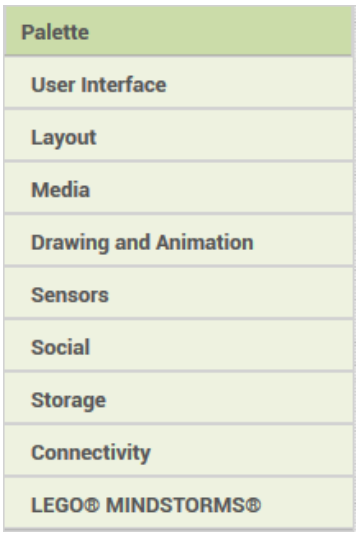


Figura 40: Paleta de categorías inicial

De estas 9 categorías, solamente serán necesarias la categoría ***Sensors*** y añadir una nueva categoría llamada ***Actuators*** y se deberán cambiar de categoría o eliminar ciertos componentes.

En este apartado, se eliminarán ciertos componentes "visualmente", es decir, la funcionalidad estará implementada pero no será posible acceder a ese componente desde la aplicación web. Esto es debido a la complejidad de eliminar cada uno de los componentes por completo de todos los ficheros del árbol de directorios, por lo que la tarea de eliminar la funcionalidad de estos componentes se pospondrá, ya que no es necesario para el correcto funcionamiento de la aplicación web.

Para ello hay que seguir los siguientes pasos:

1. Lo primero es localizar el fichero **ComponentCategory.java** situado en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\common*** y eliminar el código marcado en rojo a continuación:

```
48    public enum ComponentCategory {
49      // TODO(user): i18n category names
50      USERINTERFACE("User Interface"),
51      LAYOUT("Layout"),
52      MEDIA("Media"),
53      ANIMATION("Drawing and Animation"),
54      SENSORS("Sensors"),
55      SOCIAL("Social"),
56      STORAGE("Storage"),
57      CONNECTIVITY("Connectivity"),
58      LEGOMINDSTORMS("LEGO\u00AE MINDSTORMS\u00AE"),
59      //EXPERIMENTAL("Experimental"),
60      INTERNAL("For internal use only"),
61      // UNINITIALIZED is used as a default value so Swing libraries can still compile
62      UNINITIALIZED("Uninitialized");
63
64
65      // Mapping of component categories to names consisting only of lower-case letters,
66      // suitable for appearing in URLs.
67      private static final Map<String, String> DOC_MAP = new HashMap<String, String>();
68      static {
69        DOC_MAP.put("User Interface", "userinterface");
70        DOC_MAP.put("Layout", "layout");
71        DOC_MAP.put("media", "media");
72        DOC_MAP.put("Drawing and Animation", "animation");
73        DOC_MAP.put("Sensors", "sensors");
74        DOC_MAP.put("Social", "social");
75        DOC_MAP.put("Storage", "storage");
76        DOC_MAP.put("Connectivity", "connectivity");
77        DOC_MAP.put("LEGO\u00AE MINDSTORMS\u00AE", "legomindstorms");
78        //DOC_MAP.put("Experimental", "experimental");
79      }
```

Figura 41: Eliminando categorías 1

No hay que olvidar añadir las líneas necesarias para la nueva categoría ***Actuators***. Dichas líneas aparecen marcadas en **azul** en la siguiente captura:

```
48    public enum ComponentCategory {
49      // TODO(user): i18n category names
50      ACTUATORS("Actuators"),
51      SENSORS("Sensors"),
52      INTERNAL("For internal use only"),
53      // UNINITIALIZED is used as a default value so Swing libraries can still compile
54      UNINITIALIZED("Uninitialized");
55
56
57      // Mapping of component categories to names consisting only of lower-case letters,
58      // suitable for appearing in URLs.
59      private static final Map<String, String> DOC_MAP = new HashMap<String, String>();
60      static {
61        DOC_MAP.put("Actuators", "actuators");
62        DOC_MAP.put("Sensors", "sensors");
63      }
```

Figura 42: Añadiendo nueva categoría 1

2. Ahora se debe localizar el fichero ***DocumentationGenerator.java*** situado en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\scripts*** y eliminar el código marcado en rojo en la siguiente figura:

```
127     // Specify which categories are in which output column.
128     final ComponentCategory[][] categories = {
129       // Column one categories
130       {
131         ComponentCategory.USERINTERFACE,
132         ComponentCategory.LAYOUT,
133         ComponentCategory.MEDIA,
134         ComponentCategory.ANIMATION,
135         ComponentCategory.SOCIAL
136       },
137       // Column two categories
138       {
139         ComponentCategory.STORAGE,
140         ComponentCategory.CONNECTIVITY,
141         ComponentCategory.SENSORS,
142         ComponentCategory.LEGOMINDSTORMS,
143         //ComponentCategory.EXPERIMENTAL
144       }
145     };
```

Figura 43: Eliminando categorías 2

Eliminamos las líneas correspondientes a las categorías que no nos interesan y añadimos la correspondiente a la nueva categoría ***Actuators*** (marcada en azul):

```
127     // Specify which categories are in which output column.
128     final ComponentCategory[][] categories = {
129       // Column one categories
130       {
131         ComponentCategory.ACTUATORS
132       },
133       // Column two categories
134       {
135         ComponentCategory.SENSORS
136       }
137     };
```

Figura 44: Añadiendo nueva categoría 2

3. Ahora es necesario eliminar la asociación de los componentes con las categorías que se han eliminado junto con los componentes de la categoría ***Sensors*** que no serán utilizados. Para ello, primero se debe localizar el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime***. En ese directorio encontramos una serie de ficheros java correspondientes a cada uno de los componentes.

   Se ha de acceder a los ficheros indicados en la tabla siguiente y comentar la línea que corresponda a la asociación del componente con su categoría:

Fichero	Categoría
ActivityStarter.java	Connectivity
Ball.java	Drawing and Animation
BarcodeScanner.java	Sensors
BluetoothClient.java	Connectivity

BluetoothServer.java	Connectivity
Button.java	User Interface
Camcorder.java	Media
Canvas.java	Drawing and Animation
Checkbox.java	User Interface
ContactPicker.java	Social
DatePicker.java	User Interface
EmailPicker.java	Social
File.java	Storage
Form.java	Layout
FusiontablesControl.java	Storage
HorizontalArrangement.java	Layout
Image.java	User Interface
ImagePicker.java	Media
ImageSprite.java	Drawing and Animation
Label.java	User Interface
ListPicker.java	User Interface
ListView.java	User Interface
NearField.java	Sensors
Notifier.java	User Interface
NxtColorSensor.java	Lego Mindstorms
NxtDirectCommands.java	Lego Mindstorms
NxtDrive.java	Lego Mindstorms
NxtLightSensor.java	Lego Mindstorms
NxtSoundSensor.java	Lego Mindstorms
NxtTouchSensor.java	Lego Mindstorms
NxtUltrasonicSensor.java	Lego Mindstorms
PasswordTextBox.java	User Interface
PhoneCall.java	Social
PhoneNumberPicker.java	Social
Player.java	Media
Sharing.java	Social
Slider.java	User Interface
SoundRecorder.java	Media
SpeechRecognizer.java	Media
Spinner.java	User Interface
TableArrangement.java	Layout
TextBox.java	User Interface
Texting.java	Social
TextToSpeech.java	Media
TimePicker.java	User Interface
TinyDB.java	Storage
TinyWebDB.java	Storage
Twitter.java	Social
VerticalArrangement.java	Layout
VideoPlayer.java	Media
Web.java	Connectivity
WebViewer.java	User Interface

Tabla 5: Ficheros de componentes a eliminar

Como ejemplo, se realizará la modificación del fichero ***Ball.java***. Para ello se debe buscar y comentar la línea marcada en rojo a continuación:

```
26  @DesignerComponent(version = YaVersion.BALL_COMPONENT_VERSION,
27      description = "<p>A round 'sprite' that can be placed on a " +
28      "<code>Canvas</code>, where it can react to touches and drags, " +
29      "interact with other sprites (<code>ImageSprite</code>s and other " +
30      "<code>Ball</code>s) and the edge of the Canvas, and move according " +
31      "to its property values.</p>" +
32      "<p>For example, to have a <code>Ball</code> move 4 pixels toward the " +
33      "top of a <code>Canvas</code> every 500 milliseconds (half second), " +
34      "you would set the <code>Speed</code> property to 4 [pixels], the " +
35      "<code>Interval</code> property to 500 [milliseconds], the " +
36      "<code>Heading</code> property to 90 [degrees], and the " +
37      "<code>Enabled</code> property to <code>True</code>.  These and its " +
38      "other properties can be changed at any time.</p>" +
39      "<p>The difference between a Ball and an <code>ImageSprite</code> is " +
40      "that the latter can get its appearance from an image file, while a " +
41      "Ball's appearance can only be changed by varying its " +
42      "<code>PaintColor</code> and <code>Radius</code> properties.</p>",
43      category = ComponentCategory.ANIMATION)
```

Figura 45: Eliminando asociación de componente a categoría

El resultado sería el siguiente:

```
26  @DesignerComponent(version = YaVersion.BALL_COMPONENT_VERSION,
27      description = "<p>A round 'sprite' that can be placed on a " +
28      "<code>Canvas</code>, where it can react to touches and drags, " +
29      "interact with other sprites (<code>ImageSprite</code>s and other " +
30      "<code>Ball</code>s) and the edge of the Canvas, and move according " +
31      "to its property values.</p>" +
32      "<p>For example, to have a <code>Ball</code> move 4 pixels toward the " +
33      "top of a <code>Canvas</code> every 500 milliseconds (half second), " +
34      "you would set the <code>Speed</code> property to 4 [pixels], the " +
35      "<code>Interval</code> property to 500 [milliseconds], the " +
36      "<code>Heading</code> property to 90 [degrees], and the " +
37      "<code>Enabled</code> property to <code>True</code>.  These and its " +
38      "other properties can be changed at any time.</p>" +
39      "<p>The difference between a Ball and an <code>ImageSprite</code> is " +
40      "that the latter can get its appearance from an image file, while a " +
41      "Ball's appearance can only be changed by varying its " +
42      "<code>PaintColor</code> and <code>Radius</code> properties.</p>"
43      //category = ComponentCategory.ANIMATION
44  )
```

Figura 46: Eliminada asociación de componente a categoría

Al observar las capturas detenidamente, se puede apreciar que se ha eliminado la coma situada en la penúltima línea y se ha mantenido sin comentar el paréntesis ‘)’ que cerraba ***@DesignerComponent***. Se deberá tener cuidado a la hora de modificar los ficheros ya que en algunos de ellos (no en todos) se dará esta misma situación.

4. A continuación, es necesario cambiar la categoría asociada en los ficheros ***Camera.java*** y ***Sound.java*** de manera que se añada ***Camera.java*** a la categoría ***Sensors*** y ***Sound.java*** a la categoría ***Actuators***. Para ello, el atributo ‘category’ tendrá que ser modificado de la siguiente manera:

- **Camera.java**

```
43      category = ComponentCategory.SENSORS,
```

Figura 47: Camera asociada a categoría Sensors

- **Sound.java**

```
56      category = ComponentCategory.ACTUATORS,
```

Figura 48: Sound asociado a categoría Actuators

Para ***Sound.java***, además debemos de renombrarlo a ***Buzzer.java*** y sustituir el nombre de su clase y constructor por ***Buzzer*** tal y como se muestra en rojo en las siguientes capturas:

```
61  public class Buzzer extends AndroidNonvisibleComponent
62      implements Component, OnResumeListener, OnStopListener, OnDestroyListener, Deleteable {
```

Figura 49: Renombrando la clase Sound por Buzzer

```
116     public Buzzer(ComponentContainer container) {
117       super(container.$form());
118       thisComponent = this;
```

Figura 50: Renombrando el constructor Sound por Buzzer

Con estas modificaciones, ya hemos adaptado la paleta de categorías junto con la categoría a la que pertenece cada uno de los componentes ya incluidos que necesitaremos. El resultado final será el siguiente:

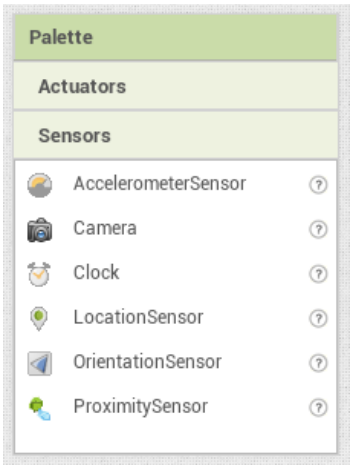


Figura 51: Paleta de categorías final

4.3 Adición de nuevos componentes

En este apartado se explicará paso a paso como añadir nuevos componentes al entorno gráfico web. Estos componentes representarán los distintos componentes físicos que pueden formar parte de los drones con los que se trabajará, y ya que no todos aparecen contemplados por defecto, será necesario añadirlos.

Se centrará en el proceso de creación del componente sin hacer mención a la adición de atributos o funciones asociadas a cada componente, ya que a esto se discutirá en el siguiente apartado.

Para llevar a cabo esta tarea se realizará un ejemplo con el componente ***GPS***, el cual queremos añadir a la categoría ***Sensors***. Es necesario mencionar que cuando un componente pueda actuar como un sensor o como un actuador, se crearán dos componentes diferentes. Por ejemplo, en el caso del componente ***GPS*** que vamos a crear, se deberá añadir otro componente llamado ***GPSController*** el cual estará ubicado en la categoría ***Actuators***. Habrá algunos componentes que sólo serán sensores y otros que sólo serán actuadores, por lo que no siempre será necesario la creación de un mismo componente en ambas categorías.

A continuación, se explican los pasos a seguir para la creación del componente ***GPS***:

1. Lo primero que hay que hacer es crear el fichero ***GPS.java*** en la ruta ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime*** en este fichero se debe escribir el siguiente código:

```
1  package com.google.appinventor.components.runtime;
2
3  import com.google.appinventor.components.annotations.DesignerComponent;
4  import com.google.appinventor.components.annotations.DesignerProperty;
5  import com.google.appinventor.components.annotations.PropertyCategory;
6  import com.google.appinventor.components.annotations.SimpleEvent;
7  import com.google.appinventor.components.annotations.SimpleFunction;
8  import com.google.appinventor.components.annotations.SimpleObject;
9  import com.google.appinventor.components.annotations.SimpleProperty;
10 import com.google.appinventor.components.common.ComponentCategory;
11 import com.google.appinventor.components.common.PropertyTypeConstants;
12 import com.google.appinventor.components.common.YaVersion;
13 import java.io.IOException;
14
15 @DesignerComponent(version = 1,
16     description = "<p>GPS description</p>",
17     category = ComponentCategory.SENSORS,
18     nonVisible = true,
19     iconName = "images/gps.png")
20 @SimpleObject
21 public class GPS
22 {
23   public GPS()
24   {
25   }
26
27 }
```

Figura 52: Código inicial de nuevo componente

Tal y como se muestra en la captura anterior, se ha creado el nuevo componente únicamente con la clase GPS y el constructor GPS vacío.

Todos los componentes que serán creados tendrán esta estructura inicial, a excepción de las líneas marcadas en azul, correspondientes a los atributos ***description***, ***category***, ***iconName*** y la ***clase*** y el ***constructor*** del componente, los cuales variarán sus valores en función del componente. En la siguiente tabla se explica brevemente su utilidad:

Campo	Función
description	Breve descripción del componente.
category	Indica la categoría en la que se englobará el componente.
iconName	Ruta relativa a la imagen que representará el componente.
clase	Parte del código en el que se añadirán todos los atributos y funciones del componente. Debe tener el mismo nombre que el fichero.
constructor	Necesario en cada uno de los componentes. Debe tener el mismo nombre que la clase. El constructor permanecerá vacío para cada componente que sea creado.

Tabla 6: Atributos del código de un componente

2. En el código insertado anteriormente hemos visto que se introduce la ruta relativa a la imagen que va a representar el componente, sin embargo, para asociar correctamente la imagen al componente se deben realizar una serie de modificaciones las cuales se describen a continuación:

   Se debe abrir el fichero ***Images.java*** localizado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y añadir el código siguiente al final de dicho fichero:

```
@Source("com/google/appinventor/images/gps.png")
ImageResource gps();
```

Figura 53: Entrada de GPS en Images.java

   El código anterior se tendrá que añadir para cada componente nuevo que sea añadido, solo teniendo que modificar el código marcado en azul. Por lo tanto, todas las imágenes asociadas a los componentes serán almacenadas en el mismo directorio.

   Ahora se ha de acceder al fichero ***SimpleComponentDescriptor.java*** localizado en el directorio ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client\editor\simple\palette*** y añadir la línea marcada en rojo:

```
88    private static void initBundledImages() {
89      bundledImages.put("images/accelerometersensor.png", images.accelerometersensor());
90      bundledImages.put("images/camera.png", images.camera());
91      bundledImages.put("images/clock.png", images.clock());
92      bundledImages.put("images/gameClient.png", images.gameclient());
93      bundledImages.put("images/locationSensor.png", images.locationSensor());
94      bundledImages.put("images/orientationsensor.png", images.orientationsensor());
95      bundledImages.put("images/pedometer.png", images.pedometerComponent());
96      bundledImages.put("images/phoneip.png", images.phonestatusComponent());
97      bundledImages.put("images/player.png", images.player());
98      bundledImages.put("images/soundEffect.png", images.soundeffect());
99      bundledImages.put("images/soundRecorder.png", images.soundRecorder());
100     bundledImages.put("images/speechRecognizer.png", images.speechRecognizer());
101     bundledImages.put("images/textToSpeech.png", images.textToSpeech());
102     bundledImages.put("images/voting.png", images.voting());
103     bundledImages.put("images/yandex.png", images.yandex());
104     bundledImages.put("images/proximitysensor.png", images.proximitysensor());
105     bundledImages.put("images/gps.png", images.gps());
106     imagesInitialized = true;
107   }
```

Figura 54: Entrada de GPS en SimpleComponentDescriptor.java 1

De dicha línea, hay que tener en cuenta que los fragmentos de códigos marcados en **azul** en la siguiente figura se deben de corresponder con los fragmentos marcados en **azul** que ya se mostraron en la **Figura 27**:

```
105     bundledImages.put("images/gps.png", images.gps());
```

Figura 55: Entrada de GPS en SimpleComponentDescriptor.java 2

3. Por último, hay que copiar la imagen que se usará para el componente al directorio ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\images***. Es muy importante que la imagen tenga un tamaño de **16x16 pixeles** y que se llame igual que aparece en las **Figuras 27, 28 y 29**.

Finalmente, si compilamos el código y ejecutamos la aplicación podremos observar el siguiente componente en la paleta ***Sensors***:

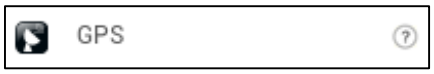


Figura 56: Componente GPS en paleta de categorías

Estos pasos se han de repetir para cada uno de los componentes que será necesario implementar. A continuación se adjunta una tabla con la información necesaria sobre cada uno de los componentes a añadir:

Componente	Categoría
Barometer	Sensors
Battery	Sensors
GPS	Sensors
GPSController	Actuators
Gyroscope	Sensors
LEDs	Actuators

Motor	Sensors
MotorController	Actuators
UltrasoundsSensor	Sensors

Tabla 7: Nuevos componentes a añadir

Si se ha repetido correctamente cada uno de los pasos para cada uno de los componentes, se tendrá una paleta de categoría similar a las siguientes capturas:

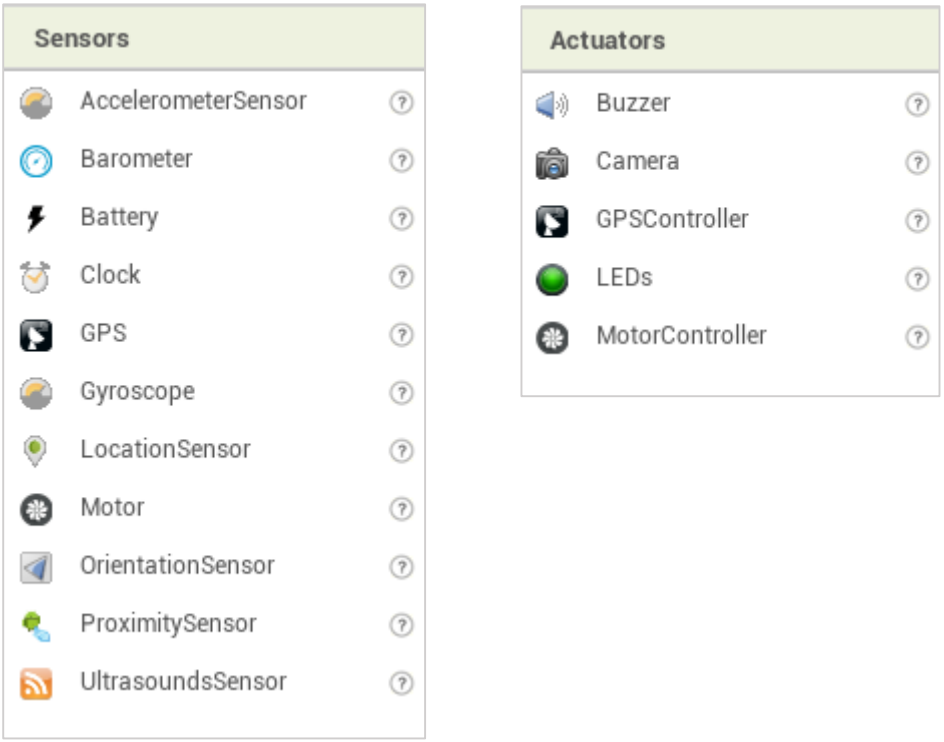


Figura 57: Componentes añadidos

4.4 Modificar componentes existentes

En el apartado anterior se ha visto cómo añadir un nuevo componente, pero no se debe pasar por alto que de los componentes que ya existían, y que hemos mantenido en el diseño, tienen ciertas propiedades y funciones (usadas en forma de bloques en el editor de bloques) las cuales no serán contempladas en el diseño final.

Algunas de ellas no necesitaremos eliminarlas por completo, ya que no estorbarán y su eliminación puede suponer alargar la finalización del diseño más de lo debido, por lo que simplemente se ocultarán visualmente (como se hizo con ciertos componentes en el **apartado 4.2**).

Los componentes constan de propiedades las cuales se pueden modificar con un valor inicial desde la parte de diseño. Por ejemplo, si un **Acelerómetro** es añadido al proyecto y es seleccionado en la paleta de componentes añadidos, se podrán ver y modificar las siguientes propiedades en la parte derecha del entorno web:

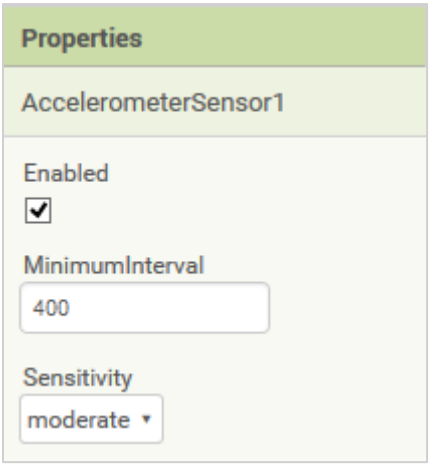


Figura 58: Propiedades por defecto del Acelerómetro

Hay que tener en cuenta que no todas las propiedades tienen por qué ser modificables desde este menú. Algunas tendrán que ser modificadas directamente desde el editor de bloques (veremos más sobre cómo funciona el editor de bloques a continuación y en los siguientes apartados y capítulos).

También se pueden ejecutar funciones específicas de cada componente y modificar las propiedades antes mencionadas de manera dinámica en la parte del editor en forma de bloques. Siguiendo con el mismo ejemplo, los bloques asociados al **Acelerómetro** serían los siguientes:

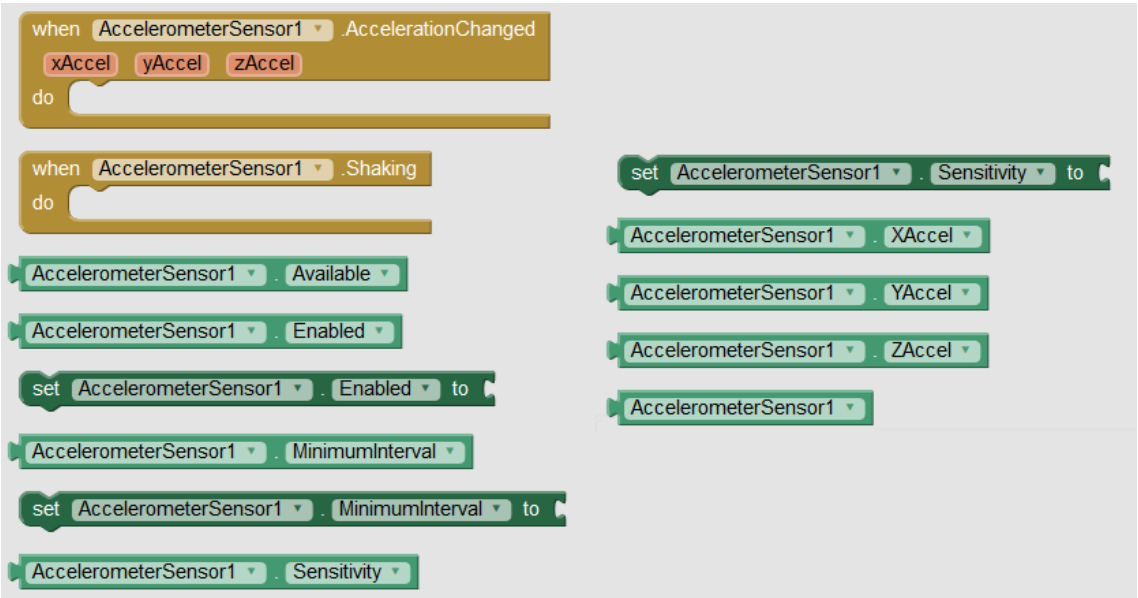


Figura 59: Bloques por defecto del Acelerómetro

Cada uno de los componentes que se han mantenido de App Inventor, tienen propiedades y bloques al igual que en el componente tomado como ejemplo. En cada uno de estos componentes, será necesario eliminar o modificar algunas de estas propiedades o bloques.

Por ejemplo, fijándose en la **Figura 33**, se aprecian dos bloques ***when…do***. Estos bloques no son necesarios, por lo que posteriormente serán eliminados.

Antes de comenzar, se adjunta una tabla que indica cada uno de los componentes que necesitan ser modificados y qué parte será necesario modificar:

Componente	Propiedades	Bloques
AccelerometerSensor		X
Clock		X
LocationSensor		X
OrientationSensor		X
ProximitySensor		X
Buzzer	X	X
Camera	X	X

Tabla 8: Partes de los componentes que se han de modificar

A continuación se explicará, para cada uno de los componentes mencionados, las modificaciones necesarias.

4.4.1 AccelerometerSensor

En este componente, tal y como se ha visto anteriormente, sólo se necesitará eliminar los dos bloques ***when…do***. También eliminaremos otras funciones que no usaremos.

1. Para eliminar el bloque correspondiente a la función ***AccelerationChanged***, debemos acceder al fichero ***AccelerometerSensor.java*** localizado en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime*** y comentar la línea marcada en **azul**:

```
192   /**
193    * Indicates the acceleration changed in the X, Y, and/or Z dimensions.
194    */
195   @SimpleEvent
196   public void AccelerationChanged(float xAccel, float yAccel, float zAccel) {
197     this.xAccel = xAccel;
198     this.yAccel = yAccel;
199     this.zAccel = zAccel;
200
201     addToSensorCache(X_CACHE, xAccel);
202     addToSensorCache(Y_CACHE, yAccel);
203     addToSensorCache(Z_CACHE, zAccel);
204
205     long currentTime = System.currentTimeMillis();
206
207     //Checks whether the phone is shaking and the minimum interval
208     //has elapsed since the last registered a shaking event.
209     if ((isShaking(X_CACHE, xAccel) || isShaking(Y_CACHE, yAccel) || isShaking(Z_CACHE, zAccel))
210         && (timeLastShook == 0 || currentTime >= timeLastShook + minimumInterval)){
211       timeLastShook = currentTime;
212       Shaking();
213     }
214
215     EventDispatcher.dispatchEvent(this, "AccelerationChanged", xAccel, yAccel, zAccel);
216   }
```

Figura 60: Eliminando bloque when AccelerationChanged

Para evitar problemas al eliminar otros fragmentos de código en los puntos posteriores, también eliminaremos el código marcado en **rojo** en la captura anterior.

2. Para eliminar el bloque correspondiente a la función ***Shaking***, debemos eliminar el código contenido en la siguiente figura:

```
218   /**
219    * Indicates the device started being shaken or continues to be shaken.
220    */
221   @SimpleEvent
222   public void Shaking() {
223     EventDispatcher.dispatchEvent(this, "Shaking");
224   }
```

Figura 61: Eliminando bloque when Shaking

3. Ahora que hemos eliminado ambos bloques ***when***, vamos a eliminar ciertos fragmentos de código que ya no son de utilidad (aunque no es estrictamente necesario su eliminación para el correcto funcionamiento del componente). Debido a la extensión de las modificaciones a realizar y con el objetivo de no alargar innecesariamente la extensión de la memoria, estos cambios se reflejarán en el **Anexo 2** y el **Anexo 3**.

4.4.2 Clock

En el caso del componente **Clock**, sólo será necesario eliminar el siguiente bloque:

Figura 62: Bloque when Timer de Clock

Para lograrlo solamente será necesario comentar las líneas marcadas en rojo en la siguiente captura:

```
78    /**
79     * Default Timer event handler.
80     */
81    @SimpleEvent(
82        description = "Timer has gone off.")
83    public void Timer() {
84      if (timerAlwaysFires || onScreen) {
85        EventDispatcher.dispatchEvent(this, "Timer");
86      }
87    }
```

Figura 63: Eliminando bloque when Timer

4.4.3 LocationSensor

Para el componente **LocationSensor** sólo se tendrán que eliminar los siguientes bloques:

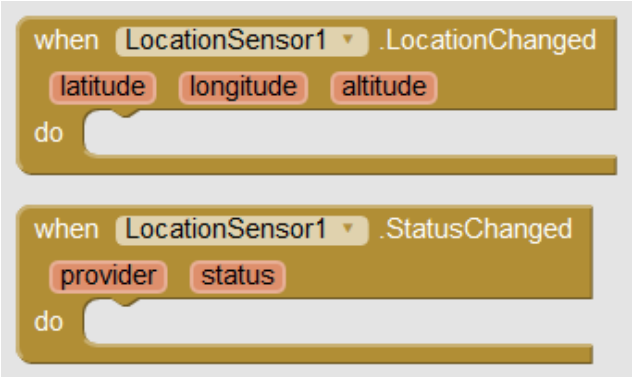


Figura 64: Bloques when LocationChanged y when StatusChanged de LocationSensor

Bastará con comentar las líneas marcadas en rojo en la siguiente figura:

```
210   /**
211    * Indicates that a new location has been detected.
212    */
213   @SimpleEvent
214   public void LocationChanged(double latitude, double longitude, double altitude) {
215     if (enabled) {
216       EventDispatcher.dispatchEvent(this, "LocationChanged", latitude, longitude, altitude);
217     }
218   }
219
220   /**
221    * Indicates that the status of the location provider service has changed, such as when a
222    * provider is lost or a new provider starts being used.
223    */
224   @SimpleEvent
225   public void StatusChanged(String provider, String status) {
226     if (enabled) {
227       EventDispatcher.dispatchEvent(this, "StatusChanged", provider, status);
228     }
229   }
```

Figura 65: Eliminando bloques when LocationChanged y when StatusChanged

4.4.4 OrientationSensor

En el componente **OrientationSensor** bastará con eliminar un bloque *when*:

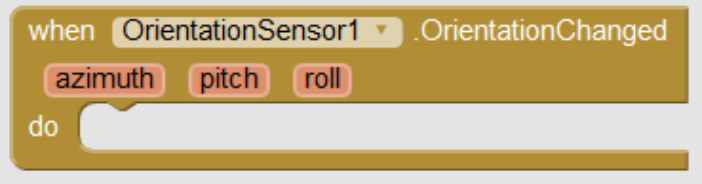


Figura 66: Bloque when OrientationChanged de OrientationSensor

De manera similar a lo realizado en los apartados anteriores, comentamos la línea marcada en rojo:

```
153   @SimpleEvent
154   public void OrientationChanged(float azimuth, float pitch, float roll) {
155     EventDispatcher.dispatchEvent(this, "OrientationChanged", azimuth, pitch, roll);
156   }
```

Figura 67: Eliminando bloque when OrientationChanged

4.4.5 ProximitySensor

En el componente **ProximitySensor**, el bloque *when* del cuál prescindiremos es el siguiente:

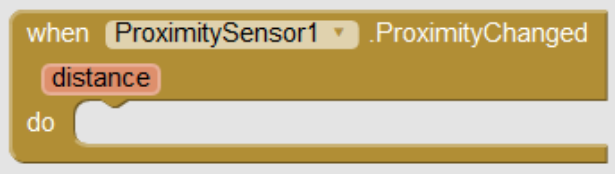


Figura 68: Bloque when ProximityChanged de ProximitySensor

Comentamos la siguiente línea marcada en rojo:

```
200   @SimpleEvent(description = "Triggered when distance (in cm) of the object to the device changes. ")
201   public void ProximityChanged(float distance) {
202       this.distance = distance;
203       EventDispatcher.dispatchEvent(this, "ProximityChanged", this.distance);
204   }
```

Figura 69: Eliminando bloque when ProximityChanged

4.4.6 Buzzer

El compoente **Buzzer** lleva incluidas por defecto las siguientes propiedades:

Figura 70: Propiedades iniciales de Buzzer

En este caso, debemos eliminar el atributo ***Source*** y sustituirlo por el booleano ***On***, el cuál indicará si el **Buzzer** está activado o no. Para ello se realizarán las siguientes modificaciones:

1. Primero se eliminará la propiedad ***Source***. Para ello, hay que abrir el fichero **Buzzer.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime*** y eliminar las siguientes líneas de código:

```
141   /**
142    * Returns the sound's filename.
143    */
144   @SimpleProperty(
145       category = PropertyCategory.BEHAVIOR,
146       description = "The name of the sound file.  Only certain " +
147       "formats are supported.  See http://developer.android.com/guide/appendix/media-formats.html.")
148   public String Source() {
149     return sourcePath;
150   }
```

Figura 71: Eliminando función Source() 1

```
152   /**
153    * Sets the sound source
154    *
155    * <p/>See {@link MediaUtil#determineMediaSource} for information about what
156    * a path can be.
157    *
158    * @param path  the path to the sound source
159    */
160   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ASSET,
161       defaultValue = "")
162   @SimpleProperty
163   public void Source(String path) {
164     sourcePath = (path == null) ? "" : path;
165
166     // Clear the previous sound.
167     if (streamId != 0) {
168       soundPool.stop(streamId);
169       streamId = 0;
170     }
171     soundId = 0;
172
173     if (sourcePath.length() != 0) {
174       Integer existingSoundId = soundMap.get(sourcePath);
175       if (existingSoundId != null) {
176         soundId = existingSoundId;
177
178       } else {
179         Log.i("Sound", "No existing sound with path " + sourcePath + ".");
180         try {
181           int newSoundId = MediaUtil.loadSoundPool(soundPool, form, sourcePath);
182           if (newSoundId != 0) {
183             soundMap.put(sourcePath, newSoundId);
184             Log.i("Sound", "Successfully began loading sound: setting soundId to " + newSoundId + ".");
185             soundId = newSoundId;
186             // set flag to show that loading has begun
187             loadComplete = false;
188           } else {
189             form.dispatchErrorOccurredEvent(this, "Source",
190                 ErrorMessages.ERROR_UNABLE_TO_LOAD_MEDIA, sourcePath);
191           }
192         } catch (IOException e) {
193           form.dispatchErrorOccurredEvent(this, "Source",
194               ErrorMessages.ERROR_UNABLE_TO_LOAD_MEDIA, sourcePath);
195         }
196       }
197     }
198   }
```

Figura 72: Eliminando función Source() 2

2. Ahora se va a introducir el booleano ***On*** en el componente. Lo primero es crear la variable booleana de instancia que almacenará el valor. Esta variable se debe crear **antes del constructor de la clase Buzzer**. A continuación se muestra el código equivalente a dicha variable:

```
115     // Power
116     private boolean isOn;
```

Figura 73: Variable de instancia isOn en Buzzer

El booleano ***On*** tendrá asociado dos funciones: una servirá para obtener el valor actual de dicha variable (función Get marcada en azul) y la otra para la modificación de su valor (función Set marcada en rojo). Dichas funciones deben declararse **después del constructor de la clase Buzzer**:

```
141     /**
142      * On/Off
143      */
144     @SimpleProperty(
145         category = PropertyCategory.BEHAVIOR)
146     public boolean On() {
147       return isOn;
148     }
149
150     /**
151      * On/Off
152      */
153     @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
154         defaultValue = "False")
155     @SimpleProperty
156     public void On(boolean status) {
157       isOn = status;
158     }
```

Figura 74: Funciones Get y Set de On

En la captura anterior, se muestra que cada una de las funciones tienen asociados unos atributos, a continuación los explicamos brevemente:

Atributo	Descripción
@SimpleProperty	Mediante este atributo añadiremos al editor de bloques un bloque **Get** (azul) o **Set** (rojo) cuya funcionalidad sea la de la función asociada. El parámetro **category** indicará el tipo de propiedad (sólo se indica en la función **Get**).
@DesignerProperty	Mediante este atributo añadiremos al menú de propiedades (visto en la **Figura 44**), la propiedad **On**. El parámetro **editorType** indica el tipo de valor que almacenará la propiedad y el parámetro **defaultValue** será su valor inicial.

Tabla 9: Atributos asociados a funciones

Si se ha modificado correctamente el código, el menú de propiedades quedará como el de la siguiente figura:

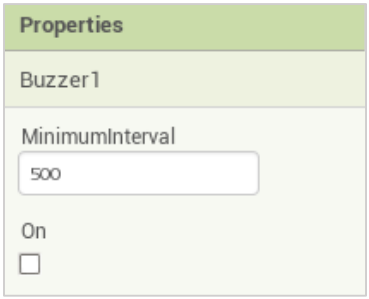


Figura 75: Propiedades finales de Buzzer

Los bloques **Get** y **Set** de **On** lucirán de la siguiente manera:

Figura 76: Bloques Get y Set de Buzzer

3. Al igual que se realizó con el componente **AccelerometerSensor** (apartado 4.4.1), se realizarán una serie de modificaciones para eliminar código que no será de utilidad en el componente. Al igual que en **AccelerometerSensor**, estas modificaciones no serán estrictamente necesarias para el correcto funcionamiento del componente. Debido a la extensión de estas modificaciones, en el **Anexo 3** se puede encontrar el código final del fichero tras realizar dichas modificaciones.

4.4.7 Camera

El componente **Camera** sólo tiene como propiedad el booleano ***UseFront***:

Figura 77: Propiedades iniciales de Camera

Sin embargo, esta propiedad no será necesaria, por lo que se procederá a su eliminación. Para ello, debemos acceder al fichero ***Camera.java*** situado en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime*** y eliminar las siguientes líneas de código (marcadas en rojo en la **Figura 52** y todas las líneas de las **Figuras 53 y 54**):

```
59    // whether to open into the front-facing camera
60    private boolean useFront;
61
62    /**
63     * Creates a Camera component.
64     *
65     * Camera has a boolean option to request the forward-facing camera via an intent extra.
66     *
67     * @param container container, component will be placed in
68     */
69    public Camera(ComponentContainer container) {
70      super(container.$form());
71      this.container = container;
72
73      // Default property values
74      UseFront(false);
75    }
```

Figura 78: Eliminando UseFront de Camera 1

```
71    /**
72     * Returns true if the front-facing camera is to be used (when available)
73     *
74     * @return {@code true} indicates front-facing is to be used, {@code false} will open default
75     */
76    @SimpleProperty(
77      category = PropertyCategory.BEHAVIOR)
78    public boolean UseFront() {
79      return useFront;
80    }
81
82    /**
83     * Specifies whether the front-facing camera should be used (when available)
84     *
85     * @param front
86     *          {@code true} for front-facing camera, {@code false} for default
87     */
88    @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue = "False")
89    @SimpleProperty(description = "Specifies whether the front-facing camera should be used (when available). "
90      + "If the device does not have a front-facing camera, this option will be ignored "
91      + "and the camera will open normally.")
92    public void UseFront(boolean front) {
93      useFront = front;
94    }
```

Figura 79: Eliminando UseFront de Camera 2

```
101         // NOTE: This uses an undocumented, testing feature (CAMERA_FACING).
102         // It may not work in the future.
103         if (useFront) {
104           intent.putExtra("android.intent.extras.CAMERA_FACING", 1);
105         }
```

Figura 80: Eliminando UseFront de Camera 3

El código perteneciente a la **Figura 54** se encuentra en el método ***TakePicture()*** de Camera.

Una vez eliminada la propiedad ***UseFront***, el siguiente paso será añadir tres nuevos bloques al componente. Estos bloques serán los correspondientes a las siguientes funciones:

Función	Descripción
orientation	Toma como parámetros la orientación vertical y horizontal en grados y permite modificar la dirección a la que apunta la cámara.
startVideo	Comienza la grabación de vídeo.
stopVideo	Finaliza la grabación de vídeo.

Tabla 10: Descripción de nuevas funciones de Camera

A continuación se detallan los pasos a seguir:

1. Para el bloque ***orientation***, lo primero es añadir las variables de instancia siguientes **antes del constructor de la clase**:

```
59      // Vertical move in degrees
60      private int tilt;
61
62      // Horizontal move in degrees
63      private int pan;
```

Figura 81: Variables de instancia de función orientation de Camera

   Las funciones ***startVideo*** y ***stopVideo*** no precisarán de variables de instancia.

2. El código necesario para implementar las tres funciones se debe escribir **después del constructor de la clase** y corresponde al de la siguiente figura:

```
77      /*
78       * Camera Orientation
79       */
80      @SimpleFunction(description = "set Camera orientation")
81      public void orientation(int tilt, int pan) {
82        if(tilt >= 0 && tilt <= 360 && pan >= 0 && pan <= 360)
83        {
84          this.tilt = tilt;
85          this.pan = pan;
86        }
87      }
88
89      /**
90       * StartVideo
91       */
92      @SimpleFunction
93      public void startVideo() {}
94
95      /**
96       * StopVideo
97       */
98      @SimpleFunction
99      public void stopVideo() {}
```

Figura 82: Funciones orientation, startVideo y stopVideo de Camera

   En la figura anterior, se puede observar que las funciones ***startVideo*** y ***stopVideo*** no tienen ninguna funcionalidad implementada. Esto es debido a que el lenguaje visual que se está desarrollando tiene como único objetivo el traducir los bloques al lenguaje TeleoR, por lo que sólo será necesario asociar estas funciones con las correspondientes en lenguaje TeleoR, sin tener que implementar funcionalidad.

   Estas funciones vacías se repetirán en los siguientes apartados cuando se añadan los nuevos bloques asociados a los componentes creados en el **apartado 4.3**.

3. Por último, es conveniente añadir que, al igual que en el componente Buzzer, se han realizado modificaciones para eliminar código inservible. Estas modificaciones también se pueden encontrar en el **Anexo 3**.

Finalmente, en el editor de bloques se podrán ver los resultados del código añadido al componente **Camera**:

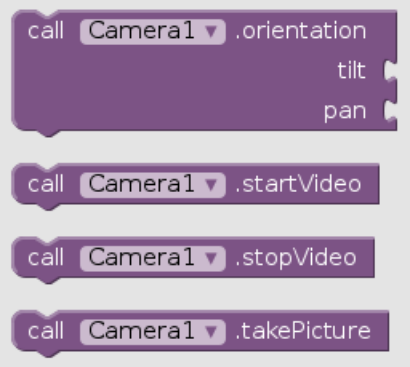


Figura 83: Bloques de Camera

4.5 Adición de bloques a los nuevos componentes

En este apartado se utilizará la misma metodología utilizada en el **apartado 4.4** para añadir nuevos bloques a los componentes creados en el **apartado 4.3**.

Los ficheros java de cada uno de los componentes que se modificará a continuación se encuentran en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime***.

A continuación se detallan los pasos a seguir para cada uno de estos componentes.

4.5.1 Battery

El componente **Battery** únicamente constará de la propiedad ***Level***, la cual almacenará la carga actual de la batería del dron, sin embargo, no se debe permitir al usuario modificar el valor inicial de esta propiedad en la parte de diseño, sino que su obtención y modificación sólo será posible de manera dinámica desde el **editor** mediante dos bloques **Get** y **Set**.

Para implementar dichos bloques, se debe acceder al fichero **Battery.java** y añadir el código marcado en rojo:

```
30    // Battery Level
31    private int level;
32
33    // Constructor
34    public Battery() {}
35
36    /**
37     * Get Level
38     */
39    @SimpleProperty(
40        category = PropertyCategory.BEHAVIOR)
41    public int Level() {
42      return level;
43    }
44
45    /**
46     * Set Level
47     */
48    @SimpleProperty
49    public void Level(int lev) {
50      level = lev;
51    }
```

Figura 84: Funciones Get y Set de Battery

Tal y como se ha ido realizando anteriormente, la variable de instancia se debe crear **antes del constructor de la clase**, y las funciones **después del constructor de la clase**.

El resultado final debería ser el siguiente:

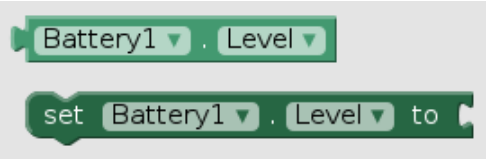


Figura 85: Bloques Get y Set de Battery

4.5.2 GPS

El GPS es un sensor el cual constará de tres propiedades: ***Latitude***, ***Longitude*** y ***Altitude***.

Al igual que en el apartado anterior, se creará un bloque **Get** y otro **Set** para cada propiedad.

Para implementar dichos bloques, se debe abrir el fichero **GPS.java** y seguir las siguientes instrucciones:

1. Lo primero es añadir las variables de instancia para cada una de las propiedades añadiendo el código siguiente **antes del constructor de la clase**:

```
23    // Current position
24    private double latitude;
25    private double longitude;
26    private double altitude;
```

Figura 86: Variables de instancia de GPS

2. A continuación se insertará el siguiente código, correspondiente a los bloques **Get** de cada propiedad:

```
30    /**
31     * Get Latitude
32     */
33    @SimpleProperty(
34        category = PropertyCategory.BEHAVIOR)
35    public double Latitude() {
36      return latitude;
37    }
38
39    /**
40     * Get Longitude
41     */
42    @SimpleProperty(
43        category = PropertyCategory.BEHAVIOR)
44    public double Longitude() {
45      return longitude;
46    }
47
48    /**
49     * Get Altitude
50     */
51    @SimpleProperty(
52        category = PropertyCategory.BEHAVIOR)
53    public double Altitude() {
54      return altitude;
55    }
```

Figura 87: Funciones Get de GPS

3. Por último, añadir el código correspondiente a los bloques **Set**:

```
57    /**
58     * Set Latitude
59     */
60    @SimpleProperty
61    public void Latitude(double lat) {
62      latitude = lat;
63    }
64
65    /**
66     * Set Longitude
67     */
68    @SimpleProperty
69    public void Longitude(double lon) {
70      longitude = lon;
71    }
72
73    /**
74     * Set Altitude
75     */
76    @SimpleProperty
77    public void Altitude(double alt) {
78      altitude = alt;
79    }
```

Figura 88: Funciones Set de GPS

Si se han seguido todos los pasos correctamente, el resultado será el mostrado en la siguiente figura:

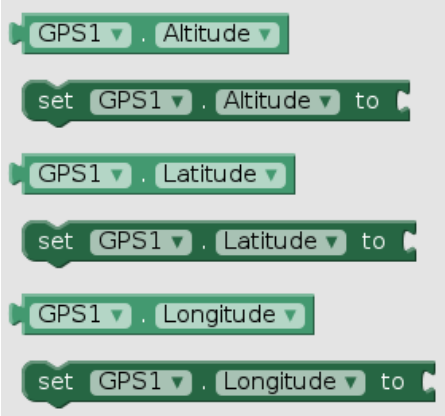


Figura 89: Bloques Get y Set de GPS

4.5.3 GPSController

El componente **GPSController** es un actuador el cual se compone de dos funciones: ***resetHome*** y ***setHome*.**

Las coordenadas **Home** se corresponderán con la de la base o punto de partida. Estas dos funciones nos permitirán modificar los valores de las coordenadas Home o resetear dichas coordenadas a un valor por defecto. Las coordenadas se compondrán de tres valores: latitud, longitud y altitud (definidos en el **apartado 4.5.2**).

Para añadir dichas funciones, se debe acceder al fichero **GPSController.java** y añadir el siguiente código marcado en rojo después del constructor de la clase:

```
public GPSController() {}

/*
 * Change Home Coordinates
 */
@SimpleFunction
public void setHome(double latitude, double longitude, double altitude) {}

/**
 * Reset Home Coordinates
 */
@SimpleFunction
public void resetHome() {}
```

Figura 90: Funciones resetHome y setHome de GPSController

Como ya se comentó anteriormente, no es necesario añadir funcionalidad a las funciones implementadas, ya que sólo traduciremos dichas funciones a comandos específicos del lenguaje TeleoR.

Es conveniente destacar que en este caso, a diferencia de los anteriores, se ha usado el atributo ***@SimpleFunction***, el cuál creará un tipo de bloque diferente a los **Get** y **Set** implementados hasta ahora. Como en el caso de ***setHome***, los tres parámetros que se le pasan como argumento influirán en el efecto de dicha función (en este caso, modificar las coordenadas base).

Los nuevos bloques creados lucirán como los mostrados a continuación:

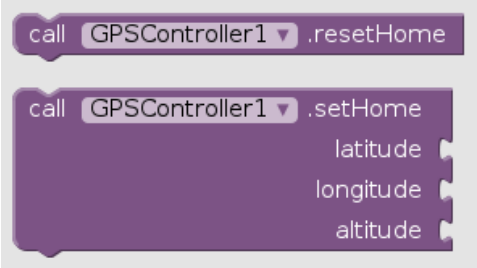


Figura 91: Bloques resetHome y setHome de GPSController

4.5.4 Gyroscope

Para el componente **Gyroscope** sólo será necesario poder obtener el valor de sus coordenadas ***x***, ***y*** y ***z***. Por ello, en este caso sólo habrá que añadir tres bloques ***Get*** correspondientes a dichas coordenadas, los cuales se llamarán: ***XGyr***, ***YGyr*** y ***ZGyr***.

Para añadir las tres funciones que harán posible la creación de los bloques, se debe abrir el fichero **Gyroscope.java** y añadir el siguiente código marcado en rojo tal y como se hizo en puntos anteriores:

```
28    // Backing for sensor values
29    private float xGyr;
30    private float yGyr;
31    private float zGyr;
32
33    public Gyroscope () {}
34
35    /**
36     * Get XGyr
37     */
38    @SimpleProperty(
39        category = PropertyCategory.BEHAVIOR)
40    public float XGyr() {
41      return xGyr;
42    }
43
44    /**
45     * Get YGyr
46     */
47    @SimpleProperty(
48        category = PropertyCategory.BEHAVIOR)
49    public float YGyr() {
50      return yGyr;
51    }
52
53    /**
54     * Get ZGyr
55     */
56    @SimpleProperty(
57        category = PropertyCategory.BEHAVIOR)
58    public float ZGyr() {
59      return zGyr;
60    }
```

Figura 92: Funciones Get de Gyroscope

Los bloques generados serán los siguientes:

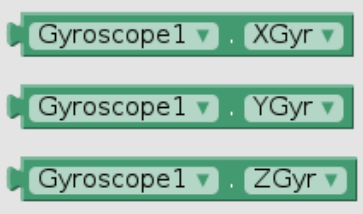


Figura 93: Bloques Get de Gyroscope

4.5.5 LEDs

En el componente **LEDs** solamente será necesaria la opción de activar o desactivar los leds así como poder comprobar y modificar dinámicamente desde el editor su activación.

Para ello se creará, al igual que se hizo en el componente **Buzzer** (ver **apartado 4.4.6**), la propiedad booleana ***On*** con sus correspondientes bloques **Get** y **Set**.

Para crear la propiedad, mostrarla en el menú de propiedades y crear los bloques Get y Set asociados a ella, se debe abrir el fichero **LEDs.java** y añadir el código marcado en rojo en la siguiente captura:

```
28    // Power
29    private boolean isOn;
30
31    public LEDs() {}
32
33    /**
34     * Get On/Off
35     */
36    @SimpleProperty(
37        category = PropertyCategory.BEHAVIOR)
38    public boolean On() {
39      return isOn;
40    }
41
42    /**
43     * Set On/Off
44     */
45    @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
46        defaultValue = "False")
47    @SimpleProperty
48    public void On(boolean status) {
49      isOn = status;
50    }
```

Figura 94: Funciones Get y Set de LEDs

Los bloques resultantes serán los siguientes:

Figura 95: Bloque Get y Set de LEDs

4.5.6 Motor

El sensor **Motor** tiene un amplio número de propiedades booleanas y numéricas las cuales es necesario que sean contempladas por en el lenguaje visual. Estas propiedades sólo serán accesibles mediante el editor de bloques.

A continuación se muestra una tabla con dichas propiedades y el tipo de dato correspondiente a cada una de ellas:

Propiedad	Tipo de dato
isLanded	Booleano
isTakingOff	Booleano
isHovering	Booleano
isFlying	Booleano
isLanding	Booleano
inEmergency	Booleano
Latitude	Double
Longitude	Double
Altitude	Double
SpeedX	Double
SpeedY	Double
SpeedZ	Double
Roll	Entero
Pitch	Entero
Yaw	Entero

Tabla 11: Nuevas propiedades de Motor

Mediante la tabla anterior, se puede proceder a la creación de los bloques **Get** y **Set** correspondientes a cada una de dichas propiedades.

Debido al gran número de bloques que se deben crear y con el objetivo de no alargar innecesariamente la memoria, se explicará cómo añadir únicamente los bloques Get y Set pertenecientes a la propiedad ***isLanded***, ya que el resto de bloques se pueden introducir de manera análoga. El fichero **Motor.java** con cada una de las funciones necesarias para la creación de estos bloques se encuentra en el **Anexo 3**.

Lo primero es abrir el fichero **Motor.java** e introducir la siguiente línea de código antes del constructor de la clase:

```
28    private boolean landed;
```

Figura 96: Variable de instancia de propiedad isLanded

Posteriormente introducimos el siguiente código después del constructor:

```
32    /**
33     * Get isLanded
34     */
35    @SimpleProperty(
36        category = PropertyCategory.BEHAVIOR)
37    public boolean isLanded() {
38      return landed;
39    }
40
41    /**
42     * Set isLanded
43     */
44    @SimpleProperty
45    public void isLanded(boolean landed) {
46      this.landed = landed;
47    }
```

Figura 97: Funciones Get y Set de isLanded

Con esto ya se habrán añadido correctamente los bloques Get y Set de isLanded. Si se repiten los pasos para cada una de las propiedades (teniendo en cuenta el tipo de dato de cada una de ellas), se obtienen los siguientes bloques:

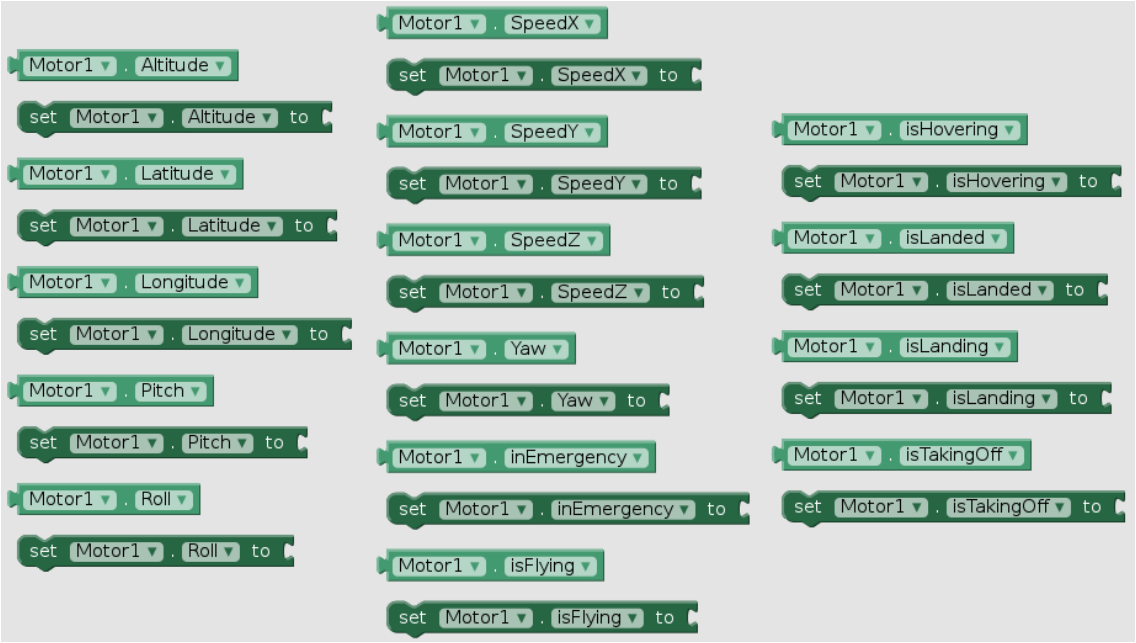


Figura 98: Bloques Get y Set de Motor

4.5.7 MotorController

En el actuador **MotorController** se añadirán nuevos bloques que ejecutarán comandos específicos en lenguaje TeleoR. Las funciones que crearán dichos bloques estarán vacías por las mismas razones vistas en los bloques de **GPSController** (ver **apartado 4.5.3**).

A continuación se muestra en una tabla cada una de dichas funciones junto con los parámetros que empleará como argumentos cada una de ellas junto con el tipo de dato de dichos parámetros:

Función	Parámetros	Tipo de dato
flatTrim	-	-
takeOff	-	-
moveForward	SpeedX SpeedY SpeedZ	Double
moveBackward	SpeedX SpeedY SpeedZ	Double
moveRight	SpeedX SpeedY SpeedZ	Double
moveLeft	SpeedX SpeedY SpeedZ	Double
up	SpeedX SpeedY SpeedZ	Double
down	SpeedX SpeedY SpeedZ	Double
land	-	-
startNavigateHome	-	-
stopNavigateHome	-	-
emergencyMode	-	-
flip	direction	String

Tabla 12: Nuevas funciones de MotorController

Para no alargar la extensión de la memoria, se explicará únicamente cómo añadir las funciones ***flatTrim***, ***takeOff*** y ***moveForward***.

Para ello hay que abrir el fichero **MotorController.java** y añadir el siguiente código después del constructor de la clase:

```
30    /**
31     * flatTrim
32     */
33    @SimpleFunction
34    public void flatTrim() {}
35
36    /**
37     * takeOff
38     */
39    @SimpleFunction
40    public void takeOff() {}
41
42    /**
43     * moveForward
44     */
45    @SimpleFunction
46    public void moveForward(double SpeedX, double SpeedY, double SpeedZ) {}
```

Figura 99: Funciones flatTrim, takeOff y moveForward de MotorController

Introduciendo de igual manera el resto de funciones, se obtienen finalmente los siguientes bloques:

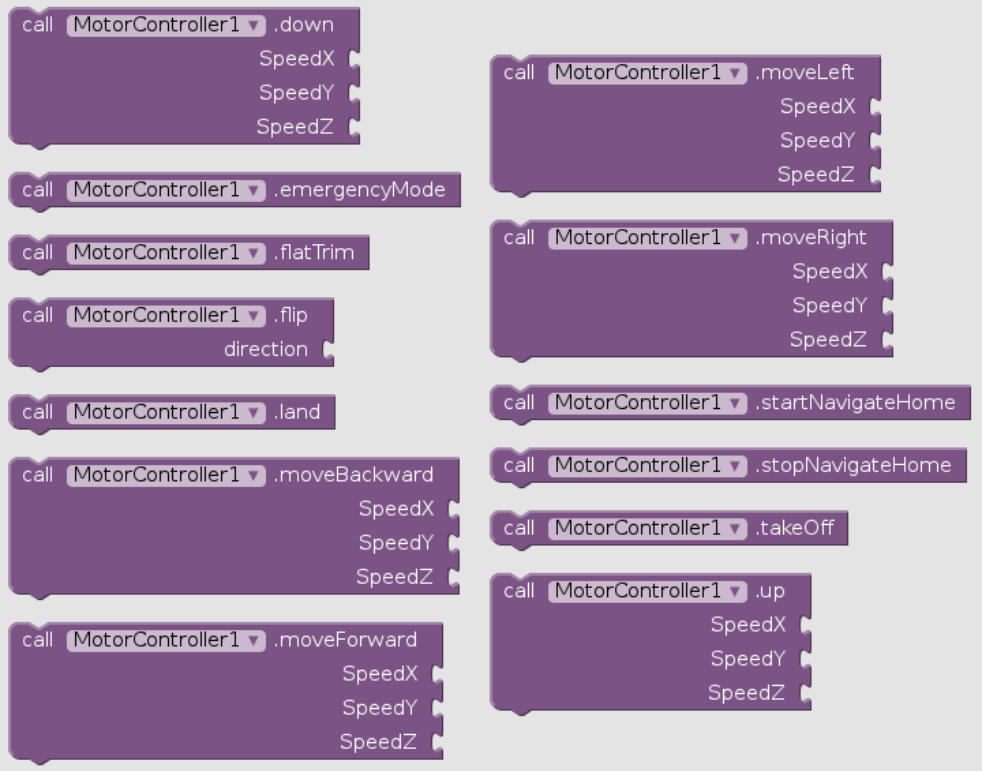


Figura 100: Bloques de MotorController

4.5.8 UltrasoundsSensor

Por último, en el componente **UltrasoundsSensor** sólo será necesario poder obtener mediante un bloque **Get** la distancia entre el dron y el objeto detectado mediante el sensor.

Se debe abrir el fichero **UltrasoundsSensor.java** e introducir el siguiente código marcado en rojo:

```
28    private float distance;
29
30    public UltrasoundsSensor() {}
31
32    /**
33     * Distance
34     */
35    @SimpleProperty(
36        category = PropertyCategory.BEHAVIOR)
37    public float Distance() {
38      return distance;
39    }
```

Figura 101: Función Get de UltrasoundSensor

El bloque resultante será el siguiente:

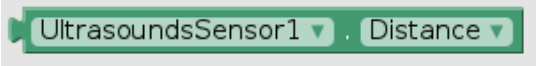


Figura 102: Bloque Get de UltrasoundSensor

4.6 Modificación del nombre del entorno gráfico web

Como último punto a tener en cuenta en este capítulo, se le debe dar un nuevo nombre a al entorno gráfico web que se está creando.

El nombre elegido será: **TeleoR Factory**.

Para lograr cambiar y mostrar correctamente el nuevo nombre, se debe acceder al fichero **TopPanel.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y buscar y modificar la siguiente línea de código marcada en rojo:

```
265     panel.add(logo);
266     panel.setCellWidth(logo, "50px");
267     Label title = new Label("MIT App Inventor 2");
268     Label version = new Label("Beta");
```

Figura 103: Modificando el nombre del entorno web

Dicha línea será sustituida por la siguiente:

```
267     Label title = new Label("TeleoR Factory");
```

Figura 104: Nuevo nombre del entorno web

Si se accede de nuevo al entorno web, se podrá visualizar el nuevo nombre:

Figura 105: Logo del entorno web

4.7 Otras modificaciones (Opcional)

En este último apartado se realizarán una serie de modificaciones adicionales las cuales, aunque no sean indispensables para el funcionamiento del entorno visual, si que es interesante el tenerlas en cuenta. En primer lugar se **eliminará funcionalidad incluida en la Screen** la cual ha dejado de tener utilidad, y en segundo lugar se permitirá **generar los ficheros de cada proyecto** creado con TeleoR Factory en una ruta específica.

4.7.1 Eliminar funcionalidad de la Screen

Dentro del entorno web, hay un componente del cuál no se ha tenido en cuenta sus propiedades y bloques: la **Screen**.

Cuando en la parte de diseño no se selecciona ningún componente, aparecen las propiedades de la Screen. Estas propiedades, las cuales tienen asociados una serie de bloques, aparecen disponibles en todos los proyectos creados para el uso del programador. Sin embargo, para los fines perseguidos en este proyecto, no tienen ninguna utilidad.

Por ello, a pesar de que las modificaciones realizadas a continuación no son estrictamente necesarias, sí que es interesante eliminar funcionalidad innecesaria.

En la siguiente figura se muestran las propiedades asociadas a la Screen:

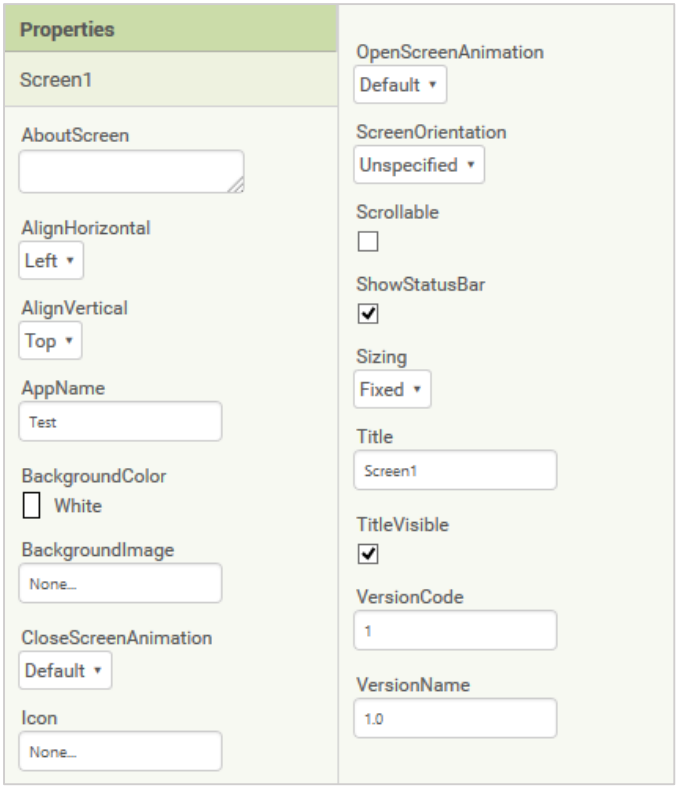


Figura 106: Propiedades de la Screen

Para lograr eliminarlas **visualmente** (eliminarlas por completo requeriría una eliminación más exhaustiva de código en varios ficheros), se debe acceder al fichero **Form.java** situado en el directorio ***<RAÍZ_APPINVENTOR>\components\src\com\google\appinventor\components\runtime*** y buscar las funciones mostradas en la **Figura 79**. Algunas de estas propiedades tendrán asociadas más de una función, pudiendo darse el caso de que una propiedad, como por ejemplo la propiedad ***BackgroundColor***, la cual tendrá asociados un bloque **Get** y otro **Set**.

Debido a la extensión de este apartado, sólo se mostrará la eliminación visual de las propiedades tomando como ejemplo la propiedad ***BackgroundColor***, sin embargo en el **Anexo 3** se podrá encontrar el fichero **Form.java** con todas las modificaciones realizadas.

En el fichero **Form.java** se han de buscar las siguientes líneas marcadas en rojo y comentarlas:

```
665   /**
666    * BackgroundColor property getter method.
667    *
668    * @return  background RGB color with alpha
669    */
670   @SimpleProperty(category = PropertyCategory.APPEARANCE)
671   public int BackgroundColor() {
672     return backgroundColor;
673   }
674
675   /**
676    * BackgroundColor property setter method.
677    *
678    * @param argb  background RGB color with alpha
679    */
680   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_COLOR,
681       defaultValue = Component.DEFAULT_VALUE_COLOR_WHITE)
682   @SimpleProperty
683   public void BackgroundColor(int argb) {
684     backgroundColor = argb;
685     if (argb != Component.COLOR_DEFAULT) {
686       viewLayout.getLayoutManager().setBackgroundColor(argb);
687       // Just setting the background color on the layout manager is insufficient.
688       frameLayout.setBackgroundColor(argb);
689     } else {
690       viewLayout.getLayoutManager().setBackgroundColor(Component.COLOR_WHITE);
691       // Just setting the background color on the layout manager is insufficient.
692       frameLayout.setBackgroundColor(Component.COLOR_WHITE);
693     }
694   }
```

Figura 107: Eliminando propiedad BackgroundColor de Screen

Un importante dato a tener en cuenta, es que por razones aún en investigación, las propiedades ***Scrollable***, ***AlignVertical*** y sus bloques asociados son las únicas propiedades que **no se pueden eliminar**, ya que al recompilar e intentar abrir un proyecto da un error e impide abrir dicho proyecto.

4.7.2 Generación de los ficheros del proyecto actual

Aunque no tiene nada que ver con el **modo diseño** tratado en este capítulo, será conveniente ser capaz de localizar los ficheros de cada proyecto creado con el entorno visual. Por defecto, estos ficheros son creados en una carpeta aleatoria en el momento de la compilación en APK y luego son eliminados automáticamente. Sin embargo, con el objetivo de poder visualizar dichos ficheros, se ha realizado una pequeña modificación que permite que el proyecto sea guardado en un ruta fija en el ordenador y que además dichos ficheros del proyecto no sean eliminados.

Para ello, se debe acceder a la ruta ***<RAÍZ_APPINVENTOR>\buildserver\src\com\google\appinventor\buildserver*** y localizar el fichero **ProjectBuilder.java**. En dicho fichero se debe buscar la siguiente función y eliminar su contenido (código marcado en rojo):

```
96   private static File createNewTempDir() {
97     File baseDir = new File(System.getProperty("java.io.tmpdir"));
98     String baseNamePrefix = System.currentTimeMillis() + "_" + Math.random() + "-";
99
100    final int TEMP_DIR_ATTEMPTS = 10000;
101    for (int counter = 0; counter < TEMP_DIR_ATTEMPTS; counter++) {
102      File tempDir = new File(baseDir, baseNamePrefix + counter);
103      if (tempDir.exists()) {
104        continue;
105      }
106      if (tempDir.mkdir()) {
107        return tempDir;
108      }
109    }
110    throw new IllegalStateException("Failed to create directory within "
111        + TEMP_DIR_ATTEMPTS + " attempts (tried "
112        + baseNamePrefix + "0 to " + baseNamePrefix + (TEMP_DIR_ATTEMPTS - 1) + ')');
113  }
```

Figura 108: Función generadora de ficheros del proyecto actual

A continuación se debe sustituir por el siguiente código (marcado en azul):

```
private static File createNewTempDir() {
  File baseDir = new File("/root/tmpDirProject");
  return baseDir;
}
```

Figura 109: Nueva función generadora de ficheros de proyecto

El directorio ***/root/tmpDirProject*** especificado entre comillas dobles se corresponde con la ruta absoluta donde se almacenarán los ficheros del proyecto.

5. Construcción de la parte del editor

En este capítulo se diseñará el verdadero corazón del entorno web. En esta parte se realizará la programación mediante el lenguaje visual por bloques que se está desarrollando, mediante la cual se generará posteriormente el código TeleoR equivalente.

Este editor utiliza Google Blockly, como ya se ha comentado anteriormente, por lo que la documentación proporcionada por Google será capaz de ayudar en algunas partes del proceso de modificación.

Lo primero que se debe saber es que en la parte del editor existen dos tipos diferentes de bloques:

- **Built-in**: Los bloques Built-in están separados en categorías y constituyen el conjunto de bloques predefinido de Google Blockly. El código que implementa estos bloques es diferente del visto hasta ahora, pero posteriormente se verá que su modificación y creación puede resultar muy simple.
- **Bloques asociados a componentes**: Estos bloques son los generados por los componentes que se han añadido en la parte de diseño. El código que implementa estos bloques está escrito en Java, y se encuentra en el fichero java de su componente correspondiente tal cual se vio en el **Capítulo 4**.

En cuanto a los bloques ***Built-in***, algunos de estos bloques serán necesarios y otros no, por lo que se eliminarán aquellos bloques prescindibles. También será necesario añadir una nueva categoría dentro de los bloques ***Built-in*** llamada ***Qulog***, la cual almacenara los **procesos Qulog** necesarios para la programación en TeleoR.

Los **bloques asociados a componentes** deberían haber sido correctamente definidos al modificar e implementar componentes junto con sus propiedades en el capítulo anterior, por lo que no será necesario abordarlos en este capítulo.

Los bloques Built-in están agrupados en una serie de categorías, tal y como se muestra en la siguiente figura:

Figura 110: Categorías Built-in

La categoría ***Colors*** será eliminada por completo (categoría y bloques que contiene), pero en este capítulo primero se centrará en los bloques pertenecientes al resto de categorías.

Un dato importante a tener en cuenta, es que algunos bloques tendrán la opción ***mutator***. Si un bloque lleva incorporada esta opción, aparecerá el símbolo en la esquina superior izquierda. Con esta opción, se pueden añadir atributos extras a los bloques, modificándolos de manera dinámica en el propio editor. Esta información será útil a la hora de modificar el código de los bloques, ya que los bloques que incorporan los ***mutator*** suelen tener mayor código y bloques asociados.

Como ejemplo, el bloque ***if…then*** de la categoría ***Control*** se puede modificar mediante los ***mutator*** para lograr bloques como los siguientes:

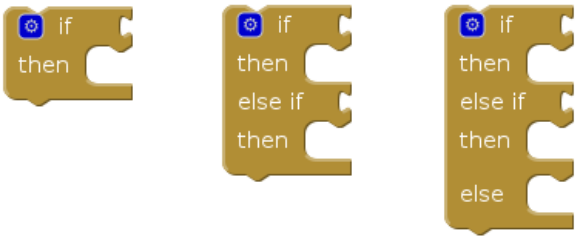


Figura 111: Ejemplo de mutator en bloque if...then

5.1 Control

En la categoría **Control** se localizan bloques que servirán para realizar bucles con determinadas condiciones, entre otros. Los bloques que interesará mantener son los correspondientes a la siguiente figura:

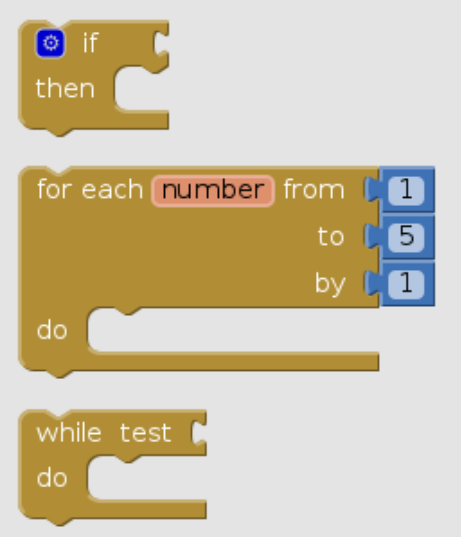


Figura 112: Bloques a mantener en categoría Control

El código fuente que genera los bloques para cada una de las categorías se encuentra en el directorio **<*RAÍZ_APPINVENTOR*>*blocklyeditor**src**blocks***. Concretamente en este caso se debe abrir el fichero ***control.js***, en el cuál se encuentra el código que genera los bloques de ***Control***.

Por ejemplo, el siguiente sería el código correspondiente al bloque ***while test… do***

```
480  Blockly.Blocks['controls_while'] = {
481    // While condition.
482    category: 'Control',
483    helpUrl: Blockly.Msg.LANG_CONTROLS_WHILE_HELPURL,
484    init: function () {
485      this.setColour(Blockly.CONTROL_CATEGORY_HUE);
486      this.appendValueInput('TEST')
487          .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
488          .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_TITLE)
489          .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_TEST)
490          .setAlign(Blockly.ALIGN_RIGHT);
491      this.appendStatementInput('DO')
492          .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_DO)
493          .setAlign(Blockly.ALIGN_RIGHT);
494      this.setPreviousStatement(true);
495      this.setNextStatement(true);
496      this.setTooltip(Blockly.Msg.LANG_CONTROLS_WHILE_TOOLTIP);
497    },
498    typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_WHILE_TITLE}]
499  };
```

Figura 113: Código de bloque while test...do

En la figura anterior se observar un fragmento marcado en rojo. Este fragmento se corresponde con el identificador del bloque, siendo en este caso '**controls_while'** (importante que todos los identificadores vayan entre comillas como en el ejemplo). Mediante estos identificadores se localizará en este capítulo el código correspondiente a los bloques que se desean mantener y eliminar.

En este caso, se deben **mantener** los bloques los identificadores mostrados en la siguiente tabla:

Bloque	Identificador
if…	controls_if
if…if…	controls_if_if
if…elseif…	controls_if_elseif

if…else…	controls_if_else
for…each…	Controls_forRange
while test…do…	controls_while

También se deben buscar los ficheros que indican el código generado por cada bloque en función de los parámetros que se le añadan en el editor. Estos ficheros se encuentran en el directorio ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail***. En este caso, el fichero buscado es ***control.js***.

A continuación se muestra como ejemplo el código que generará el bloque ***while test… do…***

```
115  Blockly.Yail['controls_while'] = function() {
116    // While condition.
117    var test = Blockly.Yail.valueToCode(this, 'TEST', Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
118    var toDo = Blockly.Yail.statementToCode(this, 'DO') || Blockly.Yail.YAIL_FALSE;
119    var code = Blockly.Yail.YAIL_WHILE + test + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN + toDo +
120    Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_CLOSE_COMBINATION;
121    return code;
122  };
```

Figura 114: Código generado por bloque while test…do

En este caso, se deben **mantener** los bloques con los siguientes identificadores:

Bloque	Identificador	¿Mutator?
if…	controls_if	
for…each…	controls_forRange	
while test…do…	controls_while	

5.2 Logic

En la categoría **Logic** se encuentran los bloques necesarios para realizar comparaciones booleanas y operaciones lógicas elementales (NOT, AND, OR,…). Los bloques que interesará conservar son los correspondientes a la siguiente figura:

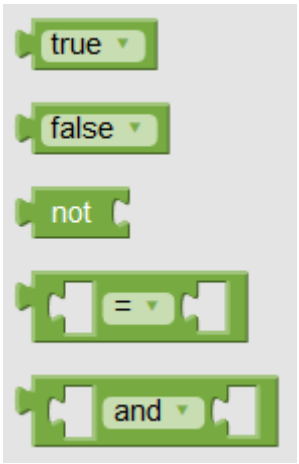


Figura 115: Bloques a mantener en categoría Logic

El único bloque que en este caso no será necesario es el correspondiente a la operación lógica OR, ya que esta operación no está contemplada en el lenguaje TeleoR. A parte de

eliminar el bloque OR, se deberá eliminar la opción OR en la lista desplegable del bloque AND.

Para ello, hay que acceder al fichero ***logic.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** y eliminar el código marcado en rojo en las siguientes capturas:

```
182     }, {
183       translatedName: Blockly.Msg.LANG_LOGIC_OPERATION_OR,
184       dropDown: {
185         titleName: 'OP',
186         value: 'OR'
187       }
188     }]
189   };
190
191   Blockly.Blocks.logic_operation.OPERATORS = function () {
192     return [
193       [Blockly.Msg.LANG_LOGIC_OPERATION_AND, 'AND'],
194       [Blockly.Msg.LANG_LOGIC_OPERATION_OR, 'OR']
195     ]
196   };
197
198   Blockly.Blocks.logic_operation.HELPURLS = function () {
199     return {
200       AND: Blockly.Msg.LANG_LOGIC_OPERATION_HELPURL_AND,
201       OR: Blockly.Msg.LANG_LOGIC_OPERATION_HELPURL_OR
202     }
203   };
204   Blockly.Blocks.logic_operation.TOOLTIPS = function () {
205     return {
206       AND: Blockly.Msg.LANG_LOGIC_OPERATION_TOOLTIP_AND,
207       OR: Blockly.Msg.LANG_LOGIC_OPERATION_TOOLTIP_OR
208     }
209   };
```

Figura 116: Eliminando opción OR en lista desplegable de bloque AND

```
210
211 Blockly.Blocks['logic_or'] = {
212   // Logical operations: 'and', 'or'.
213   category: 'Logic',
214   init: function () {
215     this.setColour(Blockly.LOGIC_CATEGORY_HUE);
216     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
217     this.appendValueInput('A')
218         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT));
219     this.appendValueInput('B')
220         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
221         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.logic_operation.OPERATORS), 'OP');
222     this.setFieldValue('OR', 'OP');
223     this.setInputsInline(true);
224     // Assign 'this' to a variable for use in the tooltip closure below.
225     var thisBlock = this;
226     this.setTooltip(function () {
227       var op = thisBlock.getFieldValue('OP');
228       return Blockly.Blocks.logic_operation.TOOLTIPS[op];
229     });
230   },
231   helpUrl: function () {
232     var op = this.getFieldValue('OP');
233     return Blockly.Blocks.logic_operation.HELPURLS[op];
234   }
235 };
```

Figura 117: Eliminando código del bloque OR

Por último, acceder al fichero ***logic.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail*** y eliminar el código marcado en rojo en la siguiente captura:

```
58  Blockly.Yail.logic_operation.OPERATORS = {
59    AND : [ 'and-delayed', Blockly.Yail.ORDER_NONE ],
60    OR : [ 'or-delayed', Blockly.Yail.ORDER_NONE ]
61  };
62
63  Blockly.Yail['logic_or'] = function() {
64    return Blockly.Yail.logic_operation.call(this);
65  }
66
```

Figura 118: Eliminando código generado por bloque OR y opción OR

5.3 Math

En la categoría **Math** están localizados los bloques necesarios para realizar operaciones matemáticas elementales (sumas, restas, divisiones, comparaciones,…). Los bloques que interesará conservar son los correspondientes a la siguiente figura:

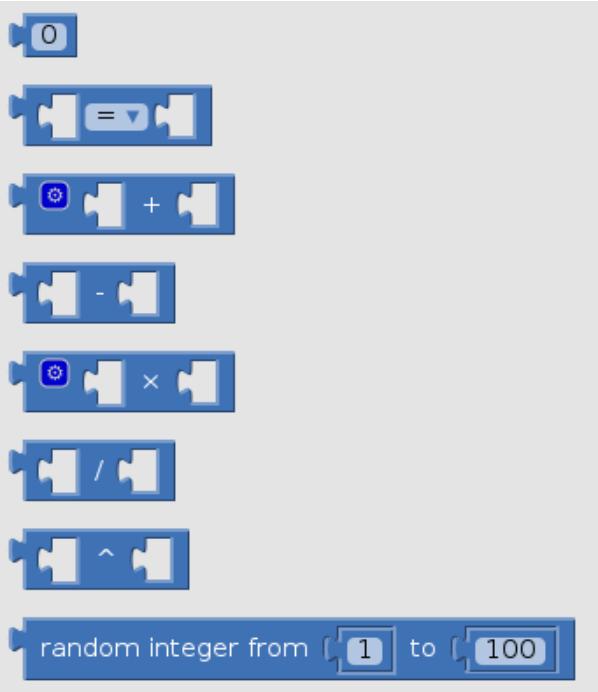


Figura 119: Bloques a mantener en categoría Math

En este caso, ha sido necesario eliminar gran parte de los bloques que contiene. Para ello, hay que acceder al fichero ***math.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** y eliminar los bloques con los identificadores mostrados en la siguiente tabla:

Bloque	Identificador
random fraction	math_random_float
random set seed to…	math_random_set_seed
on list	math_on_list
square root	math_single
abs	math_abs

neg	math_neg
round	math_round
ceiling	math_ceiling
floor	math_floor
divide	math_divide
sin	math_trig
cos	math_cos
tan	math_tan
atan2	math_atan2
convert angles	math_convert_angles
format as decimal	math_format_as_decimal
is number	math_is_a_number

Para mayor simplicidad, bastará con eliminar todo el código **a partir de la línea 340** de dicho fichero.

Por último, hay que acceder al fichero ***math.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail*** y eliminar el código situado **a partir de la línea 128** exceptuando el siguiente fragmento:

```
182  Blockly.Yail['math_random_int'] = function() {
183    // Random integer between [X] and [Y].
184    var argument0 = Blockly.Yail.valueToCode(this, 'FROM',
185      Blockly.Yail.ORDER_NONE) || 0;
186    var argument1 = Blockly.Yail.valueToCode(this, 'TO',
187      Blockly.Yail.ORDER_NONE) || 0;
188    var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-integer"
189        + Blockly.Yail.YAIL_SPACER;
190    code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
191        + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
192        + argument0 + Blockly.Yail.YAIL_SPACER + argument1
193        + Blockly.Yail.YAIL_CLOSE_COMBINATION;
194    code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
195        + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
196        + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
197    code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random integer"
198        + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
199    return [ code, Blockly.Yail.ORDER_ATOMIC ];
200  };
```

Figura 120: Fragmento de código a mantener en categoría Math

En el caso de Math, también será necesario añadir los siguientes bloques:

Figura 121: Nuevos bloques de categoría Math

Para crear estos bloques se ha empleado el editor de bloques online de Google Blockly. Dicho editor es accesible desde el siguiente enlace:

https://blockly-demo.appspot.com/static/demos/blocklyfactory/index.html

Como ejemplo, el siguiente enlace muestra el bloque ***now*** en el editor de bloques de Google:

https://blockly-demo.appspot.com/static/demos/blockfactory/index.html#8anzbv

El siguiente código debe ser introducido en el fichero ***math.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** para añadir los tres bloques previamente mencionados:

```
Blockly.Blocks['now_time'] = {
  category: 'Math',
  init: function() {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setHelpUrl('http://www.example.com/');
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown([["now", "NOW_T"], ["exec_time", "EXEC_T"], ["start_time", "START_T"]]), "FUNC");
    this.setOutput(true, "Number");
    this.setTooltip('');
  }
};

Blockly.Blocks['exec_time'] = {
  category: 'Math',
  init: function() {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setHelpUrl('http://www.example.com/');
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown([["exec_time", "EXEC_T"], ["now", "NOW_T"], ["start_time", "START_T"]]), "FUNC");
    this.setOutput(true, "Number");
    this.setTooltip('');
  }
};

Blockly.Blocks['start_time'] = {
  category: 'Math',
  init: function() {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setHelpUrl('http://www.example.com/');
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown([["start_time", "START_T"], ["now", "NOW_T"], ["exec_time", "EXEC_T"]]), "FUNC");
    this.setOutput(true, "Number");
    this.setTooltip('');
  }
};
```

Figura 122: Código de nuevos bloques de Math

Por último, añadiendo el siguiente código en el fichero ***math.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail***, será implementada la lógica asociada a los tres bloques:

```
Blockly.Yail['now_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};

Blockly.Yail['exec_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};

Blockly.Yail['start_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};
```

Figura 123: Código generado por nuevos bloques de Math

5.4 Text

En la categoría **Text** se ubican diversos bloques para manipular y comparar cadenas de caracteres. En este caso, sólo interesará conservar un único bloque, el cual permitirá crear una cadena de caracteres:

Figura 124: Bloque a mantener en categoría Text

Se debe acceder al fichero ***text.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** y eliminar todo el código excepto el marcado en la siguiente figura:

```
'use strict';

goog.provide('Blockly.Blocks.text');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['text'] = {
  // Text value.
  category: 'Text',
  helpUrl: Blockly.Msg.LANG_TEXT_TEXT_HELPURL,
  init: function () {
    this.setColour(Blockly.TEXT_CATEGORY_HUE);
    this.appendDummyInput().appendField(Blockly.Msg.LANG_TEXT_TEXT_LEFT_QUOTE).appendField(
        new Blockly.FieldTextBlockInput(''),
        'TEXT').appendField(Blockly.Msg.LANG_TEXT_TEXT_RIGHT_QUOTE);
    this.setOutput(true, [Blockly.Blocks.text.connectionCheck]);
    this.setTooltip(Blockly.Msg.LANG_TEXT_TEXT_TOOLTIP);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_CATEGORY_TEXT}]
};
```

Figura 125: Código de bloque a mantener en Text

De igual manera, abrimos el fichero ***text.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail*** y eliminamos todo el código exceptuando el de la siguiente figura:

```
'use strict';

goog.provide('Blockly.Yail.text');

Blockly.Yail['text'] = function() {
  // Text value.
  var code = Blockly.Yail.quote_(this.getFieldValue('TEXT'));
  return [code, Blockly.Yail.ORDER_ATOMIC];
};
```

Figura 126: Código generado por bloque a mantener en Text

5.5 Lists

En la categoría Lists encontraremos los bloques necesarios para crear listas, las cuales podrán contener diferentes tipos de datos. También encontraremos los bloques que nos

servirán para crear tipos de datos específicos utilizados en los TRs. Los bloques que nos interesará conservar son los correspondientes a la siguiente figura:

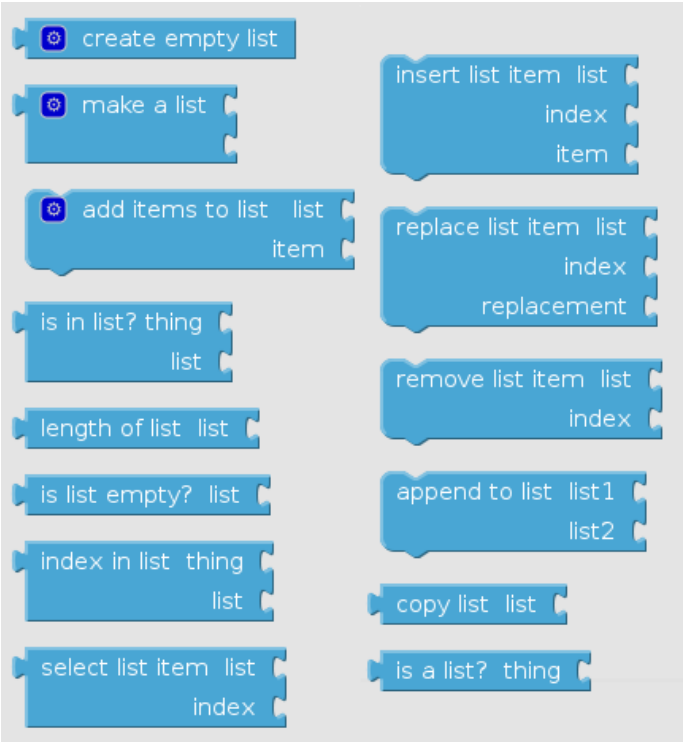


Figura 127: Bloques a mantener en categoría Lists

Los bloques que vamos a eliminar poseen los identificadores mostrados en la siguiente tabla:

Bloque	Identificador
pick a random item	list_pick_random_item
list to csv row	lists_to_csv_row
list to csv table	lists_to_csv_table
list from csv row	lists_from_csv_row
list from csv table	lists_from_csv_table
lookup in pairs	lists_lookup_in_pairs

El código correspondiente al primero de los bloques a eliminar se muestra en la siguiente figura:

```
182  Blockly.Blocks['lists_pick_random_item'] = {
183    // Length of list.
184    category : 'Lists',
185    helpUrl : Blockly.Msg.LANG_LISTS_PICK_RANDOM_ITEM_HELPURL,
186    init : function() {
187      this.setColour(Blockly.LIST_CATEGORY_HUE);
188      this.setOutput(true, null);
189      this.appendValueInput('LIST')
190        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks.Utilities.INPUT))
191        .appendField(Blockly.Msg.LANG_LISTS_PICK_RANDOM_TITLE_PICK_RANDOM)
192        .appendField(Blockly.Msg.LANG_LISTS_PICK_RANDOM_ITEM_INPUT_LIST);
193      this.setTooltip(Blockly.Msg.LANG_LISTS_PICK_RANDOM_TOOLTIP);
194    },
195    typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_PICK_RANDOM_TITLE_PICK_RANDOM }]
196  };
```

Figura 128: Eliminando código de bloque pick random item

Tras eliminar el bloque anterior, sólo será necesario eliminar todo el código situado a partir de la línea 335.

5.6 Variables

En la categoría **Variables** se encuentran los bloques necesarios para crear variables con valores determinados (numéricos, cadenas de texto, etc.), para poder usarlas en la lógica de la aplicación. En este caso, se necesitarán todos los bloques contenidos en esta categoría, los cuáles son los siguientes:

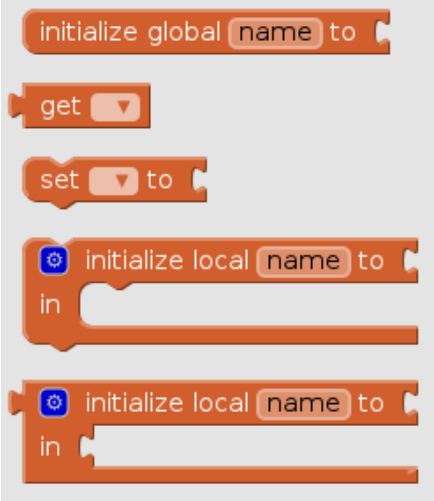


Figura 129: Bloques a mantener en categoría Variables

Se deberán crear dos nuevos bloques, los cuales se muestran en la siguiente figura:

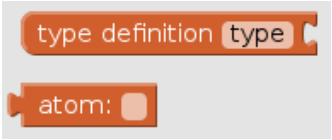


Figura 130: Nuevos bloques de categoría Variables

El bloque ***type definition*** nos permitirá definir tipos personalizados en lenguaje TeleoR, y el bloque ***atom*** nos permitirá definir un tipo de dato propio de TeleoR pero que no está contemplado por defecto en Google Blockly.

El siguiente código debe ser introducido en el fichero ***lexical-variables.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** para añadir los dos bloques previamente mencionados:

```
695   Blockly.Blocks['type_def'] = {
696     category: 'Variables',
697     init: function() {
698       this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
699       this.appendValueInput("DEF")
700           .setCheck("Array")
701           .appendField("type definition")
702           .appendField(new Blockly.FieldTextInput("type"), "TYPE");
703       this.setTooltip('');
704     }
705   };
706
707   Blockly.Blocks['atom_def'] = {
708     category: 'Variables',
709     init: function() {
710       this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
711       this.appendDummyInput()
712           .appendField("atom:")
713           .appendField(new Blockly.FieldTextInput(""), "VALUE");
714       this.setOutput(true, "Array");
715       this.setTooltip('');
716     }
717   };
```

Figura 131: Código de nuevos bloques de Variables

Por último, añadiendo el siguiente código en el fichero ***variables.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail***, se implementará la lógica asociada a los dos bloques:

```
141   Blockly.Yail['type_def'] = function(block) {
142     var value_def = Blockly.Yail.valueToCode(block, 'DEF', Blockly.Yail.ORDER_ATOMIC);
143     var text_type = block.getFieldValue('TYPE');
144     // TODO: Assemble Yail into code variable.
145     var code = '...';
146     return code;
147   };
148
149   Blockly.Yail['atom_def'] = function(block) {
150     var text_value = block.getFieldValue('VALUE');
151     // TODO: Assemble Yail into code variable.
152     var code = '...';
153     // TODO: Change ORDER_NONE to the correct strength.
154     return [code, Blockly.YAIL.ORDER_NONE];
155   };
```

Figura 132: Código generado por nuevos bloques de Variables

5.7 Procedures

En la categoría **Procedures** se encuentran los bloques necesarios para declarar procesos con un determinado objetivo. Utilizaremos estos bloques parar representar ***Goals*** y ***Subgoals*** en lenguaje TeleoR. En este caso, se mantendrán todos los bloques contenidos en esta categoría, los cuáles son los siguientes:

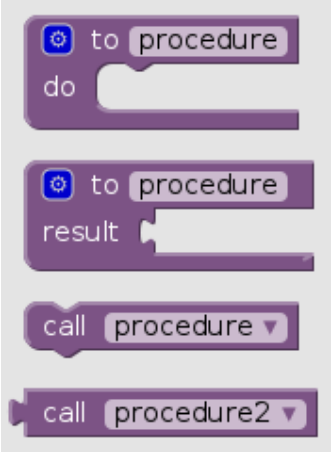


Figura 133: Bloques a mantener en categoría Procedures

Sin embargo, será necesario realizar los siguientes cambios:

- Modificar los dos últimos bloques ***call*** para añadir tres parámetros por defecto.
- Añadir un nuevo bloque llamado ***nil***, el cuál representará una acción nula.

Lo primero es acceder el fichero ***procedures.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** y realizar los siguientes cambios:

- Añadir el siguiente código en el bloque con identificador ***procedures_callnoreturn***:

```
725  Blockly.Blocks['procedures_callnoreturn'] = {
726    // Call a procedure with no return value.
727    category: 'Procedures',  // Procedures are handled specially.
728    helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_HELPURL,
729    init: function() {
730      this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
731      this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(false);};
732
733      this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldProcedure.onChange);
734      this.procDropDown.block = this;
735      this.appendDummyInput()
736          .appendField(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_CALL)
737          .appendField(this.procDropDown,"PROCNAME");
738      this.setPreviousStatement(true);
739      this.setNextStatement(true);
740      this.appendValueInput("wait_time")
741          .setCheck("Number")
742          .setAlign(Blockly.ALIGN_RIGHT)
743          .appendField("wait_time");
744      this.appendValueInput("wait_trials")
745          .setCheck("Number")
746          .setAlign(Blockly.ALIGN_RIGHT)
747          .appendField("wait_trials");
748      this.appendValueInput("exec_time")
749          .setCheck("Number")
750          .setAlign(Blockly.ALIGN_RIGHT)
751          .appendField("exec_time");
752      this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_TOOLTIP);
753      this.arguments_ = [];
```

Figura 134: Añadiendo parámetros a bloque procedures_callnoreturn

- Añadir el siguiente código en el bloque con identificador ***procedures_callreturn***:

```
929   Blockly.Blocks['procedures_callreturn'] = {
930     // Call a procedure with a return value.
931     category: 'Procedures',  // Procedures are handled specially.
932     helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLRETURN_HELPURL,
933     init: function() {
934       this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
935       this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(true);};
936
937       this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldProcedure.onChange);
938       this.procDropDown.block = this;
939       this.appendDummyInput()
940           .appendField(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_CALL)
941           .appendField(this.procDropDown,"PROCNAME");
942       this.appendValueInput("wait_time")
943           .setCheck("Number")
944           .setAlign(Blockly.ALIGN_RIGHT)
945           .appendField("wait_time");
946       this.appendValueInput("wait_trials")
947           .setCheck("Number")
948           .setAlign(Blockly.ALIGN_RIGHT)
949           .appendField("wait_trials");
950       this.appendValueInput("exec_time")
951           .setCheck("Number")
952           .setAlign(Blockly.ALIGN_RIGHT)
953           .appendField("exec_time");
954       this.setOutput(true, null);
955       this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_TOOLTIP);
956       this.arguments_ = [];
```

Figura 135: Añadiendo parámetros a bloque procedures_callreturn

- Añadir el siguiente código al final del fichero para implementar el bloque ***nil***:

```
978   Blockly.Blocks['nil_proc'] = {
979     category: 'Procedures',
980     init: function() {
981       this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
982       this.setHelpUrl('http://www.example.com/');
983       this.appendDummyInput()
984           .appendField("nil");
985       this.setPreviousStatement(true);
986       this.setNextStatement(true);
987       this.setTooltip('');
988     }
989   };
```

Figura 136: Código de nuevo bloque de Procedures

Por último, acceder al fichero ***procedures.js*** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail*** y añadir el siguiente código marcado en rojo:

```
64  Blockly.Yail['procedures_callnoreturn'] = function() {
65    var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
66    var wait_time = Blockly.Yail.valueToCode(block, 'wait_time', Blockly.Yail.ORDER_ATOMIC);
67    var wait_trials = Blockly.Yail.valueToCode(block, 'wait_trials', Blockly.Yail.ORDER_ATOMIC);
68    var exec_time = Blockly.Yail.valueToCode(block, 'exec_time', Blockly.Yail.ORDER_ATOMIC);
69    var argCode = [];
70    for ( var x = 0;this.getInput("ARG" + x); x++) {
71      argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
72    }
73    var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + procName
74        + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.join(' ')
75        + Blockly.Yail.YAIL_CLOSE_COMBINATION;
76    return code;
77  };
78
79  // Generator code for procedure call with return
80  Blockly.Yail['procedures_callreturn'] = function() {
81    var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
82    var wait_time = Blockly.Yail.valueToCode(block, 'wait_time', Blockly.Yail.ORDER_ATOMIC);
83    var wait_trials = Blockly.Yail.valueToCode(block, 'wait_trials', Blockly.Yail.ORDER_ATOMIC);
84    var exec_time = Blockly.Yail.valueToCode(block, 'exec_time', Blockly.Yail.ORDER_ATOMIC);
85    var argCode = [];
86    for ( var x = 0; this.getInput("ARG" + x); x++) {
87      argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
88    }
89    var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + procName
90        + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.join(' ')
91        + Blockly.Yail.YAIL_CLOSE_COMBINATION;
92    return [ code, Blockly.Yail.ORDER_ATOMIC ];
93  };
94
95  Blockly.Yail['nil_proc'] = function(block) {
96    // TODO: Assemble Yail into code variable.
97    var code = '...';
98    return code;
99  };
```

Figura 137: Nuevo código generado por bloques en Procedures

Tras las modificaciones antes mencionadas, los bloques de la categoría **Procedures** tendrán el siguiente aspecto:

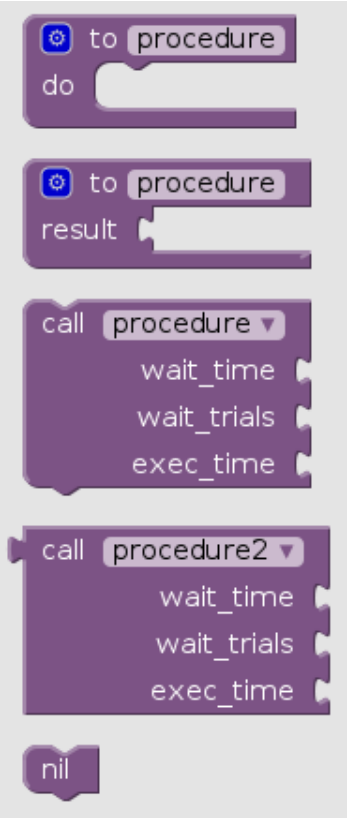


Figura 138: Bloques modificados de categoría Procedures

5.8 Colors

Tal y como se comentó anteriormente, la categoría **Colors** no será de utilidad en el editor de bloques, por lo que se procederá a su eliminación.

Lo primero que se debe hacer es abrir el fichero **BlockSelectorBox.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client\boxes*** y eliminar el código marcado en rojo en las siguientes capturas:

```
73    private static final String BUILTIN_DRAWER_NAMES[] = { "Control", "Logic", "Math", "Text",
74        "Lists", "Colors", "Variables", "Procedures" };
```

Figura 139: Eliminando categoría Colors 1

```
108   private static void initBundledImages() {
109     bundledImages.put("Control", images.control());
110     bundledImages.put("Logic", images.logic());
111     bundledImages.put("Math", images.math());
112     bundledImages.put("Text", images.text());
113     bundledImages.put("Lists", images.lists());
114     bundledImages.put("Colors", images.colors());
115     bundledImages.put("Variables", images.variables());
116     bundledImages.put("Procedures", images.procedures());
117   }
```

Figura 140: Eliminando categoría Colors 2

```
156   private String getBuiltinDrawerNames(String drawerName) {
157     String name;
158
159     OdeLog.wlog("getBuiltinDrawerNames: drawerName = " + drawerName);
160
161     if (drawerName.equals("Control")) {
162       name = MESSAGES.builtinControlLabel();
163     } else if (drawerName.equals("Logic")) {
164       name = MESSAGES.builtinLogicLabel();
165     } else if (drawerName.equals("Math")) {
166       name = MESSAGES.builtinMathLabel();
167     } else if (drawerName.equals("Text")) {
168       name = MESSAGES.builtinTextLabel();
169     } else if (drawerName.equals("Lists")) {
170       name = MESSAGES.builtinListsLabel();
171     } else if (drawerName.equals("Colors")) {
172       name = MESSAGES.builtinColorsLabel();
173     } else if (drawerName.equals("Variables")) {
174       name = MESSAGES.builtinVariablesLabel();
175     } else if (drawerName.equals("Procedures")) {
176       name = MESSAGES.builtinProceduresLabel();
177     } else {
178       name = drawerName;
179     }
180     return name;
181   }
```

Figura 141: Eliminando categoría Colors 3

A continuación, acceder al fichero **OdeMessages.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y eliminar el código mostrado en la siguiente captura:

```
748   @DefaultMessage("Colors")
749   @Description("Label on built-in-Colors-blocks branch of block selector tree")
750   String builtinColorsLabel();
```

Figura 142: Eliminando etiqueta de categoría Colors

Para eliminar la imagen asociada a la categoría se debe abrir el fichero **Images.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y eliminar el código mostrado a continuación:

```
452   /**
453    * Built in drawer item: colors
454    */
455   @Source("com/google/appinventor/images/colors.png")
456   ImageResource colors();
```

Figura 143: Eliminando imagen de categoría Colors

La imagen *colors.png* no será eliminada, ya que más tarde será renombrada y reutilizada para la nueva categoría que se creará en el siguiente apartado.

Para eliminar el color asociado a la categoría, se debe abrir el fichero **blockColors.js** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src*** y eliminar la siguiente línea:

```
12   'use strict';
13
14   Blockly.HSV_SATURATION = 0.7;
15   Blockly.CONTROL_CATEGORY_HUE = [177, 142, 53];
16   Blockly.LOGIC_CATEGORY_HUE = [119, 171, 65];
17   Blockly.MATH_CATEGORY_HUE = [63, 113,181];
18   Blockly.TEXT_CATEGORY_HUE = [179, 45, 94];
19   Blockly.LIST_CATEGORY_HUE = [73,166,212];
20   Blockly.COLOR_CATEGORY_HUE = [125,125,125];
21   Blockly.VARIABLE_CATEGORY_HUE = [208,95,45];
22   Blockly.PROCEDURE_CATEGORY_HUE = [124,83,133];
```

Figura 144: Eliminando color asociado a la categoría Colors

Por último, se debe acceder al fichero **ploverConfig.js** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor*** y eliminar las líneas marcadas en rojo:

```
88      //blocks files
89      './src/blocks/control.js',
90      './src/blocks/logic.js',
91      './src/blocks/text.js',
92      './src/blocks/lists.js',
93      './src/blocks/math.js',
94      './src/blocks/utilities.js',
95      './src/blocks/procedures.js',
96      './src/blocks/lexical-variables.js',
97      './src/blocks/colors.js',
98      './src/blocks/components.js',
99
100     //generator files
101     "./src/generators/yail.js",
102     "./src/generators/yail/componentblock.js",
103     "./src/generators/yail/lists.js",
104     "./src/generators/yail/math.js",
105     "./src/generators/yail/control.js",
106     "./src/generators/yail/logic.js",
107     "./src/generators/yail/text.js",
108     "./src/generators/yail/colors.js",
109     "./src/generators/yail/variables.js",
110     "./src/generators/yail/procedures.js",
```

Figura 145: Eliminando colors.js de la lista de ficheros a compilar

Ahora la lista de categorías Built-in lucirá de la siguiente manera:

Figura 146: Categorías Built-in tras eliminar Colors

5.9 Qulog

En este apartado se creará la categoría **Qulog**, en la cual se añadirán los llamados ***Qulog Procedures***. Los Qulog Procedures son procesos específicos del lenguaje TeleoR, los cuales siempre son llamados tras finalizar la llamada a un proceso de la categoría **Procedures**. La categoría **Qulog** contendrá un total de dos bloques representando los dos Qulog Procedures disponibles en TeleoR.

Para empezar, se deben modificar las categorías Built-in para añadir la nueva categoría **Qulog**, a la cual se le asignará el color de la categoría **Colors** que fue eliminada previamente. Los ficheros a modificar serán los mismos que se modificaron para eliminar la categoría **Colors** en el apartado anterior.

Para comenzar, hay que acceder al fichero **BlockSelectorBox.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client\boxes*** y añadir el código marcado en rojo en las siguientes capturas:

```
73    private static final String BUILTIN_DRAWER_NAMES[] = { "Control", "Logic", "Math", "Text",
74        "Lists", "Variables", "Procedures", "Qulog" };
```

Figura 147: Añadiendo categoría Qulog 1

```
108   private static void initBundledImages() {
109     bundledImages.put("Control", images.control());
110     bundledImages.put("Logic", images.logic());
111     bundledImages.put("Math", images.math());
112     bundledImages.put("Text", images.text());
113     bundledImages.put("Lists", images.lists());
114     bundledImages.put("Variables", images.variables());
115     bundledImages.put("Procedures", images.procedures());
116     bundledImages.put("Qulog", images.qulog());
117   }
```

Figura 148: Añadiendo categoría Qulog 2

```
157   private String getBuiltinDrawerNames(String drawerName) {
158     String name;
159
160     OdeLog.wlog("getBuiltinDrawerNames: drawerName = " + drawerName);
161
162     if (drawerName.equals("Control")) {
163       name = MESSAGES.builtinControlLabel();
164     } else if (drawerName.equals("Logic")) {
165       name = MESSAGES.builtinLogicLabel();
166     } else if (drawerName.equals("Math")) {
167       name = MESSAGES.builtinMathLabel();
168     } else if (drawerName.equals("Text")) {
169       name = MESSAGES.builtinTextLabel();
170     } else if (drawerName.equals("Lists")) {
171       name = MESSAGES.builtinListsLabel();
172     } else if (drawerName.equals("Variables")) {
173       name = MESSAGES.builtinVariablesLabel();
174     } else if (drawerName.equals("Procedures")) {
175       name = MESSAGES.builtinProceduresLabel();
176     } else if (drawerName.equals("Qulog")) {
177       name = MESSAGES.builtinQulogLabel();
178     } else {
179       name = drawerName;
180     }
181     return name;
182   }
```

Figura 149: Añadiendo categoría Qulog 3

A continuación, acceder al fichero **OdeMessages.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y añadir el código mostrado en la siguiente captura:

```
5371    // QULOG PROCEDURES
5372    @DefaultMessage("Qulog")
5373    @Description("Label on built-in-Qulog-Procedures-blocks branch of block selector tree")
5374    String builtinQulogLabel();
```

Figura 150: Añadiendo etiqueta de categoría Qulog

Todo el código de este fichero está englobado en el método ***public interface OdeMessages***, por lo que el código anterior debe ser añadido dentro de dicho método y por lo tanto antes de la llave "}" situada al final del mismo.

Para asociar una imagen a la nueva categoría se debe abrir el fichero **Images.java** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\client*** y añadir el código mostrado en la siguiente captura:

```
532     /**
533      * Built in drawer item: qulog
534      */
535     @Source("com/google/appinventor/images/qulog.png")
536     ImageResource qulog();
```

Figura 151: Añadiendo imagen de categoría Qulog

De manera similar al fichero **OdeMessages.java**, todo el código de este fichero está englobado en el método ***public interface Images***, por lo que el código anterior debe ser añadido dentro de dicho método.

En la captura anterior se puede apreciar que se ha seleccionado como imagen el fichero ***qulog.png*** situado en la ruta ***<RAÍZ_APPINVENTOR>\appengine\src\com\google\appinventor\images***. Esta imagen no existe actualmente, por lo que se utilizará la imagen ***colors.png*** que quedó en desuso en el apartado anterior renombrándola a ***qulog.png***.

Para añadir el color asociado a la categoría, de debe abrir el fichero **blockColors.js** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src*** y añadir la línea marcada en rojo:

```
12  'use strict';
13
14  Blockly.HSV_SATURATION = 0.7;
15  Blockly.CONTROL_CATEGORY_HUE = [177, 142, 53];
16  Blockly.LOGIC_CATEGORY_HUE = [119, 171, 65];
17  Blockly.MATH_CATEGORY_HUE = [63, 113,181];
18  Blockly.TEXT_CATEGORY_HUE = [179, 45, 94];
19  Blockly.LIST_CATEGORY_HUE = [73,166,212];
20  Blockly.VARIABLE_CATEGORY_HUE = [208,95,45];
21  Blockly.PROCEDURE_CATEGORY_HUE = [124,83,133];
22  Blockly.QULOG_CATEGORY_HUE = [125,125,125];
```

Figura 152: Añadiendo color asociado a la categoría Qulog

De esta manera, estamos asociando el color previamente utilizado por la categoría **Colors** a la nueva categoría **Qulog**.

Por último, se debe acceder al fichero **ploverConfig.js** situado en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor*** y añadir las líneas marcadas en rojo:

```
88      //blocks files
89      './src/blocks/control.js',
90      './src/blocks/logic.js',
91      './src/blocks/text.js',
92      './src/blocks/lists.js',
93      './src/blocks/math.js',
94      './src/blocks/utilities.js',
95      './src/blocks/procedures.js',
96      './src/blocks/lexical-variables.js',
97      './src/blocks/components.js',
98      './src/blocks/qulog.js',
99
100     //generator files
101     "./src/generators/yail.js",
102     "./src/generators/yail/componentblock.js",
103     "./src/generators/yail/lists.js",
104     "./src/generators/yail/math.js",
105     "./src/generators/yail/control.js",
106     "./src/generators/yail/logic.js",
107     "./src/generators/yail/text.js",
108     "./src/generators/yail/variables.js",
109     "./src/generators/yail/procedures.js",
110     "./src/generators/yail/qulog.js",
```

Figura 153: Añadiendo qulog.js a la lista de ficheros a compilar

Tras realizar todos los cambios indicados, las categorías Built-in se mostrarán como en la siguiente captura:

Figura 154: Categorías Built-in tras añadir Qulog

Una vez implementada la categoría, se han de añadir los bloques correspondientes a dicha categoría. Los bloques que se añadirán a continuación son los siguientes:

Figura 155: Bloques de nueva categoría Qulog

Se debe crear un nuevo fichero llamado ***qulog.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\blocks*** y añadir el siguiente código:

```
'use strict';

goog.provide('Blockly.Blocks.qulog');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['remember_call'] = {
  category: 'Qulog',
  init: function() {
    this.setColour(Blockly.QULOG_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField("call")
        .appendField(new Blockly.FieldDropdown([["remember", "REMEMBER"], ["forget", "FORGET"]]), "DD");
    this.appendValueInput("belief")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("belief");
    this.appendValueInput("time")
        .setCheck("Number")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("time");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setInputsInline(false);
    this.setTooltip('');
  }
};

Blockly.Blocks['forget_call'] = {
  category: 'Qulog',
  init: function() {
    this.setColour(Blockly.QULOG_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField("call")
        .appendField(new Blockly.FieldDropdown([["forget", "FORGET"], ["remember", "REMEMBER"]]), "DD");
    this.appendValueInput("belief")
        .setCheck("String")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("belief");
    this.appendValueInput("time")
        .setCheck("Number")
        .setAlign(Blockly.ALIGN_RIGHT)
        .appendField("time");
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip('');
  }
};
```

Figura 156: Código de bloques de nueva categoría Qulog

Por último, debemos crear un nuevo fichero llamado ***qulog.js*** en la ruta ***<RAÍZ_APPINVENTOR>\blocklyeditor\src\generators\yail*** y añadir el siguiente código:

```
'use strict';

goog.provide('Blockly.Yail.qulog');

Blockly.Yail['remember_call'] = function(block) {
  var dropdown_dd = block.getFieldValue('DD');
  var value_belief = Blockly.Yail.valueToCode(block, 'belief', Blockly.Yail.ORDER_ATOMIC);
  var value_time = Blockly.Yail.valueToCode(block, 'time', Blockly.Yail.ORDER_ATOMIC);
  // TODO: Assemble Yail into code variable.
  var code = '...';
  return code;
};

Blockly.Yail['forget_call'] = function(block) {
  var dropdown_dd = block.getFieldValue('DD');
  var value_belief = Blockly.Yail.valueToCode(block, 'belief', Blockly.Yail.ORDER_ATOMIC);
  var value_time = Blockly.Yail.valueToCode(block, 'time', Blockly.Yail.ORDER_ATOMIC);
  // TODO: Assemble Yail into code variable.
  var code = '...';
  return code;
};
```

Figura 157: Código generado por bloques de nueva categoría Qulog

6. Conclusiones y trabajos futuros

Tras realizar paso a paso las instrucciones indicadas en cada uno de los capítulos, se obtiene un entorno web y un lenguaje visual muy sencillos de manejar para cualquier programador, los cuales se acercan bastante a la idea inicial que se pretendía desarrollar.

El proceso ha sido tedioso debido a la extensión del código fuente y el tiempo requerido para compilarlo y visualizar los cambios realizados, pero ha merecido la pena el esfuerzo.

El entorno de programación web **TeleoR Factory** presenta gran potencial, aunque quedan pendientes para trabajo futuro una serie de modificaciones esenciales para su funcionamiento, las cuales no han podido ser abordadas en este proyecto tales como:

- Generación de código TeleoR mediante el fichero XML generado por el proyecto.
- Eliminación de funcionalidad no requerida por TeleoR Factory.
- Limpieza y eliminación de código innecesario.
- Opción de selección entre varios modelos de dron.
- Inclusión de nuevas opciones necesarias omitidas en este proyecto. Por ejemplo, el envío de mensajes entre diferentes drones.

Se espera que esta memoria sirva como guía para el programador que se aventure a realizar estas modificaciones en el futuro y a mejorar en general su aspecto y funcionalidad.

7. Ejemplo de uso

A continuación, se mostrará un simple ejemplo de uso del editor para construir lógica TeleoR mediante bloques. El objetivo es mostrar una especificación TeleoR sencilla y su equivalencia en lenguaje de bloques en **TeleoR Factory**.

Se dispone de un dron cuya misión es llegar a una altura determinada para echar una foto. El fichero de código que muestra el código asociado a dicha misión es el siguiente:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TYPES
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

speed ::= (1..3).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% PERCEPTS / BELIEFS / ACTIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

percept
    altitude : (nat). % Positive integer

discrete
    photo : ().

durative
    up : (speed),
    down : (speed).


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% TR
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

mainTask : nat ~>
mainTask(A){
    altitude(CurrentA) &
        (CurrentA > 0.9*A) &
        (CurrentA < 1.1*A)              ~> photo
    altitude(A_higher) & A_higher > A   ~> down(2)
    altitude(A_lower) & A_lower < A     ~> up(2)
    true                                ~> ()
}.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% LAUNCH
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

go : nat
go(A) ~>>
    start_agent testAgent env@localhost all;
    start_task main mainTask(A).
```

Figura 158: Ejemplo de especificación TR textual

Del fichero anterior se puede distinguir lo siguiente:

- Tiene definido un tipo: ***speed***. Es para la velocidad de movimiento.
- Tiene un percept, ***altitude(nat)***, que tiene un parámetro que es la altitud (tipo entero sin signo).
- Una acción discreta, ***photo***, la cual ordenará a la cámara que tome una fotografía.
- Tenemos dos acciones durativas, ***up(speed)*** y ***down(speed)***, para mover el dron arriba y abajo. Reciben como parámetro la velocidad del movimiento.
- ***mainTask*** es el procedimiento TR. Recibe un parámetro al inicio, **A**, que es la altitud a la que tiene que llegar el dron. Si está a esa altitud, hace una foto, si no,

se mueve arriba o abajo hasta que llegue. A la hora de comprobar la altitud se comprueba dentro de un margen del 10%.

- La función qulog de abajo del todo, ***go***, es para arrancar el agente y la tarea principal. Es la función que se llama desde el intérprete en la terminal para ejecutar el programa.

Los componentes que serán necesarios añadir en el **modo diseño** del entorno web serán los siguientes:

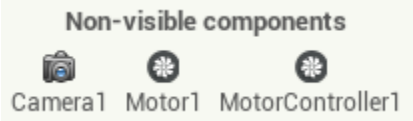


Figura 159: Componentes necesarios para el ejemplo de especificación TeleoR

Estos componentes proporcionarán los bloques que serán necesarios para construir el programa en el **editor de bloques**.

A continuación, se muestra el equivalente de dicha especificación al lenguaje de bloques que se ha desarrollado:

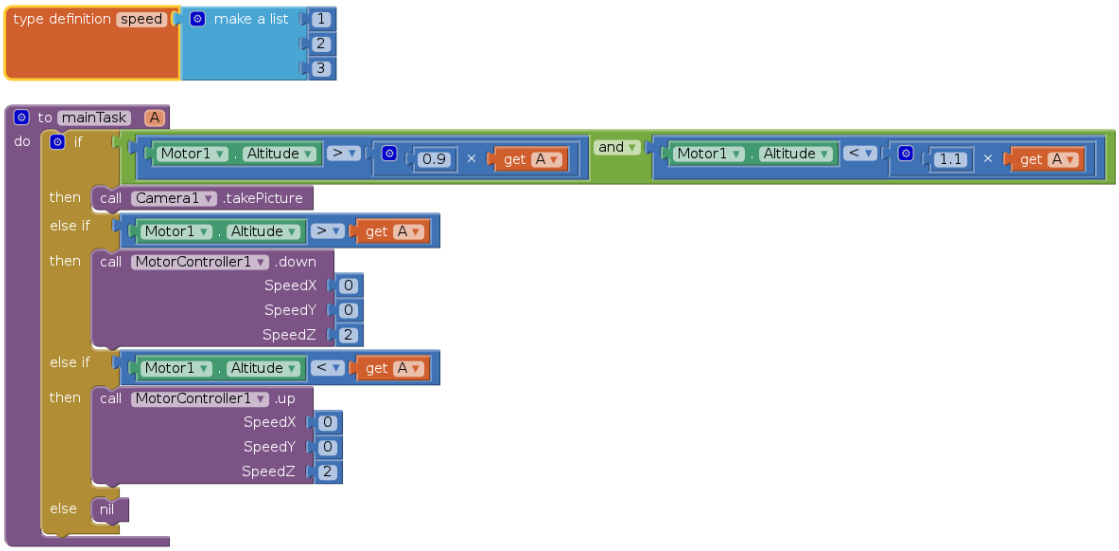


Figura 160: Ejemplo de especificación TeleoR mediante bloques

ANEXO 1: Descarga y compilación del código fuente de MIT App Inventor

T todo el proceso de modificación y compilación del código fuente de App Inventor se realizará utilizando Linux.

Con el objetivo de poder probar las modificaciones que se vayan realizando en el código fuente, es necesario ser capaces de compilar y mostrar la aplicación Web resultante de manera local. Para ello, se deben instalar una serie de herramientas siguiendo los siguientes pasos:

1. Descargamos el código fuente de App Inventor introduciendo en un terminal el siguiente comando:

   git clone https://github.com/mit-cml/appinventor-sources.git

2. En la carpeta ***appinventor-sources*** que será creada, hay buscar un fichero llamado ***sample-.gitignore*** *y renombrarlo a* ***.gitignore***.

3. En la carpeta ***appinventor-sources*** se haya una carpeta llamada ***appinventor***. De ahora en adelante esta carpeta será referida como el **directorio raíz** de appinventor.

4. A continuación, hay que descargar e instalar el ***Appengine SDK***, necesario para compilar el código de App Inventor. Para ello se debe acceder al siguiente enlace:

   https://cloud.google.com/appengine/downloads

   Seleccionar la opción ***Google App Engine SDK for Java*** y descargar la última versión disponible.

   Después descomprimir el fichero zip descargado en una ruta de libre elección a la que se llamará **directorio appengine**.

5. Instalar las aplicaciones necesarias mediante el siguiente comando (nótese que serán necesarios permisos de administrador para realizar la instalación):

   sudo apt-get install openjdk-7-jdk ant lib32z1 lib32ncurses5 lib32bz2-1.0 lib32stdc++6

6. Se necesitará instalar la herramienta ***apache2*** para poder ejecutar el servidor de la aplicación web de manera local mediante el siguiente comando:

   sudo apt-get install apache2

7. Una vez realizados todos los pasos, sólo queda compilar el código fuente de App Inventor posicionándose en consola en el **directorio raíz** de App Inventor y ejecutando el comando ***ant*** (aunque es conveniente eliminar previamente los ficheros de la compilación anterior mediante ***ant clean***):

   cd <RAÍZ_APPINVENTOR>
   ant clean
   ant

8. Ya se puede arrancar App Inventor mediante el comando:

   <DIRECTORIO_APPENGINE>/bin/dev_appserver.sh --port=8888 --address=0.0.0.0 <RAÍZ_APPINVENTOR>/appengine/build/war/

   Posteriormente, escribir en el navegador la siguiente URL para mostrar la aplicación web:

   http://localhost:8888

9. Como último dato, aunque no será necesario ejecutarlo por el momento, antes de compilar una aplicación creada mediante App Inventor, se deberá tener ejecutando el ***BuildServer*** mediante los siguientes comandos:

   cd <RAÍZ_APPINVENTOR>/buildserver
   ant RunLocalBuildServer

Para más información, se proporciona el enlace original de la documentación de App Inventor:

https://docs.google.com/document/pub?id=1Xc9yt02x3BRoq5m1PJHBr81OOv69rEBy8LVG_84j9jc#h.6y8gcyo56euq

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2009-2011 Google, All Rights reserved
    3: // Copyright 2011-2012 MIT, All rights reserved
    4: // Released under the Apache License, Version 2.0
    5: // http://www.apache.org/licenses/LICENSE-2.0
    6:
    7: package com.google.appinventor.client.editor.simple.components;
    8:
    9: import static com.google.appinventor.client.Ode.MESSAGES;
   10:
   11: import java.util.HashMap;
   12: import java.util.HashSet;
   13: import java.util.Map;
   14:
   15: import com.google.appinventor.client.editor.simple.SimpleEditor;
   16: import com.google.appinventor.client.editor.simple.components.utils.PropertiesUtil;
   17: import com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidLeng
thPropertyEditor;
   18: import com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidVert
icalAlignmentChoicePropertyEditor;
   19: import com.google.appinventor.client.output.OdeLog;
   20: import com.google.appinventor.client.properties.BadPropertyEditorException;
   21: import com.google.appinventor.shared.settings.SettingsConstants;
   22: import com.google.gwt.dom.client.DivElement;
   23: import com.google.gwt.dom.client.Document;
   24: import com.google.gwt.dom.client.Element;
   25: import com.google.gwt.dom.client.Style;
   26: import com.google.gwt.user.client.ui.AbsolutePanel;
   27: import com.google.gwt.user.client.ui.Composite;
   28: import com.google.gwt.user.client.ui.DockPanel;
   29: import com.google.gwt.user.client.ui.HorizontalPanel;
   30: import com.google.gwt.user.client.ui.Image;
   31: import com.google.gwt.user.client.ui.Label;
   32: import com.google.gwt.user.client.ui.ScrollPanel;
   33: import com.google.gwt.user.client.ui.TreeItem;
   34:
   35: /**
   36:  * Mock Form component.
   37:  *
   38:  */
   39: public final class MockForm extends MockContainer {
   40:
   41:   /*
   42:    * Widget for the mock form title bar.
   43:    */
   44:   private class TitleBar extends Composite {
   45:     private static final int HEIGHT = 24;
   46:
   47:
   48:
   49:     // UI elements
   50:     private Label title;
   51:     private AbsolutePanel bar;
   52:
   53:     /*
   54:      * Creates a new title bar.
   55:      */
   56:     TitleBar() {
   57:       title = new Label();
   58:       title.setStylePrimaryName("ode-SimpleMockFormTitle");
   59:       title.setHorizontalAlignment(Label.ALIGN_LEFT);
   60:
   61:       bar = new AbsolutePanel();
   62:       bar.add(title, 12, 4);
   63:
   64:       initWidget(bar);
   65:
   66:       setStylePrimaryName("ode-SimpleMockFormTitleBar");
   67:       setSize("100%", HEIGHT + "px");
   68:     }
   69:
   70:     /*
   71:      * Changes the title in the title bar.
   72:      */
   73:     void changeTitle(String newTitle) {
   74:       title.setText(newTitle);
   75:     }
   76:   }
   77:
   78:   /*
   79:    * Widget for a mock phone status bar.
   80:    */
   81:   private class PhoneBar extends Composite {
   82:     private static final int HEIGHT = 24;
   83:
   84:     // UI elements
   85:     private DockPanel bar;
   86:     private Image phoneBarImage;
   87:
   88:     /*
   89:      * Creates a new phone status bar.
   90:      */
   91:     PhoneBar() {
   92:       phoneBarImage = new Image(images.phonebar());
   93:
   94:       bar = new DockPanel();
   95:       bar.setHorizontalAlignment(HorizontalPanel.ALIGN_RIGHT);
   96:       bar.add(phoneBarImage, DockPanel.EAST);
   97:
   98:       initWidget(bar);
   99:
  100:       setStylePrimaryName("ode-SimpleMockFormPhoneBar");
  101:       setSize("100%", HEIGHT + "px");
  102:     }
  103:   }
  104:
  105:   /**
  106:    * Component type name.
  107:    */
  108:   public static final String TYPE = "Form";
  109:
  110:   private static final String VISIBLE_TYPE = "Screen";
  111:
  112:   // TODO(lizlooney) 320x480 is the resolution of the G1. Do we want to change this
to the
  113:   // resolution of the Nexus One?
  114:   private static final int PORTRAIT_WIDTH = 320;
  115:   private static final int PORTRAIT_HEIGHT = 480;
  116:   private static final int LANDSCAPE_WIDTH = 480;
  117:   private static final int LANDSCAPE_HEIGHT = 320;
  118:
  119:   // Property names
  120:   private static final String PROPERTY_NAME_TITLE = "Title";
  121:   private static final String PROPERTY_NAME_SCREEN_ORIENTATION = "ScreenOrientation"
```

```
;
  122:   private static final String PROPERTY_NAME_SCROLLABLE = "Scrollable";
  123:   private static final String PROPERTY_NAME_ICON = "Icon";
  124:   private static final String PROPERTY_NAME_VCODE = "VersionCode";
  125:   private static final String PROPERTY_NAME_VNAME = "VersionName";
  126:   private static final String PROPERTY_NAME_ANAME = "AppName";
  127:
  128:   // Form UI components
  129:   AbsolutePanel formWidget;
  130:   ScrollPanel scrollPanel;
  131:   private TitleBar titleBar;
  132:   private MockComponent selectedComponent;
  133:
  134:   private int screenWidth;
  135:   private int screenHeight;
  136:   private int usableScreenHeight;
  137:
  138:   // Set of listeners for any changes of the form
  139:   final HashSet<FormChangeListener> formChangeListeners = new HashSet<FormChangeList
ener>();
  140:
  141:   // Don't access the verticalScrollbarWidth field directly. Use getVerticalScrollba
rWidth().
  142:   private static int verticalScrollbarWidth;
  143:
  144:   private MockFormLayout myLayout;
  145:
  146:   // flag to control attempting to enable/disable vertical
  147:   // alignment when scrollable property is changed
  148:   private boolean initialized = false;
  149:
  150:   private YoungAndroidVerticalAlignmentChoicePropertyEditor myVAlignmentPropertyEdit
or;
  151:
  152:   public static final String PROPERTY_NAME_HORIZONTAL_ALIGNMENT = "AlignHorizontal";
  153:   public static final String PROPERTY_NAME_VERTICAL_ALIGNMENT = "AlignVertical";
  154:
  155:   /**
  156:    * Creates a new MockForm component.
  157:    *
  158:    * @param editor  editor of source file the component belongs to
  159:    */
  160:   public MockForm(SimpleEditor editor) {
  161:     // Note(Hal): This helper thing is a kludge because I really want to write:
  162:     // myLayout = new MockHVLayout(orientation);
  163:     // super(editor, type, icon, myLayout);
  164:     // but Java won't let me do that.
  165:
  166:     super(editor, TYPE, images.form(), MockFormHelper.makeLayout());
  167:     // Note(hal): There better not be any calls to MockFormHelper before the
  168:     // next instruction.  Note that the Helper methods are synchronized to avoid pos
sible
  169:     // future problems if we ever have threads creating forms in parallel.
  170:     myLayout = MockFormHelper.getLayout();
  171:
  172:     formWidget = new AbsolutePanel();
  173:     formWidget.setStylePrimaryName("ode-SimpleMockForm");
  174:
  175:     // Initialize mock form UI by adding the phone bar and title bar.
  176:     formWidget.add(new PhoneBar());
  177:     titleBar = new TitleBar();
  178:     formWidget.add(titleBar);
  179:
  180:     // Put a ScrollPanel around the rootPanel.
  181:     scrollPanel = new ScrollPanel(rootPanel);
  182:     formWidget.add(scrollPanel);
  183:
  184:     screenWidth = PORTRAIT_WIDTH;
  185:     screenHeight = PORTRAIT_HEIGHT;
  186:     usableScreenHeight = screenHeight - PhoneBar.HEIGHT - TitleBar.HEIGHT;
  187:
  188:     // This is just the initial size of the form. It will be resized in refresh();
  189:     rootPanel.setPixelSize(screenWidth, usableScreenHeight);
  190:     resizePanels();
  191:
  192:     initComponent(formWidget);
  193:
  194:     // Set up the initial state of the vertical alignment property editor and its
  195:     // dropdowns
  196:     try {
  197:       myVAlignmentPropertyEditor = PropertiesUtil.getVAlignmentEditor(properties);
  198:     } catch (BadPropertyEditorException e) {
  199:       OdeLog.log(MESSAGES.badAlignmentPropertyEditorForArrangement());
  200:       return;
  201:     }
  202:     enableAndDisableDropdowns();
  203:     initialized = true;
  204:     // Now that the default for Scrollable is false, we need to force setting the pr
operty when creating the MockForm
  205:     setScrollableProperty(getPropertyValue(PROPERTY_NAME_SCROLLABLE));
  206:   }
  207:
  208:   /*
  209:    * Resizes the scrollPanel and formWidget based on the screen size.
  210:    */
  211:   private void resizePanels() {
  212:     // Set the scrollPanel's width to account for the width of the vertical scrollba
r.
  213:     int vertScrollbarWidth = getVerticalScrollbarWidth();
  214:     scrollPanel.setPixelSize(screenWidth + vertScrollbarWidth, usableScreenHeight);
  215:     formWidget.setPixelSize(screenWidth + vertScrollbarWidth, screenHeight);
  216:   }
  217:
  218:   /*
  219:    * Returns the width of a vertical scroll bar, calculating it if necessary.
  220:    */
  221:   private static int getVerticalScrollbarWidth() {
  222:     // We only calculate the vertical scroll bar width once, then we store it in the
 static field
  223:     // verticalScrollbarWidth. If the field is non-zero, we don't need to calculate
it again.
  224:     if (verticalScrollbarWidth == 0) {
  225:       // The following code will calculate (on the fly) the width of a vertical scro
ll bar.
  226:       // We'll create two divs, one inside the other and add the outer div to the do
cument body,
  227:       // but off-screen where the user won't see it.
  228:       // We'll measure the width of the inner div twice: (first) when the outer div'
s vertical
  229:       // scrollbar is hidden and (second) when the outer div's vertical scrollbar is
 visible.
  230:       // The width of inner div will be smaller when outer div's vertical scrollbar
is visible.
  231:       // By subtracting the two measurements, we can calculate the width of the vert
```

```
ical scrollbar.
  232:
  233:       // I used code from the following websites as reference material:
  234:       // http://jdsharp.us/jQuery/minute/calculate-scrollbar-width.php
  235:       // http://www.fleegix.org/articles/2006-05-30-getting-the-scrollbar-width-in-p
ixels
  236:
  237:       Document document = Document.get();
  238:
  239:       // Create an outer div.
  240:       DivElement outerDiv = document.createDivElement();
  241:       Style outerDivStyle = outerDiv.getStyle();
  242:       // Use absolute positioning and set the top/left so that it is off-screen.
  243:       // We don't want the user to see anything while we do this calculation.
  244:       outerDivStyle.setProperty("position", "absolute");
  245:       outerDivStyle.setProperty("top", "-1000px");
  246:       outerDivStyle.setProperty("left", "-1000px");
  247:       // Set the width and height of the outer div to a fixed size in pixels.
  248:       outerDivStyle.setProperty("width", "100px");
  249:       outerDivStyle.setProperty("height", "50px");
  250:       // Hide the outer div's scrollbar by setting the "overflow" property to "hidde
n".
  251:       outerDivStyle.setProperty("overflow", "hidden");
  252:
  253:       // Create an inner div and put it inside the outer div.
  254:       DivElement innerDiv = document.createDivElement();
  255:       Style innerDivStyle = innerDiv.getStyle();
  256:       // Set the height of the inner div to be 4 times the height of the outer div s
o that a
  257:       // vertical scrollbar will be necessary (but hidden for now) on the outer div.
  258:       innerDivStyle.setProperty("height", "200px");
  259:       outerDiv.appendChild(innerDiv);
  260:
  261:       // Temporarily add the outer div to the document body. It's off-screen so the
user won't
  262:       // actually see anything.
  263:       Element bodyElement = document.getElementsByTagName("body").getItem(0);
  264:       bodyElement.appendChild(outerDiv);
  265:
  266:       // Get the width of the inner div while the outer div's vertical scrollbar is
hidden.
  267:       int widthWithoutScrollbar = innerDiv.getOffsetWidth();
  268:       // Show the outer div's vertical scrollbar by setting the "overflow" property
to "auto".
  269:       outerDivStyle.setProperty("overflow", "auto");
  270:       // Now, get the width of the inner div while the vertical scrollbar is visible
.
  271:       int widthWithScrollbar = innerDiv.getOffsetWidth();
  272:
  273:       // Remove the outer div from the document body.
  274:       bodyElement.removeChild(outerDiv);
  275:
  276:       // Calculate the width of the vertical scrollbar by subtracting the two widths
.
  277:       verticalScrollbarWidth = widthWithoutScrollbar - widthWithScrollbar;
  278:     }
  279:
  280:     return verticalScrollbarWidth;
  281:   }
  282:
  283:   @Override
  284:   public final MockForm getForm() {
  285:     return this;
  286:   }
  287:
  288:   @Override
  289:   public boolean isForm() {
  290:     return true;
  291:   }
  292:
  293:
  294:   @Override
  295:   public String getVisibleTypeName() {
  296:     return VISIBLE_TYPE;
  297:   }
  298:
  299:   @Override
  300:   protected void addWidthHeightProperties() {
  301:     addProperty(PROPERTY_NAME_WIDTH, "" + PORTRAIT_WIDTH, null,
  302:         new YoungAndroidLengthPropertyEditor());
  303:     addProperty(PROPERTY_NAME_HEIGHT, "" + LENGTH_PREFERRED, null,
  304:         new YoungAndroidLengthPropertyEditor());
  305:   }
  306:
  307:   @Override
  308:   public boolean isPropertyPersisted(String propertyName) {
  309:     // We use the Width and Height properties to make the form appear correctly in t
he designer,
  310:     // but they aren't actually persisted to the .scm file.
  311:     if (propertyName.equals(PROPERTY_NAME_WIDTH) ||
  312:         propertyName.equals(PROPERTY_NAME_HEIGHT)) {
  313:       return false;
  314:     }
  315:     return super.isPropertyPersisted(propertyName);
  316:   }
  317:
  318:   @Override
  319:   protected boolean isPropertyVisible(String propertyName) {
  320:     if (propertyName.equals(PROPERTY_NAME_WIDTH) ||
  321:         propertyName.equals(PROPERTY_NAME_HEIGHT)) {
  322:       return false;
  323:     }
  324:
  325:     if (propertyName.equals(PROPERTY_NAME_ICON)) {
  326:       // The Icon property actually applies to the application and is only visible o
n Screen1.
  327:       return editor.isScreen1();
  328:     }
  329:
  330:     if (propertyName.equals(PROPERTY_NAME_VNAME)) {
  331:       // The VersionName property actually applies to the application and is only vi
sible on Screen1.
  332:       return editor.isScreen1();
  333:     }
  334:
  335:     if (propertyName.equals(PROPERTY_NAME_VCODE)) {
  336:       // The VersionCode property actually applies to the application and is only vi
sible on Screen1.
  337:       return editor.isScreen1();
  338:     }
  339:
  340:     if (propertyName.equals(PROPERTY_NAME_ANAME)) {
  341:       // The AppName property actually applies to the application and is only visibl
e on Screen1.
```

```
342:       return editor.isScreen1();
343:     }
344:
345:     return super.isPropertyVisible(propertyName);
346:   }
347:
348:   /*
349:    * Sets the form's BackgroundColor property to a new value.
350:    */
351:   private void setBackgroundColorProperty(String text) {
352:     if (MockComponentsUtil.isNoneColor(text)) {
353:       text = "&HFF000000";  // black
354:     } else if (MockComponentsUtil.isDefaultColor(text)) {
355:       text = "&HFFFFFFFF";  // white
356:     }
357:     MockComponentsUtil.setWidgetBackgroundColor(rootPanel, text);
358:   }
359:
360:   /*
361:    * Sets the form's BackgroundImage property to a new value.
362:    */
363:   private void setBackgroundImageProperty(String text) {
364:     String url = convertImagePropertyValueToUrl(text);
365:     if (url == null) {
366:       // text was not recognized as an asset.
367:       url = "";
368:     }
369:     MockComponentsUtil.setWidgetBackgroundImage(rootPanel, url);
370:   }
371:
372:   private void setScreenOrientationProperty(String text) {
373:     if (hasProperty(PROPERTY_NAME_WIDTH) && hasProperty(PROPERTY_NAME_HEIGHT) &&
374:         hasProperty(PROPERTY_NAME_SCROLLABLE)) {
375:       if (text.equalsIgnoreCase("landscape")) {
376:         screenWidth = LANDSCAPE_WIDTH;
377:         screenHeight = LANDSCAPE_HEIGHT;
378:       } else {
379:         screenWidth = PORTRAIT_WIDTH;
380:         screenHeight = PORTRAIT_HEIGHT;
381:       }
382:       usableScreenHeight = screenHeight - PhoneBar.HEIGHT - TitleBar.HEIGHT;
383:       resizePanels();
384:
385:       changeProperty(PROPERTY_NAME_WIDTH, "" + screenWidth);
386:       boolean scrollable = Boolean.parseBoolean(getPropertyValue(PROPERTY_NAME_SCROL
LABLE));
387:       if (!scrollable) {
388:         changeProperty(PROPERTY_NAME_HEIGHT, "" + usableScreenHeight);
389:       }
390:     }
391:   }
392:
393:   private void setScrollableProperty(String text) {
394:     if (hasProperty(PROPERTY_NAME_HEIGHT)) {
395:       final boolean scrollable = Boolean.parseBoolean(text);
396:       int heightHint = scrollable ? LENGTH_PREFERRED : usableScreenHeight;
397:       changeProperty(PROPERTY_NAME_HEIGHT, "" + heightHint);
398:     }
399:   }
400:
401:   private void setIconProperty(String icon) {
402:     // The Icon property actually applies to the application and is only visible on
Screen1.
403:     // When we load a form that is not Screen1, this method will be called with the
default value
404:     // for icon (empty string). We need to ignore that.
405:     if (editor.isScreen1()) {
406:       editor.getProjectEditor().changeProjectSettingsProperty(
407:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
408:           SettingsConstants.YOUNG_ANDROID_SETTINGS_ICON, icon);
409:     }
410:   }
411:
412:   private void setVCodeProperty(String vcode) {
413:     // The VersionCode property actually applies to the application and is only visi
ble on Screen1.
414:     // When we load a form that is not Screen1, this method will be called with the
default value
415:     // for VersionCode (1). We need to ignore that.
416:     if (editor.isScreen1()) {
417:       editor.getProjectEditor().changeProjectSettingsProperty(
418:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
419:           SettingsConstants.YOUNG_ANDROID_SETTINGS_VERSION_CODE, vcode);
420:     }
421:   }
422:
423:   private void setVNameProperty(String vname) {
424:     // The VersionName property actually applies to the application and is only visi
ble on Screen1.
425:     // When we load a form that is not Screen1, this method will be called with the
default value
426:     // for VersionName (1.0). We need to ignore that.
427:     if (editor.isScreen1()) {
428:       editor.getProjectEditor().changeProjectSettingsProperty(
429:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
430:           SettingsConstants.YOUNG_ANDROID_SETTINGS_VERSION_NAME, vname);
431:     }
432:   }
433:
434:   private void setANameProperty(String aname) {
435:     // The AppName property actually applies to the application and is only visible
on Screen1.
436:     // When we load a form that is not Screen1, this method will be called with the
default value
437:     if (editor.isScreen1()) {
438:       editor.getProjectEditor().changeProjectSettingsProperty(
439:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
440:           SettingsConstants.YOUNG_ANDROID_SETTINGS_APP_NAME, aname);
441:     }
442:   }
443:
444:   /**
445:    * Forces a re-layout of the child components of the container.
446:    */
447:   public final void refresh() {
448:     Map<MockComponent, LayoutInfo> layoutInfoMap = new HashMap<MockComponent, Layout
Info>();
449:
450:     collectLayoutInfos(layoutInfoMap, this);
451:
452:     LayoutInfo formLayoutInfo = layoutInfoMap.get(this);
453:     layout.layoutChildren(formLayoutInfo);
454:     rootPanel.setPixelSize(formLayoutInfo.width,
455:         Math.max(formLayoutInfo.height, usableScreenHeight));
```

```
456:
457:     for (LayoutInfo layoutInfo : layoutInfoMap.values()) {
458:       layoutInfo.cleanUp();
459:     }
460:     layoutInfoMap.clear();
461:   }
462:
463:   /*
464:    * Collects the LayoutInfo of the given component and, recursively, all of
465:    * its children.
466:    *
467:    * If a component's width/height hint is automatic, the corresponding
468:    * LayoutInfo's width/height will be set to the calculated width/height.
469:    * If a component's width/height hint is fill parent, the corresponding
470:    * LayoutInfo's width/height may be set to fill parent. This will be resolved
471:    * when layoutChildren is called.
472:    */
473:   private static void collectLayoutInfos(Map<MockComponent, LayoutInfo> layoutInfoMa
p,
474:       MockComponent component) {
475:
476:     LayoutInfo layoutInfo = component.createLayoutInfo(layoutInfoMap);
477:
478:     // If this component is a container, collect the LayoutInfos of its children.
479:     if (component instanceof MockContainer) {
480:       if (!layoutInfo.visibleChildren.isEmpty()) {
481:         // We resize the container to be very large so that we get accurate
482:         // results when we ask for a child's size using getOffsetWidth/getOffsetHeig
ht.
483:         // If the container is its normal size (or perhaps the default empty
484:         // size), then the browser won't give us anything bigger than that
485:         // when we ask for a child's size.
486:         if (component.isForm()) {
487:           ((MockForm) component).rootPanel.setPixelSize(1000, 1000);
488:         } else {
489:           component.setPixelSize(1000, 1000);
490:         }
491:
492:         // Show children that should be shown and collect their layoutInfos.
493:         // Note that some MockLayout implementations may hide children that are in t
he
494:         // visibleChildren list. For example, in MockTableLayout, if two or more chi
ldren occupy
495:         // the same cell in the table, all but one of the children are hidden.
496:         for (MockComponent child : layoutInfo.visibleChildren) {
497:           child.setVisible(true);
498:           collectLayoutInfos(layoutInfoMap, child);
499:         }
500:       }
501:
502:       // Hide children that should be hidden.
503:       for (MockComponent child : component.getHiddenVisibleChildren()) {
504:         child.setVisible(false);
505:       }
506:     }
507:
508:     layoutInfo.gatherDimensions();
509:   }
510:
511:   /**
512:    * Adds an {@link FormChangeListener} to the listener set if it isn't already in t
here.
513:    *
514:    * @param listener  the {@code FormChangeListener} to be added
515:    */
516:   public void addFormChangeListener(FormChangeListener listener) {
517:     formChangeListeners.add(listener);
518:   }
519:
520:   /**
521:    * Removes an {@link FormChangeListener} from the listener list.
522:    *
523:    * @param listener  the {@code FormChangeListener} to be removed
524:    */
525:   public void removeFormChangeListener(FormChangeListener listener) {
526:     formChangeListeners.remove(listener);
527:   }
528:
529:   /**
530:    * Triggers a component property change event to be sent to the listener on the li
stener list.
531:    */
532:   protected void fireComponentPropertyChanged(MockComponent component,
533:       String propertyName, String propertyValue) {
534:     for (FormChangeListener listener : formChangeListeners) {
535:       listener.onComponentPropertyChanged(component, propertyName, propertyValue);
536:     }
537:   }
538:
539:   /**
540:    * Triggers a component removed event to be sent to the listener on the listener l
ist.
541:    */
542:   protected void fireComponentRemoved(MockComponent component, boolean permanentlyDe
leted) {
543:     for (FormChangeListener listener : formChangeListeners) {
544:       listener.onComponentRemoved(component, permanentlyDeleted);
545:     }
546:   }
547:
548:   /**
549:    * Triggers a component added event to be sent to the listener on the listener lis
t.
550:    */
551:   protected void fireComponentAdded(MockComponent component) {
552:     for (FormChangeListener listener : formChangeListeners) {
553:       listener.onComponentAdded(component);
554:     }
555:   }
556:
557:   /**
558:    * Triggers a component renamed event to be sent to the listener on the listener l
ist.
559:    */
560:   protected void fireComponentRenamed(MockComponent component, String oldName) {
561:     for (FormChangeListener listener : formChangeListeners) {
562:       listener.onComponentRenamed(component, oldName);
563:     }
564:   }
565:
566:   /**
567:    * Triggers a component selection change event to be sent to the listener on the l
istener list.
568:    */
```

```
  protected void fireComponentSelectionChange(MockComponent component, boolean selected) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentSelectionChange(component, selected);
    }
  }

  /**
   * Changes the component that is currently selected in the form.
   * <p>
   * There will always be exactly one component selected in a form
   * at any given time.
   */
  public final void setSelectedComponent(MockComponent newSelectedComponent) {
    MockComponent oldSelectedComponent = selectedComponent;

    if (newSelectedComponent == null) {
      throw new IllegalArgumentException("at least one component must always be selected");
    }
    if (newSelectedComponent == oldSelectedComponent) {
      return;
    }

    selectedComponent = newSelectedComponent;

    if (oldSelectedComponent != null) {     // Can be null initially
      oldSelectedComponent.onSelectedChange(false);
    }
    newSelectedComponent.onSelectedChange(true);
  }

  public final MockComponent getSelectedComponent() {
    return selectedComponent;
  }

  /**
   * Builds a tree of the component hierarchy of the form for display in the
   * {@code SourceStructureExplorer}.
   *
   * @return  tree showing the component hierarchy of the form
   */
  public TreeItem buildComponentsTree() {
    return buildTree();
  }

  // PropertyChangeListener implementation

  @Override
  public void onPropertyChange(String propertyName, String newValue) {
    super.onPropertyChange(propertyName, newValue);

    // Apply changed properties to the mock component
    if (propertyName.equals(PROPERTY_NAME_BACKGROUNDCOLOR)) {
      setBackgroundColorProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_BACKGROUNDIMAGE)) {
      setBackgroundImageProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_SCREEN_ORIENTATION)) {
      setScreenOrientationProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_SCROLLABLE)) {
      setScrollableProperty(newValue);
      adjustAlignmentDropdowns();
    } else if (propertyName.equals(PROPERTY_NAME_TITLE)) {
      titleBar.changeTitle(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_ICON)) {
      setIconProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_VCODE)) {
      setVCodeProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_VNAME)) {
      setVNameProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_ANAME)) {
      setANameProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_HORIZONTAL_ALIGNMENT)) {
      myLayout.setHAlignmentFlags(newValue);
      refreshForm();
    } else if (propertyName.equals(PROPERTY_NAME_VERTICAL_ALIGNMENT)) {
    myLayout.setVAlignmentFlags(newValue);
    refreshForm();
    }
  }

  // enableAndDisable It should not be called until the component is initialized.
  // Otherwise, we'll get NPEs in trying to use myAlignmentPropertyEditor.
  private void adjustAlignmentDropdowns() {
    if (initialized) enableAndDisableDropdowns();
  }

  // Don't forget to call this on initialization!!!
  // If scrollable is True, the selector for vertical alignment should be disabled.
  private void enableAndDisableDropdowns() {
    String scrollable = properties.getProperty(PROPERTY_NAME_SCROLLABLE).getValue();
    if (scrollable.equals("True")) {
      myVAlignmentPropertyEditor.disable();
    } else myVAlignmentPropertyEditor.enable();
  }
}
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0


package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.ErrorMessages;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;

import java.util.LinkedList;
import java.util.List;
import java.util.Queue;

/**
 * Physical world component that can detect shaking and measure
 * acceleration in three dimensions.  It is implemented using
 * android.hardware.SensorListener
 * (http://developer.android.com/reference/android/hardware/SensorListener.html).
 *
 * <p>From the Android documentation:
 * "Sensor values are acceleration in the X, Y and Z axis, where the X axis
 * has positive direction toward the right side of the device, the Y axis has
 * positive direction toward the top of the device and the Z axis has
 * positive direction toward the front of the device. The direction of the
 * force of gravity is indicated by acceleration values in the X, Y and Z
 * axes. The typical case where the device is flat relative to the surface of
 * the Earth appears as -STANDARD_GRAVITY in the Z axis and X and Y values
 * close to zero. Acceleration values are given in SI units (m/s^2)."
 *
 */
// TODO(user): ideas - event for knocking
@DesignerComponent(version = YaVersion.ACCELEROMETERSENSOR_COMPONENT_VERSION,
    description = "Non-visible component that can detect shaking and " +
    "measure acceleration approximately in three dimensions using SI units " +
    "(m/s<sup>2</sup>).  The components are: <ul>\n" +
    "<li> <strong>xAccel</strong>: 0 when the phone is at rest on a flat " +
    "     surface, positive when the phone is tilted to the right (i.e., " +
    "     its left side is raised), and negative when the phone is tilted " +
    "     to the left (i.e., its right size is raised).</li>\n " +
    "<li> <strong>yAccel</strong>: 0 when the phone is at rest on a flat " +
    "     surface, positive when its bottom is raised, and negative when " +
    "     its top is raised. </li>\n " +
    "<li> <strong>zAccel</strong>: Equal to -9.8 (earth's gravity in meters per " +
    "     second per second when the device is at rest parallel to the ground " +
    "     with the display facing up, " +
    "     0 when perpindicular to the ground, and +9.8 when facing down.  " +
    "     The value can also be affected by accelerating it with or against " +
    "     gravity. </li></ul>",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/accelerometersensor.png")
@SimpleObject
public class AccelerometerSensor extends AndroidNonvisibleComponent
    implements OnStopListener, OnResumeListener, SensorComponent, SensorEventListener, Deleteable {

  // Shake thresholds - derived by trial
  private static final double weakShakeThreshold = 5.0;
  private static final double moderateShakeThreshold = 13.0;
  private static final double strongShakeThreshold = 20.0;

  // Cache for shake detection
  private static final int SENSOR_CACHE_SIZE = 10;
  private final Queue<Float> X_CACHE = new LinkedList<Float>();
  private final Queue<Float> Y_CACHE = new LinkedList<Float>();
  private final Queue<Float> Z_CACHE = new LinkedList<Float>();

  // Backing for sensor values
  private float xAccel;
  private float yAccel;
  private float zAccel;

  private int accuracy;

  private int sensitivity;

  // Sensor manager
  private final SensorManager sensorManager;

  // Indicates whether the accelerometer should generate events
  private boolean enabled;

  //Specifies the minimum time interval between calls to Shaking()
  private int minimumInterval;

  //Specifies the time when Shaking() was last called
  private long timeLastShook;

  private Sensor accelerometerSensor;

  /**
   * Creates a new AccelerometerSensor component.
   *
   * @param container  ignored (because this is a non-visible component)
   */
  public AccelerometerSensor(ComponentContainer container) {
    super(container.$form());
    form.registerForOnResume(this);
    form.registerForOnStop(this);

    enabled = true;
    sensorManager = (SensorManager) container.$context().getSystemService(Context.SENSOR_SERVICE);
    accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    startListening();
    MinimumInterval(400);
    Sensitivity(Component.ACCELEROMETER_SENSITIVITY_MODERATE);
```

```
  123:   }
  124:
  125:
  126:   /**
  127:    * Returns the minimum interval required between calls to Shaking(),
  128:    * in milliseconds.
  129:    * Once the phone starts being shaken, all further Shaking() calls will be ignored
  130:    * until the interval has elapsed.
  131:    * @return  minimum interval in ms
  132:    */
  133:   @SimpleProperty(
  134:       category = PropertyCategory.BEHAVIOR,
  135:       description = "The minimum interval between phone shakes")
  136:   public int MinimumInterval() {
  137:     return minimumInterval;
  138:   }
  139:
  140:   /**
  141:    * Specifies the minimum interval required between calls to Shaking(),
  142:    * in milliseconds.
  143:    * Once the phone starts being shaken, all further Shaking() calls will be ignored
  144:    * until the interval has elapsed.
  145:    * @param interval  minimum interval in ms
  146:    */
  147:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_IN
TEGER,
  148:       defaultValue = "400") //Default value derived by trial of 12 people on 3 diffe
rent devices
  149:   @SimpleProperty
  150:   public void MinimumInterval(int interval) {
  151:     minimumInterval = interval;
  152:   }
  153:
  154:   /**
  155:    * Returns a number that encodes how sensitive the AccelerometerSensor is.
  156:    * The choices are: 1 = weak, 2 = moderate, 3 = strong.
  157:    *
  158:    * @return  one of {@link Component#ACCELEROMETER_SENSITIVITY_WEAK},
  159:    *          {@link Component#ACCELEROMETER_SENSITIVITY_MODERATE} or
  160:    *          {@link Component#ACCELEROMETER_SENSITIVITY_STRONG}
  161:    */
  162:   @SimpleProperty(
  163:       category = PropertyCategory.APPEARANCE,
  164:       description = "A number that encodes how sensitive the accelerometer is. " +
  165:               "The choices are: 1 = weak, 2 = moderate, " +
  166:               " 3 = strong.")
  167:   public int Sensitivity() {
  168:     return sensitivity;
  169:   }
  170:
  171:   /**
  172:    * Specifies the sensitivity of the accelerometer
  173:    * and checks that the argument is a legal value.
  174:    *
  175:    * @param sensitivity one of {@link Component#ACCELEROMETER_SENSITIVITY_WEAK},
  176:    *          {@link Component#ACCELEROMETER_SENSITIVITY_MODERATE} or
  177:    *          {@link Component#ACCELEROMETER_SENSITIVITY_STRONG}
  178:    *
  179:    */
  180:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ACCELEROMETER_S
ENSITIVITY,
  181:       defaultValue = Component.ACCELEROMETER_SENSITIVITY_MODERATE + "")
  182:   @SimpleProperty
  183:   public void Sensitivity(int sensitivity) {
  184:     if ((sensitivity == 1) || (sensitivity == 2) || (sensitivity == 3)) {
  185:       this.sensitivity = sensitivity;
  186:     } else {
  187:       form.dispatchErrorOccurredEvent(this, "Sensitivity",
  188:           ErrorMessages.ERROR_BAD_VALUE_FOR_ACCELEROMETER_SENSITIVITY, sensitivity);
  189:     }
  190:   }
  191:
  192:   /**
  193:    * Indicates the acceleration changed in the X, Y, and/or Z dimensions.
  194:    */
  195:   @SimpleEvent
  196:   public void AccelerationChanged(float xAccel, float yAccel, float zAccel) {
  197:     this.xAccel = xAccel;
  198:     this.yAccel = yAccel;
  199:     this.zAccel = zAccel;
  200:
  201:     addToSensorCache(X_CACHE, xAccel);
  202:     addToSensorCache(Y_CACHE, yAccel);
  203:     addToSensorCache(Z_CACHE, zAccel);
  204:
  205:     long currentTime = System.currentTimeMillis();
  206:
  207:     //Checks whether the phone is shaking and the minimum interval
  208:     //has elapsed since the last registered a shaking event.
  209:     if ((isShaking(X_CACHE, xAccel) || isShaking(Y_CACHE, yAccel) || isShaking(Z_CAC
HE, zAccel))
  210:         && (timeLastShook == 0 || currentTime >= timeLastShook + minimumInterval)){
  211:       timeLastShook = currentTime;
  212:       Shaking();
  213:     }
  214:
  215:     EventDispatcher.dispatchEvent(this, "AccelerationChanged", xAccel, yAccel, zAcce
l);
  216:   }
  217:
  218:   /**
  219:    * Indicates the device started being shaken or continues to be shaken.
  220:    */
  221:   @SimpleEvent
  222:   public void Shaking() {
  223:     EventDispatcher.dispatchEvent(this, "Shaking");
  224:   }
  225:
  226:   /**
  227:    * Available property getter method (read-only property).
  228:    *
  229:    * @return {@code true} indicates that an accelerometer sensor is available,
  230:    *         {@code false} that it isn't
  231:    */
  232:   @SimpleProperty(
  233:       category = PropertyCategory.BEHAVIOR)
  234:   public boolean Available() {
  235:     List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);
  236:     return (sensors.size() > 0);
  237:   }
  238:
  239:   /**
  240:    * If true, the sensor will generate events.  Otherwise, no events
  241:    * are generated even if the device is accelerated or shaken.
```

```
   *
   * @return {@code true} indicates that the sensor generates events,
   *         {@code false} that it doesn't
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR)
  public boolean Enabled() {
    return enabled;
  }

  // Assumes that sensorManager has been initialized, which happens in constructor
  private void startListening() {
    sensorManager.registerListener(this, accelerometerSensor, SensorManager.SENSOR_D
ELAY_GAME);
  }

  // Assumes that sensorManager has been initialized, which happens in constructor
  private void stopListening() {
    sensorManager.unregisterListener(this);
  }

  /**
   * Specifies whether the sensor should generate events.  If true,
   * the sensor will generate events.  Otherwise, no events are
   * generated even if the device is accelerated or shaken.
   *
   * @param enabled  {@code true} enables sensor event generation,
   *                 {@code false} disables it
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
      defaultValue = "True")
  @SimpleProperty
  public void Enabled(boolean enabled) {
    if (this.enabled == enabled) {
      return;
    }
    this.enabled = enabled;
    if (enabled) {
      startListening();
    } else {
      stopListening();
    }
  }

  /**
   * Returns the acceleration in the X-dimension in SI units (m/s^2).
   * The sensor must be enabled to return meaningful values.
   *
   * @return  X acceleration
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR)
  public float XAccel() {
    return xAccel;
  }

  /**
   * Returns the acceleration in the Y-dimension in SI units (m/s^2).
   * The sensor must be enabled to return meaningful values.
   *
   * @return  Y acceleration
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR)
  public float YAccel() {
    return yAccel;
  }

  /**
   * Returns the acceleration in the Z-dimension in SI units (m/s^2).
   * The sensor must be enabled to return meaningful values.
   *
   * @return  Z acceleration
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR)
  public float ZAccel() {
    return zAccel;
  }

  /*
   * Updating sensor cache, replacing oldest values.
   */
  private void addToSensorCache(Queue<Float> cache, float value) {
    if (cache.size() >= SENSOR_CACHE_SIZE) {
      cache.remove();
    }
    cache.add(value);
  }

  /*
   * Indicates whether there was a sudden, unusual movement.
   */
  // TODO(user): Maybe this can be improved.
  // See http://www.utdallas.edu/~rxb023100/pubs/Accelerometer_WBSN.pdf.
  private boolean isShaking(Queue<Float> cache, float currentValue) {
    float average = 0;
    for (float value : cache) {
      average += value;
    }

    average /= cache.size();

    if (Sensitivity() == 1) { //sensitivity is weak
      return Math.abs(average - currentValue) > strongShakeThreshold;
    } else if (Sensitivity() == 2) { //sensitivity is moderate
      return ((Math.abs(average - currentValue) > moderateShakeThreshold)
        && (Math.abs(average - currentValue) < strongShakeThreshold));
    } else { //sensitivity is strong
      return ((Math.abs(average - currentValue) > weakShakeThreshold)
        && (Math.abs(average - currentValue) < moderateShakeThreshold));
    }
  }

  // SensorListener implementation
  @Override
  public void onSensorChanged(SensorEvent sensorEvent) {
    if (enabled) {
      final float[] values = sensorEvent.values;
      xAccel = values[0];
      yAccel = values[1];
      zAccel = values[2];
      accuracy = sensorEvent.accuracy;
      AccelerationChanged(xAccel, yAccel, zAccel);
```

```
365:     }
366:   }
367:
368:   @Override
369:   public void onAccuracyChanged(Sensor sensor, int accuracy) {
370:     // TODO(markf): Figure out if we actually need to do something here.
371:   }
372:
373:   // OnResumeListener implementation
374:
375:   @Override
376:   public void onResume() {
377:     if (enabled) {
378:       startListening();
379:     }
380:   }
381:
382:   // OnStopListener implementation
383:
384:   @Override
385:   public void onStop() {
386:     if (enabled) {
387:       stopListening();
388:     }
389:   }
390:
391:   // Deleteable implementation
392:
393:   @Override
394:   public void onDelete() {
395:     if (enabled) {
396:       stopListening();
397:     }
398:   }
399: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.ErrorMessages;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Intent;
import android.net.Uri;
import android.os.Environment;
import android.provider.MediaStore;
import android.util.Log;

import java.io.File;
import java.util.Date;

/**
 * Camera provides access to the phone's camera
 *
 *
 */
@DesignerComponent(version = YaVersion.CAMERA_COMPONENT_VERSION,
    description = "A component to take a picture using the device's camera. " +
        "After the picture is taken, the name of the file on the phone " +
        "containing the picture is available as an argument to the " +
        "AfterPicture event. The file name can be used, for example, to set " +
        "the Picture property of an Image component.",
    category = ComponentCategory.MEDIA,
    nonVisible = true,
    iconName = "images/camera.png")
@SimpleObject
public class Camera extends AndroidNonvisibleComponent
     implements ActivityResultListener, Component {

  private static final String CAMERA_INTENT = "android.media.action.IMAGE_CAPTURE";
  private static final String CAMERA_OUTPUT = "output";
  private final ComponentContainer container;
  private Uri imageFile;

  /* Used to identify the call to startActivityForResult. Will be passed back
  into the resultReturned() callback method. */
  private int requestCode;

  // whether to open into the front-facing camera
  private boolean useFront;

  /**
   * Creates a Camera component.
   *
   * Camera has a boolean option to request the forward-facing camera via an intent extra.
   *
   * @param container container, component will be placed in
   */
  public Camera(ComponentContainer container) {
    super(container.$form());
    this.container = container;

    // Default property values
    UseFront(false);
  }

  /**
   * Returns true if the front-facing camera is to be used (when available)
   *
   * @return {@code true} indicates front-facing is to be used, {@code false} will open default
   */
  @SimpleProperty(
    category = PropertyCategory.BEHAVIOR)
  public boolean UseFront() {
    return useFront;
  }

  /**
   * Specifies whether the front-facing camera should be used (when available)
   *
   * @param front
   *          {@code true} for front-facing camera, {@code false} for default
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaultValue = "False")
  @SimpleProperty(description = "Specifies whether the front-facing camera should be used (when available). "
      + "If the device does not have a front-facing camera, this option will be ignored "
      + "and the camera will open normally.")
  public void UseFront(boolean front) {
    useFront = front;
  }

  /**
   * Takes a picture, then raises the AfterPicture event.
   * If useFront is true, adds an extra to the intent that requests the front-facing camera.
   */
  @SimpleFunction
  public void TakePicture() {
    Date date = new Date();
    String state = Environment.getExternalStorageState();

    if (Environment.MEDIA_MOUNTED.equals(state)) {
      Log.i("CameraComponent", "External storage is available and writable");

      imageFile = Uri.fromFile(new File(Environment.getExternalStorageDirectory(),
        "/Pictures/app_inventor_" + date.getTime()
        + ".jpg"));

      ContentValues values = new ContentValues();
```

```
      values.put(MediaStore.Images.Media.DATA, imageFile.getPath());
      values.put(MediaStore.Images.Media.MIME_TYPE, "image/jpeg");
      values.put(MediaStore.Images.Media.TITLE, imageFile.getLastPathSegment());

      if (requestCode == 0) {
        requestCode = form.registerForActivityResult(this);
      }

      Uri imageUri = container.$context().getContentResolver().insert(
        MediaStore.Images.Media.INTERNAL_CONTENT_URI, values);
      Intent intent = new Intent(CAMERA_INTENT);
      intent.putExtra(CAMERA_OUTPUT, imageUri);

      // NOTE: This uses an undocumented, testing feature (CAMERA_FACING).
      // It may not work in the future.
      if (useFront) {
        intent.putExtra("android.intent.extras.CAMERA_FACING", 1);
      }

      container.$context().startActivityForResult(intent, requestCode);
    } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
      form.dispatchErrorOccurredEvent(this, "TakePicture",
          ErrorMessages.ERROR_MEDIA_EXTERNAL_STORAGE_READONLY);
    } else {
      form.dispatchErrorOccurredEvent(this, "TakePicture",
          ErrorMessages.ERROR_MEDIA_EXTERNAL_STORAGE_NOT_AVAILABLE);
    }
  }

  @Override
  public void resultReturned(int requestCode, int resultCode, Intent data) {
    Log.i("CameraComponent",
      "Returning result. Request code = " + requestCode + ", result code = " + resul
tCode);
    if (requestCode == this.requestCode && resultCode == Activity.RESULT_OK) {
      File image = new File(imageFile.getPath());
      if (image.length() != 0) {
        AfterPicture(imageFile.toString());
      } else {
        deleteFile(imageFile);  // delete empty file
        // see if something useful got returned in the data
        if (data != null && data.getData() != null) {
          Uri tryImageUri = data.getData();
          Log.i("CameraComponent", "Calling Camera.AfterPicture with image path "
              + tryImageUri.toString());
          AfterPicture(tryImageUri.toString());
        } else {
          Log.i("CameraComponent", "Couldn't find an image file from the Camera resu
lt");

          form.dispatchErrorOccurredEvent(this, "TakePicture",
              ErrorMessages.ERROR_CAMERA_NO_IMAGE_RETURNED);
        }
      }
    } else {
      // delete empty file
      deleteFile(imageFile);
    }
  }

  private void deleteFile(Uri fileUri) {
    File fileToDelete = new File(fileUri.getPath());
    try {
      if (fileToDelete.delete()) {
        Log.i("CameraComponent", "Deleted file " + fileUri.toString());
      } else {
        Log.i("CameraComponent", "Could not delete file " + fileUri.toString());
      }
    } catch (SecurityException e) {
      Log.i("CameraComponent", "Got security exception trying to delete file "
          + fileUri.toString());
    }
  }

  /**
   * Indicates that a photo was taken with the camera and provides the path to
   * the stored picture.
   */
  @SimpleEvent
  public void AfterPicture(String image) {
    EventDispatcher.dispatchEvent(this, "AfterPicture", image);
  }
}
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.errors.YailRuntimeError;
import com.google.appinventor.components.runtime.util.Dates;
import com.google.appinventor.components.runtime.util.TimerInternal;

import java.util.Calendar;

/**
 * Clock provides the phone's clock, a timer, calendar and time calculations.
 * Everything is represented in milliseconds.
 *
 */

@DesignerComponent(version = YaVersion.CLOCK_COMPONENT_VERSION,
    description = "Non-visible component that provides the instant in "
    + "time using the internal clock on the phone. It can fire a "
    + "timer at regularly set intervals and perform time "
    + "calculations, manipulations, and conversions. Methods to "
    + "format the date and time are also available.",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/clock.png")
@SimpleObject
public final class Clock extends AndroidNonvisibleComponent
    implements Component, AlarmHandler, OnStopListener, OnResumeListener, OnDestroyL
istener,
               Deleteable {
  private static final int DEFAULT_INTERVAL = 1000;  // ms
  private static final boolean DEFAULT_ENABLED = true;

  private TimerInternal timerInternal;
  private boolean timerAlwaysFires = true;
  private boolean onScreen = false;

  /**
   * Creates a new Clock component.
   *
   * @param container ignored (because this is a non-visible component)
   */
  public Clock(ComponentContainer container) {
    super(container.$form());
    timerInternal = new TimerInternal(this, DEFAULT_ENABLED, DEFAULT_INTERVAL);

    // Set up listeners
    form.registerForOnResume(this);
    form.registerForOnStop(this);
    form.registerForOnDestroy(this);

    if (form instanceof ReplForm) {
      // In REPL, if this Clock component was added to the project after the onResum
e call occurred,
      // then onScreen would be false, but the REPL app is, in fact, on screen.
      onScreen = true;
    }
  }

  // Only the above constructor should be used in practice.
  public Clock() {
    super(null);
    // To allow testing without Timer
  }

  /**
   * Default Timer event handler.
   */
  @SimpleEvent(
      description = "Timer has gone off.")
  public void Timer() {
    if (timerAlwaysFires || onScreen) {
      EventDispatcher.dispatchEvent(this, "Timer");
    }
  }

  /**
   * Interval property getter method.
   *
   * @return timer interval in ms
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description ="Interval between timer events in ms")
  public int TimerInterval() {
    return timerInternal.Interval();
  }

  /**
   * Interval property setter method: sets the interval between timer events.
   *
   * @param interval timer interval in ms
   */
  @DesignerProperty(
      editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
      defaultValue = DEFAULT_INTERVAL + "")
  @SimpleProperty
  public void TimerInterval(int interval) {
    timerInternal.Interval(interval);
  }

  /**
   * Enabled property getter method.
   *
   * @return {@code true} indicates a running timer, {@code false} a stopped
   *         timer
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "Fires timer if true")
```

```
123:   public boolean TimerEnabled() {
124:     return timerInternal.Enabled();
125:   }
126:
127:   /**
128:    * Enabled property setter method: starts or stops the timer.
129:    *
130:    * @param enabled {@code true} starts the timer, {@code false} stops it
131:    */
132:   @DesignerProperty(
133:       editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
134:       defaultValue = DEFAULT_ENABLED ? "True" : "False")
135:   @SimpleProperty
136:   public void TimerEnabled(boolean enabled) {
137:     timerInternal.Enabled(enabled);
138:   }
139:
140:   /**
141:    * TimerAlwaysFires property getter method.
142:    *
143:    *  return {@code true} if the timer event will fire even if the application
144:    *   is not on the screen
145:    */
146:   @SimpleProperty(
147:       category = PropertyCategory.BEHAVIOR,
148:       description = "Will fire even when application is not showing on the "
149:       + "screen if true")
150:   public boolean TimerAlwaysFires() {
151:     return timerAlwaysFires;
152:   }
153:
154:   /**
155:    * TimerAlwaysFires property setter method: instructs when to disable
156:    *
157:    *  @param always {@code true} if the timer event should fire even if the
158:    *  application is not on the screen
159:    */
160:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaul
tValue = "True")
161:   @SimpleProperty
162:   public void TimerAlwaysFires(boolean always) {
163:     timerAlwaysFires = always;
164:   }
165:
166:   // AlarmHandler implementation
167:
168:   @Override
169:   public void alarm() {
170:     Timer();
171:   }
172:
173:   /**
174:    * Returns the current system time in milliseconds.
175:    *
176:    * @return  current system time in milliseconds
177:    */
178:   @SimpleFunction (description = "The phone's internal time")
179:   public static long SystemTime() {
180:     return Dates.Timer();
181:   }
182:
183:   @SimpleFunction(description = "The current instant in time read from "
184:       + "phone's clock")
185:   public static Calendar Now() {
186:     return Dates.Now();
187:   }
188:
189:   /**
190:    * An instant in time specified by MM/DD/YYYY hh:mm:ss or MM/DD/YYYY or hh:mm
191:    * where MM is the month (01-12), DD the day (01-31), YYYY the year
192:    * (0000-9999), hh the hours (00-23), mm the minutes (00-59) and ss
193:    * the seconds (00-59).
194:    *
195:    * @param from  string to convert
196:    * @return  date
197:    */
198:   @SimpleFunction(
199:       description = "An instant specified by MM/DD/YYYY hh:mm:ss or MM/DD/YYYY or hh
:mm")
200:   public static Calendar MakeInstant(String from) {
201:     try {
202:       return Dates.DateValue(from);
203:     } catch (IllegalArgumentException e) {
204:       throw new YailRuntimeError(
205:           "Argument to MakeInstant should have form MM/DD/YYYY, hh:mm:ss, or MM/DD/Y
YYY or hh:mm",
206:           "Sorry to be so picky.");
207:     }
208:   }
209:
210:   /**
211:    * Create an Calendar from ms since 1/1/1970 00:00:00.0000
212:    * Probably should go in Calendar.
213:    *
214:    * @param millis raw millisecond number.
215:    */
216:   @SimpleFunction(description = "An instant in time specified by the milliseconds si
nce 1970.")
217:   public static Calendar MakeInstantFromMillis(long millis) {
218:     Calendar instant = Dates.Now(); // just to get our hands on an instant
219:     instant.setTimeInMillis(millis);
220:     return instant;
221:   }
222:
223:   /**
224:    * Calendar property getter method: gets the raw millisecond representation of
225:    *  a Calendar.
226:    * @param instant Calendar
227:    * @return milliseconds since 1/1/1970.
228:    */
229:   @SimpleFunction (description = "The instant in time measured as milliseconds since
 1970.")
230:   public static long GetMillis(Calendar instant) {
231:     return instant.getTimeInMillis();
232:   }
233:
234:   @SimpleFunction(description = "An instant in time some seconds after the argument"
)
235:   public static Calendar AddSeconds(Calendar instant, int seconds) {
236:     Calendar newInstant = (Calendar) instant.clone();
237:     Dates.DateAdd(newInstant, Calendar.SECOND, seconds);
238:     return newInstant;
239:   }
240:
```

```
  @SimpleFunction(description = "An instant in time some minutes after the argument"
)
  public static Calendar AddMinutes(Calendar instant, int minutes) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.MINUTE, minutes);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some hours after the argument")
  public static Calendar AddHours(Calendar instant, int hours) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.HOUR_OF_DAY, hours);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some days after the argument")
  public static Calendar AddDays(Calendar instant, int days) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.DATE, days);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some weeks after the argument")
  public static Calendar AddWeeks(Calendar instant, int weeks) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.WEEK_OF_YEAR, weeks);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some months after the argument")
  public static Calendar AddMonths(Calendar instant, int months) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.MONTH, months);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some years after the argument")
  public static Calendar AddYears(Calendar instant, int years) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.YEAR, years);
    return newInstant;
  }

  /**
   * Returns the milliseconds by which end follows start (+ or -)
   *
   * @param start beginning instant
   * @param end ending instant
   * @return  milliseconds
   */
  @SimpleFunction (description = "Milliseconds elapsed between instants")
  public static long Duration(Calendar start, Calendar end) {
    return end.getTimeInMillis() - start.getTimeInMillis();
  }

  /**
   * Returns the seconds for the given instant.
   *
   * @param instant  instant to use seconds of
   * @return  seconds (range 0 - 59)
   */
  @SimpleFunction (description = "The second of the minute")
  public static int Second(Calendar instant) {
    return Dates.Second(instant);
  }

/**
   * Returns the minutes for the given date.
   *
   * @param instant instant to use minutes of
   * @return  minutes (range 0 - 59)
   */
  @SimpleFunction(description = "The minute of the hour")
  public static int Minute(Calendar instant) {
    return Dates.Minute(instant);
  }

  /**
   * Returns the hours for the given date.
   *
   * @param instant Calendar to use hours of
   * @return  hours (range 0 - 23)
   */
  @SimpleFunction (description = "The hour of the day")
  public static int Hour(Calendar instant) {
    return Dates.Hour(instant);
  }

  /**
   * Returns the day of the month.
   *
   * @param instant  instant to use day of the month of
   * @return  day: [1...31]
   */
  @SimpleFunction (description = "The day of the month")
  public static int DayOfMonth(Calendar instant) {
    return Dates.Day(instant);
  }

  /**
   * Returns the weekday for the given instant.
   *
   * @param instant  instant to use day of week of
   * @return day of week: [1...7] starting with Sunday
   */
  @SimpleFunction (description = "The day of the week represented as a "
      + "number from 1 (Sunday) to 7 (Saturday)")
  public static int Weekday(Calendar instant) {
    return Dates.Weekday(instant);
  }

  /**
   * Returns the name of the weekday for the given instant.
   *
   * @param instant  instant to use weekday of
   * @return  weekday, as a string.
   */
  @SimpleFunction (description = "The name of the day of the week")
  public static String WeekdayName(Calendar instant) {
    return Dates.WeekdayName(instant);
  }

  /**
   * Returns the number of the month for the given instant.
```

```
364:    *
365:    * @param instant  instant to use month of
366:    * @return  number of month
367:    */
368:   @SimpleFunction (description = "The month of the year represented as a "
369:       + "number from 1 to 12)")
370:   public static int Month(Calendar instant) {
371:     return Dates.Month(instant) + 1;
372:   }
373:
374:   /**
375:    * Returns the name of the month for the given instant.
376:    *
377:    * @param instant  instant to use month of
378:    * @return  name of month
379:    */
380:   @SimpleFunction (description = "The name of the month")
381:   public static String MonthName(Calendar instant) {
382:     return Dates.MonthName(instant);
383:   }
384:
385:  /**
386:    * Returns the year of the given instant.
387:    *
388:    * @param instant  instant to use year of
389:    * @return  year
390:    */
391:   @SimpleFunction(description = "The year")
392:   public static int Year(Calendar instant) {
393:     return Dates.Year(instant);
394:   }
395:
396:   /**
397:    * Converts and formats the given instant into a string.   *
398:    *
399:    * @param instant  instant to format
400:    * @return  formatted instant
401:    */
402:   @SimpleFunction (description = "Text representing the date and time of an"
403:       + " instant")
404:   public static String FormatDateTime(Calendar instant) {
405:     return Dates.FormatDateTime(instant);
406:   }
407:
408:   /**
409:    * Converts and formats the given instant into a string.
410:    *
411:    * @param instant  instant to format
412:    * @return  formatted instant
413:    */
414:   @SimpleFunction (description = "Text representing the date of an instant")
415:   public static String FormatDate(Calendar instant) {
416:     return Dates.FormatDate(instant);
417:   }
418:
419:   /**
420:    * Converts and formats the given instant into a string.
421:    *
422:    * @param instant  instant to format
423:    * @return  formatted instant
424:    */
425:   @SimpleFunction (description = "Text representing the time of an instant")
426:   public static String FormatTime(Calendar instant) {
427:     return Dates.FormatTime(instant);
428:   }
429:
430:   @Override
431:   public void onStop() {
432:     onScreen = false;
433:   }
434:
435:   @Override
436:   public void onResume() {
437:     onScreen = true;
438:   }
439:
440:   @Override
441:   public void onDestroy() {
442:     timerInternal.Enabled(false);
443:   }
444:
445:   @Override
446:   public void onDelete() {
447:     timerInternal.Enabled(false);
448:   }
449: }
```

```
    1:  // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2009-2011 Google, All Rights reserved
    3: // Copyright 2011-2012 MIT, All rights reserved
    4: // Released under the Apache License, Version 2.0
    5: // http://www.apache.org/licenses/LICENSE-2.0
    6:
    7: // ********************************************
    8: // If we're not going to go this route with onDestroy, then at least get rid of the
DEBUG flag.
    9:
   10: package com.google.appinventor.components.runtime;
   11:
   12: import java.io.IOException;
   13: import java.lang.reflect.InvocationTargetException;
   14: import java.lang.reflect.Method;
   15: import java.util.HashMap;
   16: import java.util.List;
   17: import java.util.Map;
   18: import java.util.Set;
   19:
   20: import org.json.JSONException;
   21:
   22: import android.app.Activity;
   23: import android.app.Dialog;
   24: import android.content.ActivityNotFoundException;
   25: import android.content.Intent;
   26: import android.content.pm.ActivityInfo;
   27: import android.content.res.Configuration;
   28: import android.graphics.drawable.Drawable;
   29: import android.os.Bundle;
   30: import android.os.Handler;
   31: import android.text.Html;
   32: import android.util.Log;
   33: import android.view.KeyEvent;
   34: import android.view.Menu;
   35: import android.view.MenuItem;
   36: import android.view.MenuItem.OnMenuItemClickListener;
   37: import android.view.ViewGroup;
   38: import android.view.WindowManager;
   39: import android.widget.FrameLayout;
   40: import android.widget.ScrollView;
   41:
   42: import com.google.appinventor.components.annotations.DesignerComponent;
   43: import com.google.appinventor.components.annotations.DesignerProperty;
   44: import com.google.appinventor.components.annotations.PropertyCategory;
   45: import com.google.appinventor.components.annotations.SimpleEvent;
   46: import com.google.appinventor.components.annotations.SimpleObject;
   47: import com.google.appinventor.components.annotations.SimpleProperty;
   48: import com.google.appinventor.components.annotations.UsesPermissions;
   49: import com.google.appinventor.components.common.ComponentCategory;
   50: import com.google.appinventor.components.common.ComponentConstants;
   51: import com.google.appinventor.components.common.PropertyTypeConstants;
   52: import com.google.appinventor.components.common.YaVersion;
   53: import com.google.appinventor.components.runtime.collect.Lists;
   54: import com.google.appinventor.components.runtime.collect.Maps;
   55: import com.google.appinventor.components.runtime.collect.Sets;
   56: import com.google.appinventor.components.runtime.util.AlignmentUtil;
   57: import com.google.appinventor.components.runtime.util.AnimationUtil;
   58: import com.google.appinventor.components.runtime.util.ErrorMessages;
   59: import com.google.appinventor.components.runtime.util.FullScreenVideoUtil;
   60: import com.google.appinventor.components.runtime.util.JsonUtil;
   61: import com.google.appinventor.components.runtime.util.MediaUtil;
   62: import com.google.appinventor.components.runtime.util.OnInitializeListener;
   63: import com.google.appinventor.components.runtime.util.SdkLevel;
   64: import com.google.appinventor.components.runtime.util.ViewUtil;
   65:
   66: /**
   67:  * Component underlying activities and UI apps, not directly accessible to Simple pr
ogrammers.
   68:  *
   69:  * <p>This is the root container of any Android activity and also the
   70:  * superclass for for Simple/Android UI applications.
   71:  *
   72:  * The main form is always named "Screen1".
   73:  *
   74:  */
   75: @DesignerComponent(version = YaVersion.FORM_COMPONENT_VERSION,
   76:     category = ComponentCategory.LAYOUT,
   77:     description = "Top-level component containing all other components in the progra
m",
   78:     showOnPalette = false)
   79: @SimpleObject
   80: @UsesPermissions(permissionNames = "android.permission.INTERNET,android.permission.A
CCESS_WIFI_STATE,android.permission.ACCESS_NETWORK_STATE")
   81: public class Form extends Activity
   82:     implements Component, ComponentContainer, HandlesEventDispatching {
   83:   private static final String LOG_TAG = "Form";
   84:
   85:   private static final String RESULT_NAME = "APP_INVENTOR_RESULT";
   86:
   87:   private static final String ARGUMENT_NAME = "APP_INVENTOR_START";
   88:
   89:   public static final String APPINVENTOR_URL_SCHEME = "appinventor";
   90:
   91:   // Keep track of the current form object.
   92:   // activeForm always holds the Form that is currently handling event dispatching s
o runtime.scm
   93:   // can lookup symbols in the correct environment.
   94:   // There is at least one case where an event can be fired when the activity is not
 the foreground
   95:   // activity: if a Clock component's TimerAlwaysFires property is true, the Clock c
omponent's
   96:   // Timer event will still fire, even when the activity is no longer in the foregro
und. For this
   97:   // reason, we cannot assume that the activeForm is the foreground activity.
   98:   protected static Form activeForm;
   99:
  100:   // applicationIsBeingClosed is set to true during closeApplication.
  101:   private static boolean applicationIsBeingClosed;
  102:
  103:   private final Handler androidUIHandler = new Handler();
  104:
  105:   protected String formName;
  106:
  107:   private boolean screenInitialized;
  108:
  109:   private static final int SWITCH_FORM_REQUEST_CODE = 1;
  110:   private static int nextRequestCode = SWITCH_FORM_REQUEST_CODE + 1;
  111:
  112:   // Backing for background color
  113:   private int backgroundColor;
  114:
  115:   // Information string the app creator can set.  It will be shown when
  116:   // "about this application" menu item is selected.
```

```
  117:   private String aboutScreen;
  118:
  119:   private String backgroundImagePath = "";
  120:   private Drawable backgroundDrawable;
  121:
  122:   // Layout
  123:   private LinearLayout viewLayout;
  124:
  125:   // translates App Inventor alignment codes to Android gravity
  126:   private AlignmentUtil alignmentSetter;
  127:
  128:   // the alignment for this component's LinearLayout
  129:   private int horizontalAlignment;
  130:   private int verticalAlignment;
  131:
  132:   // String representing the transition animation type
  133:   private String openAnimType;
  134:   private String closeAnimType;
  135:
  136:   private FrameLayout frameLayout;
  137:   private boolean scrollable;
  138:
  139:   // Application lifecycle related fields
  140:   private final HashMap<Integer, ActivityResultListener> activityResultMap = Maps.ne
wHashMap();
  141:   private final Set<OnStopListener> onStopListeners = Sets.newHashSet();
  142:   private final Set<OnNewIntentListener> onNewIntentListeners = Sets.newHashSet();
  143:   private final Set<OnResumeListener> onResumeListeners = Sets.newHashSet();
  144:   private final Set<OnPauseListener> onPauseListeners = Sets.newHashSet();
  145:   private final Set<OnDestroyListener> onDestroyListeners = Sets.newHashSet();
  146:
  147:   // AppInventor lifecycle: listeners for the Initialize Event
  148:   private final Set<OnInitializeListener> onInitializeListeners = Sets.newHashSet();
  149:
  150:   // Set to the optional String-valued Extra passed in via an Intent on startup.
  151:   // This is passed directly in the Repl.
  152:   protected String startupValue = "";
  153:
  154:   // To control volume of error complaints
  155:   private static long minimumToastWait = 10000000000L; // 10 seconds
  156:   private long lastToastTime = System.nanoTime() - minimumToastWait;
  157:
  158:   // In a multiple screen application, when a secondary screen is opened, nextFormNa
me is set to
  159:   // the name of the secondary screen. It is saved so that it can be passed to the O
therScreenClosed
  160:   // event.
  161:   private String nextFormName;
  162:
  163:   private FullScreenVideoUtil fullScreenVideoUtil;
  164:
  165:   @Override
  166:   public void onCreate(Bundle icicle) {
  167:     // Called when the activity is first created
  168:     super.onCreate(icicle);
  169:
  170:     // Figure out the name of this form.
  171:     String className = getClass().getName();
  172:     int lastDot = className.lastIndexOf('.');
  173:     formName = className.substring(lastDot + 1);
  174:     Log.d(LOG_TAG, "Form " + formName + " got onCreate");
  175:
  176:     activeForm = this;
  177:     Log.i(LOG_TAG, "activeForm is now " + activeForm.formName);
  178:
  179:     viewLayout = new LinearLayout(this, ComponentConstants.LAYOUT_ORIENTATION_VERTIC
AL);
  180:     alignmentSetter = new AlignmentUtil(viewLayout);
  181:
  182:     defaultPropertyValues();
  183:
  184:     // Get startup text if any before adding components
  185:     Intent startIntent = getIntent();
  186:     if (startIntent != null && startIntent.hasExtra(ARGUMENT_NAME)) {
  187:       startupValue = startIntent.getStringExtra(ARGUMENT_NAME);
  188:     }
  189:
  190:     fullScreenVideoUtil = new FullScreenVideoUtil(this, androidUIHandler);
  191:
  192:     // Set soft keyboard to not cover the focused UI element, e.g., when you are typ
ing
  193:     // into a textbox near the bottom of the screen.
  194:     WindowManager.LayoutParams params = getWindow().getAttributes();
  195:     int softInputMode = params.softInputMode;
  196:     getWindow().setSoftInputMode(
  197:         softInputMode | WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE);
  198:
  199:     // Add application components to the form
  200:     $define();
  201:
  202:     // Special case for Event.Initialize(): all other initialize events are triggere
d after
  203:     // completing the constructor. This doesn't work for Android apps though because
 this method
  204:     // is called after the constructor completes and therefore the Initialize event
would run
  205:     // before initialization finishes. Instead the compiler suppresses the invocatio
n of the
  206:     // event and leaves it up to the library implementation.
  207:     Initialize();
  208:   }
  209:
  210:   private void defaultPropertyValues() {
  211:     Scrollable(false); // frameLayout is created in Scrollable()
  212:     BackgroundImage("");
  213:     AboutScreen("");
  214:     BackgroundColor(Component.COLOR_WHITE);
  215:     AlignHorizontal(ComponentConstants.GRAVITY_LEFT);
  216:     AlignVertical(ComponentConstants.GRAVITY_TOP);
  217:     Title("");
  218:   }
  219:
  220:   @Override
  221:   public void onConfigurationChanged(Configuration newConfig) {
  222:     super.onConfigurationChanged(newConfig);
  223:
  224:     final int newOrientation = newConfig.orientation;
  225:     if (newOrientation == Configuration.ORIENTATION_LANDSCAPE ||
  226:         newOrientation == Configuration.ORIENTATION_PORTRAIT) {
  227:       // At this point, the screen has not be resized to match the new orientation.
  228:       // We use Handler.post so that we'll dispatch the ScreenOrientationChanged eve
nt after the
  229:       // screen has been resized to match the new orientation.
  230:
```

```
231:       androidUIHandler.post(new Runnable() {
232:         public void run() {
233:           boolean dispatchEventNow = false;
234:           if (frameLayout != null) {
235:             if (newOrientation == Configuration.ORIENTATION_LANDSCAPE) {
236:               if (frameLayout.getWidth() >= frameLayout.getHeight()) {
237:                 dispatchEventNow = true;
238:               }
239:             } else { // Portrait
240:               if (frameLayout.getHeight() >= frameLayout.getWidth()) {
241:                 dispatchEventNow = true;
242:               }
243:             }
244:           }
245:           if (dispatchEventNow) {
246:             ScreenOrientationChanged();
247:           } else {
248:             // Try again later.
249:             androidUIHandler.post(this);
250:           }
251:         }
252:       });
253:     }
254:   }
255:
256:   /*
257:    * Here we override the hardware back button, just to make sure
258:    * that the closing screen animation is applied. (In API level
259:    * 5, we can simply override the onBackPressed method rather
260:    * than bothering with onKeyDown)
261:    */
262:   @Override
263:   public boolean onKeyDown(int keyCode, KeyEvent event) {
264:     if (keyCode == KeyEvent.KEYCODE_BACK) {
265:       if (!BackPressed()) {
266:         boolean handled = super.onKeyDown(keyCode, event);
267:         AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
268:         return handled;
269:       } else {
270:         return true;
271:       }
272:     }
273:     return super.onKeyDown(keyCode, event);
274:   }
275:
276:   @SimpleEvent(description = "Device back button pressed.")
277:   public boolean BackPressed() {
278:     return EventDispatcher.dispatchEvent(this, "BackPressed");
279:   }
280:
281:   // onActivityResult should be triggered in only two cases:
282:   // (1) The result is for some other component in the app, not this Form itself
283:   // (2) This page started another page, and that page is closing, and passing
284:   // its value back as a JSON-encoded string in the intent.
285:
286:   @Override
287:   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
288:     Log.i(LOG_TAG, "Form " + formName + " got onActivityResult, requestCode = " +
289:         requestCode + ", resultCode = " + resultCode);
290:     if (requestCode == SWITCH_FORM_REQUEST_CODE) {
291:       // Assume this is a multiple screen application, and a secondary
292:       // screen has closed.  Process the result as a JSON-encoded string.
293:       // This can also happen if the user presses the back button, in which case
294:       // there's no data.
295:       String resultString;
296:       if (data != null && data.hasExtra(RESULT_NAME)) {
297:         resultString = data.getStringExtra(RESULT_NAME);
298:       } else {
299:         resultString = "";
300:       }
301:       Object decodedResult = decodeJSONStringForForm(resultString, "other screen clo
sed");
302:       // nextFormName was set when this screen opened the secondary screen
303:       OtherScreenClosed(nextFormName, decodedResult);
304:     } else {
305:       // Another component (such as a ListPicker, ActivityStarter, etc) is expecting
 this result.
306:       ActivityResultListener component = activityResultMap.get(requestCode);
307:       if (component != null) {
308:         component.resultReturned(requestCode, resultCode, data);
309:       }
310:     }
311:   }
312:
313:   // functionName is a string to include in the error message that will be shown
314:   // if the JSON decoding fails
315:   private  static Object decodeJSONStringForForm(String jsonString, String functionN
ame) {
316:     Log.i(LOG_TAG, "decodeJSONStringForForm -- decoding JSON representation:" + json
String);
317:     Object valueFromJSON = "";
318:     try {
319:       valueFromJSON = JsonUtil.getObjectFromJson(jsonString);
320:       Log.i(LOG_TAG, "decodeJSONStringForForm -- got decoded JSON:" + valueFromJSON.
toString());
321:     } catch (JSONException e) {
322:       activeForm.dispatchErrorOccurredEvent(activeForm, functionName,
323:           // showing the start value here will produce an ugly error on the phone, b
ut it's
324:           // more useful than not showing the value
325:           ErrorMessages.ERROR_SCREEN_BAD_VALUE_RECEIVED, jsonString);
326:     }
327:     return valueFromJSON;
328:   }
329:
330:   public int registerForActivityResult(ActivityResultListener listener) {
331:     int requestCode = generateNewRequestCode();
332:     activityResultMap.put(requestCode, listener);
333:     return requestCode;
334:   }
335:
336:   public void unregisterForActivityResult(ActivityResultListener listener) {
337:     List<Integer> keysToDelete = Lists.newArrayList();
338:     for (Map.Entry<Integer, ActivityResultListener> mapEntry : activityResultMap.ent
rySet()) {
339:       if (listener.equals(mapEntry.getValue())) {
340:         keysToDelete.add(mapEntry.getKey());
341:       }
342:     }
343:     for (Integer key : keysToDelete) {
344:       activityResultMap.remove(key);
345:     }
346:   }
347:
```

```
348:   private static int generateNewRequestCode() {
349:     return nextRequestCode++;
350:   }
351:
352:   @Override
353:   protected void onResume() {
354:     super.onResume();
355:     Log.i(LOG_TAG, "Form " + formName + " got onResume");
356:     activeForm = this;
357:
358:     // If applicationIsBeingClosed is true, call closeApplication() immediately to c
ontinue
359:     // unwinding through all forms of a multi-screen application.
360:     if (applicationIsBeingClosed) {
361:       closeApplication();
362:       return;
363:     }
364:
365:     for (OnResumeListener onResumeListener : onResumeListeners) {
366:       onResumeListener.onResume();
367:     }
368:   }
369:
370:   public void registerForOnResume(OnResumeListener component) {
371:     onResumeListeners.add(component);
372:   }
373:
374:   /**
375:    * An app can register to be notified when App Inventor's Initialize
376:    * block has fired.  They will be called in Initialize().
377:    *
378:    * @param component
379:    */
380:   public void registerForOnInitialize(OnInitializeListener component) {
381:     onInitializeListeners.add(component);
382:   }
383:
384:   @Override
385:   protected void onNewIntent(Intent intent) {
386:     super.onNewIntent(intent);
387:     Log.d(LOG_TAG, "Form " + formName + " got onNewIntent " + intent);
388:     for (OnNewIntentListener onNewIntentListener : onNewIntentListeners) {
389:       onNewIntentListener.onNewIntent(intent);
390:     }
391:   }
392:
393:   public void registerForOnNewIntent(OnNewIntentListener component) {
394:     onNewIntentListeners.add(component);
395:   }
396:
397:   @Override
398:   protected void onPause() {
399:     super.onPause();
400:     Log.i(LOG_TAG, "Form " + formName + " got onPause");
401:     for (OnPauseListener onPauseListener : onPauseListeners) {
402:       onPauseListener.onPause();
403:     }
404:   }
405:
406:   public void registerForOnPause(OnPauseListener component) {
407:     onPauseListeners.add(component);
408:   }
409:
410:   @Override
411:   protected void onStop() {
412:     super.onStop();
413:     Log.i(LOG_TAG, "Form " + formName + " got onStop");
414:     for (OnStopListener onStopListener : onStopListeners) {
415:       onStopListener.onStop();
416:     }
417:   }
418:
419:   public void registerForOnStop(OnStopListener component) {
420:     onStopListeners.add(component);
421:   }
422:
423:   @Override
424:   protected void onDestroy() {
425:     super.onDestroy();
426:     // for debugging and future growth
427:     Log.i(LOG_TAG, "Form " + formName + " got onDestroy");
428:
429:     // Unregister events for components in this form.
430:     EventDispatcher.removeDispatchDelegate(this);
431:
432:     for (OnDestroyListener onDestroyListener : onDestroyListeners) {
433:       onDestroyListener.onDestroy();
434:     }
435:   }
436:
437:   public void registerForOnDestroy(OnDestroyListener component) {
438:     onDestroyListeners.add(component);
439:   }
440:
441:   public Dialog onCreateDialog(int id) {
442:     switch(id) {
443:     case FullScreenVideoUtil.FULLSCREEN_VIDEO_DIALOG_FLAG:
444:       return fullScreenVideoUtil.createFullScreenVideoDialog();
445:     default:
446:       return super.onCreateDialog(id);
447:     }
448:   }
449:
450:   public void onPrepareDialog(int id, Dialog dialog) {
451:     switch(id) {
452:     case FullScreenVideoUtil.FULLSCREEN_VIDEO_DIALOG_FLAG:
453:       fullScreenVideoUtil.prepareFullScreenVideoDialog(dialog);
454:       break;
455:     default:
456:       super.onPrepareDialog(id, dialog);
457:     }
458:   }
459:
460:   /**
461:    * Compiler-generated method to initialize and add application components to
462:    * the form.  We just provide an implementation here to artificially make
463:    * this class concrete so that it is included in the documentation and
464:    * Codeblocks language definition file generated by
465:    * {@link com.google.appinventor.components.scripts.DocumentationGenerator} and
466:    * {@link com.google.appinventor.components.scripts.LangDefXmlGenerator},
467:    * respectively.  The actual implementation appears in {@code runtime.scm}.
468:    */
469:   protected void $define() {    // This must be declared protected because we are ca
lled from Screen1 which subclasses
```

```
470:                                  // us and isn't in our package.
471:     throw new UnsupportedOperationException();
472:   }
473:
474:   @Override
475:   public boolean canDispatchEvent(Component component, String eventName) {
476:     // Events can only be dispatched after the screen initialized event has complete
d.
477:     boolean canDispatch = screenInitialized ||
478:         (component == this && eventName.equals("Initialize"));
479:
480:     if (canDispatch) {
481:       // Set activeForm to this before the event is dispatched.
482:       // runtime.scm will call getActiveForm() when the event handler executes.
483:       activeForm = this;
484:     }
485:
486:     return canDispatch;
487:   }
488:
489:   /**
490:    * A trivial implementation to artificially make this class concrete so
491:    * that it is included in the documentation and
492:    * Codeblocks language definition file generated by
493:    * {@link com.google.appinventor.components.scripts.DocumentationGenerator} and
494:    * {@link com.google.appinventor.components.scripts.LangDefXmlGenerator},
495:    * respectively.  The actual implementation appears in {@code runtime.scm}.
496:    */
497:   @Override
498:   public boolean dispatchEvent(Component component, String componentName, String eve
ntName,
499:       Object[] args) {
500:     throw new UnsupportedOperationException();
501:   }
502:
503:
504:   /**
505:    * Initialize event handler.
506:    */
507:   @SimpleEvent(description = "Screen starting")
508:   public void Initialize() {
509:     // Dispatch the Initialize event only after the screen's width and height are no
 longer zero.
510:     androidUIHandler.post(new Runnable() {
511:       public void run() {
512:         if (frameLayout != null && frameLayout.getWidth() != 0 && frameLayout.getHei
ght() != 0) {
513:           EventDispatcher.dispatchEvent(Form.this, "Initialize");
514:           screenInitialized = true;
515:
516:           //  Call all apps registered to be notified when Initialize Event is dispa
tched
517:           for (OnInitializeListener onInitializeListener : onInitializeListeners) {
518:             onInitializeListener.onInitialize();
519:           }
520:           if (activeForm instanceof ReplForm) { // We are the Companion
521:             ((ReplForm)activeForm).HandleReturnValues();
522:           }
523:         } else {
524:           // Try again later.
525:           androidUIHandler.post(this);
526:         }
527:       }
528:     });
529:   }
530:
531:   @SimpleEvent(description = "Screen orientation changed")
532:   public void ScreenOrientationChanged() {
533:     EventDispatcher.dispatchEvent(this, "ScreenOrientationChanged");
534:   }
535:
536:   /**
537:    * ErrorOccurred event handler.
538:    */
539:   @SimpleEvent(
540:       description = "Event raised when an error occurs. Only some errors will " +
541:       "raise this condition.  For those errors, the system will show a notification
" +
542:       "by default.  You can use this event handler to prescribe an error " +
543:       "behavior different than the default.")
544:   public void ErrorOccurred(Component component, String functionName, int errorNumbe
r,
545:       String message) {
546:     String componentType = component.getClass().getName();
547:     componentType = componentType.substring(componentType.lastIndexOf(".") + 1);
548:     Log.e(LOG_TAG, "Form " + formName + " ErrorOccurred, errorNumber = " + errorNumb
er +
549:         ", componentType = " + componentType + ", functionName = " + functionName +
550:         ", messages = " + message);
551:     if ((!(EventDispatcher.dispatchEvent(
552:         this, "ErrorOccurred", component, functionName, errorNumber, message)))
553:         && screenInitialized)  {
554:       // If dispatchEvent returned false, then no user-supplied error handler was ru
n.
555:       // If in addition, the screen initializer was run, then we assume that the
556:       // user did not provide an error handler.   In this case, we run a default
557:       // error handler, namely, showing a notification to the end user of the app.
558:       // The app writer can override this by providing an error handler.
559:       new Notifier(this).ShowAlert("Error " + errorNumber + ": " + message);
560:     }
561:   }
562:
563:
564:   public void ErrorOccurredDialog(Component component, String functionName, int erro
rNumber,
565:       String message, String title, String buttonText) {
566:     String componentType = component.getClass().getName();
567:     componentType = componentType.substring(componentType.lastIndexOf(".") + 1);
568:     Log.e(LOG_TAG, "Form " + formName + " ErrorOccurred, errorNumber = " + errorNumb
er +
569:         ", componentType = " + componentType + ", functionName = " + functionName +
570:         ", messages = " + message);
571:     if ((!(EventDispatcher.dispatchEvent(
572:         this, "ErrorOccurred", component, functionName, errorNumber, message)))
573:         && screenInitialized)  {
574:       // If dispatchEvent returned false, then no user-supplied error handler was ru
n.
575:       // If in addition, the screen initializer was run, then we assume that the
576:       // user did not provide an error handler.   In this case, we run a default
577:       // error handler, namely, showing a message dialog to the end user of the app.
578:       // The app writer can override this by providing an error handler.
579:       new Notifier(this).ShowMessageDialog("Error " + errorNumber + ": " + message,
title, buttonText);
580:     }
```

```
  581:   }
  582:
  583:
  584:   public void dispatchErrorOccurredEvent(final Component component, final String fun
ctionName,
  585:       final int errorNumber, final Object... messageArgs) {
  586:     runOnUiThread(new Runnable() {
  587:       public void run() {
  588:         String message = ErrorMessages.formatMessage(errorNumber, messageArgs);
  589:         ErrorOccurred(component, functionName, errorNumber, message);
  590:       }
  591:     });
  592:   }
  593:
  594:   // This is like dispatchErrorOccurred, except that it defaults to showing
  595:   // a message dialog rather than an alert.   The app writer can override either of
these behaviors,
  596:   // but using the event dialog version frees the app writer of the need to explicit
ly override
  597:   // the alert behavior in the case
  598:   // where a message dialog is what's generally needed.
  599:   public void dispatchErrorOccurredEventDialog(final Component component, final Stri
ng functionName,
  600:       final int errorNumber, final Object... messageArgs) {
  601:     runOnUiThread(new Runnable() {
  602:       public void run() {
  603:         String message = ErrorMessages.formatMessage(errorNumber, messageArgs);
  604:         ErrorOccurredDialog(
  605:             component,
  606:             functionName,
  607:             errorNumber,
  608:             message,
  609:             "Error in " + functionName,
  610:             "Dismiss");
  611:       }
  612:     });
  613:   }
  614:
  615:
  616:
  617:   /**
  618:    * Scrollable property getter method.
  619:    *
  620:    * @return  true if the screen is vertically scrollable
  621:    */
  622:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
  623:     description = "When checked, there will be a vertical scrollbar on the "
  624:     + "screen, and the height of the application can exceed the physical "
  625:     + "height of the device. When unchecked, the application height is "
  626:     + "constrained to the height of the device.")
  627:   public boolean Scrollable() {
  628:     return scrollable;
  629:   }
  630:
  631:   /**
  632:    * Scrollable property setter method.
  633:    *
  634:    * @param scrollable  true if the screen should be vertically scrollable
  635:    */
  636:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
  637:     defaultValue = "False")
  638:   @SimpleProperty
  639:   public void Scrollable(boolean scrollable) {
  640:     if (this.scrollable == scrollable && frameLayout != null) {
  641:       return;
  642:     }
  643:
  644:     // Remove our view from the current frameLayout.
  645:     if (frameLayout != null) {
  646:       frameLayout.removeAllViews();
  647:     }
  648:
  649:     this.scrollable = scrollable;
  650:
  651:     frameLayout = scrollable ? new ScrollView(this) : new FrameLayout(this);
  652:     frameLayout.addView(viewLayout.getLayoutManager(), new ViewGroup.LayoutParams(
  653:         ViewGroup.LayoutParams.MATCH_PARENT,
  654:         ViewGroup.LayoutParams.MATCH_PARENT));
  655:
  656:     frameLayout.setBackgroundColor(backgroundColor);
  657:     if (backgroundDrawable != null) {
  658:       ViewUtil.setBackgroundImage(frameLayout, backgroundDrawable);
  659:     }
  660:
  661:     setContentView(frameLayout);
  662:     frameLayout.requestLayout();
  663:   }
  664:
  665:   /**
  666:    * BackgroundColor property getter method.
  667:    *
  668:    * @return  background RGB color with alpha
  669:    */
  670:   @SimpleProperty(category = PropertyCategory.APPEARANCE)
  671:   public int BackgroundColor() {
  672:     return backgroundColor;
  673:   }
  674:
  675:   /**
  676:    * BackgroundColor property setter method.
  677:    *
  678:    * @param argb  background RGB color with alpha
  679:    */
  680:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_COLOR,
  681:       defaultValue = Component.DEFAULT_VALUE_COLOR_WHITE)
  682:   @SimpleProperty
  683:   public void BackgroundColor(int argb) {
  684:     backgroundColor = argb;
  685:     if (argb != Component.COLOR_DEFAULT) {
  686:       viewLayout.getLayoutManager().setBackgroundColor(argb);
  687:       // Just setting the background color on the layout manager is insufficient.
  688:       frameLayout.setBackgroundColor(argb);
  689:     } else {
  690:       viewLayout.getLayoutManager().setBackgroundColor(Component.COLOR_WHITE);
  691:       // Just setting the background color on the layout manager is insufficient.
  692:       frameLayout.setBackgroundColor(Component.COLOR_WHITE);
  693:     }
  694:   }
  695:
  696:   /**
  697:    * Returns the path of the background image.
  698:    *
  699:    * @return  the path of the background image
  700:    */
```

```
701:   @SimpleProperty(
702:       category = PropertyCategory.APPEARANCE,
703:       description = "The screen background image.")
704:   public String BackgroundImage() {
705:     return backgroundImagePath;
706:   }
707:
708:
709:   /**
710:    * Specifies the path of the background image.
711:    *
712:    * <p/>See {@link MediaUtil#determineMediaSource} for information about what
713:    * a path can be.
714:    *
715:    * @param path the path of the background image
716:    */
717:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ASSET,
718:       defaultValue = "")
719:   @SimpleProperty(
720:       category = PropertyCategory.APPEARANCE,
721:       description = "The screen background image.")
722:   public void BackgroundImage(String path) {
723:     backgroundImagePath = (path == null) ? "" : path;
724:
725:     try {
726:       backgroundDrawable = MediaUtil.getBitmapDrawable(this, backgroundImagePath);
727:     } catch (IOException ioe) {
728:       Log.e(LOG_TAG, "Unable to load " + backgroundImagePath);
729:       backgroundDrawable = null;
730:     }
731:
732:     ViewUtil.setBackgroundImage(frameLayout, backgroundDrawable);
733:     frameLayout.invalidate();
734:   }
735:
736:   /**
737:    * Title property getter method.
738:    *
739:    * @return  form caption
740:    */
741:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
742:       description = "The caption for the form, which apears in the title bar")
743:   public String Title() {
744:     return getTitle().toString();
745:   }
746:
747:   /**
748:    * Title property setter method: sets a new caption for the form in the
749:    * form's title bar.
750:    *
751:    * @param title  new form caption
752:    */
753:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
754:       defaultValue = "")
755:   @SimpleProperty
756:   public void Title(String title) {
757:     setTitle(title);
758:   }
759:
760:
761:   /**
762:    * AboutScreen property getter method.
763:    *
764:    * @return  AboutScreen string
765:    */
766:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
767:       description = "Information about the screen.  It appears when \"About this App
lication\" "
768:       + "is selected from the system menu. Use it to inform people about your app.
In multiple "
769:       + "screen apps, each screen has its own AboutScreen info.")
770:   public String AboutScreen() {
771:     return aboutScreen;
772:   }
773:
774:   /**
775:    * AboutScreen property setter method: sets a new aboutApp string for the form in
the
776:    * form's "About this application" menu.
777:    *
778:    * @param title  new form caption
779:    */
780:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_TEXTAREA,
781:       defaultValue = "")
782:   @SimpleProperty
783:   public void AboutScreen(String aboutScreen) {
784:     this.aboutScreen = aboutScreen;
785:   }
786:
787:   /**
788:    * The requested screen orientation. Commonly used values are
789:       unspecified (-1), landscape (0), portrait (1), sensor (4), and user (2).  " +
790:       "See the Android developer documentation for ActivityInfo.Screen_Orientation f
or the " +
791:       "complete list of possible settings.
792:    *
793:    * ScreenOrientation property getter method.
794:    *
795:    * @return  screen orientation
796:    */
797:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
798:       description = "The requested screen orientation, specified as a text value.  "
 +
799:       "Commonly used values are " +
800:       "landscape, portrait, sensor, user and unspecified.  " +
801:       "See the Android developer documentation for ActivityInfo.Screen_Orientation f
or the " +
802:       "complete list of possible settings.")
803:   public String ScreenOrientation() {
804:     switch (getRequestedOrientation()) {
805:       case ActivityInfo.SCREEN_ORIENTATION_BEHIND:
806:         return "behind";
807:       case ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE:
808:         return "landscape";
809:       case ActivityInfo.SCREEN_ORIENTATION_NOSENSOR:
810:         return "nosensor";
811:       case ActivityInfo.SCREEN_ORIENTATION_PORTRAIT:
812:         return "portrait";
813:       case ActivityInfo.SCREEN_ORIENTATION_SENSOR:
814:         return "sensor";
815:       case ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED:
816:         return "unspecified";
817:       case ActivityInfo.SCREEN_ORIENTATION_USER:
818:         return "user";
```

```
819:       case 10: // ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR
820:         return "fullSensor";
821:       case 8: // ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE
822:         return "reverseLandscape";
823:       case 9: // ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT
824:         return "reversePortrait";
825:       case 6: // ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE
826:         return "sensorLandscape";
827:       case 7: // ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT
828:         return "sensorPortrait";
829:     }
830:
831:     return "unspecified";
832:   }
833:
834:   /**
835:    * ScreenOrientation property setter method: sets the screen orientation for
836:    * the form.
837:    *
838:    * @param screenOrientation  the screen orientation as a string
839:    */
840:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SCREEN_ORIENTATION,
841:       defaultValue = "unspecified")
842:   @SimpleProperty(category = PropertyCategory.APPEARANCE)
843:   public void ScreenOrientation(String screenOrientation) {
844:     if (screenOrientation.equalsIgnoreCase("behind")) {
845:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_BEHIND);
846:     } else if (screenOrientation.equalsIgnoreCase("landscape")) {
847:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
848:     } else if (screenOrientation.equalsIgnoreCase("nosensor")) {
849:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_NOSENSOR);
850:     } else if (screenOrientation.equalsIgnoreCase("portrait")) {
851:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
852:     } else if (screenOrientation.equalsIgnoreCase("sensor")) {
853:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR);
854:     } else if (screenOrientation.equalsIgnoreCase("unspecified")) {
855:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
856:     } else if (screenOrientation.equalsIgnoreCase("user")) {
857:       setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_USER);
858:     } else if (SdkLevel.getLevel() >= SdkLevel.LEVEL_GINGERBREAD) {
859:       if (screenOrientation.equalsIgnoreCase("fullSensor")) {
860:         setRequestedOrientation(10); // ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR
861:       } else if (screenOrientation.equalsIgnoreCase("reverseLandscape")) {
862:         setRequestedOrientation(8); // ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDSCAPE
863:       } else if (screenOrientation.equalsIgnoreCase("reversePortrait")) {
864:         setRequestedOrientation(9); // ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT
865:       } else if (screenOrientation.equalsIgnoreCase("sensorLandscape")) {
866:         setRequestedOrientation(6); // ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE
867:       } else if (screenOrientation.equalsIgnoreCase("sensorPortrait")) {
868:         setRequestedOrientation(7); // ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT
869:       } else {
870:         dispatchErrorOccurredEvent(this, "ScreenOrientation",
871:             ErrorMessages.ERROR_INVALID_SCREEN_ORIENTATION, screenOrientation);
872:       }
873:     } else {
874:       dispatchErrorOccurredEvent(this, "ScreenOrientation",
875:           ErrorMessages.ERROR_INVALID_SCREEN_ORIENTATION, screenOrientation);
876:     }
877:   }
878:
879:
880:   // Note(halabelson): This section on centering is duplicated between Form and HVArrangement
881:   // I did not see a clean way to abstract it.  Someone should have a look.
882:
883:   // Note(halabelson): The numeric encodings of the alignment specifications are specified
884:   // in ComponentConstants
885:
886:   /**
887:    * Returns a number that encodes how contents of the screen are aligned horizontally.
888:    * The choices are: 1 = left aligned, 2 = horizontally centered, 3 = right aligned
889:    */
890:   @SimpleProperty(
891:       category = PropertyCategory.APPEARANCE,
892:       description = "A number that encodes how contents of the screen are aligned " +
893:           " horizontally. The choices are: 1 = left aligned, 2 = horizontally centered, " +
894:           " 3 = right aligned.")
895:   public int AlignHorizontal() {
896:     return horizontalAlignment;
897:   }
898:
899:   /**
900:    * Sets the horizontal alignment for contents of the screen
901:    *
902:    * @param alignment
903:    */
904:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_HORIZONTAL_ALIGNMENT,
905:       defaultValue = ComponentConstants.HORIZONTAL_ALIGNMENT_DEFAULT + "")
906:   @SimpleProperty
907:   public void AlignHorizontal(int alignment) {
908:     try {
909:       // notice that the throw will prevent the alignment from being changed
910:       // if the argument is illegal
911:       alignmentSetter.setHorizontalAlignment(alignment);
912:       horizontalAlignment = alignment;
913:     } catch (IllegalArgumentException e) {
914:       this.dispatchErrorOccurredEvent(this, "HorizontalAlignment",
915:           ErrorMessages.ERROR_BAD_VALUE_FOR_HORIZONTAL_ALIGNMENT, alignment);
916:     }
917:   }
918:
919:   /**
920:    * Returns a number that encodes how contents of the arrangement are aligned vertically.
921:    * The choices are: 1 = top, 2 = vertically centered, 3 = aligned at the bottom.
922:    * Vertical alignment has no effect if the screen is scrollable.
923:    */
924:   @SimpleProperty(
925:       category = PropertyCategory.APPEARANCE,
926:       description = "A number that encodes how the contents of the arrangement are aligned " +
927:       "vertically. The choices are: 1 = aligned at the top, 2 = vertically centered, " +
928:       "3 = aligned at the bottom. Vertical alignment has no effect if the screen is scrollable.")
```

```
  929:  public int AlignVertical() {
  930:    return verticalAlignment;
  931:  }
  932:
  933:  /**
  934:   * Sets the vertical alignment for contents of the screen
  935:   *
  936:   * @param alignment
  937:   */
  938:  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_VERTICAL_ALIGNME
NT,
  939:      defaultValue = ComponentConstants.VERTICAL_ALIGNMENT_DEFAULT + "")
  940:  @SimpleProperty
  941:  public void AlignVertical(int alignment) {
  942:    try {
  943:      // notice that the throw will prevent the alignment from being changed
  944:      // if the argument is illegal
  945:      alignmentSetter.setVerticalAlignment(alignment);
  946:      verticalAlignment = alignment;
  947:    } catch (IllegalArgumentException e) {
  948:      this.dispatchErrorOccurredEvent(this, "VerticalAlignment",
  949:          ErrorMessages.ERROR_BAD_VALUE_FOR_VERTICAL_ALIGNMENT, alignment);
  950:    }
  951:  }
  952:
  953:  /**
  954:   * Returns the type of open screen animation (default, fade, zoom, slidehorizontal,
  955:   * slidevertical and none).
  956:   *
  957:   * @return open screen animation
  958:   */
  959:  @SimpleProperty(category = PropertyCategory.APPEARANCE,
  960:    description = "The animation for switching to another screen. Valid" +
  961:    " options are default, fade, zoom, slidehorizontal, slidevertical, and none"
)
  962:  public String OpenScreenAnimation() {
  963:    return openAnimType;
  964:  }
  965:
  966:  /**
  967:   * Sets the animation type for the transition to another screen.
  968:   *
  969:   * @param animType the type of animation to use for the transition
  970:   */
  971:  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SCREEN_ANIMATIO
N,
  972:    defaultValue = "default")
  973:  @SimpleProperty
  974:  public void OpenScreenAnimation(String animType) {
  975:    if ((animType != "default") &&
  976:      (animType != "fade") && (animType != "zoom") && (animType != "slidehorizontal"
) &&
  977:      (animType != "slidevertical") && (animType != "none")) {
  978:      this.dispatchErrorOccurredEvent(this, "Screen",
  979:        ErrorMessages.ERROR_SCREEN_INVALID_ANIMATION, animType);
  980:      return;
  981:    }
  982:    openAnimType = animType;
  983:  }
  984:
  985:  /**
  986:   * Returns the type of close screen animation (default, fade, zoom, slidehorizontal
,
  987:   * slidevertical and none).
  988:   *
  989:   * @return open screen animation
  990:   */
  991:  @SimpleProperty(category = PropertyCategory.APPEARANCE,
  992:    description = "The animation for closing current screen and returning " +
  993:    " to the previous screen. Valid options are default, fade, zoom, slidehorizontal
, " +
  994:    "slidevertical, and none")
  995:  public String CloseScreenAnimation() {
  996:    return closeAnimType;
  997:  }
  998:
  999:  /**
 1000:   * Sets the animation type for the transition of this form closing and returning
 1001:   * to a form behind it in the activity stack.
 1002:   *
 1003:   * @param animType the type of animation to use for the transition
 1004:   */
 1005:  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SCREEN_ANIMATIO
N,
 1006:    defaultValue = "default")
 1007:  @SimpleProperty
 1008:  public void CloseScreenAnimation(String animType) {
 1009:    if ((animType != "default") &&
 1010:      (animType != "fade") && (animType != "zoom") && (animType != "slidehorizontal"
) &&
 1011:      (animType != "slidevertical") && (animType != "none")) {
 1012:      this.dispatchErrorOccurredEvent(this, "Screen",
 1013:        ErrorMessages.ERROR_SCREEN_INVALID_ANIMATION, animType);
 1014:      return;
 1015:    }
 1016:    closeAnimType = animType;
 1017:  }
 1018:
 1019:  /*
 1020:   * Used by ListPicker, and ActivityStarter to get this Form's current opening tran
sition
 1021:   * animation
 1022:   */
 1023:  public String getOpenAnimType() {
 1024:    return openAnimType;
 1025:  }
 1026:
 1027:  /**
 1028:   * Specifies the name of the application icon.
 1029:   *
 1030:   * @param name the name of the application icon
 1031:   */
 1032:  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ASSET,
 1033:      defaultValue = "")
 1034:  @SimpleProperty(userVisible = false)
 1035:  public void Icon(String name) {
 1036:    // We don't actually need to do anything.
 1037:  }
 1038:
 1039:  /**
 1040:   * Specifies the Version Code.
 1041:   *
 1042:   * @param vCode the version name of the application
 1043:   */
```

```
1044:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_IN
TEGER,
1045:     defaultValue = "1")
1046:   @SimpleProperty(userVisible = false,
1047:     description = "An integer value which must be incremented each time a new Androi
d "
1048:     +  "Application Package File (APK) is created for the Google Play Store.")
1049:   public void VersionCode(int vCode) {
1050:     // We don't actually need to do anything.
1051:   }
1052:
1053:   /**
1054:    * Specifies the Version Name.
1055:    *
1056:    * @param vName the version name of the application
1057:    */
1058:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
1059:     defaultValue = "1.0")
1060:   @SimpleProperty(userVisible = false,
1061:     description = "A string which can be changed to allow Google Play "
1062:     + "Store users to distinguish between different versions of the App.")
1063:   public void VersionName(String vName) {
1064:     // We don't actually need to do anything.
1065:   }
1066:
1067:   /**
1068:    * Specifies the App Name.
1069:    *
1070:    * @param aName the display name of the installed application in the phone
1071:    */
1072:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
1073:     defaultValue = "")
1074:   @SimpleProperty(userVisible = false,
1075:     description = "This is the display name of the installed application in the phon
e." +
1076:                   "If the AppName is blank, it will be set to the name of the project
when the project is built.")
1077:   public void AppName(String aName) {
1078:     // We don't actually need to do anything.
1079:   }
1080:
1081:   /**
1082:    * Width property getter method.
1083:    *
1084:    * @return  width property used by the layout
1085:    */
1086:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
1087:     description = "Screen width (x-size).")
1088:   public int Width() {
1089:     return frameLayout.getWidth();
1090:   }
1091:
1092:   /**
1093:    * Height property getter method.
1094:    *
1095:    * @return  height property used by the layout
1096:    */
1097:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
1098:     description = "Screen height (y-size).")
1099:   public int Height() {
1100:     return frameLayout.getHeight();
1101:   }
1102:
1103:   /**
1104:    * Display a new form.
1105:    *
1106:    * @param nextFormName the name of the new form to display
1107:    */
1108:   // This is called from runtime.scm when a "open another screen" block is executed.
1109:   public static void switchForm(String nextFormName) {
1110:     if (activeForm != null) {
1111:       activeForm.startNewForm(nextFormName, null);
1112:     } else {
1113:       throw new IllegalStateException("activeForm is null");
1114:     }
1115:   }
1116:
1117:   /**
1118:    * Display a new form and pass a startup value to the new form.
1119:    *
1120:    * @param nextFormName the name of the new form to display
1121:    * @param startValue the start value to pass to the new form
1122:    */
1123:   // This is called from runtime.scm when a "open another screen with start value" b
lock is
1124:   // executed.  Note that startNewForm will JSON encode the start value
1125:   public static void switchFormWithStartValue(String nextFormName, Object startValue
) {
1126:     Log.i(LOG_TAG, "Open another screen with start value:" + nextFormName);
1127:     if (activeForm != null) {
1128:       activeForm.startNewForm(nextFormName, startValue);
1129:       } else {
1130:         throw new IllegalStateException("activeForm is null");
1131:     }
1132:   }
1133:
1134:   // This JSON encodes the startup value
1135:   protected void startNewForm(String nextFormName, Object startupValue) {
1136:     Log.i(LOG_TAG, "startNewForm:" + nextFormName);
1137:     Intent activityIntent = new Intent();
1138:     // Note that the following is dependent on form generated class names being the
same as
1139:     // their form names and all forms being in the same package.
1140:     activityIntent.setClassName(this, getPackageName() + "." + nextFormName);
1141:     String functionName = (startupValue == null) ? "open another screen" :
1142:       "open another screen with start value";
1143:     String jValue;
1144:     if (startupValue != null) {
1145:       Log.i(LOG_TAG, "StartNewForm about to JSON encode:" + startupValue);
1146:       jValue = jsonEncodeForForm(startupValue, functionName);
1147:       Log.i(LOG_TAG, "StartNewForm got JSON encoding:" + jValue);
1148:     } else{
1149:       jValue = "";
1150:     }
1151:     activityIntent.putExtra(ARGUMENT_NAME, jValue);
1152:     // Save the nextFormName so that it can be passed to the OtherScreenClosed event
in the
1153:     // future.
1154:     this.nextFormName = nextFormName;
1155:     Log.i(LOG_TAG, "about to start new form" + nextFormName);
1156:     try {
1157:       Log.i(LOG_TAG, "startNewForm starting activity:" + activityIntent);
1158:       startActivityForResult(activityIntent, SWITCH_FORM_REQUEST_CODE);
1159:       AnimationUtil.ApplyOpenScreenAnimation(this, openAnimType);
```

```
1160:     } catch (ActivityNotFoundException e) {
1161:       dispatchErrorOccurredEvent(this, functionName,
1162:           ErrorMessages.ERROR_SCREEN_NOT_FOUND, nextFormName);
1163:     }
1164:   }
1165:
1166:   // functionName is used for including in the error message to be shown
1167:   // if the JSON encoding fails
1168:   protected static String jsonEncodeForForm(Object value, String functionName) {
1169:     String jsonResult = "";
1170:     Log.i(LOG_TAG, "jsonEncodeForForm -- creating JSON representation:" + value.toSt
ring());
1171:     try {
1172:       // TODO(hal): check that this is OK for raw strings
1173:       jsonResult = JsonUtil.getJsonRepresentation(value);
1174:       Log.i(LOG_TAG, "jsonEncodeForForm -- got JSON representation:" + jsonResult);
1175:     } catch (JSONException e) {
1176:       activeForm.dispatchErrorOccurredEvent(activeForm, functionName,
1177:           // showing the bad value here will produce an ugly error on the phone, but
it's
1178:           // more useful than not showing the value
1179:           ErrorMessages.ERROR_SCREEN_BAD_VALUE_FOR_SENDING, value.toString());
1180:     }
1181:     return jsonResult;
1182:   }
1183:
1184:   @SimpleEvent(description = "Event raised when another screen has closed and contro
l has " +
1185:       "returned to this screen.")
1186:   public void OtherScreenClosed(String otherScreenName, Object result) {
1187:     Log.i(LOG_TAG, "Form " + formName + " OtherScreenClosed, otherScreenName = " +
1188:         otherScreenName + ", result = " + result.toString());
1189:     EventDispatcher.dispatchEvent(this, "OtherScreenClosed", otherScreenName, result
);
1190:   }
1191:
1192:
1193:   // Component implementation
1194:
1195:   @Override
1196:   public HandlesEventDispatching getDispatchDelegate() {
1197:     return this;
1198:   }
1199:
1200:   // ComponentContainer implementation
1201:
1202:   @Override
1203:   public Activity $context() {
1204:     return this;
1205:   }
1206:
1207:   @Override
1208:   public Form $form() {
1209:     return this;
1210:   }
1211:
1212:   @Override
1213:   public void $add(AndroidViewComponent component) {
1214:     viewLayout.add(component);
1215:   }
1216:
1217:   @Override
1218:   public void setChildWidth(AndroidViewComponent component, int width) {
1219:     // A form is a vertical layout.
1220:     ViewUtil.setChildWidthForVerticalLayout(component.getView(), width);
1221:   }
1222:
1223:   @Override
1224:   public void setChildHeight(AndroidViewComponent component, int height) {
1225:     // A form is a vertical layout.
1226:     ViewUtil.setChildHeightForVerticalLayout(component.getView(), height);
1227:   }
1228:
1229:   /*
1230:    * This is called from runtime.scm at the beginning of each event handler.
1231:    * It allows runtime.scm to know which form environment should be used for
1232:    * looking up symbols. The active form is the form that is currently
1233:    * (or was most recently) dispatching an event.
1234:    */
1235:   public static Form getActiveForm() {
1236:     return activeForm;
1237:   }
1238:
1239:
1240:   /**
1241:    * Returns the string that was passed to this screen when it was opened
1242:    *
1243:    * @return StartupText
1244:    */
1245:   // This is called from runtime.scm when a "get plain start text" block is executed
.
1246:   public static String getStartText() {
1247:     if (activeForm != null) {
1248:       return activeForm.startupValue;
1249:     } else {
1250:       throw new IllegalStateException("activeForm is null");
1251:     }
1252:   }
1253:
1254:   /**
1255:    * Returns the value that was passed to this screen when it was opened
1256:    *
1257:    * @return StartValue
1258:    */
1259:   // TODO(hal): cache this?
1260:   // Note: This is called as a primitive from runtime.scm and it returns an arbitrar
y Java object.
1261:   // Therefore it must be explicitly sanitized by runtime, unlike methods, which
1262:   // are sanitized via call-component-method.
1263:   public static Object getStartValue() {
1264:     if (activeForm != null) {
1265:       return decodeJSONStringForForm(activeForm.startupValue, "get start value");
1266:     } else {
1267:       throw new IllegalStateException("activeForm is null");
1268:     }
1269:   }
1270:
1271:
1272:   /**
1273:    * Closes the current screen, as opposed to finishApplication, which
1274:    * exits the entire application.
1275:    */
1276:   // This is called from runtime.scm when a "close screen" block is executed.
1277:   public static void finishActivity() {
```

```
1278:     if (activeForm != null) {
1279:       activeForm.closeForm(null);
1280:     } else {
1281:       throw new IllegalStateException("activeForm is null");
1282:     }
1283:   }
1284:
1285:   // This is called from runtime.scm when a "close screen with value" block is execu
ted.
1286:   public static void finishActivityWithResult(Object result) {
1287:     if (activeForm != null) {
1288:       if (activeForm instanceof ReplForm) {
1289:         ((ReplForm)activeForm).setResult(result);
1290:         activeForm.closeForm(null);        // This will call RetValManager.popScreen
()
1291:       } else {
1292:         String jString = jsonEncodeForForm(result, "close screen with value");
1293:         Intent resultIntent = new Intent();
1294:         resultIntent.putExtra(RESULT_NAME, jString);
1295:         activeForm.closeForm(resultIntent);
1296:       }
1297:     } else {
1298:       throw new IllegalStateException("activeForm is null");
1299:     }
1300:   }
1301:
1302:   // This is called from runtime.scm when a "close screen with plain text" block is
executed.
1303:   public static void finishActivityWithTextResult(String result) {
1304:     if (activeForm != null) {
1305:       Intent resultIntent = new Intent();
1306:       resultIntent.putExtra(RESULT_NAME, result);
1307:       activeForm.closeForm(resultIntent);
1308:     } else {
1309:       throw new IllegalStateException("activeForm is null");
1310:     }
1311:   }
1312:
1313:
1314:   protected void closeForm(Intent resultIntent) {
1315:     if (resultIntent != null) {
1316:       setResult(Activity.RESULT_OK, resultIntent);
1317:     }
1318:     finish();
1319:     AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
1320:   }
1321:
1322:   // This is called from runtime.scm when a "close application" block is executed.
1323:   public static void finishApplication() {
1324:     if (activeForm != null) {
1325:       activeForm.closeApplicationFromBlocks();
1326:     } else {
1327:       throw new IllegalStateException("activeForm is null");
1328:     }
1329:   }
1330:
1331:   protected void closeApplicationFromBlocks() {
1332:     closeApplication();
1333:   }
1334:
1335:   private void closeApplicationFromMenu() {
1336:     closeApplication();
1337:   }
1338:
1339:   private void closeApplication() {
1340:     // In a multi-screen application, only Screen1 can successfully call System.exit
(0). Here, we
1341:     // set applicationIsBeingClosed to true. If this is not Screen1, when we call fi
nish() below,
1342:     // the previous form's onResume method will be called. In onResume, we check
1343:     // applicationIsBeingClosed and call closeApplication again. The stack of forms
will unwind
1344:     // until we get back to Screen1; then we'll call System.exit(0) below.
1345:     applicationIsBeingClosed = true;
1346:
1347:     finish();
1348:
1349:     if (formName.equals("Screen1")) {
1350:       // I know that this is frowned upon in Android circles but I really think that
it's
1351:       // confusing to users if the exit button doesn't really stop everything, inclu
ding other
1352:       // forms in the app (when we support them), non-UI threads, etc.  We might nee
d to be
1353:       // careful about this is we ever support services that start up on boot (since
it might
1354:       // mean that the only way to restart that service) is to reboot but that's a l
ong way off.
1355:       System.exit(0);
1356:     }
1357:   }
1358:
1359:   // Configure the system menu to include items to kill the application and to show
"about"
1360:   // information
1361:
1362:   @Override
1363:   public boolean onCreateOptionsMenu(Menu menu) {
1364:     // This procedure is called only once.  To change the items dynamically
1365:     // we would use onPrepareOptionsMenu.
1366:     super.onCreateOptionsMenu(menu);
1367:     // add the menu items
1368:     // Comment out the next line if we don't want the exit button
1369:     addExitButtonToMenu(menu);
1370:     addAboutInfoToMenu(menu);
1371:     return true;
1372:   }
1373:
1374:   public void addExitButtonToMenu(Menu menu) {
1375:     MenuItem stopApplicationItem = menu.add(Menu.NONE, Menu.NONE, Menu.FIRST,
1376:     "Stop this application")
1377:     .setOnMenuItemClickListener(new OnMenuItemClickListener() {
1378:       public boolean onMenuItemClick(MenuItem item) {
1379:         showExitApplicationNotification();
1380:         return true;
1381:       }
1382:     });
1383:     stopApplicationItem.setIcon(android.R.drawable.ic_notification_clear_all);
1384:   }
1385:
1386:   public void addAboutInfoToMenu(Menu menu) {
1387:     MenuItem aboutAppItem = menu.add(Menu.NONE, Menu.NONE, 2,
1388:     "About this application")
1389:     .setOnMenuItemClickListener(new OnMenuItemClickListener() {
```

```
1390:       public boolean onMenuItemClick(MenuItem item) {
1391:         showAboutApplicationNotification();
1392:         return true;
1393:       }
1394:     });
1395:     aboutAppItem.setIcon(android.R.drawable.sym_def_app_icon);
1396:   }
1397:
1398:   private void showExitApplicationNotification() {
1399:     String title = "Stop application?";
1400:     String message = "Stop this application and exit? You'll need to relaunch " +
1401:         "the application to use it again.";
1402:     String positiveButton = "Stop and exit";
1403:     String negativeButton = "Don't stop";
1404:     // These runnables are passed to twoButtonAlert.  They perform the corresponding
 actions
1405:     // when the button is pressed.   Here there's nothing to do for "don't stop" and
 cancel
1406:     Runnable stopApplication = new Runnable() {public void run () {closeApplicationF
romMenu();}};
1407:     Runnable doNothing = new Runnable () {public void run() {}};
1408:     Notifier.twoButtonDialog(
1409:         this,
1410:         message,
1411:         title,
1412:         positiveButton,
1413:         negativeButton,
1414:         false, // cancelable is false
1415:         stopApplication,
1416:         doNothing,
1417:         doNothing);
1418:   }
1419:
1420:   private String yandexTranslateTagline = "";
1421:
1422:   void setYandexTranslateTagline(){
1423:     yandexTranslateTagline = "<p><small>Language translation powered by Yandex.Trans
late</small></p>";
1424:   }
1425:
1426:   private void showAboutApplicationNotification() {
1427:     String title = "About this app";
1428:     String MITtagline = "<p><small><em>Invented with MIT App Inventor<br>appinventor
.mit.edu</em></small></p>";
1429:     // Users can hide the taglines by including an HTML open comment <!-- in the abo
ut screen message
1430:     String message = aboutScreen + MITtagline + yandexTranslateTagline;
1431:     message = message.replaceAll("\\n", "<br>"); // Allow for line breaks in the str
ing.
1432:     String buttonText ="Got it";
1433:     Notifier.oneButtonAlert(this, message, title, buttonText);
1434:   }
1435:
1436:   // This is called from clear-current-form in runtime.scm.
1437:   public void clear() {
1438:     viewLayout.getLayoutManager().removeAllViews();
1439:     // Set all screen properties to default values.
1440:     defaultPropertyValues();
1441:     screenInitialized = false;
1442:   }
1443:
1444:   public void deleteComponent(Object component) {
1445:     if (component instanceof OnStopListener) {
1446:       OnStopListener onStopListener = (OnStopListener) component;
1447:       if (onStopListeners.contains(onStopListener)) {
1448:         onStopListeners.remove(onStopListener);
1449:       }
1450:     }
1451:     if (component instanceof OnResumeListener) {
1452:       OnResumeListener onResumeListener = (OnResumeListener) component;
1453:       if (onResumeListeners.contains(onResumeListener)) {
1454:         onResumeListeners.remove(onResumeListener);
1455:       }
1456:     }
1457:     if (component instanceof OnPauseListener) {
1458:       OnPauseListener onPauseListener = (OnPauseListener) component;
1459:       if (onPauseListeners.contains(onPauseListener)) {
1460:         onPauseListeners.remove(onPauseListener);
1461:       }
1462:     }
1463:     if (component instanceof OnDestroyListener) {
1464:       OnDestroyListener onDestroyListener = (OnDestroyListener) component;
1465:       if (onDestroyListeners.contains(onDestroyListener)) {
1466:         onDestroyListeners.remove(onDestroyListener);
1467:       }
1468:     }
1469:     if (component instanceof Deleteable) {
1470:       ((Deleteable) component).onDelete();
1471:     }
1472:   }
1473:
1474:   public void dontGrabTouchEventsForComponent() {
1475:     // The following call results in the Form not grabbing our events and
1476:     // handling dragging on its own, which it wants to do to handle scrolling.
1477:     // Its effect only lasts long as the current set of motion events
1478:     // generated during this touch and drag sequence.  Consequently, if a
1479:     // component wants to handle dragging it needs to call this in the
1480:     // onTouchEvent of its View.
1481:     frameLayout.requestDisallowInterceptTouchEvent(true);
1482:   }
1483:
1484:
1485:   // This is used by Repl to throttle error messages which can get out of
1486:   // hand, e.g. if triggered by Accelerometer.
1487:   protected boolean toastAllowed() {
1488:     long now = System.nanoTime();
1489:     if (now > lastToastTime + minimumToastWait) {
1490:       lastToastTime = now;
1491:       return true;
1492:     }
1493:     return false;
1494:   }
1495:
1496:   // This is used by runtime.scm to call the Initialize of a component.
1497:   public void callInitialize(Object component) throws Throwable {
1498:     Method method;
1499:     try {
1500:       method = component.getClass().getMethod("Initialize", (Class<?>[]) null);
1501:     } catch (SecurityException e) {
1502:       Log.i(LOG_TAG, "Security exception " + e.getMessage());
1503:       return;
1504:     } catch (NoSuchMethodException e) {
1505:       //This is OK.
1506:       return;
```

```
 1507:     }
 1508:     try {
 1509:       Log.i(LOG_TAG, "calling Initialize method for Object " + component.toString())
;
 1510:       method.invoke(component, (Object[]) null);
 1511:     } catch (InvocationTargetException e){
 1512:       Log.i(LOG_TAG, "invoke exception: " + e.getMessage());
 1513:       throw e.getTargetException();
 1514:     }
 1515:   }
 1516:
 1517:   /**
 1518:    * Perform some action related to fullscreen video display.
 1519:    * @param action
 1520:    *          Can be any of the following:
 1521:    *          <ul>
 1522:    *          <li>
 1523:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_DURATION}
 1524:    *          </li>
 1525:    *          <li>
 1526:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_FULLSCREEN}
 1527:    *          </li>
 1528:    *          <li>
 1529:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_PAUSE}
 1530:    *          </li>
 1531:    *          <li>
 1532:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_PLAY}
 1533:    *          </li>
 1534:    *          <li>
 1535:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_SEEK}
 1536:    *          </li>
 1537:    *          <li>
 1538:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_SOURCE}
 1539:    *          </li>
 1540:    *          <li>
 1541:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_STOP}
 1542:    *          </li>
 1543:    *          </ul>
 1544:    * @param source
 1545:    *          The VideoPlayer to use in some actions.
 1546:    * @param data
 1547:    *          Used by the method. This object varies depending on the action.
 1548:    * @return Varies depending on what action was passed in.
 1549:    */
 1550:   public synchronized Bundle fullScreenVideoAction(int action, VideoPlayer source, O
bject data) {
 1551:     return fullScreenVideoUtil.performAction(action, source, data);
 1552:   }
 1553: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.ErrorMessages;

import android.content.Context;
import android.location.Address;
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.location.LocationProvider;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Sensor that can provide information on longitude, latitude, and altitude.
 *
 */
@DesignerComponent(version = YaVersion.LOCATIONSENSOR_COMPONENT_VERSION,
    description = "Non-visible component providing location information, " +
    "including longitude, latitude, altitude (if supported by the device), " +
    "and address.  This can also perform \"geocoding\", converting a given " +
    "address (not necessarily the current one) to a latitude (with the " +
    "<code>LatitudeFromAddress</code> method) and a longitude (with the " +
    "<code>LongitudeFromAddress</code> method).</p>\n" +
    "<p>In order to function, the component must have its " +
    "<code>Enabled</code> property set to True, and the device must have " +
    "location sensing enabled through wireless networks or GPS " +
    "satellites (if outdoors).</p>\n" +
    "Location information might not be immediately available when an app starts.  Yo
u'll have to wait a short time for " +
    "a location provider to be found and used, or wait for the OnLocationChanged eve
nt",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/locationSensor.png")
@SimpleObject
@UsesPermissions(permissionNames =
                 "android.permission.ACCESS_FINE_LOCATION," +
                 "android.permission.ACCESS_COARSE_LOCATION," +
                 "android.permission.ACCESS_MOCK_LOCATION," +
                 "android.permission.ACCESS_LOCATION_EXTRA_COMMANDS")
public class LocationSensor extends AndroidNonvisibleComponent
    implements Component, OnStopListener, OnResumeListener, Deleteable {

  /**
   * Class that listens for changes in location, raises appropriate events,
   * and provides properties.
   *
   */
  private class MyLocationListener implements LocationListener {
    @Override
    // This sets fields longitude, latitude, altitude, hasLocationData, and
    // hasAltitude, then calls LocationSensor.LocationChanged(), alll in the
    // enclosing class LocationSensor.
    public void onLocationChanged(Location location) {
      lastLocation = location;
      longitude = location.getLongitude();
      latitude = location.getLatitude();
      // If the current location doesn't have altitude information, the prior
      // altitude reading is retained.
      if (location.hasAltitude()) {
        hasAltitude = true;
        altitude = location.getAltitude();
      }
      hasLocationData = true;
      LocationChanged(latitude, longitude, altitude);
    }

    @Override
    public void onProviderDisabled(String provider) {
      StatusChanged(provider, "Disabled");
      stopListening();
      if (enabled) {
        RefreshProvider();
      }
    }

    @Override
    public void onProviderEnabled(String provider) {
      StatusChanged(provider, "Enabled");
      RefreshProvider();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
      switch (status) {
        // Ignore TEMPORARILY_UNAVAILABLE, because service usually returns quickly.
        case LocationProvider.TEMPORARILY_UNAVAILABLE:
          StatusChanged(provider, "TEMPORARILY_UNAVAILABLE");
          break;
        case LocationProvider.OUT_OF_SERVICE:
          // If the provider we were listening to is no longer available,
          // find another.
          StatusChanged(provider, "OUT_OF_SERVICE");

          if (provider.equals(providerName)) {
            stopListening();
            RefreshProvider();
          }
          break;
```

```
123:         case LocationProvider.AVAILABLE:
124:           // If another provider becomes available and is one we hadn't known
125:           // about see if it is better than the one we're currently using.
126:           StatusChanged(provider, "AVAILABLE");
127:           if (!provider.equals(providerName) &&
128:               !allProviders.contains(provider)) {
129:             RefreshProvider();
130:           }
131:           break;
132:       }
133:     }
134:   }
135:
136:   /**
137:    * Constant returned by {@link #Longitude()}, {@link #Latitude()}, and
138:    * {@link #Altitude()} if no value could be obtained for them.  The client
139:    * can find this out directly by calling {@link #HasLongitudeLatitude()} or
140:    * {@link #HasAltitude()}.
141:    */
142:   public static final int UNKNOWN_VALUE = 0;
143:
144:   // These variables contain information related to the LocationProvider.
145:   private final Criteria locationCriteria;
146:   private final Handler handler;
147:   private final LocationManager locationManager;
148:
149:   private boolean providerLocked = false; // if true we can't change providerName
150:   private String providerName;
151:   // Invariant: providerLocked => providerName is non-empty
152:
153:   private int timeInterval;
154:   private int distanceInterval;
155:
156:   private MyLocationListener myLocationListener;
157:
158:   private LocationProvider locationProvider;
159:   private boolean listening = false;
160:     // Invariant: listening <=> a myLocationListener is registered with locationMana
ger
161:     // Invariant: !listening <=> locationProvider == null
162:
163:   //This holds all the providers available when we last chose providerName.
164:   //The reported best provider is first, possibly duplicated.
165:   private List<String> allProviders;
166:
167:   // These location-related values are set in MyLocationListener.onLocationChanged()
.
168:   private Location lastLocation;
169:   private double longitude = UNKNOWN_VALUE;
170:   private double latitude = UNKNOWN_VALUE;
171:   private double altitude = UNKNOWN_VALUE;
172:   private boolean hasLocationData = false;
173:   private boolean hasAltitude = false;
174:
175:   // This is used in reverse geocoding.
176:   private Geocoder geocoder;
177:
178:   // User-settable properties
179:   private boolean enabled = true;  // the default value is true
180:
181:   /**
182:    * Creates a new LocationSensor component.
183:    *
184:    * @param container  ignored (because this is a non-visible component)
185:    */
186:   public LocationSensor(ComponentContainer container) {
187:     super(container.$form());
188:     handler = new Handler();
189:     // Set up listener
190:     form.registerForOnResume(this);
191:     form.registerForOnStop(this);
192:
193:     // Initialize sensor properties (60 seconds; 5 meters)
194:     timeInterval = 60000;
195:     distanceInterval = 5;
196:
197:     // Initialize location-related fields
198:     Context context = container.$context();
199:     geocoder = new Geocoder(context);
200:     locationManager = (LocationManager) context.getSystemService(Context.LOCATION_SE
RVICE);
201:     locationCriteria = new Criteria();
202:     myLocationListener = new MyLocationListener();
203:     allProviders = new ArrayList<String>();
204:     // Do some initialization depending on the initial enabled state
205:     Enabled(enabled);
206:   }
207:
208:   // Events
209:
210:   /**
211:    * Indicates that a new location has been detected.
212:    */
213:   @SimpleEvent
214:   public void LocationChanged(double latitude, double longitude, double altitude) {
215:     if (enabled) {
216:       EventDispatcher.dispatchEvent(this, "LocationChanged", latitude, longitude, al
titude);
217:     }
218:   }
219:
220:   /**
221:    * Indicates that the status of the location provider service has changed, such as
 when a
222:    * provider is lost or a new provider starts being used.
223:    */
224:   @SimpleEvent
225:   public void StatusChanged(String provider, String status) {
226:     if (enabled) {
227:       EventDispatcher.dispatchEvent(this, "StatusChanged", provider, status);
228:     }
229:   }
230:
231:   // Properties
232:
233:   /**
234:    * Indicates the source of the location information.  If there is no provider, the
235:    * string "NO PROVIDER" is returned.  This is useful primarily for debugging.
236:    */
237:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
238:   public String ProviderName() {
239:     if (providerName == null) {
240:       return "NO PROVIDER";
241:     } else {
```

```
  242:       return providerName;
  243:     }
  244:   }
  245:
  246:   /**
  247:    * Change the location provider.
  248:    * If the blocks program changes the name, try to change the provider.
  249:    * Whatever happens now, the provider and the reported name may be switched to
  250:    * Android's preferred provider later. This is primarily for debugging.
  251:    */
  252:   @SimpleProperty
  253:   public void ProviderName(String providerName) {
  254:     this.providerName = providerName;
  255:     if (!empty(providerName) && startProvider(providerName)) {
  256:       return;
  257:     } else {
  258:       RefreshProvider();
  259:     }
  260:   }
  261:
  262:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  263:   public boolean ProviderLocked() {
  264:     return providerLocked;
  265:   }
  266:
  267:   /**
  268:    * Indicates whether the sensor should allow the developer to
  269:    * manually change the provider (GPS, GSM, Wifi, etc.)
  270:    * from which location updates are received.
  271:    */
  272:   @SimpleProperty
  273:   public void ProviderLocked(boolean lock) {
  274:       providerLocked = lock;
  275:   }
  276:
  277:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SENSOR_TIME_INT
ERVAL,
  278:       defaultValue = "60000")
  279:   @SimpleProperty
  280:   public void TimeInterval(int interval) {
  281:
  282:       // make sure that the provided value is a valid one.
  283:       // choose 1000000 miliseconds to be the upper limit
  284:       if (interval < 0 || interval > 1000000)
  285:           return;
  286:
  287:       timeInterval = interval;
  288:
  289:       // restart listening for location updates, using the new time interval
  290:       if (enabled) {
  291:           RefreshProvider();
  292:       }
  293:   }
  294:
  295:   @SimpleProperty(
  296:       description = "Determines the minimum time interval, in milliseconds, that the
 sensor will try " +
  297:           "to use for sending out location updates. However, location updates will o
nly be received " +
  298:           "when the location of the phone actually changes, and use of the specified
 time interval " +
  299:           "is not guaranteed. For example, if 1000 is used as the time interval, loc
ation updates will " +
  300:           "never be fired sooner than 1000ms, but they may be fired anytime after.",
  301:           category = PropertyCategory.BEHAVIOR)
  302:   public int TimeInterval() {
  303:       return timeInterval;
  304:   }
  305:
  306:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SENSOR_DIST_INT
ERVAL,
  307:       defaultValue = "5")
  308:   @SimpleProperty
  309:   public void DistanceInterval(int interval) {
  310:
  311:       // make sure that the provided value is a valid one.
  312:       // choose 1000 meters to be the upper limit
  313:       if (interval < 0 || interval > 1000)
  314:           return;
  315:
  316:       distanceInterval = interval;
  317:
  318:       // restart listening for location updates, using the new distance interval
  319:       if (enabled) {
  320:           RefreshProvider();
  321:       }
  322:   }
  323:
  324:   @SimpleProperty(
  325:       description = "Determines the minimum distance interval, in meters, that the s
ensor will try " +
  326:       "to use for sending out location updates. For example, if this is set to 5, th
en the sensor will " +
  327:       "fire a LocationChanged event only after 5 meters have been traversed. However
, the sensor does " +
  328:       "not guarantee that an update will be received at exactly the distance interva
l. It may take more " +
  329:       "than 5 meters to fire an event, for instance.",
  330:       category = PropertyCategory.BEHAVIOR)
  331:   public int DistanceInterval() {
  332:       return distanceInterval;
  333:   }
  334:
  335:   /**
  336:    * Indicates whether longitude and latitude information is available.  (It is
  337:    * always the case that either both or neither are.)
  338:    */
  339:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  340:   public boolean HasLongitudeLatitude() {
  341:     return hasLocationData && enabled;
  342:   }
  343:
  344:   /**
  345:    * Indicates whether altitude information is available.
  346:    */
  347:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  348:   public boolean HasAltitude() {
  349:     return hasAltitude && enabled;
  350:   }
  351:
  352:   /**
  353:    * Indicates whether information about location accuracy is available.
  354:    */
  355:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
```

```
356:   public boolean HasAccuracy() {
357:     return Accuracy() != UNKNOWN_VALUE && enabled;
358:   }
359:
360:   /**
361:    * The most recent available longitude value.  If no value is available,
362:    * 0 will be returned.
363:    */
364:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
365:   public double Longitude() {
366:     return longitude;
367:   }
368:
369:   /**
370:    * The most recently available latitude value.  If no value is available,
371:    * 0 will be returned.
372:    */
373:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
374:   public double Latitude() {
375:       return latitude;
376:   }
377:
378:   /**
379:    * The most recently available altitude value, in meters.  If no value is
380:    * available, 0 will be returned.
381:    */
382:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
383:   public double Altitude() {
384:     return altitude;
385:   }
386:
387:   /**
388:    * The most recent measure of accuracy, in meters.  If no value is available,
389:    * 0 will be returned.
390:    */
391:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
392:   public double Accuracy() {
393:     if (lastLocation != null && lastLocation.hasAccuracy()) {
394:       return lastLocation.getAccuracy();
395:     } else if (locationProvider != null) {
396:       return locationProvider.getAccuracy();
397:     } else {
398:       return UNKNOWN_VALUE;
399:     }
400:   }
401:
402:   /**
403:    * Indicates whether the user has specified that the sensor should
404:    * listen for location changes and raise the corresponding events.
405:    */
406:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
407:   public boolean Enabled() {
408:     return enabled;
409:   }
410:
411:   /**
412:    * Indicates whether the sensor should listen for location chagnes
413:    * and raise the corresponding events.
414:    */
415:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
416:       defaultValue = "True")
417:   @SimpleProperty
418:   public void Enabled(boolean enabled) {
419:     this.enabled = enabled;
420:     if (!enabled) {
421:       stopListening();
422:     } else {
423:       RefreshProvider();
424:     }
425:   }
426:
427:   /**
428:    * Provides a textual representation of the current address or
429:    * "No address available".
430:    */
431:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
432:   public String CurrentAddress() {
433:     if (hasLocationData &&
434:         latitude <= 90 && latitude >= -90 &&
435:         longitude <= 180 || longitude >= -180) {
436:       try {
437:         List<Address> addresses = geocoder.getFromLocation(latitude, longitude, 1);
438:         if (addresses != null && addresses.size() == 1) {
439:           Address address = addresses.get(0);
440:           if (address != null) {
441:             StringBuilder sb = new StringBuilder();
442:             for (int i = 0; i <= address.getMaxAddressLineIndex(); i++) {
443:               sb.append(address.getAddressLine(i));
444:               sb.append("\n");
445:             }
446:             return sb.toString();
447:           }
448:         }
449:
450:       } catch (Exception e) {
451:         // getFromLocation can throw an IOException or an IllegalArgumentException
452:         // a bad result can give an indexOutOfBoundsException
453:         // are there others?
454:         if (e instanceof IllegalArgumentException
455:             || e instanceof IOException
456:             || e instanceof IndexOutOfBoundsException ) {
457:           Log.e("LocationSensor", "Exception thrown by getting current address " + e
.getMessage());
458:         } else {
459:           // what other exceptions can happen here?
460:           Log.e("LocationSensor",
461:               "Unexpected exception thrown by getting current address " + e.getMessa
ge());
462:         }
463:       }
464:     }
465:     return "No address available";
466:   }
467:
468:   /**
469:    * Derives Latitude from Address
470:    *
471:    * @param locationName  human-readable address
472:    *
473:    * @return latitude in degrees, 0 if not found.
474:    */
475:   @SimpleFunction(description = "Derives latitude of given address")
476:   public double LatitudeFromAddress(String locationName) {
477:     try {
```

```
478:       List<Address> addressObjs = geocoder.getFromLocationName(locationName, 1);
479:       Log.i("LocationSensor", "latitude addressObjs size is " + addressObjs.size() +
 " for " + locationName);
480:       if ( (addressObjs == null) || (addressObjs.size() == 0) ){
481:         throw new IOException("");
482:       }
483:       return addressObjs.get(0).getLatitude();
484:     } catch (IOException e) {
485:       form.dispatchErrorOccurredEvent(this, "LatitudeFromAddress",
486:           ErrorMessages.ERROR_LOCATION_SENSOR_LATITUDE_NOT_FOUND, locationName);
487:       return 0;
488:     }
489:   }
490:
491:   /**
492:    * Derives Longitude from Address
493:    * @param locationName  human-readable address
494:    *
495:    * @return longitude in degrees, 0 if not found.
496:    */
497:   @SimpleFunction(description = "Derives longitude of given address")
498:   public double LongitudeFromAddress(String locationName) {
499:     try {
500:       List<Address> addressObjs = geocoder.getFromLocationName(locationName, 1);
501:       Log.i("LocationSensor", "longitude addressObjs size is " + addressObjs.size()
+ " for " + locationName);
502:       if ( (addressObjs == null) || (addressObjs.size() == 0) ){
503:         throw new IOException("");
504:       }
505:       return addressObjs.get(0).getLongitude();
506:     } catch (IOException e) {
507:       form.dispatchErrorOccurredEvent(this, "LongitudeFromAddress",
508:           ErrorMessages.ERROR_LOCATION_SENSOR_LONGITUDE_NOT_FOUND, locationName);
509:       return 0;
510:     }
511:   }
512:
513:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
514:   public List<String> AvailableProviders () {
515:     return allProviders;
516:   }
517:
518:   // Methods to stop and start listening to LocationProviders
519:
520:   /**
521:    * Refresh provider attempts to choose and start the best provider unless
522:    * someone has set and locked the provider. Currently, blocks programmers
523:    * cannot do that because the relevant methods are not declared as properties.
524:    *
525:    */
526:
527:   // @SimpleFunction(description = "Find and start listening to a location provider.
")
528:   public void RefreshProvider() {
529:     stopListening();             // In case another provider is active.
530:     if (providerLocked && !empty(providerName)) {
531:       listening = startProvider(providerName);
532:       return;
533:     }
534:     allProviders = locationManager.getProviders(true);  // Typically it's ("network"
 "gps")
535:     String bProviderName = locationManager.getBestProvider(locationCriteria, true);
536:     if (bProviderName != null && !bProviderName.equals(allProviders.get(0))) {
537:       allProviders.add(0, bProviderName);
538:     }
539:     // We'll now try the best first and stop as soon as one successfully starts.
540:     for (String providerN : allProviders) {
541:       listening = startProvider(providerN);
542:       if (listening) {
543:         if (!providerLocked) {
544:           providerName = providerN;
545:         }
546:         return;
547:       }
548:     }
549:   }
550:
551:   /* Start listening to ProviderName.
552:    * Return true iff successful.
553:    */
554:   private boolean startProvider(String providerName) {
555:     this.providerName = providerName;
556:     LocationProvider tLocationProvider = locationManager.getProvider(providerName);
557:     if (tLocationProvider == null) {
558:       Log.d("LocationSensor", "getProvider(" + providerName + ") returned null");
559:       return false;
560:     }
561:     stopListening();
562:     locationProvider = tLocationProvider;
563:     locationManager.requestLocationUpdates(providerName, timeInterval,
564:         distanceInterval, myLocationListener);
565:     listening = true;
566:     return true;
567:   }
568:
569:   /**
570:    * This unregisters {@link #myLocationListener} as a listener to location
571:    * updates.  It is safe to call this even if no listener had been registered,
572:    * in which case it has no effect.  This also sets the value of
573:    * {@link #locationProvider} to {@code null} and sets {@link #listening}
574:    * to {@code false}.
575:    */
576:   private void stopListening() {
577:     if (listening) {
578:       locationManager.removeUpdates(myLocationListener);
579:       locationProvider = null;
580:       listening = false;
581:     }
582:   }
583:
584:
585:   // OnResumeListener implementation
586:
587:   @Override
588:   public void onResume() {
589:     if (enabled) {
590:       RefreshProvider();
591:     }
592:   }
593:
594:   // OnStopListener implementation
595:
596:   @Override
597:   public void onStop() {
```

```
598:     stopListening();
599:   }
600:
601:   // Deleteable implementation
602:
603:   @Override
604:   public void onDelete() {
605:     stopListening();
606:   }
607:
608:   private boolean empty(String s) {
609:     return s == null || s.length() == 0;
610:   }
611: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.FroyoUtil;
import com.google.appinventor.components.runtime.util.OrientationSensorUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
import android.view.Display;
import android.view.Surface;
import android.view.WindowManager;

/**
 * Sensor that can measure absolute orientation in 3 dimensions.
 *
 */
@DesignerComponent(version = YaVersion.ORIENTATIONSENSOR_COMPONENT_VERSION,
    description = "<p>Non-visible component providing information about the " +
    "device's physical orientation in three dimensions: <ul> " +
    "<li> <strong>Roll</strong>: 0 degrees when the device is level, increases to "
+
    "     90 degrees as the device is tilted up on its left side, and " +
    "     decreases to -90 degrees when the device is tilted up on its right side. "
 +
    "     </li> " +
    "<li> <strong>Pitch</strong>: 0 degrees when the device is level, up to " +
    "     90 degrees as the device is tilted so its top is pointing down, " +
    "     up to 180 degrees as it gets turned over.  Similarly, as the device " +
    "     is tilted so its bottom points down, pitch decreases to -90 " +
    "     degrees, then further decreases to -180 degrees as it gets turned all the
way " +
    "     over.</li> " +
    "<li> <strong>Azimuth</strong>: 0 degrees when the top of the device is " +
    "     pointing north, 90 degrees when it is pointing east, 180 degrees " +
    "     when it is pointing south, 270 degrees when it is pointing west, " +
    "     etc.</li></ul>" +
    "     These measurements assume that the device itself is not moving.</p>",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/orientationsensor.png")

@SimpleObject
public class OrientationSensor extends AndroidNonvisibleComponent
    implements SensorEventListener, Deleteable, OnPauseListener, OnResumeListener {
  // Constants
  private static final String LOG_TAG = "OrientationSensor";
  // offsets in array returned by SensorManager.getOrientation()
  private static final int AZIMUTH = 0;
  private static final int PITCH = 1;
  private static final int ROLL = 2;
  private static final int DIMENSIONS = 3;  // Warning: specific to our universe

  // Properties
  private boolean enabled;
  private float azimuth;    // degrees
  private float pitch;  // degrees
  private float roll;   // degrees
  private int accuracy;

  // Sensor information
  private final SensorManager sensorManager;
  private final Sensor accelerometerSensor;
  private final Sensor magneticFieldSensor;
  private boolean listening;

  // Pre-allocated arrays to hold sensor data so that we don't cause so many garbage
collections
  // while processing sensor events. All are used only in onSensorChanged.
  private final float[] accels = new float[DIMENSIONS];  // acceleration vector
  private final float[] mags = new float[DIMENSIONS];    // magnetic field vector

  // Flags to tell whether the above arrays are filled. They are set in onSensorChan
ged and cleared
  // in stopListening.
  private boolean accelsFilled;
  private boolean magsFilled;

  // Pre-allocated matrixes used to compute orientation values from acceleration and
magnetic
  // field data.
  private final float[] rotationMatrix = new float[DIMENSIONS * DIMENSIONS];
  private final float[] inclinationMatrix = new float[DIMENSIONS * DIMENSIONS];
  private final float[] values = new float[DIMENSIONS];

  /**
   * Creates a new OrientationSensor component.
   *
   * @param container  ignored (because this is a non-visible component)
   */
  public OrientationSensor(ComponentContainer container) {
    super(container.$form());

    // Get sensors, and start listening.
    sensorManager =
      (SensorManager) container.$context().getSystemService(Context.SENSOR_SERVICE);
    accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    magneticFieldSensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
;

    // Begin listening in onResume() and stop listening in onPause().
    form.registerForOnResume(this);
    form.registerForOnPause(this);

    // Set default property values.
    Enabled(true);
```

```
  }

  private void startListening() {
    if (!listening) {
      sensorManager.registerListener(this, accelerometerSensor,
                                     SensorManager.SENSOR_DELAY_NORMAL);
      sensorManager.registerListener(this, magneticFieldSensor,
                                     SensorManager.SENSOR_DELAY_NORMAL);
      listening = true;
    }
  }

  private void stopListening() {
    if (listening) {
      sensorManager.unregisterListener(this);
      listening = false;

      // Throw out sensor information that will go stale.
      accelsFilled = false;
      magsFilled = false;
    }
  }

  // Events

  /**
   * Default OrientationChanged event handler.
   *
   * <p>This event is signalled when the device's orientation has changed.  It
   * reports the new values of azimuth, pich, and roll, and it also sets the Azimuth
, Pitch,
   * and roll properties.</p>
   * <p>Azimuth is the compass heading in degrees, pitch indicates how the device
   * is tilted from top to bottom, and roll indicates how much the device is tilted
from
   * side to side.</p>
   */
  @SimpleEvent
  public void OrientationChanged(float azimuth, float pitch, float roll) {
    EventDispatcher.dispatchEvent(this, "OrientationChanged", azimuth, pitch, roll);
  }

  // Properties

  /**
   * Available property getter method (read-only property).
   *
   * @return {@code true} indicates that an orientation sensor is available,
   *         {@code false} that it isn't
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public boolean Available() {
    return sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).size() > 0
        && sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD).size() > 0;
  }

  /**
   * Enabled property getter method.
   *
   * @return {@code true} indicates that the sensor generates events,
   *         {@code false} that it doesn't
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public boolean Enabled() {
    return enabled;
  }

  /**
   * Enabled property setter method.
   *
   * @param enabled  {@code true} enables sensor event generation,
   *                 {@code false} disables it
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
      defaultValue = "True")
  @SimpleProperty
  public void Enabled(boolean enabled) {
    if (this.enabled != enabled) {
      this.enabled = enabled;
      if (enabled) {
        startListening();
      } else {
        stopListening();
      }
    }
  }

  /**
   * Pitch property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current pitch
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Pitch() {
    return pitch;
  }

  /**
   * Roll property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current roll
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Roll() {
    return roll;
  }

  /**
   * Azimuth property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current azimuth
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Azimuth() {
    return azimuth;
  }

  /**
```

```
240:    * <p>Angle property getter method (read-only property).  Specifically, this
241:    * provides the angle in which the orientation sensor is tilted, treating
242:    * -{@link #Roll()} as the x-coordinate and {@link #Pitch()} as the
243:    * y-coordinate.  For the amount of the tilt, use {@link #Magnitude()}.</p>
244:    *
245:    * <p>To return meaningful values the sensor must be enabled.</p>
246:    *
247:    * @return the angle in degrees
248:    */
249:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
250:   public float Angle() {
251:     return OrientationSensor.computeAngle(pitch, roll);
252:   }
253:
254:   /**
255:    * Computes the angle the phone is tilted.  This has been lifted out
256:    * of {@link #Angle()} for ease of testing.
257:    *
258:    * @param pitch an angle indicating how far the device is tilted vertically,
259:    *              with a value of +90 degrees if the top is pointing straight
260:    *              down, +180 degrees if the phone is entirely turned over,
261:    *              -90/+270 degrees if the top is pointing straight up, etc.
262:    * @param roll an angle indicating how far the device is tilted horizontally,
263:    *             with a value of +90 degrees if it is tilted entirely on its
264:    *             left side, -90 degrees if it is tilted entirely on its right
265:    *             side; the maximum absolute value of roll is 90 degrees, after
266:    *             which it decreases back toward 0 (flat face-up or face-down).
267:    *
268:    * @returns the corresonding angle in the range [-180, +180] degrees
269:    */
270:   static float computeAngle(float pitch, float roll) {
271:     return (float) Math.toDegrees(Math.atan2(Math.toRadians(pitch),
272:                                              // invert roll to correct sign
273:                                              -Math.toRadians(roll)));
274:   }
275:
276:   /**
277:    * Magnitude property getter method (read-only property).  Specifically, this
278:    * returns a number between 0 and 1, indicating how much the device
279:    * is tilted.  For the angle of tilt, use {@link #Angle()}.
280:    *
281:    * <p>To return meaningful values the sensor must be enabled.</p>
282:    *
283:    * @return the magnitude of the tilt, from 0 to 1
284:    */
285:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
286:   public float Magnitude() {
287:     // Limit pitch and roll to 90; otherwise, the phone is upside down.
288:     // The official documentation falsely claims that the range of pitch and
289:     // roll is [-90, 90].  If the device is upside-down, it can range from
290:     // -180 to 180.  We restrict it to the range [-90, 90].
291:     // With that restriction, if the pitch and roll angles are P and R, then
292:     // the force is given by 1 - cos(P)cos(R).  I have found a truly wonderful
293:     // proof of this theorem, but the margin enforced by Lint is too small to
294:     // contain it.
295:     final int MAX_VALUE = 90;
296:     double npitch = Math.toRadians(Math.min(MAX_VALUE, Math.abs(pitch)));
297:     double nroll = Math.toRadians(Math.min(MAX_VALUE, Math.abs(roll)));
298:     return (float) (1.0 - Math.cos(npitch) * Math.cos(nroll));
299:   }
300:
301:   // SensorListener implementation
302:
303:   /*
304:    * Returns the rotation of the screen from its "natural" orientation.
305:    * Note that this is the angle of rotation of the drawn graphics on the
306:    * screen, which is the opposite direction of the physical rotation of the
307:    * device. For example, if the device is rotated 90 degrees counter-clockwise,
308:    * to compensate rendering will be rotated by 90 degrees clockwise and thus
309:    * the returned value here will be Surface.ROTATION_90.  Return values will
310:    * be in the set Surface.ROTATION_{0,90,180,270}.
311:    */
312:   private int getScreenRotation() {
313:     Display display =
314:         ((WindowManager) form.getSystemService(Context.WINDOW_SERVICE)).
315:         getDefaultDisplay();
316:     if (SdkLevel.getLevel() >= SdkLevel.LEVEL_FROYO) {
317:       return FroyoUtil.getRotation(display);
318:     } else {
319:       return display.getOrientation();
320:     }
321:   }
322:
323:   /**
324:    * Responds to changes in the accelerometer or magnetic field sensors to
325:    * recompute orientation.  This only updates azimuth, pitch, and roll and
326:    * raises the OrientationChanged event if both sensors have reported in
327:    * at least once.
328:    *
329:    * @param sensorEvent an event from the accelerometer or magnetic field sensor
330:    */
331:   @Override
332:   public void onSensorChanged(SensorEvent sensorEvent) {
333:     if (enabled) {
334:       int eventType = sensorEvent.sensor.getType();
335:
336:       // Save the new sensor information about acceleration or the magnetic field.
337:       switch (eventType) {
338:         case Sensor.TYPE_ACCELEROMETER:
339:           // Update acceleration array.
340:           System.arraycopy(sensorEvent.values, 0, accels, 0, DIMENSIONS);
341:           accelsFilled = true;
342:           // Only update the accuracy property for the accelerometer.
343:           accuracy = sensorEvent.accuracy;
344:           break;
345:
346:         case Sensor.TYPE_MAGNETIC_FIELD:
347:           // Update magnetic field array.
348:           System.arraycopy(sensorEvent.values, 0, mags, 0, DIMENSIONS);
349:           magsFilled = true;
350:           break;
351:
352:         default:
353:           Log.e(LOG_TAG, "Unexpected sensor type: " + eventType);
354:           return;
355:       }
356:
357:       // If we have both acceleration and magnetic information, recompute values.
358:       if (accelsFilled && magsFilled) {
359:         SensorManager.getRotationMatrix(rotationMatrix,    // output
360:                                         inclinationMatrix, // output
361:                                         accels,
362:                                         mags);
363:         SensorManager.getOrientation(rotationMatrix, values);
```

```
364:
365:         // Make sure values are in expected range.
366:         azimuth = OrientationSensorUtil.normalizeAzimuth(
367:             (float) Math.toDegrees(values[AZIMUTH]));
368:         pitch = OrientationSensorUtil.normalizePitch(
369:             (float) Math.toDegrees(values[PITCH]));
370:         // Sign change for roll is for compatibility with earlier versions
371:         // of App Inventor that got orientation sensor information differently.
372:         roll = OrientationSensorUtil.normalizeRoll(
373:             (float) -Math.toDegrees(values[ROLL]));
374:
375:         // Adjust pitch and roll for phone rotation (e.g., landscape)
376:         int rotation = getScreenRotation();
377:         switch(rotation) {
378:           case Surface.ROTATION_0:  // normal rotation
379:             break;
380:           case Surface.ROTATION_90:  // phone is turned 90 degrees counter-clockwise
381:             float temp = -pitch;
382:             pitch = -roll;
383:             roll = temp;
384:             break;
385:           case Surface.ROTATION_180: // phone is rotated 180 degrees
386:             roll = -roll;
387:             break;
388:           case Surface.ROTATION_270:  // phone is turned 90 degrees clockwise
389:             temp = pitch;
390:             pitch = roll;
391:             roll = temp;
392:             break;
393:           default:
394:             Log.e(LOG_TAG, "Illegal value for getScreenRotation(): " +
395:                   rotation);
396:             break;
397:         }
398:
399:         // Raise event.
400:         OrientationChanged(azimuth, pitch, roll);
401:       }
402:     }
403:   }
404:
405:   @Override
406:   public void onAccuracyChanged(Sensor sensor, int accuracy) {
407:     // TODO(markf): Figure out if we actually need to do something here.
408:   }
409:
410:   // Deleteable implementation
411:
412:   @Override
413:   public void onDelete() {
414:     stopListening();
415:   }
416:
417:   // OnPauseListener implementation
418:
419:   public void onPause() {
420:     stopListening();
421:   }
422:
423:   // OnResumeListener implementation
424:
425:   public void onResume() {
426:     if (enabled) {
427:       startListening();
428:     }
429:   }
430: }
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2009-2011 Google, All Rights reserved
    3: // Copyright 2011-2014 MIT, All rights reserved
    4: // Released under the Apache License, Version 2.0
    5: // http://www.apache.org/licenses/LICENSE-2.0
    6:
    7: package com.google.appinventor.components.runtime;
    8:
    9: import android.content.Context;
   10: import android.hardware.Sensor;
   11: import android.hardware.SensorEvent;
   12: import android.hardware.SensorEventListener;
   13: import android.hardware.SensorManager;
   14: import com.google.appinventor.components.annotations.*;
   15: import com.google.appinventor.components.common.ComponentCategory;
   16: import com.google.appinventor.components.common.PropertyTypeConstants;
   17: import com.google.appinventor.components.common.YaVersion;
   18: import java.util.List;
   19:
   20: @DesignerComponent(version = YaVersion.PROXIMITYSENSOR_COMPONENT_VERSION,
   21:         description = "<p>Non-visible component that can measures the proximity of a
n object in cm " +
   22:                 "relative to the view screen of a device. This sensor is typically u
sed to determine " +
   23:                 "whether a handset is being held up to a persons ear; " +
   24:                 "i.e. lets you determine how far away an object is from a device. "
+
   25:                 "Many devices return the absolute distance, in cm, but some return o
nly near and far values. " +
   26:                 "In this case, the sensor usually reports its maximum range value in
 the far state " +
   27:                 "and a lesser value in the near state.</p>",
   28:         category = ComponentCategory.SENSORS,
   29:         nonVisible = true,
   30:         iconName = "images/proximitysensor.png")
   31: @SimpleObject
   32: public class ProximitySensor extends AndroidNonvisibleComponent
   33:         implements OnStopListener, OnResumeListener, SensorComponent, OnPauseListene
r,
   34:         SensorEventListener, Deleteable {
   35:
   36:     private Sensor proximitySensor;
   37:
   38:     private final SensorManager sensorManager;
   39:
   40:     // Indicates whether the sensor should generate events
   41:     private boolean enabled;
   42:     private float distance=0f;
   43:
   44:     // Indicates if the sensor should be running when screen is off (on pause)
   45:     private boolean keepRunningWhenOnPause;
   46:
   47:     /**
   48:      * Creates a new ProximitySensor component.
   49:      *
   50:      * @param container  ignored (because this is a non-visible component)
   51:      */
   52:     public ProximitySensor(ComponentContainer container) {
   53:         super(container.$form());
   54:         form.registerForOnResume(this);
   55:         form.registerForOnStop(this);
   56:         form.registerForOnPause(this);
   57:
   58:         enabled = true;
   59:         sensorManager = (SensorManager) container.$context().getSystemService(Contex
t.SENSOR_SERVICE);
   60:         proximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
   61:         startListening();
   62:     }
   63:
   64:     /**
   65:      * Used to determine if the device has ProximitySensor
   66:      *
   67:      * @return {@code true} indicates that an proximity sensor is available,
   68:      *         {@code false} that it isn't
   69:      */
   70:     @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Reports whe
ther or not the device has a proximity sensor")
   71:     public boolean Available() {
   72:         List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_PROXIMITY);
   73:         return (sensors.size() > 0);
   74:     }
   75:
   76:     @Override
   77:     public void onResume() {
   78:         if (enabled) {
   79:             startListening();
   80:         }
   81:     }
   82:
   83:     @Override
   84:     public void onStop() {
   85:         if (enabled) {
   86:             stopListening();
   87:         }
   88:     }
   89:
   90:     @Override
   91:     public void onDelete() {
   92:         if (enabled) {
   93:             stopListening();
   94:         }
   95:     }
   96:
   97:     @Override
   98:     public void onPause() {
   99:         if (enabled && !keepRunningWhenOnPause) {
  100:             stopListening();
  101:         }
  102:     }
  103:
  104:
  105:     /**
  106:      * Registers the sensor to start listening for proximity changes
  107:      */
  108:     private void startListening() {
  109:         sensorManager.registerListener(this, proximitySensor, SensorManager.SENSOR_D
ELAY_NORMAL);
  110:     }
  111:
  112:     /**
  113:      * Stops the sensors from listening to the proximity changes
  114:      */
  115:     private void stopListening() {
```

```
116:         sensorManager.unregisterListener(this);
117:     }
118:
119:     /**
120:      * Called when sensor values have changed
121:      * @param sensorEvent holds information such as the sensor's type,
122:      *                    the time-stamp, accuracy and sensor's data
123:      */
124:     @Override
125:     public void onSensorChanged(SensorEvent sensorEvent) {
126:         if (enabled) {
127:             final float[] values = sensorEvent.values.clone();
128:             distance = values[0];
129:             ProximityChanged(distance);
130:         }
131:     }
132:
133:     /**
134:      * Determines a sensor's maximum range. Some proximity sensors return binary val
ues
135:      * that represent "near" or "far." In this case, the sensor usually reports
136:      * its maximum range value in the far state and a lesser value in the near state
.
137:      * Typically, the far value is a value > 5 cm, but this can vary from sensor to
sensor.
138:      *
139:      * @return Sensor's maximum range.
140:      */
141:     @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Reports the
 Maximum Range of the device's ProximitySensor")
142:     public float MaximumRange() {
143:         return proximitySensor.getMaximumRange();
144:     }
145:
146:     /**
147:      * If true, the sensor will generate events.  Otherwise, no events
148:      * are generated .
149:      *
150:      * @return {@code true} indicates that the sensor generates events,
151:      *         {@code false} that it doesn't
152:      */
153:     @SimpleProperty(category = PropertyCategory.BEHAVIOR)
154:     public boolean Enabled() {
155:         return enabled;
156:     }
157:
158:     /**
159:      * Specifies whether the sensor should generate events.  If true,
160:      * the sensor will generate events.  Otherwise, no events are generated.
161:      *
162:      * @param enabled  {@code true} enables sensor event generation,
163:      *                 {@code false} disables it
164:      */
165:     @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defa
ultValue = "True")
166:     @SimpleProperty (description = "If enabled, then device will listen for changes
in proximity")
167:     public void Enabled(boolean enabled) {
168:         if (this.enabled == enabled) {
169:             return;
170:         }
171:
172:         this.enabled = enabled;
173:         if (enabled) {
174:             startListening();
175:         } else {
176:             stopListening();
177:         }
178:     }
179:
180:     /**
181:      * Returns value of keepRunningWhenOnPause
182:      */
183:     @SimpleProperty(category = PropertyCategory.BEHAVIOR)
184:     public boolean KeepRunningWhenOnPause() {
185:         return keepRunningWhenOnPause;
186:     }
187:
188:     /**
189:      * Specifies if sensor should still be listening when activity is not active
190:      *
191:      * @param enabled
192:      */
193:     @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defa
ultValue = "False")
194:     @SimpleProperty (description = "If set to true, it will keep sensing for proximi
ty changes even when the app is not visible")
195:     public void KeepRunningWhenOnPause(boolean enabled) {
196:
197:         this.keepRunningWhenOnPause = enabled;
198:     }
199:
200:     @SimpleEvent(description = "Triggered when distance (in cm) of the object to the
 device changes. ")
201:     public void ProximityChanged(float distance) {
202:         this.distance = distance;
203:         EventDispatcher.dispatchEvent(this, "ProximityChanged", this.distance);
204:     }
205:
206:     /**
207:      * Returns the distance.
208:      * The sensor must be enabled to return meaningful values.
209:      *
210:      * @return  distance
211:      */
212:     @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Returns the
distance from the object to the device")
213:     public float Distance() {
214:         return distance;
215:     }
216:
217:     /**
218:      * Called when the accuracy of the registered sensor has changed
219:      * @param sensor Sensor
220:      * @param accuracy the new accuracy of this sensor
221:      */
222:     @Override
223:     public void onAccuracyChanged(Sensor sensor, int accuracy) {
224:         //To change body of implemented methods use File | Settings | File Templates
.
225:     }
226: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
/// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.ErrorMessages;
import com.google.appinventor.components.runtime.util.MediaUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;

import android.content.Context;
import android.media.AudioManager;
import android.media.SoundPool;
import android.os.Handler;
import android.os.Vibrator;
import android.util.Log;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

/**
 * Multimedia component that plays sounds and optionally vibrates.  A
 * sound is specified via filename.  See also
 * {@link android.media.SoundPool}.
 *
 * @author sharon@google.com (Sharon Perl)
 * @author hal@mit.edu (Hal Abelson) added wait for load to complete
 */
@DesignerComponent(version = YaVersion.SOUND_COMPONENT_VERSION,
    description = "<p>A multimedia component that plays sound " +
    "files and optionally vibrates for the number of milliseconds " +
    "(thousandths of a second) specified in the Blocks Editor.  The name of " +
    "the sound file to play can be specified either in the Designer or in " +
    "the Blocks Editor.</p> <p>For supported sound file formats, see " +
    "<a href=\"http://developer.android.com/guide/appendix/media-formats.html\"" +
    " target=\"_blank\">Android Supported Media Formats</a>.</p>" +
    "<p>This <code>Sound</code> component is best for short sound files, such as sou
nd " +
    "effects, while the <code>Player</code> component is more efficient for " +
    "longer sounds, such as songs.</p>" +
    "<p>You might get an error if you attempt to play a sound " +
    "immeditely after setting the source.</p>",
    category = ComponentCategory.MEDIA,
    nonVisible = true,
    iconName = "images/soundEffect.png")
@SimpleObject
@UsesPermissions(permissionNames = "android.permission.VIBRATE, android.permission.I
NTERNET")
public class Sound extends AndroidNonvisibleComponent
    implements Component, OnResumeListener, OnStopListener, OnDestroyListener, Delet
eable {

  private boolean loadComplete;    // did the sound finish loading

  // The purpose of this class is to avoid getting rejected by the Android verifier
when the
  // Sound component code is loaded into a device with API level less than 8, where
the verifier
  // will reject OnLoadCompleteListener.  We do this trick by putting
  // the use of OnLoadCompleteListener in the class OnLoadHelper and arranging (see
below) for
  // the class to be compiled only if the API level is at least 8.
  private class OnLoadHelper {
    public void setOnloadCompleteListener (SoundPool soundPool) {
      soundPool.setOnLoadCompleteListener(new android.media.SoundPool.OnLoadComplete
Listener() {
        public void onLoadComplete(SoundPool soundPool, int sampleId, int status) {
          loadComplete = true;
        }
      });
    }
  }

  private static final int MAX_STREAMS = 10;

  // max number of consecutive delays to wait for a sound to load
  private static final int MAX_PLAY_DELAY_RETRIES = 10;
  // number of ms in each delay before retrying
  private static final int PLAY_DELAY_LENGTH = 50;

  private static final float VOLUME_FULL = 1.0f;
  private static final int LOOP_MODE_NO_LOOP = 0;
  private static final float PLAYBACK_RATE_NORMAL = 1.0f;
  private SoundPool soundPool;

  // soundMap maps sounds (assets, etc) that are loaded into soundPool to their resp
ective
  // soundIds.
  private final Map<String, Integer> soundMap;

  // We will wait for Sound loading to complete before trying to play, but only
  // if the API level is at least 8, because onLoadCompleteListener is not available
  // in earlier APIs. For those early systems, attempting to play a sound before it
is loaded
  // will fail to play the sound and there will be no retry, although there might be
 a "cannot
  // play" error.
  private final boolean waitForLoadToComplete = (SdkLevel.getLevel() >= SdkLevel.LEV
EL_FROYO);

  private String sourcePath;              // name of source
  private int soundId;                    // id of sound in the soundPool
  private int streamId;                   // stream id returned from last call to So
undPool.play
  private int minimumInterval;            // minimum interval between Play() calls
  private long timeLastPlayed;            // the system time when Play() was last ca
lled
  private final Vibrator vibe;
  private final Handler playWaitHandler = new Handler();

```

```
  //save a pointer to this Sound component to use in the error in postDelayed below
  private final Component thisComponent;


  public Sound(ComponentContainer container) {
    super(container.$form());
    thisComponent = this;
    soundPool = new SoundPool(MAX_STREAMS, AudioManager.STREAM_MUSIC, 0);
    soundMap = new HashMap<String, Integer>();
    vibe = (Vibrator) form.getSystemService(Context.VIBRATOR_SERVICE);
    sourcePath = "";
    loadComplete = true;  //nothing to wait for until we attempt to load
    form.registerForOnResume(this);
    form.registerForOnStop(this);
    form.registerForOnDestroy(this);

    // Make volume buttons control media, not ringer.
    form.setVolumeControlStream(AudioManager.STREAM_MUSIC);

    // Default property values
    MinimumInterval(500);

    if (waitForLoadToComplete) {
      new OnLoadHelper().setOnloadCompleteListener(soundPool);
    }
  }



  /**
   * Returns the sound's filename.
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "The name of the sound file.  Only certain " +
      "formats are supported.  See http://developer.android.com/guide/appendix/media-formats.html.")
  public String Source() {
    return sourcePath;
  }

  /**
   * Sets the sound source
   *
   * <p/>See {@link MediaUtil#determineMediaSource} for information about what
   * a path can be.
   *
   * @param path  the path to the sound source
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ASSET,
      defaultValue = "")
  @SimpleProperty
  public void Source(String path) {
    sourcePath = (path == null) ? "" : path;

    // Clear the previous sound.
    if (streamId != 0) {
      soundPool.stop(streamId);
      streamId = 0;
    }
    soundId = 0;

    if (sourcePath.length() != 0) {
      Integer existingSoundId = soundMap.get(sourcePath);
      if (existingSoundId != null) {
        soundId = existingSoundId;

      } else {
        Log.i("Sound", "No existing sound with path " + sourcePath + ".");
        try {
          int newSoundId = MediaUtil.loadSoundPool(soundPool, form, sourcePath);
          if (newSoundId != 0) {
            soundMap.put(sourcePath, newSoundId);
            Log.i("Sound", "Successfully began loading sound: setting soundId to " + newSoundId + ".");
            soundId = newSoundId;
            // set flag to show that loading has begun
            loadComplete = false;
          } else {
            form.dispatchErrorOccurredEvent(this, "Source",
                ErrorMessages.ERROR_UNABLE_TO_LOAD_MEDIA, sourcePath);
          }
        } catch (IOException e) {
          form.dispatchErrorOccurredEvent(this, "Source",
              ErrorMessages.ERROR_UNABLE_TO_LOAD_MEDIA, sourcePath);
        }
      }
    }
  }

  /**
   * Returns the minimum interval required between calls to Play(), in
   * milliseconds.
   * Once the sound starts playing, all further Play() calls will be ignored
   * until the interval has elapsed.
   * @return  minimum interval in ms
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "The minimum interval between sounds.  If you play a sound, " +
      "all further Play() calls will be ignored until the interval has elapsed.")
  public int MinimumInterval() {
    return minimumInterval;
  }

  /**
   * Specify the minimum interval required between calls to Play(), in
   * milliseconds.
   * Once the sound starts playing, all further Play() calls will be ignored
   * until the interval has elapsed.
   * @param interval  minimum interval in ms
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
      defaultValue = "500")
  @SimpleProperty
  public void MinimumInterval(int interval) {
    minimumInterval = interval;
  }


  // number of retries remaining before signaling an error
  private int delayRetries;

```

```
  233:   /**
  234:    * Plays the sound.
  235:    */
  236:   @SimpleFunction(description = "Plays the sound specified by the Source property.")
  237:   public void Play() {
  238:     if (soundId != 0) {
  239:       long currentTime = System.currentTimeMillis();
  240:       if (timeLastPlayed == 0 || currentTime >= timeLastPlayed + minimumInterval) {
  241:         timeLastPlayed = currentTime;
  242:         delayRetries =  MAX_PLAY_DELAY_RETRIES;
  243:         playWhenLoadComplete();
  244:       } else {
  245:         // fail silently
  246:         Log.i("Sound", "Unable to play because MinimumInterval has not elapsed since
 last play.");
  247:       }
  248:     } else {
  249:       // Alert the user that the sound is bad, but would need to look in the log to
distinguish
  250:       // this error from the UNABLE_TO_PLAY_MEDIA error in playAndCheck.
  251:       Log.i("Sound", "Sound Id was 0. Did you remember to set the Source property?")
;
  252:       form.dispatchErrorOccurredEvent(this, "Play",
  253:           ErrorMessages.ERROR_UNABLE_TO_PLAY_MEDIA, sourcePath);
  254:     }
  255:   }
  256:
  257:   // Attempt to play the sound, possibly after a delay to allow the sound to load.
  258:   private void playWhenLoadComplete() {
  259:     if (loadComplete || !waitForLoadToComplete) {
  260:       playAndCheckResult();
  261:     } else {
  262:       Log.i("Sound", "Sound not ready:  retrying.  Remaining retries = " + delayRetr
ies);
  263:       // if the sound wasn't ready we retry after a delay. We implement the delay by
 posting
  264:       // to a separate handler: using a loop with a sleep might seem simpler, but it
 would block
  265:       // the UI thread.
  266:       playWaitHandler.postDelayed(new Runnable() {
  267:         @Override
  268:         public void run() {
  269:           if (loadComplete) {
  270:             playAndCheckResult();
  271:           } else if (delayRetries > 0) {
  272:             delayRetries--;
  273:             playWhenLoadComplete();
  274:           } else {
  275:             form.dispatchErrorOccurredEvent(thisComponent, "Play",
  276:                 ErrorMessages.ERROR_SOUND_NOT_READY, sourcePath);
  277:           }
  278:         }
  279:       }, PLAY_DELAY_LENGTH);
  280:     }
  281:   }
  282:
  283:   private void playAndCheckResult() {
  284:     streamId = soundPool.play(soundId, VOLUME_FULL, VOLUME_FULL, 0, LOOP_MODE_NO_LOO
P,
  285:         PLAYBACK_RATE_NORMAL);
  286:   Log.i("Sound", "SoundPool.play returned stream id " + streamId);
  287:   if (streamId == 0) {
  288:     form.dispatchErrorOccurredEvent(this, "Play",
  289:         ErrorMessages.ERROR_UNABLE_TO_PLAY_MEDIA, sourcePath);
  290: }
  291:   }
  292:
  293:
  294:   /**
  295:    * Pauses playing the sound if it is being played.
  296:    */
  297:   @SimpleFunction(description = "Pauses playing the sound if it is being played.")
  298:   public void Pause() {
  299:     if (streamId != 0) {
  300:       soundPool.pause(streamId);
  301:     } else {
  302:       Log.i("Sound", "Unable to pause. Did you remember to call the Play function?")
;
  303:     }
  304:   }
  305:
  306:   /**
  307:    * Resumes playing the sound after a pause.
  308:    */
  309:   @SimpleFunction(description = "Resumes playing the sound after a pause.")
  310:   public void Resume() {
  311:     if (streamId != 0) {
  312:       soundPool.resume(streamId);
  313:     } else {
  314:       Log.i("Sound", "Unable to resume. Did you remember to call the Play function?"
);
  315:     }
  316:   }
  317:
  318:   /**
  319:    * Stops playing the sound if it is being played.
  320:    */
  321: @SimpleFunction(description = "Stops playing the sound if it is being played.")
  322:   public void Stop() {
  323:     if (streamId != 0) {
  324:       soundPool.stop(streamId);
  325:       streamId = 0;
  326:     } else {
  327:       Log.i("Sound", "Unable to stop. Did you remember to call the Play function?");
  328:     }
  329:   }
  330:
  331:   /**
  332:    * Vibrates for the specified number of milliseconds.
  333:    */
  334:   @SimpleFunction(description = "Vibrates for the specified number of milliseconds."
)
  335:   public void Vibrate(int millisecs) {
  336:     vibe.vibrate(millisecs);
  337:   }
  338:
  339:   @SimpleEvent(description = "The SoundError event is no longer used. " +
  340:       "Please use the Screen.ErrorOccurred event instead.",
  341:       userVisible = false)
  342:   public void SoundError(String message) {
  343:   }
  344:
  345:   // OnStopListener implementation
  346:
```

```
347:   @Override
348:   public void onStop() {
349:     Log.i("Sound", "Got onStop");
350:     if (streamId != 0) {
351:       soundPool.pause(streamId);
352:     }
353:   }
354:
355:   // OnResumeListener implementation
356:
357:   @Override
358:   public void onResume() {
359:     Log.i("Sound", "Got onResume");
360:     if (streamId != 0) {
361:       soundPool.resume(streamId);
362:     }
363:   }
364:
365:   // OnDestroyListener implementation
366:
367:   @Override
368:   public void onDestroy() {
369:     prepareToDie();
370:   }
371:
372:   // Deletable implementation
373:
374:   @Override
375:   public void onDelete() {
376:     prepareToDie();
377:   }
378:
379:   private void prepareToDie() {
380:     if (streamId != 0) {
381:       soundPool.stop(streamId);
382:       soundPool.unload(streamId);
383:     }
384:     soundPool.release();
385:     vibe.cancel();
386:     // The documentation for SoundPool suggests setting the reference to null;
387:     soundPool = null;
388:   }
389: }
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @fileoverview Control blocks for Blockly, modified for App Inventor
    7:  * @author fraser@google.com (Neil Fraser)
    8:  * @author andrew.f.mckinney@gmail.com (Andrew F. McKinney)
    9:  * Due to the frequency of long strings, the 80-column wrap rule need not apply
   10:  * to language files.
   11:  */
   12:
   13: /**
   14:  * Lyn's History:
   15:  * [lyn, written 11/16-17/13, added 07/01/14] Added freeVariables, renameFree, and r
enameBound to forRange and forEach loops
   16:  * [lyn, 10/27/13] Specify direction of flydowns
   17:  * [lyn, 10/25/13] Made collapsed block labels more sensible.
   18:  * [lyn, 10/10-14/13]
   19:  *   + Installed flydown index variable declarations in forRange and forEach loops
   20:  *   + Abstracted over string labels on all blocks using constants defined in en/_me
ssages.js
   21:  *   + Renamed "for <i> start [] end [] step []" block to "for each <number> from []
 to [] by []"
   22:  *   + Renamed "for each <i> in list []" block to "for each <item> in list []"
   23:  *   + Renamed "choose test [] then-return [] else-return []" to "if [] then [] else
 []"
   24:  *     (TODO: still needs to have a mutator like  the "if" statement blocks).
   25:  *   + Renamed "evaluate" block to "evaluate but ignore result"
   26:  *   + Renamed "do {} then-return []" block to "do {} result []" and re-added this b
lock
   27:  *     to the Control drawer (who removed it?)
   28:  *   + Removed get block (still in Variable drawer; no longer needed with parameter
flydowns)
   29:  * [lyn, 11/29-30/12]
   30:  *   + Change forEach and forRange loops to take name as input text rather than via
plug.
   31:  *   + For these blocks, add extra methods to support renaming.
   32:  */
   33:
   34: 'use strict';
   35:
   36: goog.provide('Blockly.Blocks.control');
   37:
   38: goog.require('Blockly.Blocks.Utilities');
   39:
   40: Blockly.Blocks['controls_if'] = {
   41:   // If/elseif/else condition.
   42:   category: 'Control',
   43:   helpUrl: Blockly.Msg.LANG_CONTROLS_IF_HELPURL,
   44:   init: function () {
   45:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
   46:     this.appendValueInput('IF0')
   47:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
   48:         .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_IF);
   49:     this.appendStatementInput('DO0')
   50:         .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
   51:     this.setPreviousStatement(true);
   52:     this.setNextStatement(true);
   53:     this.setMutator(new Blockly.Mutator(['controls_if_elseif',
   54:       'controls_if_else']));
   55:     // Assign 'this' to a variable for use in the tooltip closure below.
   56:     var thisBlock = this;
   57:     this.setTooltip(function () {
   58:       if (!thisBlock.elseifCount_ && !thisBlock.elseCount_) {
   59:         return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_1;
   60:       } else if (!thisBlock.elseifCount_ && thisBlock.elseCount_) {
   61:         return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_2;
   62:       } else if (thisBlock.elseifCount_ && !thisBlock.elseCount_) {
   63:         return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_3;
   64:       } else if (thisBlock.elseifCount_ && thisBlock.elseCount_) {
   65:         return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_4;
   66:       }
   67:       return '';
   68:     });
   69:     this.elseifCount_ = 0;
   70:     this.elseCount_ = 0;
   71:     this.warnings = [{name: "checkEmptySockets", sockets: [{baseName: "IF"}, {baseNa
me: "DO"}]}];
   72:   },
   73:   mutationToDom: function () {
   74:     if (!this.elseifCount_ && !this.elseCount_) {
   75:       return null;
   76:     }
   77:     var container = document.createElement('mutation');
   78:     if (this.elseifCount_) {
   79:       container.setAttribute('elseif', this.elseifCount_);
   80:     }
   81:     if (this.elseCount_) {
   82:       container.setAttribute('else', 1);
   83:     }
   84:     return container;
   85:   },
   86:   domToMutation: function (xmlElement) {
   87:     if (xmlElement.getAttribute('elseif') === null) {
   88:       this.elseifCount_ = 0;
   89:     } else {
   90:       this.elseifCount_ = window.parseInt(xmlElement.getAttribute('elseif'), 10);
   91:     }
   92:
   93:     this.elseCount_ = window.parseInt(xmlElement.getAttribute('else'), 10);
   94:     for (var x = 1; x <= this.elseifCount_; x++) {
   95:       this.appendValueInput('IF' + x)
   96:           .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockl
y.Blocks.Utilities.INPUT))
   97:           .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSEIF);
   98:       this.appendStatementInput('DO' + x)
   99:           .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
  100:     }
  101:     if (this.elseCount_) {
  102:       this.appendStatementInput('ELSE')
  103:           .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSE);
  104:     }
  105:   },
  106:   decompose: function (workspace) {
  107:     var containerBlock = new Blockly.Block.obtain(workspace, 'controls_if_if');
  108:     containerBlock.initSvg();
  109:     var connection = containerBlock.getInput('STACK').connection;
  110:     for (var x = 1; x <= this.elseifCount_; x++) {
  111:       var elseifBlock = new Blockly.Block.obtain(workspace, 'controls_if_elseif');
  112:       elseifBlock.initSvg();
  113:       connection.connect(elseifBlock.previousConnection);
  114:       connection = elseifBlock.nextConnection;
```

```
    }
    if (this.elseCount_) {
      var elseBlock = new Blockly.Block.obtain(workspace, 'controls_if_else');
      elseBlock.initSvg();
      connection.connect(elseBlock.previousConnection);
    }
    return containerBlock;
  },
  compose: function (containerBlock) {
    // Disconnect the else input blocks and destroy the inputs.
    if (this.elseCount_) {
      this.removeInput('ELSE');
    }
    this.elseCount_ = 0;
    // Disconnect all the elseif input blocks and destroy the inputs.
    for (var x = this.elseifCount_; x > 0; x--) {
      this.removeInput('IF' + x);
      this.removeInput('DO' + x);
    }
    this.elseifCount_ = 0;
    // Rebuild the block's optional inputs.
    var clauseBlock = containerBlock.getInputTargetBlock('STACK');
    while (clauseBlock) {
      switch (clauseBlock.type) {
        case 'controls_if_elseif':
          this.elseifCount_++;
          var ifInput = this.appendValueInput('IF' + this.elseifCount_)
              .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Bl
ockly.Blocks.Utilities.INPUT))
              .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSEIF);
          var doInput = this.appendStatementInput('DO' + this.elseifCount_);
          doInput.appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
          // Reconnect any child blocks.
          if (clauseBlock.valueConnection_) {
            ifInput.connection.connect(clauseBlock.valueConnection_);
          }
          if (clauseBlock.statementConnection_) {
            doInput.connection.connect(clauseBlock.statementConnection_);
          }
          break;
        case 'controls_if_else':
          this.elseCount_++;
          var elseInput = this.appendStatementInput('ELSE');
          elseInput.appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSE);
          // Reconnect any child blocks.
          if (clauseBlock.statementConnection_) {
            elseInput.connection.connect(clauseBlock.statementConnection_);
          }
          break;
        default:
          throw 'Unknown block type.';
      }
      clauseBlock = clauseBlock.nextConnection &&
      clauseBlock.nextConnection.targetBlock();
    }
  },
  saveConnections: function (containerBlock) {
    // Store a pointer to any connected child blocks.
    var inputDo;
    var clauseBlock = containerBlock.getInputTargetBlock('STACK');
    var x = 1;
    while (clauseBlock) {
      switch (clauseBlock.type) {
        case 'controls_if_elseif':
          var inputIf = this.getInput('IF' + x);
          inputDo = this.getInput('DO' + x);
          clauseBlock.valueConnection_ =
              inputIf && inputIf.connection.targetConnection;
          clauseBlock.statementConnection_ =
              inputDo && inputDo.connection.targetConnection;
          x++;
          break;
        case 'controls_if_else':
          inputDo = this.getInput('ELSE');
          clauseBlock.statementConnection_ =
              inputDo && inputDo.connection.targetConnection;
          break;
        default:
          throw 'Unknown block type.';
      }
      clauseBlock = clauseBlock.nextConnection &&
      clauseBlock.nextConnection.targetBlock();
    }
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_IF_IF_TITLE_IF}]
};

Blockly.Blocks['controls_if_if'] = {
  // If condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_IF_TITLE_IF);
    this.appendStatementInput('STACK');
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_IF_TOOLTIP);
    this.contextMenu = false;
  }
};

Blockly.Blocks['controls_if_elseif'] = {
  // Else-If condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_ELSEIF_TITLE_ELSEIF);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_ELSEIF_TOOLTIP);
    this.contextMenu = false;
  }
};

Blockly.Blocks['controls_if_else'] = {
  // Else condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_ELSE_TITLE_ELSE);
    this.setPreviousStatement(true);
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_ELSE_TOOLTIP);
    this.contextMenu = false;
  }
};
```

```
Blockly.Blocks['controls_forRange'] = {
  // For range.
  category: 'Control',
  helpUrl: Blockly.Msg.LANG_CONTROLS_FORRANGE_HELPURL,
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    //this.setOutput(true, null);
    // Need to deal with variables here
    // [lyn, 11/30/12] Changed variable to be text input box that does renaming right (i.e., avoids variable capture)
    // Old code:
    // this.appendValueInput('VAR').appendField('for range').appendField('variable').setAlign(Blockly.ALIGN_RIGHT);
    // this.appendValueInput('START').setCheck(Number).appendField('start').setAlign(Blockly.ALIGN_RIGHT);
    this.appendValueInput('START')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_ITEM)
        .appendField(new Blockly.FieldParameterFlydown(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_VAR, true, Blockly.FieldFlydown.DISPLAY_BELOW), 'VAR')
        .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_START)
        .setAlign(Blockly.ALIGN_RIGHT);
    this.appendValueInput('END')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_END)
        .setAlign(Blockly.ALIGN_RIGHT);
    this.appendValueInput('STEP')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_STEP)
        .setAlign(Blockly.ALIGN_RIGHT);
    this.appendStatementInput('DO')
        .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_DO)
        .setAlign(Blockly.ALIGN_RIGHT);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_FORRANGE_TOOLTIP);
  },
  getVars: function () {
    return [this.getFieldValue('VAR')];
  },
  blocksInScope: function () {
    var doBlock = this.getInputTargetBlock('DO');
    if (doBlock) {
      return [doBlock];
    } else {
      return [];
    }
  },
  declaredNames: function () {
    return [this.getFieldValue('VAR')];
  },
  renameVar: function (oldName, newName) {
    if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
      this.setFieldValue(newName, 'VAR');
    }
  },
  renameBound: function (boundSubstitution, freeSubstitution) {
    Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('START'), freeSubstitution);
    Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('END'), freeSubstitution);
    Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('STEP'), freeSubstitution);
    var oldIndexVar = this.getFieldValue('VAR');
    var newIndexVar = boundSubstitution.apply(oldIndexVar);
    if (newIndexVar !== oldIndexVar) {
      this.renameVar(oldIndexVar, newIndexVar);
      var indexSubstitution = Blockly.Substitution.simpleSubstitution(oldIndexVar, newIndexVar);
      var extendedFreeSubstitution = freeSubstitution.extend(indexSubstitution);
      Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), extendedFreeSubstitution);
    } else {
      var removedFreeSubstitution = freeSubstitution.remove([oldIndexVar]);
      Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), removedFreeSubstitution);
    }
    if (this.nextConnection) {
      var nextBlock = this.nextConnection.targetBlock();
      Blockly.LexicalVariable.renameFree(nextBlock, freeSubstitution);
    }
  },
  renameFree: function (freeSubstitution) {
    var indexVar = this.getFieldValue('VAR');
    var bodyFreeVars = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('DO'));
    bodyFreeVars.deleteName(indexVar);
    var renamedBodyFreeVars = bodyFreeVars.renamed(freeSubstitution);
    if (renamedBodyFreeVars.isMember(indexVar)) { // Variable capture!
      var newIndexVar = Blockly.FieldLexicalVariable.nameNotIn(indexVar, renamedBodyFreeVars.toList());
      var boundSubstitution = Blockly.Substitution.simpleSubstitution(indexVar, newIndexVar);
      this.renameBound(boundSubstitution, freeSubstitution);
    } else {
      this.renameBound(new Blockly.Substitution(), freeSubstitution);
    }
  },
  freeVariables: function () { // return the free variables of this block
    var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('DO'));
    result.deleteName(this.getFieldValue('VAR')); // Remove bound index variable from body free vars
    result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('START')));
    result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('END')));
    result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('STEP')));
    if (this.nextConnection) {
      var nextBlock = this.nextConnection.targetBlock();
      result.unite(Blockly.LexicalVariable.freeVariables(nextBlock));
    }
    return result;
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_ITEM}]
};

Blockly.Blocks['controls_forEach'] = {
  // For each loop.
  category: 'Control',
```

```
  341:   helpUrl: Blockly.Msg.LANG_CONTROLS_FOREACH_HELPURL,
  342:   init: function () {
  343:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
  344:     //this.setOutput(true, null);
  345:     // [lyn, 10/07/13] Changed default name from "i" to "item"
  346:     // [lyn, 11/29/12] Changed variable to be text input box that does renaming righ
t (i.e., avoids variable capture)
  347:     // Old code:
  348:     // this.appendValueInput('VAR').appendField('for range').appendField('variable')
.setAlign(Blockly.ALIGN_RIGHT);
  349:     // this.appendValueInput('START').setCheck(Number).appendField('start').setAlign
(Blockly.ALIGN_RIGHT);
  350:     this.appendValueInput('LIST')
  351:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list", Blockly.Blo
cks.Utilities.INPUT))
  352:         .appendField(Blockly.Msg.LANG_CONTROLS_FOREACH_INPUT_ITEM)
  353:         .appendField(new Blockly.FieldParameterFlydown(Blockly.Msg.LANG_CONTROLS_FOR
EACH_INPUT_VAR,
  354:             true, Blockly.FieldFlydown.DISPLAY_BELOW), 'VAR')
  355:         .appendField(Blockly.Msg.LANG_CONTROLS_FOREACH_INPUT_INLIST)
  356:         .setAlign(Blockly.ALIGN_RIGHT);
  357:     this.appendStatementInput('DO').appendField(Blockly.Msg.LANG_CONTROLS_FOREACH_IN
PUT_DO);
  358:     this.setPreviousStatement(true);
  359:     this.setNextStatement(true);
  360:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_FOREACH_TOOLTIP);
  361:   },
  362:   getVars: function () {
  363:     return [this.getFieldValue('VAR')];
  364:   },
  365:   blocksInScope: function () {
  366:     var doBlock = this.getInputTargetBlock('DO');
  367:     if (doBlock) {
  368:       return [doBlock];
  369:     } else {
  370:       return [];
  371:     }
  372:   },
  373:   declaredNames: function () {
  374:     return [this.getFieldValue('VAR')];
  375:   },
  376:   renameVar: function (oldName, newName) {
  377:     if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
  378:       this.setFieldValue(newName, 'VAR');
  379:     }
  380:   },
  381:   renameBound: function (boundSubstitution, freeSubstitution) {
  382:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('LIST'), freeSubstit
ution);
  383:     var oldIndexVar = this.getFieldValue('VAR');
  384:     var newIndexVar = boundSubstitution.apply(oldIndexVar);
  385:     if (newIndexVar !== oldIndexVar) {
  386:       this.renameVar(oldIndexVar, newIndexVar);
  387:       var indexSubstitution = Blockly.Substitution.simpleSubstitution(oldIndexVar, n
ewIndexVar);
  388:       var extendedFreeSubstitution = freeSubstitution.extend(indexSubstitution);
  389:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), extendedFre
eSubstitution);
  390:     } else {
  391:       var removedFreeSubstitution = freeSubstitution.remove([oldIndexVar]);
  392:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), removedFree
Substitution);
  393:     }
  394:     if (this.nextConnection) {
  395:       var nextBlock = this.nextConnection.targetBlock();
  396:       Blockly.LexicalVariable.renameFree(nextBlock, freeSubstitution);
  397:     }
  398:   },
  399:   renameFree: function (freeSubstitution) {
  400:     var indexVar = this.getFieldValue('VAR');
  401:     var bodyFreeVars = Blockly.LexicalVariable.freeVariables(this.getInputTargetBloc
k('DO'));
  402:     bodyFreeVars.deleteName(indexVar);
  403:     var renamedBodyFreeVars = bodyFreeVars.renamed(freeSubstitution);
  404:     if (renamedBodyFreeVars.isMember(indexVar)) { // Variable capture!
  405:       var newIndexVar = Blockly.FieldLexicalVariable.nameNotIn(indexVar, renamedBody
FreeVars.toList());
  406:       var boundSubstitution = Blockly.Substitution.simpleSubstitution(indexVar, newI
ndexVar);
  407:       this.renameBound(boundSubstitution, freeSubstitution);
  408:     } else {
  409:       this.renameBound(new Blockly.Substitution(), freeSubstitution);
  410:     }
  411:   },
  412:   freeVariables: function () { // return the free variables of this block
  413:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('DO'
));
  414:     result.deleteName(this.getFieldValue('VAR')); // Remove bound index variable fro
m body free vars
  415:     result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('LIS
T')));
  416:     if (this.nextConnection) {
  417:       var nextBlock = this.nextConnection.targetBlock();
  418:       result.unite(Blockly.LexicalVariable.freeVariables(nextBlock));
  419:     }
  420:     return result;
  421:   },
  422:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_FOREACH_INPUT_ITEM}]
  423: };
  424:
  425: /* [lyn 10/10/13] With parameter flydown changes,
  426: * I don't think a special GET block in the Control drawer is necesssary
  427: Blockly.Blocks.for_lexical_variable_get = {
  428:   // Variable getter.
  429:   category: 'Control',
  430:   helpUrl: Blockly.Msg.LANG_CONTROLS_GET_HELPURL,
  431:   init: function() {
  432:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
  433:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
  434:     this.fieldVar_.setBlock(this);
  435:     this.appendDummyInput()
  436:         .appendField("get")
  437:         .appendField(this.fieldVar_, 'VAR');
  438:     this.setOutput(true, null);
  439:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_GET_TOOLTIP);
  440:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
  441:   },
  442:   getVars: function() {
  443:     return [this.getFieldValue('VAR')];
  444:   },
  445:   onchange: function() {
  446:     // [lyn, 11/10/12] Checks if parent has changed. If so, checks if curent variab
le name
```

```
447:      //    is still in scope. If so, keeps it as is; if not, changes to ???
448:      //    *** NEED TO MAKE THIS BEHAVIOR BETTER!
449:     if (this.fieldVar_) {
450:        var currentName = this.fieldVar_.getText();
451:        var nameList = this.fieldVar_.getNamesInScope();
452:        var cachedParent = this.fieldVar_.getCachedParent();
453:        var currentParent = this.fieldVar_.getBlock().getParent();
454:        // [lyn, 11/10/12] Allow current name to stay if block moved to workspace in
"untethered" way.
455:        //   Only changed to ??? if tether an untethered block.
456:        if (currentParent != cachedParent) {
457:          this.fieldVar_.setCachedParent(currentParent);
458:          if  (currentParent !== null) {
459:            for (var i = 0; i < nameList.length; i++ ) {
460:              if (nameList[i] === currentName) {
461:                return; // no change
462:              }
463:            }
464:            // Only get here if name not in list
465:            this.fieldVar_.setText(" ");
466:          }
467:        }
468:     }
469:     Blockly.WarningHandler.checkErrors.call(this);
470:   },
471:   renameLexicalVar: function(oldName, newName) {
472:     // console.log("Renaming lexical variable from " + oldName + " to " + newName);
473:     if (oldName === this.getFieldValue('VAR')) {
474:         this.setFieldValue(newName, 'VAR');
475:     }
476:   }
477: };
478: */
479:
480: Blockly.Blocks['controls_while'] = {
481:   // While condition.
482:   category: 'Control',
483:   helpUrl: Blockly.Msg.LANG_CONTROLS_WHILE_HELPURL,
484:   init: function () {
485:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
486:     this.appendValueInput('TEST')
487:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
488:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_TITLE)
489:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_TEST)
490:         .setAlign(Blockly.ALIGN_RIGHT);
491:     this.appendStatementInput('DO')
492:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_DO)
493:         .setAlign(Blockly.ALIGN_RIGHT);
494:     this.setPreviousStatement(true);
495:     this.setNextStatement(true);
496:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_WHILE_TOOLTIP);
497:   },
498:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_WHILE_TITLE}]
499: };
500:
501: // [lyn, 01/15/2013] Remove DO C-sockets because now handled more modularly by DO-TH
EN-RETURN block.
502: Blockly.Blocks['controls_choose'] = {
503:   // Choose.
504:   category: 'Control',
505:   helpUrl: Blockly.Msg.LANG_CONTROLS_CHOOSE_HELPURL,
506:   init: function () {
507:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
508:     this.setOutput(true, null);
509:     this.appendValueInput('TEST')
510:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
511:         .appendField(Blockly.Msg.LANG_CONTROLS_CHOOSE_TITLE)
512:         .appendField(Blockly.Msg.LANG_CONTROLS_CHOOSE_INPUT_TEST)
513:         .setAlign(Blockly.ALIGN_RIGHT);
514:     // this.appendStatementInput('DO0').appendField('then-do').setAlign(Blockly.ALIG
N_RIGHT);
515:     this.appendValueInput('THENRETURN')
516:         .appendField(Blockly.Msg.LANG_CONTROLS_CHOOSE_INPUT_THEN_RETURN)
517:         .setAlign(Blockly.ALIGN_RIGHT);
518:     // this.appendStatementInput('ELSE').appendField('else-do').setAlign(Blockly.ALI
GN_RIGHT);
519:     this.appendValueInput('ELSERETURN')
520:         .appendField(Blockly.Msg.LANG_CONTROLS_CHOOSE_INPUT_ELSE_RETURN)
521:         .setAlign(Blockly.ALIGN_RIGHT);
522:     /* this.setTooltip('If the condition being tested is true, the agent will '
523:      + 'run all the blocks attached to the \'then-do\' section and return the value
attached '
524:      + 'to the \'then-return\'slot. Otherwise, the agent will run all blocks attache
d to '
525:      + 'the \'else-do\' section and return the value in the \'else-return\' slot.');
526:      */
527:     // [lyn, 01/15/2013] Edit description to be consistent with changes to slots.
528:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_CHOOSE_TOOLTIP);
529:   },
530:   typeblock: [{
531:     translatedName: Blockly.Msg.LANG_CONTROLS_CHOOSE_TITLE + ' ' +
532:     Blockly.Msg.LANG_CONTROLS_CHOOSE_INPUT_THEN_RETURN + ' ' +
533:     Blockly.Msg.LANG_CONTROLS_CHOOSE_INPUT_ELSE_RETURN
534:   }]
535: };
536:
537: // [lyn, 10/10/13] This used to be in the control drawer as well as the procedure dr
awer
538: // but someone removed it from the control drawer. I think it still belongs here.
539: Blockly.Blocks['controls_do_then_return'] = {
540:   // String length.
541:   category: 'Control',
542:   helpUrl: Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_HELPURL,
543:   init: function () {
544:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
545:     this.appendStatementInput('STM')
546:         .appendField(Blockly.Msg.LANG_CONTROLS_DO_THEN_RETURN_INPUT_DO);
547:     this.appendValueInput('VALUE')
548:         .appendField(Blockly.Msg.LANG_CONTROLS_DO_THEN_RETURN_INPUT_RETURN)
549:         .setAlign(Blockly.ALIGN_RIGHT);
550:     this.setOutput(true, null);
551:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_DO_THEN_RETURN_TOOLTIP);
552:   },
553:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_DO_THEN_RETURN_TITLE}]
554: };
555:
556: // [lyn, 01/15/2013] Added
557: Blockly.Blocks['controls_eval_but_ignore'] = {
558:   category: 'Control',
559:   helpUrl: Blockly.Msg.LANG_CONTROLS_EVAL_BUT_IGNORE_HELPURL,
560:   init: function () {
561:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
```

```
562:     this.appendValueInput('VALUE')
563:         .appendField(Blockly.Msg.LANG_CONTROLS_EVAL_BUT_IGNORE_TITLE);
564:     this.setPreviousStatement(true);
565:     this.setNextStatement(true);
566:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_EVAL_BUT_IGNORE_TOOLTIP);
567:   },
568:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_EVAL_BUT_IGNORE_TITLE}]
569: };
570:
571: /* [lyn 10/10/13] Hal doesn't like NOTHING. Must rethink
572:  // [lyn, 01/15/2013] Added
573:  Blockly.Blocks.controls_nothing = {
574:  // Expression for the nothing value
575:  category: 'Control',
576:  helpUrl: Blockly.Msg.LANG_CONTROLS_NOTHING_HELPURL,
577:  init: function() {
578:  this.setColour(Blockly.CONTROL_CATEGORY_HUE);
579:  this.appendDummyInput()
580:  .appendField("nothing");
581:  this.setOutput(true, null);
582:  this.setTooltip(Blockly.Msg.LANG_CONTROLS_NOTHING_TOOLTIP);
583:  },
584:  onchange: Blockly.WarningHandler.checkErrors,
585:  typeblock: [{ translatedName: Blockly.Msg.LANG_CONTROLS_NOTHING_TITLE }]
586:  };
587:  */
588:
589: Blockly.Blocks['controls_openAnotherScreen'] = {
590:   // Open another screen
591:   category: 'Control',
592:   helpUrl: Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_HELPURL,
593:   init: function () {
594:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
595:     this.appendValueInput('SCREEN')
596:         .appendField(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_TITLE)
597:         .appendField(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_INPUT_SCREENNAME)
598:         .setAlign(Blockly.ALIGN_RIGHT)
599:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT));
600:     this.setPreviousStatement(true);
601:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_TOOLTIP);
602:   },
603:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_TITLE}]
604: };
605:
606: Blockly.Blocks['controls_openAnotherScreenWithStartValue'] = {
607:   // Open another screen with start value
608:   category: 'Control',
609:   helpUrl: Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_START_VALUE_HELPURL,
610:   init: function () {
611:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
612:     this.appendValueInput('SCREENNAME')
613:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
614:         .appendField(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_START_VALUE_
TITLE)
615:         .appendField(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_START_VALUE_
INPUT_SCREENNAME)
616:         .setAlign(Blockly.ALIGN_RIGHT);
617:     this.appendValueInput('STARTVALUE')
618:         .appendField(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_START_VALUE_
INPUT_STARTVALUE)
619:         .setAlign(Blockly.ALIGN_RIGHT);
620:     this.setPreviousStatement(true);
621:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_START_VALUE_T
OOLTIP);
622:   },
623:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_OPEN_ANOTHER_SCREEN_WITH_ST
ART_VALUE_TITLE}]
624: };
625:
626: Blockly.Blocks['controls_getStartValue'] = {
627:   // Get start value
628:   category: 'Control',
629:   helpUrl: Blockly.Msg.LANG_CONTROLS_GET_START_VALUE_HELPURL,
630:   init: function () {
631:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
632:     this.setOutput(true, null);
633:     this.appendDummyInput()
634:         .appendField(Blockly.Msg.LANG_CONTROLS_GET_START_VALUE_TITLE);
635:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_GET_START_VALUE_TOOLTIP);
636:   },
637:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_GET_START_VALUE_TITLE}]
638: };
639:
640: Blockly.Blocks['controls_closeScreen'] = {
641:   // Close screen
642:   category: 'Control',
643:   helpUrl: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_HELPURL,
644:   init: function () {
645:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
646:     this.appendDummyInput()
647:         .appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_TITLE);
648:     this.setPreviousStatement(true);
649:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_TOOLTIP);
650:   },
651:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_TITLE}]
652: };
653:
654: Blockly.Blocks['controls_closeScreenWithValue'] = {
655:   // Close screen with value
656:   category: 'Control',
657:   helpUrl: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_VALUE_HELPURL,
658:   init: function () {
659:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
660:     this.appendValueInput('SCREEN')
661:         .appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_VALUE_TITLE)
662:         .appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_VALUE_INPUT_RESULT)
663:         .setAlign(Blockly.ALIGN_RIGHT);
664:     this.setPreviousStatement(true);
665:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_VALUE_TOOLTIP);
666:   },
667:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_VALUE_TIT
LE}]
668: };
669:
670: Blockly.Blocks['controls_closeApplication'] = {
671:   // Close application
672:   category: 'Control',
673:   helpUrl: Blockly.Msg.LANG_CONTROLS_CLOSE_APPLICATION_HELPURL,
674:   init: function () {
675:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
676:     this.appendDummyInput().appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_APPLICATION_
TITLE);
```

```
  677:     this.setPreviousStatement(true);
  678:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_CLOSE_APPLICATION_TOOLTIP);
  679:   },
  680:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_CLOSE_APPLICATION_TITLE}]
  681: };
  682:
  683: Blockly.Blocks['controls_getPlainStartText'] = {
  684:   // Get plain start text
  685:   category: 'Control',
  686:   helpUrl: Blockly.Msg.LANG_CONTROLS_GET_PLAIN_START_TEXT_HELPURL,
  687:   init: function () {
  688:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
  689:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
  690:     this.appendDummyInput()
  691:         .appendField(Blockly.Msg.LANG_CONTROLS_GET_PLAIN_START_TEXT_TITLE);
  692:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_GET_PLAIN_START_TEXT_TOOLTIP);
  693:   },
  694:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_GET_PLAIN_START_TEXT_TITLE}
]
  695: };
  696:
  697: Blockly.Blocks['controls_closeScreenWithPlainText'] = {
  698:   // Close screen with plain text
  699:   category: 'Control',
  700:   helpUrl: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_PLAIN_TEXT_HELPURL,
  701:   init: function () {
  702:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
  703:     this.appendValueInput('TEXT')
  704:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  705:         .appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_PLAIN_TEXT_TITLE)
  706:         .appendField(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_PLAIN_TEXT_INPUT_TE
XT)
  707:         .setAlign(Blockly.ALIGN_RIGHT);
  708:     this.setPreviousStatement(true);
  709:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_PLAIN_TEXT_TOOLTIP);
  710:   },
  711:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_CLOSE_SCREEN_WITH_PLAIN_TEX
T_TITLE}]
  712: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Logic blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Blocks.logic');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['logic_boolean'] = {
  // Boolean data type: true and false.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'BOOL');
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('BOOL');
      return Blockly.Blocks.logic_boolean.TOOLTIPS()[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('BOOL');
    return Blockly.Blocks.logic_boolean.HELPURLS()[op];
  },
  typeblock: [{
    translatedName: Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE,
    dropDown: {
      titleName: 'BOOL',
      value: 'TRUE'
    }
  }, {
    translatedName: Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE,
    dropDown: {
      titleName: 'BOOL',
      value: 'FALSE'
    }
  }]
};

Blockly.Blocks.logic_boolean.OPERATORS = function () {
  return [
    [Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE, 'TRUE'],
    [Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE, 'FALSE']
  ];
};

Blockly.Blocks.logic_boolean.TOOLTIPS = function () {
  return {
    TRUE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TOOLTIP_TRUE,
    FALSE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TOOLTIP_FALSE
  }
};

Blockly.Blocks.logic_boolean.HELPURLS = function () {
  return {
    TRUE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE_HELPURL,
    FALSE: Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE_HELPURL
  }
};

Blockly.Blocks['logic_false'] = {
  // Boolean data type: true and false.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown(Blockly.Blocks.logic_boolean.OPERATORS), 'BOOL');
    this.setFieldValue('FALSE', 'BOOL');
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('BOOL');
      return Blockly.Blocks.logic_boolean.TOOLTIPS()[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('BOOL');
    return Blockly.Blocks.logic_boolean.HELPURLS()[op];
  }
};

Blockly.Blocks['logic_negate'] = {
  // Negation.
  category: 'Logic',
  helpUrl: Blockly.Msg.LANG_LOGIC_NEGATE_HELPURL,
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('BOOL')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_LOGIC_NEGATE_INPUT_NOT);
    this.setTooltip(Blockly.Msg.LANG_LOGIC_NEGATE_TOOLTIP);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_LOGIC_NEGATE_INPUT_NOT}]
};

Blockly.Blocks['logic_compare'] = {
  // Comparison operator.
  category: 'Logic',
  helpUrl: function () {
    var mode = this.getFieldValue('OP');
    return Blockly.Blocks.logic_compare.HELPURLS()[mode];
  },
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A');
    this.appendValueInput('B')
```

```
        .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      var mode = thisBlock.getFieldValue('OP');
      return Blockly.Blocks.logic_compare.TOOLTIPS()[mode];
    });
  },
  // Potential clash with Math =, so using 'logic equal' for now
  typeblock: [{translatedName: Blockly.Msg.LANG_LOGIC_COMPARE_TRANSLATED_NAME}]
};

Blockly.Blocks.logic_compare.TOOLTIPS = function () {
  return {
    EQ: Blockly.Msg.LANG_LOGIC_COMPARE_TOOLTIP_EQ,
    NEQ: Blockly.Msg.LANG_LOGIC_COMPARE_TOOLTIP_NEQ
  }
};

Blockly.Blocks.logic_compare.HELPURLS = function () {
  return {
    EQ: Blockly.Msg.LANG_LOGIC_COMPARE_HELPURL_EQ,
    NEQ: Blockly.Msg.LANG_LOGIC_COMPARE_HELPURL_NEQ
  }
};

Blockly.Blocks.logic_compare.OPERATORS = function () {
  return [
    [Blockly.Msg.LANG_LOGIC_COMPARE_EQ, 'EQ'],
    [Blockly.Msg.LANG_LOGIC_COMPARE_NEQ, 'NEQ']
  ];
};

Blockly.Blocks['logic_operation'] = {
  // Logical operations: 'and', 'or'.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
        .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('OP');
      return Blockly.Blocks.logic_operation.TOOLTIPS()[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('OP');
    return Blockly.Blocks.logic_operation.HELPURLS()[op];
  },
  typeblock: [{
    translatedName: Blockly.Msg.LANG_LOGIC_OPERATION_AND,
    dropDown: {
      titleName: 'OP',
      value: 'AND'
    }
  }, {
    translatedName: Blockly.Msg.LANG_LOGIC_OPERATION_OR,
    dropDown: {
      titleName: 'OP',
      value: 'OR'
    }
  }]
};

Blockly.Blocks.logic_operation.OPERATORS = function () {
  return [
    [Blockly.Msg.LANG_LOGIC_OPERATION_AND, 'AND'],
    [Blockly.Msg.LANG_LOGIC_OPERATION_OR, 'OR']
  ]
};

Blockly.Blocks.logic_operation.HELPURLS = function () {
  return {
    AND: Blockly.Msg.LANG_LOGIC_OPERATION_HELPURL_AND,
    OR: Blockly.Msg.LANG_LOGIC_OPERATION_HELPURL_OR
  }
};
Blockly.Blocks.logic_operation.TOOLTIPS = function () {
  return {
    AND: Blockly.Msg.LANG_LOGIC_OPERATION_TOOLTIP_AND,
    OR: Blockly.Msg.LANG_LOGIC_OPERATION_TOOLTIP_OR
  }
};

Blockly.Blocks['logic_or'] = {
  // Logical operations: 'and', 'or'.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
        .appendField(new Blockly.FieldDropdown(Blockly.Blocks.logic_operation.OPERAT
ORS), 'OP');
    this.setFieldValue('OR', 'OP');
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('OP');
      return Blockly.Blocks.logic_operation.TOOLTIPS[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('OP');
    return Blockly.Blocks.logic_operation.HELPURLS[op];
  }
};
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @license
    7:  * @fileoverview Math blocks for Blockly, modified for MIT App Inventor.
    8:  * @author mckinney@mit.edu (Andrew F. McKinney)
    9:  */
   10:
   11: 'use strict';
   12:
   13: goog.provide('Blockly.Blocks.math');
   14:
   15: goog.require('Blockly.Blocks.Utilities');
   16:
   17: Blockly.Blocks['math_number'] = {
   18:   // Numeric value.
   19:   category: 'Math',
   20:   helpUrl: Blockly.Msg.LANG_MATH_NUMBER_HELPURL,
   21:   init: function () {
   22:     this.setColour(Blockly.MATH_CATEGORY_HUE);
   23:     this.appendDummyInput().appendField(
   24:         new Blockly.FieldTextInput('0', Blockly.Blocks.math_number.validator), 'NUM'
);
   25:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
   26:     this.setTooltip(Blockly.Msg.LANG_MATH_NUMBER_TOOLTIP);
   27:   },
   28:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_MUTATOR_ITEM_INPUT_NUMBER}]
   29: };
   30:
   31: Blockly.Blocks.math_number.validator = function (text) {
   32:   // Ensure that only a number may be entered.
   33:   // TODO: Handle cases like 'o', 'ten', '1,234', '3,14', etc.
   34:   var n = window.parseFloat(text || 0);
   35:   return window.isNaN(n) ? null : String(n);
   36: };
   37:
   38: Blockly.Blocks['math_compare'] = {
   39:   // Basic arithmetic operator.
   40:   // TODO(Andrew): equality block needs to have any on the sockets.
   41:   category: 'Math',
   42:   helpUrl: function () {
   43:     var mode = this.getFieldValue('OP');
   44:     return Blockly.Blocks.math_compare.HELPURLS()[mode];
   45:   },
   46:   init: function () {
   47:     this.setColour(Blockly.MATH_CATEGORY_HUE);
   48:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
   49:     this.appendValueInput('A')
   50:         .setCheck(null);
   51:     this.appendValueInput('B')
   52:         .setCheck(null)
   53:         .appendField(new Blockly.FieldDropdown(this.OPERATORS, Blockly.Blocks.math_c
ompare.onchange), 'OP');
   54:     this.setInputsInline(true);
   55:     // Assign 'this' to a variable for use in the tooltip closure below.
   56:     var thisBlock = this;
   57:     this.setTooltip(function () {
   58:       var mode = thisBlock.getFieldValue('OP');
   59:       return Blockly.Blocks.math_compare.TOOLTIPS()[mode];
   60:     });
   61:   },
   62:   // Potential clash with logic equal, using '=' for now
   63:   typeblock: [{
   64:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_EQ,
   65:     dropDown: {
   66:       titleName: 'OP',
   67:       value: 'EQ'
   68:     }
   69:   }, {
   70:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_NEQ,
   71:     dropDown: {
   72:       titleName: 'OP',
   73:       value: 'NEQ'
   74:     }
   75:   }, {
   76:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_LT,
   77:     dropDown: {
   78:       titleName: 'OP',
   79:       value: 'LT'
   80:     }
   81:   }, {
   82:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_LTE,
   83:     dropDown: {
   84:       titleName: 'OP',
   85:       value: 'LTE'
   86:     }
   87:   }, {
   88:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_GT,
   89:     dropDown: {
   90:       titleName: 'OP',
   91:       value: 'GT'
   92:     }
   93:   }, {
   94:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_GTE,
   95:     dropDown: {
   96:       titleName: 'OP',
   97:       value: 'GTE'
   98:     }
   99:   }]
  100: };
  101:
  102: Blockly.Blocks.math_compare.onchange = function (value) {
  103:   if (!this.sourceBlock_) {
  104:     return;
  105:   }
  106:   if (value == "EQ" || value == "NEQ") {
  107:     this.sourceBlock_.getInput("A").setCheck(null);
  108:     this.sourceBlock_.getInput("B").setCheck(null);
  109:   } else {
  110:     this.sourceBlock_.getInput("A")
  111:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  112:     this.sourceBlock_.getInput("B")
  113:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  114:   }
  115: };
  116:
  117: Blockly.Blocks.math_compare.OPERATORS = function () {
  118:   return [[Blockly.Msg.LANG_MATH_COMPARE_EQ, 'EQ'],
```

```
119:     [Blockly.Msg.LANG_MATH_COMPARE_NEQ, 'NEQ'],
120:     [Blockly.Msg.LANG_MATH_COMPARE_LT, 'LT'],
121:     [Blockly.Msg.LANG_MATH_COMPARE_LTE, 'LTE'],
122:     [Blockly.Msg.LANG_MATH_COMPARE_GT, 'GT'],
123:     [Blockly.Msg.LANG_MATH_COMPARE_GTE, 'GTE']];
124: };
125:
126: Blockly.Blocks.math_compare.TOOLTIPS = function () {
127:   return {
128:     EQ: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_EQ,
129:     NEQ: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_NEQ,
130:     LT: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_LT,
131:     LTE: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_LTE,
132:     GT: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_GT,
133:     GTE: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_GTE
134:   }
135: };
136:
137: Blockly.Blocks.math_compare.HELPURLS = function () {
138:   return {
139:     EQ: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_EQ,
140:     NEQ: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_NEQ,
141:     LT: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_LT,
142:     LTE: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_LTE,
143:     GT: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_GT,
144:     GTE: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_GTE
145:   }
146: };
147:
148: Blockly.Blocks['math_add'] = {
149:   // Basic arithmetic operator.
150:   category: 'Math',
151:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_ADD,
152:   init: function () {
153:     this.setColour(Blockly.MATH_CATEGORY_HUE);
154:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
155:     this.appendValueInput('NUM0')
156:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
157:     // append the title on a separate line to avoid overly long lines
158:     this.appendValueInput('NUM1')
159:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
160:         .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_ADD);
161:     this.setInputsInline(true);
162:     // Assign 'this' to a variable for use in the tooltip closure below.
163:     var thisBlock = this;
164:     this.setTooltip(function () {
165:       return Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_ADD;
166:     });
167:     this.setMutator(new Blockly.Mutator(['math_mutator_item']));
168:     this.emptyInputName = 'EMPTY';
169:     this.repeatingInputName = 'NUM';
170:     this.itemCount_ = 2;
171:   },
172:   mutationToDom: Blockly.mutationToDom,
173:   domToMutation: Blockly.domToMutation,
174:   decompose: function (workspace) {
175:     return Blockly.decompose(workspace, 'math_mutator_item', this);
176:   },
177:   compose: Blockly.compose,
178:   saveConnections: Blockly.saveConnections,
179:   addEmptyInput: function () {
180:     var input = this.appendDummyInput(this.emptyInputName);
181:   },
182:   addInput: function (inputNum) {
183:     var input = this.appendValueInput(this.repeatingInputName + inputNum)
184:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
185:     if (inputNum !== 0) {
186:       input.appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_ADD);
187:     }
188:     return input;
189:   },
190:   updateContainerBlock: function (containerBlock) {
191:     containerBlock.setFieldValue("+", "CONTAINER_TEXT");
192:   },
193:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_ADD}]
194: };
195:
196: Blockly.Blocks['math_mutator_item'] = {
197:   // Add items.
198:   init: function () {
199:     this.setColour(Blockly.MATH_CATEGORY_HUE);
200:     this.appendDummyInput()
201:       //.appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TITLE);
202:         .appendField("number");
203:     this.setPreviousStatement(true);
204:     this.setNextStatement(true);
205:     //this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TOOLTIP_1);
206:     this.contextMenu = false;
207:   }
208: };
209:
210: Blockly.Blocks['math_subtract'] = {
211:   // Basic arithmetic operator.
212:   category: 'Math',
213:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_MINUS,
214:   init: function () {
215:     this.setColour(Blockly.MATH_CATEGORY_HUE);
216:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
217:     this.appendValueInput('A')
218:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
219:     this.appendValueInput('B')
220:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
221:         .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_MINUS);
222:     this.setInputsInline(true);
223:     // Assign 'this' to a variable for use in the tooltip closure below.
224:     this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_MINUS);
225:   },
226:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_MINUS}]
227: };
228:
229: Blockly.Blocks['math_multiply'] = {
230:   // Basic arithmetic operator.
231:   category: 'Math',
232:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_MULTIPLY,
233:   init: function () {
234:     this.setColour(Blockly.MATH_CATEGORY_HUE);
235:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
```

```
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('NUM0')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('NUM1')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Blocks.Utilities.times_symbol);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      return Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_MULTIPLY;
    });
    this.setMutator(new Blockly.Mutator(['math_mutator_item']));
    this.emptyInputName = 'EMPTY';
    this.repeatingInputName = 'NUM';
    this.itemCount_ = 2;
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function (workspace) {
    return Blockly.decompose(workspace, 'math_mutator_item', this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function () {
    var input = this.appendDummyInput(this.emptyInputName);
  },
  addInput: function (inputNum) {
    var input = this.appendValueInput(this.repeatingInputName + inputNum)
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    if (inputNum !== 0) {
      input.appendField(Blockly.Blocks.Utilities.times_symbol);
    }
    return input;
  },
  updateContainerBlock: function (containerBlock) {
    containerBlock.setFieldValue(Blockly.Blocks.Utilities.times_symbol, "CONTAINER_T
EXT");
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_MULTIPLY}]
};

Blockly.Blocks['math_division'] = {
  // Basic arithmetic operator.
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_DIVIDE,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_DIVIDE);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_DIVIDE);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_DIVIDE}]
};

Blockly.Blocks['math_power'] = {
  // Basic arithmetic operator.
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_POWER,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_POWER);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_POWER);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_POWER}]
};

Blockly.Blocks['math_random_int'] = {
  // Random integer between [X] and [Y].
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_RANDOM_INT_HELPURL,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));

    var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", B
lockly.Blocks.Utilities.INPUT);
    this.interpolateMsg(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT,
        ['FROM', checkTypeNumber, Blockly.ALIGN_RIGHT],
        ['TO', checkTypeNumber, Blockly.ALIGN_RIGHT],
        Blockly.ALIGN_RIGHT)
    /*this.appendValueInput('FROM')
     .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Block
s.Utilities.INPUT))
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_TITLE_RANDOM)
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT_FROM);
     this.appendValueInput('TO')
     .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Block
s.Utilities.INPUT))
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT_TO);*/
    this.setInputsInline(true);
    this.setTooltip(Blockly.Msg.LANG_MATH_RANDOM_INT_TOOLTIP);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_RANDOM_INT_TITLE_RANDOM}]
};

Blockly.Blocks['math_random_float'] = {
  // Random fraction between 0 and 1.
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_RANDOM_FLOAT_HELPURL,
```

```
  345:   init: function () {
  346:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  347:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  348:     this.appendDummyInput()
  349:         .appendField(Blockly.Msg.LANG_MATH_RANDOM_FLOAT_TITLE_RANDOM);
  350:     this.setTooltip(Blockly.Msg.LANG_MATH_RANDOM_FLOAT_TOOLTIP);
  351:   },
  352:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_RANDOM_FLOAT_TITLE_RANDOM}]
  353: };
  354:
  355: Blockly.Blocks['math_random_set_seed'] = {
  356:   // Set the seed of the radom number generator
  357:   category: 'Math',
  358:   helpUrl: Blockly.Msg.LANG_MATH_RANDOM_SEED_HELPURL,
  359:   init: function () {
  360:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  361:     this.setOutput(false, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", B
lockly.Blocks.Utilities.OUTPUT));
  362:     this.appendValueInput('NUM')
  363:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  364:         .appendField(Blockly.Msg.LANG_MATH_RANDOM_SEED_TITLE_RANDOM)
  365:         .appendField(Blockly.Msg.LANG_MATH_RANDOM_SEED_INPUT_TO);
  366:     this.setPreviousStatement(true);
  367:     this.setNextStatement(true);
  368:     this.setTooltip(Blockly.Msg.LANG_MATH_RANDOM_SEED_TOOLTIP);
  369:   },
  370:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_RANDOM_SEED_TITLE_RANDOM}]
  371: };
  372:
  373: Blockly.Blocks['math_on_list'] = {
  374:   // Evaluate a list of numbers to return sum, average, min, max, etc.
  375:   // Some functions also work on text (min, max, mode, median).
  376:   category: 'Math',
  377:   helpUrl: '',
  378:   init: function () {
  379:     // Assign 'this' to a variable for use in the closures below.
  380:     var thisBlock = this;
  381:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  382:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  383:     this.appendValueInput('NUM0')
  384:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  385:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
  386:     this.appendValueInput('NUM1')
  387:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  388:     this.setInputsInline(false);
  389:     this.setTooltip(function () {
  390:       var mode = thisBlock.getFieldValue('OP');
  391:       return Blockly.Blocks.math_on_list.TOOLTIPS()[mode];
  392:     });
  393:     this.setMutator(new Blockly.Mutator(['math_mutator_item']));
  394:     this.itemCount_ = 2;
  395:     this.valuesToSave = {'OP': null};
  396:     this.emptyInputName = 'EMPTY';
  397:     this.repeatingInputName = 'NUM';
  398:   },
  399:   mutationToDom: Blockly.mutationToDom,
  400:   domToMutation: Blockly.domToMutation,
  401:   decompose: function (workspace) {
  402:     return Blockly.decompose(workspace, 'math_mutator_item', this);
  403:   },
  404:   compose: Blockly.compose,
  405:   saveConnections: Blockly.saveConnections,
  406:   addEmptyInput: function () {
  407:     var input = this.appendDummyInput(this.emptyInputName);
  408:     input.appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
  409:     this.setFieldValue(this.valuesToSave['OP'], 'OP');
  410:   },
  411:   addInput: function (inputNum) {
  412:     var input = this.appendValueInput(this.repeatingInputName + inputNum)
  413:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  414:     if (inputNum == 0) {
  415:       input.appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
  416:       this.setFieldValue(this.valuesToSave['OP'], 'OP');
  417:     }
  418:     return input;
  419:   },
  420:   updateContainerBlock: function (containerBlock) {
  421:
  422:     for (var i = 0; i < Blockly.Blocks.math_on_list.OPERATORS.length; i++) {
  423:       if (Blockly.Blocks.math_on_list.OPERATORS[i][1] == this.getFieldValue("OP")) {
  424:         containerBlock.setFieldValue(Blockly.Blocks.math_on_list.OPERATORS[i][0], "C
ONTAINER_TEXT");
  425:       }
  426:     }
  427:
  428:   },
  429:   typeblock: [{
  430:     translatedName: Blockly.Msg.LANG_MATH_ONLIST_OPERATOR_MIN,
  431:     dropDown: {
  432:       titleName: 'OP',
  433:       value: 'MIN'
  434:     }
  435:   }, {
  436:     translatedName: Blockly.Msg.LANG_MATH_ONLIST_OPERATOR_MAX,
  437:     dropDown: {
  438:       titleName: 'OP',
  439:       value: 'MAX'
  440:     }
  441:   }]
  442: };
  443:
  444: Blockly.Blocks.math_on_list.OPERATORS = function () {
  445:   return [[Blockly.Msg.LANG_MATH_ONLIST_OPERATOR_MIN, 'MIN'],
  446:     [Blockly.Msg.LANG_MATH_ONLIST_OPERATOR_MAX, 'MAX']]
  447: };
  448:
  449: Blockly.Blocks.math_on_list.TOOLTIPS = function () {
  450:   return {
  451:     MIN: Blockly.Msg.LANG_MATH_ONLIST_TOOLTIP_MIN,
  452:     MAX: Blockly.Msg.LANG_MATH_ONLIST_TOOLTIP_MAX
  453:   }
  454: };
  455:
  456: Blockly.Blocks['math_single'] = {
  457:   // Advanced math operators with single operand.
  458:   category: 'Math',
  459:   helpUrl: function () {
  460:     var mode = this.getFieldValue('OP');
```

```
461:     return Blockly.Blocks.math_single.HELPURLS()[mode];
462:   },
463:   init: function () {
464:     this.setColour(Blockly.MATH_CATEGORY_HUE);
465:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
466:     this.appendValueInput('NUM')
467:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
468:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
469:     // Assign 'this' to a variable for use in the tooltip closure below.
470:     var thisBlock = this;
471:     this.setTooltip(function () {
472:       var mode = thisBlock.getFieldValue('OP');
473:       return Blockly.Blocks.math_single.TOOLTIPS()[mode];
474:     });
475:   },
476:   typeblock: [{
477:     translatedName: Blockly.Msg.LANG_MATH_SINGLE_OP_ROOT,
478:     dropDown: {
479:       titleName: 'OP',
480:       value: 'ROOT'
481:     }
482:   }, {
483:     translatedName: Blockly.Msg.LANG_MATH_SINGLE_OP_ABSOLUTE,
484:     dropDown: {
485:       titleName: 'OP',
486:       value: 'ABS'
487:     }
488:   }, {
489:     translatedName: Blockly.Msg.LANG_MATH_SINGLE_OP_NEG,
490:     dropDown: {
491:       titleName: 'OP',
492:       value: 'NEG'
493:     }
494:   }, {
495:     translatedName: Blockly.Msg.LANG_MATH_SINGLE_OP_LN,
496:     dropDown: {
497:       titleName: 'OP',
498:       value: 'LN'
499:     }
500:   }, {
501:     translatedName: Blockly.Msg.LANG_MATH_SINGLE_OP_EXP,
502:     dropDown: {
503:       titleName: 'OP',
504:       value: 'EXP'
505:     }
506:   }, {
507:     translatedName: Blockly.Msg.LANG_MATH_ROUND_OPERATOR_ROUND,
508:     dropDown: {
509:       titleName: 'OP',
510:       value: 'ROUND'
511:     }
512:   }, {
513:     translatedName: Blockly.Msg.LANG_MATH_ROUND_OPERATOR_CEILING,
514:     dropDown: {
515:       titleName: 'OP',
516:       value: 'CEILING'
517:     }
518:   }, {
519:     translatedName: Blockly.Msg.LANG_MATH_ROUND_OPERATOR_FLOOR,
520:     dropDown: {
521:       titleName: 'OP',
522:       value: 'FLOOR'
523:     }
524:   }]
525: };
526:
527: Blockly.Blocks.math_single.OPERATORS = function () {
528:   return [[Blockly.Msg.LANG_MATH_SINGLE_OP_ROOT, 'ROOT'],
529:     [Blockly.Msg.LANG_MATH_SINGLE_OP_ABSOLUTE, 'ABS'],
530:     [Blockly.Msg.LANG_MATH_SINGLE_OP_NEG, 'NEG'],
531:     [Blockly.Msg.LANG_MATH_SINGLE_OP_LN, 'LN'],
532:     [Blockly.Msg.LANG_MATH_SINGLE_OP_EXP, 'EXP'],
533:     [Blockly.Msg.LANG_MATH_ROUND_OPERATOR_ROUND, 'ROUND'],
534:     [Blockly.Msg.LANG_MATH_ROUND_OPERATOR_CEILING, 'CEILING'],
535:     [Blockly.Msg.LANG_MATH_ROUND_OPERATOR_FLOOR, 'FLOOR']];
536: };
537:
538: Blockly.Blocks.math_single.TOOLTIPS = function () {
539:   return {
540:     ROOT: Blockly.Msg.LANG_MATH_SINGLE_TOOLTIP_ROOT,
541:     ABS: Blockly.Msg.LANG_MATH_SINGLE_TOOLTIP_ABS,
542:     NEG: Blockly.Msg.LANG_MATH_SINGLE_TOOLTIP_NEG,
543:     LN: Blockly.Msg.LANG_MATH_SINGLE_TOOLTIP_LN,
544:     EXP: Blockly.Msg.LANG_MATH_SINGLE_TOOLTIP_EXP,
545:     ROUND: Blockly.Msg.LANG_MATH_ROUND_TOOLTIP_ROUND,
546:     CEILING: Blockly.Msg.LANG_MATH_ROUND_TOOLTIP_CEILING,
547:     FLOOR: Blockly.Msg.LANG_MATH_ROUND_TOOLTIP_FLOOR
548:   }
549: };
550:
551: Blockly.Blocks.math_single.HELPURLS = function () {
552:   return {
553:     ROOT: Blockly.Msg.LANG_MATH_SINGLE_HELPURL_ROOT,
554:     ABS: Blockly.Msg.LANG_MATH_SINGLE_HELPURL_ABS,
555:     NEG: Blockly.Msg.LANG_MATH_SINGLE_HELPURL_NEG,
556:     LN: Blockly.Msg.LANG_MATH_SINGLE_HELPURL_LN,
557:     EXP: Blockly.Msg.LANG_MATH_SINGLE_HELPURL_EXP,
558:     ROUND: Blockly.Msg.LANG_MATH_ROUND_HELPURL_ROUND,
559:     CEILING: Blockly.Msg.LANG_MATH_ROUND_HELPURL_CEILING,
560:     FLOOR: Blockly.Msg.LANG_MATH_ROUND_HELPURL_FLOOR
561:   }
562: };
563:
564: Blockly.Blocks['math_abs'] = {
565:   // Advanced math operators with single operand.
566:   category: 'Math',
567:   helpUrl: function () {
568:     var mode = this.getFieldValue('OP');
569:     return Blockly.Blocks.math_single.HELPURLS[mode];
570:   },
571:   init: function () {
572:     this.setColour(Blockly.MATH_CATEGORY_HUE);
573:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
574:     this.appendValueInput('NUM')
575:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
576:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_single.OPERATORS)
, 'OP');
577:     this.setFieldValue('ABS', "OP");
578:     // Assign 'this' to a variable for use in the tooltip closure below.
579:     var thisBlock = this;
```

```
  580:     this.setTooltip(function () {
  581:       var mode = thisBlock.getFieldValue('OP');
  582:       return Blockly.Blocks.math_single.TOOLTIPS[mode];
  583:     });
  584:   }
  585: };
  586:
  587: Blockly.Blocks['math_neg'] = {
  588:   // Advanced math operators with single operand.
  589:   category: 'Math',
  590:   helpUrl: function () {
  591:     var mode = this.getFieldValue('OP');
  592:     return Blockly.Blocks.math_single.HELPURLS[mode];
  593:   },
  594:   init: function () {
  595:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  596:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  597:     this.appendValueInput('NUM')
  598:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  599:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_single.OPERATORS)
, 'OP');
  600:     this.setFieldValue('NEG', "OP");
  601:     // Assign 'this' to a variable for use in the tooltip closure below.
  602:     var thisBlock = this;
  603:     this.setTooltip(function () {
  604:       var mode = thisBlock.getFieldValue('OP');
  605:       return Blockly.Blocks.math_single.TOOLTIPS[mode];
  606:     });
  607:   }
  608: };
  609:
  610: Blockly.Blocks['math_round'] = {
  611:   // Advanced math operators with single operand.
  612:   category: 'Math',
  613:   helpUrl: function () {
  614:     var mode = this.getFieldValue('OP');
  615:     return Blockly.Blocks.math_single.HELPURLS[mode];
  616:   },
  617:   init: function () {
  618:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  619:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  620:     this.appendValueInput('NUM')
  621:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  622:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_single.OPERATORS)
, 'OP');
  623:     this.setFieldValue('ROUND', "OP");
  624:     // Assign 'this' to a variable for use in the tooltip closure below.
  625:     var thisBlock = this;
  626:     this.setTooltip(function () {
  627:       var mode = thisBlock.getFieldValue('OP');
  628:       return Blockly.Blocks.math_single.TOOLTIPS[mode];
  629:     });
  630:   }
  631: };
  632:
  633: Blockly.Blocks['math_ceiling'] = {
  634:   // Advanced math operators with single operand.
  635:   category: 'Math',
  636:   helpUrl: function () {
  637:     var mode = this.getFieldValue('OP');
  638:     return Blockly.Blocks.math_single.HELPURLS[mode];
  639:   },
  640:   init: function () {
  641:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  642:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  643:     this.appendValueInput('NUM')
  644:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  645:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_single.OPERATORS)
, 'OP');
  646:     this.setFieldValue('CEILING', "OP");
  647:     // Assign 'this' to a variable for use in the tooltip closure below.
  648:     var thisBlock = this;
  649:     this.setTooltip(function () {
  650:       var mode = thisBlock.getFieldValue('OP');
  651:       return Blockly.Blocks.math_single.TOOLTIPS[mode];
  652:     });
  653:   }
  654: };
  655:
  656: Blockly.Blocks['math_floor'] = {
  657:   // Advanced math operators with single operand.
  658:   category: 'Math',
  659:   helpUrl: function () {
  660:     var mode = this.getFieldValue('OP');
  661:     return Blockly.Blocks.math_single.HELPURLS[mode];
  662:   },
  663:   init: function () {
  664:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  665:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  666:     this.appendValueInput('NUM')
  667:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  668:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_single.OPERATORS)
, 'OP');
  669:     this.setFieldValue('FLOOR', "OP");
  670:     // Assign 'this' to a variable for use in the tooltip closure below.
  671:     var thisBlock = this;
  672:     this.setTooltip(function () {
  673:       var mode = thisBlock.getFieldValue('OP');
  674:       return Blockly.Blocks.math_single.TOOLTIPS[mode];
  675:     });
  676:   }
  677: };
  678:
  679: Blockly.Blocks['math_divide'] = {
  680:   // Remainder or quotient of a division.
  681:   category: 'Math',
  682:   helpUrl: function () {
  683:     var mode = this.getFieldValue('OP');
  684:     return Blockly.Blocks.math_divide.HELPURLS()[mode];
  685:   },
  686:   init: function () {
  687:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  688:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  689:     this.appendValueInput('DIVIDEND')
  690:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
```

```
locks.Utilities.INPUT))
          .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
      this.appendValueInput('DIVISOR')
          .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
          .appendField(Blockly.Msg.LANG_MATH_DIVIDE);
      this.setInputsInline(true);
      // Assign 'this' to a variable for use in the tooltip closure below.
      var thisBlock = this;
      this.setTooltip(function () {
        var mode = thisBlock.getFieldValue('OP');
        return Blockly.Blocks.math_divide.TOOLTIPS()[mode];
      });
    },
    typeblock: [{
      translatedName: Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_MODULO,
      dropDown: {
        titleName: 'OP',
        value: 'MODULO'
      }
    }, {
      translatedName: Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_REMAINDER,
      dropDown: {
        titleName: 'OP',
        value: 'REMAINDER'
      }
    }, {
      translatedName: Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_QUOTIENT,
      dropDown: {
        titleName: 'OP',
        value: 'QUOTIENT'
      }
    }]
};

Blockly.Blocks.math_divide.OPERATORS = function () {
  return [[Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_MODULO, 'MODULO'],
    [Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_REMAINDER, 'REMAINDER'],
    [Blockly.Msg.LANG_MATH_DIVIDE_OPERATOR_QUOTIENT, 'QUOTIENT']];
};

Blockly.Blocks.math_divide.TOOLTIPS = function () {
  return {
    MODULO: Blockly.Msg.LANG_MATH_DIVIDE_TOOLTIP_MODULO,
    REMAINDER: Blockly.Msg.LANG_MATH_DIVIDE_TOOLTIP_REMAINDER,
    QUOTIENT: Blockly.Msg.LANG_MATH_DIVIDE_TOOLTIP_QUOTIENT
  }
};

Blockly.Blocks.math_divide.HELPURLS = function () {
  return {
    MODULO: Blockly.Msg.LANG_MATH_DIVIDE_HELPURL_MODULO,
    REMAINDER: Blockly.Msg.LANG_MATH_DIVIDE_HELPURL_REMAINDER,
    QUOTIENT: Blockly.Msg.LANG_MATH_DIVIDE_HELPURL_QUOTIENT
  }
};

Blockly.Blocks['math_trig'] = {
  // Trigonometry operators.
  category: 'Math',
  helpUrl: function () {
    var mode = this.getFieldValue('OP');
    return Blockly.Blocks.math_trig.HELPURLS()[mode];
  },
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('NUM')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
    // Assign 'this' to a variable for use in the closures below.
    var thisBlock = this;
    this.setTooltip(function () {
      var mode = thisBlock.getFieldValue('OP');
      return Blockly.Blocks.math_trig.TOOLTIPS()[mode];
    });
  },
  typeblock: [{
    translatedName: Blockly.Msg.LANG_MATH_TRIG_SIN,
    dropDown: {
      titleName: 'OP',
      value: 'SIN'
    }
  }, {
    translatedName: Blockly.Msg.LANG_MATH_TRIG_COS,
    dropDown: {
      titleName: 'OP',
      value: 'COS'
    }
  }, {
    translatedName: Blockly.Msg.LANG_MATH_TRIG_TAN,
    dropDown: {
      titleName: 'OP',
      value: 'TAN'
    }
  }, {
    translatedName: Blockly.Msg.LANG_MATH_TRIG_ASIN,
    dropDown: {
      titleName: 'OP',
      value: 'ASIN'
    }
  }, {
    translatedName: Blockly.Msg.LANG_MATH_TRIG_ACOS,
    dropDown: {
      titleName: 'OP',
      value: 'ACOS'
    }
  }, {
    translatedName: Blockly.Msg.LANG_MATH_TRIG_ATAN,
    dropDown: {
      titleName: 'OP',
      value: 'ATAN'
    }
  }]
};

Blockly.Blocks.math_trig.OPERATORS = function () {
  return [[Blockly.Msg.LANG_MATH_TRIG_SIN, 'SIN'],
    [Blockly.Msg.LANG_MATH_TRIG_COS, 'COS'],
    [Blockly.Msg.LANG_MATH_TRIG_TAN, 'TAN'],
    [Blockly.Msg.LANG_MATH_TRIG_ASIN, 'ASIN'],
    [Blockly.Msg.LANG_MATH_TRIG_ACOS, 'ACOS'],
```

```
811:     [Blockly.Msg.LANG_MATH_TRIG_ATAN, 'ATAN']];
812: }
813:
814: Blockly.Blocks.math_trig.TOOLTIPS = function () {
815:   return {
816:     SIN: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_SIN,
817:     COS: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_COS,
818:     TAN: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_TAN,
819:     ASIN: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_ASIN,
820:     ACOS: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_ACOS,
821:     ATAN: Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_ATAN
822:   }
823: };
824:
825: Blockly.Blocks.math_trig.HELPURLS = function () {
826:   return {
827:     SIN: Blockly.Msg.LANG_MATH_TRIG_HELPURL_SIN,
828:     COS: Blockly.Msg.LANG_MATH_TRIG_HELPURL_COS,
829:     TAN: Blockly.Msg.LANG_MATH_TRIG_HELPURL_TAN,
830:     ASIN: Blockly.Msg.LANG_MATH_TRIG_HELPURL_ASIN,
831:     ACOS: Blockly.Msg.LANG_MATH_TRIG_HELPURL_ACOS,
832:     ATAN: Blockly.Msg.LANG_MATH_TRIG_HELPURL_ATAN
833:   }
834: };
835:
836: Blockly.Blocks['math_cos'] = {
837:   // Trigonometry operators.
838:   category: 'Math',
839:   helpUrl: function () {
840:     var mode = this.getFieldValue('OP');
841:     return Blockly.Blocks.math_trig.HELPURLS[mode];
842:   },
843:   init: function () {
844:     this.setColour(Blockly.MATH_CATEGORY_HUE);
845:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
846:     this.appendValueInput('NUM')
847:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
848:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_trig.OPERATORS),
'OP');
849:     this.setFieldValue('COS', "OP");
850:     // Assign 'this' to a variable for use in the closures below.
851:     var thisBlock = this;
852:     this.setTooltip(function () {
853:       var mode = thisBlock.getFieldValue('OP');
854:       return Blockly.Blocks.math_trig.TOOLTIPS[mode];
855:     });
856:   }
857: };
858:
859: Blockly.Blocks['math_tan'] = {
860:   // Trigonometry operators.
861:   category: 'Math',
862:   helpUrl: function () {
863:     var mode = this.getFieldValue('OP');
864:     return Blockly.Blocks.math_trig.HELPURLS[mode];
865:   },
866:   init: function () {
867:     this.setColour(Blockly.MATH_CATEGORY_HUE);
868:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
869:     this.appendValueInput('NUM')
870:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
871:         .appendField(new Blockly.FieldDropdown(Blockly.Blocks.math_trig.OPERATORS),
'OP');
872:     this.setFieldValue('TAN', "OP");
873:     // Assign 'this' to a variable for use in the closures below.
874:     var thisBlock = this;
875:     this.setTooltip(function () {
876:       var mode = thisBlock.getFieldValue('OP');
877:       return Blockly.Blocks.math_trig.TOOLTIPS[mode];
878:     });
879:   }
880: };
881:
882: Blockly.Blocks['math_atan2'] = {
883:   // Trigonometry operators.
884:   category: 'Math',
885:   helpUrl: Blockly.Msg.LANG_MATH_TRIG_HELPURL_ATAN2,
886:   init: function () {
887:     this.setColour(Blockly.MATH_CATEGORY_HUE);
888:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
889:     this.appendDummyInput().appendField(Blockly.Msg.LANG_MATH_TRIG_ATAN2);
890:     this.appendValueInput('Y')
891:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
892:         .appendField(Blockly.Msg.LANG_MATH_TRIG_ATAN2_Y)
893:         .setAlign(Blockly.ALIGN_RIGHT);
894:     this.appendValueInput('X')
895:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
896:         .appendField(Blockly.Msg.LANG_MATH_TRIG_ATAN2_X)
897:         .setAlign(Blockly.ALIGN_RIGHT);
898:     this.setInputsInline(false);
899:     this.setTooltip(Blockly.Msg.LANG_MATH_TRIG_TOOLTIP_ATAN2);
900:   },
901:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_TRIG_ATAN2}]
902: };
903:
904: Blockly.Blocks['math_convert_angles'] = {
905:   // Trigonometry operators.
906:   category: 'Math',
907:   helpUrl: function () {
908:     var mode = this.getFieldValue('OP');
909:     return Blockly.Blocks.math_convert_angles.HELPURLS()[mode];
910:   },
911:   init: function () {
912:     this.setColour(Blockly.MATH_CATEGORY_HUE);
913:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
914:     this.appendValueInput('NUM')
915:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
916:         .appendField(Blockly.Msg.LANG_MATH_CONVERT_ANGLES_TITLE_CONVERT)
917:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
918:     // Assign 'this' to a variable for use in the closures below.
919:     var thisBlock = this;
920:     this.setTooltip(function () {
921:       var mode = thisBlock.getFieldValue('OP');
922:       return Blockly.Blocks.math_convert_angles.TOOLTIPS()[mode];
923:     });
```

```
  924:   },
  925:   typeblock: [{
  926:     translatedName: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_TITLE_CONVERT +
  927:     ' ' + Blockly.Msg.LANG_MATH_CONVERT_ANGLES_OP_RAD_TO_DEG,
  928:     dropDown: {
  929:       titleName: 'OP',
  930:       value: 'RADIANS_TO_DEGREES'
  931:     }
  932:   }, {
  933:     translatedName: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_TITLE_CONVERT +
  934:     ' ' + Blockly.Msg.LANG_MATH_CONVERT_ANGLES_OP_DEG_TO_RAD,
  935:     dropDown: {
  936:       titleName: 'OP',
  937:       value: 'DEGREES_TO_RADIANS'
  938:     }
  939:   }]
  940: };
  941:
  942: Blockly.Blocks.math_convert_angles.OPERATORS = function () {
  943:   return [[Blockly.Msg.LANG_MATH_CONVERT_ANGLES_OP_RAD_TO_DEG, 'RADIANS_TO_DEGREES']
,
  944:     [Blockly.Msg.LANG_MATH_CONVERT_ANGLES_OP_DEG_TO_RAD, 'DEGREES_TO_RADIANS']]
  945: };
  946:
  947: Blockly.Blocks.math_convert_angles.TOOLTIPS = function () {
  948:   return {
  949:     RADIANS_TO_DEGREES: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_TOOLTIP_RAD_TO_DEG,
  950:     DEGREES_TO_RADIANS: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_TOOLTIP_DEG_TO_RAD
  951:   }
  952: };
  953:
  954: Blockly.Blocks.math_convert_angles.HELPURLS = function () {
  955:   return {
  956:     RADIANS_TO_DEGREES: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_HELPURL_RAD_TO_DEG,
  957:     DEGREES_TO_RADIANS: Blockly.Msg.LANG_MATH_CONVERT_ANGLES_HELPURL_DEG_TO_RAD
  958:   }
  959: };
  960:
  961: Blockly.Blocks['math_format_as_decimal'] = {
  962:   category: 'Math',
  963:   helpUrl: Blockly.Msg.LANG_MATH_FORMAT_AS_DECIMAL_HELPURL,
  964:   init: function () {
  965:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  966:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  967:
  968:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", B
lockly.Blocks.Utilities.INPUT);
  969:     this.interpolateMsg(Blockly.Msg.LANG_MATH_FORMAT_AS_DECIMAL_INPUT,
  970:         ['NUM', checkTypeNumber, Blockly.ALIGN_RIGHT],
  971:         ['PLACES', checkTypeNumber, Blockly.ALIGN_RIGHT],
  972:         Blockly.ALIGN_RIGHT);
  973:     /*this.appendDummyInput()
  974:       .appendField('format as decimal');
  975:     this.appendValueInput('NUM')
  976:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Bloc
ks.Utilities.INPUT))
  977:       .appendField('number')
  978:       .setAlign(Blockly.ALIGN_RIGHT);
  979:     this.appendValueInput('PLACES')
  980:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Bloc
ks.Utilities.INPUT))
  981:       .appendField('places')
  982:       .setAlign(Blockly.ALIGN_RIGHT);*/
  983:     this.setInputsInline(false);
  984:     this.setTooltip(Blockly.Msg.LANG_MATH_FORMAT_AS_DECIMAL_TOOLTIP);
  985:   },
  986:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_FORMAT_AS_DECIMAL_TITLE}]
  987: };
  988:
  989: Blockly.Blocks['math_is_a_number'] = {
  990:   category: 'Math',
  991:   helpUrl: Blockly.Msg.LANG_MATH_IS_A_NUMBER_HELPURL,
  992:   init: function () {
  993:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  994:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
  995:     this.appendValueInput('NUM')
  996:         .appendField(Blockly.Msg.LANG_MATH_IS_A_NUMBER_INPUT_NUM);
  997:     this.setTooltip(function () {
  998:       return Blockly.Msg.LANG_MATH_IS_A_NUMBER_TOOLTIP;
  999:     });
 1000:   },
 1001:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_IS_A_NUMBER_INPUT_NUM}]
 1002: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @license
    7:  * @fileoverview Text blocks for Blockly, modified for MIT App Inventor.
    8:  * @author mckinney@mit.edu (Andrew F. McKinney)
    9:  */
   10:
   11: 'use strict';
   12:
   13: goog.provide('Blockly.Blocks.text');
   14:
   15: goog.require('Blockly.Blocks.Utilities');
   16:
   17: Blockly.Blocks['text'] = {
   18:   // Text value.
   19:   category: 'Text',
   20:   helpUrl: Blockly.Msg.LANG_TEXT_TEXT_HELPURL,
   21:   init: function () {
   22:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
   23:     this.appendDummyInput().appendField(Blockly.Msg.LANG_TEXT_TEXT_LEFT_QUOTE).appen
dField(
   24:         new Blockly.FieldTextBlockInput(''),
   25:         'TEXT').appendField(Blockly.Msg.LANG_TEXT_TEXT_RIGHT_QUOTE);
   26:     this.setOutput(true, [Blockly.Blocks.text.connectionCheck]);
   27:     this.setTooltip(Blockly.Msg.LANG_TEXT_TEXT_TOOLTIP);
   28:   },
   29:   typeblock: [{translatedName: Blockly.Msg.LANG_CATEGORY_TEXT}]
   30: };
   31:
   32: Blockly.Blocks.text.connectionCheck = function (myConnection, otherConnection) {
   33:   var block = myConnection.sourceBlock_;
   34:   var otherTypeArray = otherConnection.check_;
   35:   for (var i = 0; i < otherTypeArray.length; i++) {
   36:     if (otherTypeArray[i] == "String") {
   37:       return true;
   38:     } else if (otherTypeArray[i] == "Number" && !isNaN(parseFloat(block.getFieldValu
e('TEXT')))) {
   39:       return true;
   40:     }
   41:   }
   42:   return false;
   43: };
   44:
   45: Blockly.Blocks['text_join'] = {
   46:   // Create a string made up of any number of elements of any type.
   47:   // TODO: (Andrew) Make this handle multiple arguments.
   48:   category: 'Text',
   49:   helpUrl: Blockly.Msg.LANG_TEXT_JOIN_HELPURL,
   50:   init: function () {
   51:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
   52:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
   53:     this.appendValueInput('ADD0')
   54:         .appendField(Blockly.Msg.LANG_TEXT_JOIN_TITLE_JOIN);
   55:     this.appendValueInput('ADD1');
   56:     this.setTooltip(Blockly.Msg.LANG_TEXT_JOIN_TOOLTIP);
   57:     this.setMutator(new Blockly.Mutator(['text_join_item']));
   58:     this.emptyInputName = 'EMPTY';
   59:     this.repeatingInputName = 'ADD';
   60:     this.itemCount_ = 2;
   61:   },
   62:   mutationToDom: Blockly.mutationToDom,
   63:   domToMutation: Blockly.domToMutation,
   64:   decompose: function (workspace) {
   65:     return Blockly.decompose(workspace, 'text_join_item', this);
   66:   },
   67:   compose: Blockly.compose,
   68:   saveConnections: Blockly.saveConnections,
   69:   addEmptyInput: function () {
   70:     this.appendDummyInput(this.emptyInputName)
   71:         .appendField(Blockly.Msg.LANG_TEXT_JOIN_TITLE_JOIN);
   72:   },
   73:   addInput: function (inputNum) {
   74:     var input = this.appendValueInput(this.repeatingInputName + inputNum).setCheck(B
lockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blocks.Utilities.INPUT));
   75:     if (inputNum === 0) {
   76:       input.appendField(Blockly.Msg.LANG_TEXT_JOIN_TITLE_JOIN);
   77:     }
   78:     return input;
   79:   },
   80:   updateContainerBlock: function (containerBlock) {
   81:     containerBlock.inputList[0].fieldRow[0].setText(Blockly.Msg.LANG_TEXT_JOIN_TITLE
_JOIN);
   82:   },
   83:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_JOIN_TITLE_JOIN}]
   84:
   85: };
   86:
   87: Blockly.Blocks['text_join_item'] = {
   88:   // Add items.
   89:   init: function () {
   90:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
   91:     this.appendDummyInput()
   92:         .appendField(Blockly.Msg.LANG_TEXT_JOIN_ITEM_TITLE_ITEM);
   93:     this.setPreviousStatement(true);
   94:     this.setNextStatement(true);
   95:     this.setTooltip(Blockly.Msg.LANG_TEXT_JOIN_ITEM_TOOLTIP);
   96:     this.contextMenu = false;
   97:   }
   98: };
   99:
  100: Blockly.Blocks['text_length'] = {
  101:   // String length.
  102:   category: 'Text',
  103:   helpUrl: Blockly.Msg.LANG_TEXT_LENGTH_HELPURL,
  104:   init: function () {
  105:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  106:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  107:     this.appendValueInput('VALUE')
  108:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  109:         .appendField(Blockly.Msg.LANG_TEXT_LENGTH_INPUT_LENGTH);
  110:     this.setTooltip(Blockly.Msg.LANG_TEXT_LENGTH_TOOLTIP);
  111:   },
  112:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_LENGTH_INPUT_LENGTH}]
  113: };
  114:
  115: Blockly.Blocks['text_isEmpty'] = {
  116:   // Is the string null?
  117:   category: 'Text',
```

```
  118:   helpUrl: Blockly.Msg.LANG_TEXT_ISEMPTY_HELPURL,
  119:   init: function () {
  120:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  121:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
  122:     this.appendValueInput('VALUE')
  123:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  124:         .appendField(Blockly.Msg.LANG_TEXT_ISEMPTY_INPUT_ISEMPTY);
  125:     this.setTooltip(Blockly.Msg.LANG_TEXT_ISEMPTY_TOOLTIP);
  126:   },
  127:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_ISEMPTY_INPUT_ISEMPTY}]
  128: };
  129:
  130: Blockly.Blocks['text_compare'] = {
  131:   // Compare two texts
  132:   category: 'Text',
  133:   helpUrl: Blockly.Msg.LANG_TEXT_COMPARE_HELPURL,
  134:   init: function () {
  135:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  136:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
  137:     this.appendValueInput('TEXT1')
  138:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  139:         .appendField(Blockly.Msg.LANG_TEXT_COMPARE_INPUT_COMPARE);
  140:     this.appendValueInput('TEXT2')
  141:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  142:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
  143:     this.setInputsInline(true);
  144:     var thisBlock = this;
  145:     this.setTooltip(function () {
  146:       var mode = thisBlock.getFieldValue('OP');
  147:       return Blockly.Blocks.text_compare.TOOLTIPS()[mode];
  148:     });
  149:   },
  150:   typeblock: [{
  151:     translatedName: Blockly.Msg.LANG_TEXT_COMPARE_INPUT_COMPARE + Blockly.Msg.LANG_T
EXT_COMPARE_LT,
  152:     dropDown: {
  153:       titleName: 'OP',
  154:       value: 'LT'
  155:     }
  156:   }, {
  157:     translatedName: Blockly.Msg.LANG_TEXT_COMPARE_INPUT_COMPARE + Blockly.Msg.LANG_T
EXT_COMPARE_EQUAL,
  158:     dropDown: {
  159:       titleName: 'OP',
  160:       value: 'EQUAL'
  161:     }
  162:   }, {
  163:     translatedName: Blockly.Msg.LANG_TEXT_COMPARE_INPUT_COMPARE + Blockly.Msg.LANG_T
EXT_COMPARE_GT,
  164:     dropDown: {
  165:       titleName: 'OP',
  166:       value: 'GT'
  167:     }
  168:   }]
  169: };
  170:
  171: Blockly.Blocks.text_compare.OPERATORS = function () {
  172:   return [
  173:     [Blockly.Msg.LANG_TEXT_COMPARE_LT, 'LT'], [Blockly.Msg.LANG_TEXT_COMPARE_EQUAL,
'EQUAL'], [Blockly.Msg.LANG_TEXT_COMPARE_GT, 'GT']
  174:   ]
  175: };
  176:
  177: Blockly.Blocks.text_compare.TOOLTIPS = function () {
  178:   return {
  179:     LT: Blockly.Msg.LANG_TEXT_COMPARE_TOOLTIP_LT,
  180:     EQUAL: Blockly.Msg.LANG_TEXT_COMPARE_TOOLTIP_EQUAL,
  181:     GT: Blockly.Msg.LANG_TEXT_COMPARE_TOOLTIP_GT
  182:   }
  183: };
  184:
  185: Blockly.Blocks['text_trim'] = {
  186:   // trim string
  187:   category: 'Text',
  188:   helpUrl: Blockly.Msg.LANG_TEXT_TRIM_HELPURL,
  189:   init: function () {
  190:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  191:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
  192:     this.appendValueInput('TEXT')
  193:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  194:         .appendField(Blockly.Msg.LANG_TEXT_TRIM_TITLE_TRIM);
  195:     this.setTooltip(Blockly.Msg.LANG_TEXT_TRIM_TOOLTIP);
  196:   },
  197:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_TRIM_TITLE_TRIM}]
  198: };
  199:
  200: Blockly.Blocks['text_changeCase'] = {
  201:   // Change capitalization.
  202:   category: 'Text',
  203:   helpUrl: function () {
  204:     var mode = this.getFieldValue('OP');
  205:     return Blockly.Blocks.text_changeCase.HELPURLS()[mode];
  206:   },
  207:   init: function () {
  208:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  209:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
  210:     this.appendValueInput('TEXT')
  211:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
  212:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
  213:     var thisBlock = this;
  214:     this.setTooltip(function () {
  215:       var mode = thisBlock.getFieldValue('OP');
  216:       return Blockly.Blocks.text_changeCase.TOOLTIPS()[mode];
  217:     });
  218:   },
  219:   typeblock: [{
  220:     translatedName: Blockly.Msg.LANG_TEXT_CHANGECASE_OPERATOR_UPPERCASE,
  221:     dropDown: {
  222:       titleName: 'OP',
  223:       value: 'UPCASE'
  224:     }
  225:   }, {
  226:     translatedName: Blockly.Msg.LANG_TEXT_CHANGECASE_OPERATOR_DOWNCASE,
  227:     dropDown: {
  228:       titleName: 'OP',
```

```
229:       value: 'DOWNCASE'
230:     }
231:   }]
232: };
233:
234: Blockly.Blocks.text_changeCase.OPERATORS = function () {
235:   return [
236:     [Blockly.Msg.LANG_TEXT_CHANGECASE_OPERATOR_UPPERCASE, 'UPCASE'], [Blockly.Msg.LA
NG_TEXT_CHANGECASE_OPERATOR_DOWNCASE, 'DOWNCASE']
237:   ]
238: };
239:
240: Blockly.Blocks.text_changeCase.TOOLTIPS = function () {
241:   return {
242:     UPCASE: Blockly.Msg.LANG_TEXT_CHANGECASE_TOOLTIP_UPPERCASE,
243:     DOWNCASE: Blockly.Msg.LANG_TEXT_CHANGECASE_TOOLTIP_DOWNCASE
244:   }
245: };
246:
247: Blockly.Blocks.text_changeCase.HELPURLS = function () {
248:   return {
249:     UPCASE: Blockly.Msg.LANG_TEXT_CHANGECASE_HELPURL_UPPERCASE,
250:     DOWNCASE: Blockly.Msg.LANG_TEXT_CHANGECASE_HELPURL_DOWNCASE
251:   }
252: };
253:
254: Blockly.Blocks['text_starts_at'] = {
255:   // return index of first occurrence.
256:   category: 'Text',
257:   helpUrl: Blockly.Msg.LANG_TEXT_STARTS_AT_HELPURL,
258:   init: function () {
259:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
260:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
261:     var checkTypeText = Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Block
ly.Blocks.Utilities.INPUT);
262:     this.interpolateMsg(Blockly.Msg.LANG_TEXT_STARTS_AT_INPUT,
263:         ['TEXT', checkTypeText, Blockly.ALIGN_RIGHT],
264:         ['PIECE', checkTypeText, Blockly.ALIGN_RIGHT],
265:         Blockly.ALIGN_RIGHT);
266:     this.setTooltip(Blockly.Msg.LANG_TEXT_STARTS_AT_TOOLTIP);
267:     this.setInputsInline(false);
268:   },
269:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_STARTS_AT_INPUT_STARTS_AT}]
270: };
271:
272: Blockly.Blocks['text_contains'] = {
273:   // Is text contained in
274:   category: 'Text',
275:   helpUrl: Blockly.Msg.LANG_TEXT_CONTAINS_HELPURL,
276:   init: function () {
277:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
278:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
279:     var checkTypeText = Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Block
ly.Blocks.Utilities.INPUT);
280:     this.interpolateMsg(Blockly.Msg.LANG_TEXT_CONTAINS_INPUT,
281:         ['TEXT', checkTypeText, Blockly.ALIGN_RIGHT],
282:         ['PIECE', checkTypeText, Blockly.ALIGN_RIGHT],
283:         Blockly.ALIGN_RIGHT);
284:     this.setTooltip(Blockly.Msg.LANG_TEXT_CONTAINS_TOOLTIP);
285:     this.setInputsInline(false);
286:   },
287:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_CONTAINS_INPUT_CONTAINS}]
288: };
289:
290: Blockly.Blocks['text_split'] = {
291:   // This includes all four split variants (modes). The name and type of the 'AT' ar
g
292:   // changes to match the selected mode.
293:   category: 'Text',
294:   helpUrl: function () {
295:     var mode = this.getFieldValue('OP');
296:     return Blockly.Blocks.text_split.HELPURLS()[mode];
297:   },
298:   init: function () {
299:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
300:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list", Bloc
kly.Blocks.Utilities.OUTPUT));
301:     this.appendValueInput('TEXT')
302:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
303:         .appendField(new Blockly.FieldDropdown(this.OPERATORS, Blockly.Blocks.text_s
plit.dropdown_onchange), 'OP')
304:         .appendField(Blockly.Msg.LANG_TEXT_SPLIT_INPUT_TEXT);
305:     this.appendValueInput('AT')
306:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
307:         .appendField(Blockly.Msg.LANG_TEXT_SPLIT_INPUT_AT, 'ARG2_NAME')
308:         .setAlign(Blockly.ALIGN_RIGHT);
309:   },
310:   // adjust for the mode when the block is read in
311:   domToMutation: function (xmlElement) {
312:     var mode = xmlElement.getAttribute('mode');
313:     Blockly.Blocks.text_split.adjustToMode(mode, this);
314:   },
315:   // put the mode in the DOM so it can be read in by domToMutation
316:   // WARNING:  Note that the 'mode' tag below is lowercase.  It would not work
317:   // to make it uppercase ('MODE').  There's a bug somewhere (in the browser?) that
318:   // writes tags as lowercase.  So writing the date with tag 'MODE' and attempting
319:   // to read with tag 'MODE' wil not work.
320:   mutationToDom: function () {
321:     var container = document.createElement('mutation');
322:     var savedMode = this.getFieldValue('OP');
323:     container.setAttribute('mode', savedMode);
324:     return container;
325:   },
326:   typeblock: [{
327:     translatedName: Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT,
328:     dropDown: {
329:       titleName: 'OP',
330:       value: 'SPLIT'
331:     }
332:   }, {
333:     translatedName: Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_FIRST,
334:     dropDown: {
335:       titleName: 'OP',
336:       value: 'SPLITATFIRST'
337:     }
338:   }, {
339:     translatedName: Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_ANY,
340:     dropDown: {
341:       titleName: 'OP',
342:       value: 'SPLITATANY'
```

```
343:     }
344:   }, {
345:     translatedName: Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_FIRST_OF_ANY,
346:     dropDown: {
347:       titleName: 'OP',
348:       value: 'SPLITATFIRSTOFANY'
349:     }
350:   }]
351: };
352:
353: // Change the name and type of ARG2 and set tooltop depending on mode
354: Blockly.Blocks.text_split.adjustToMode = function (mode, block) {
355:   if (mode == 'SPLITATFIRST' || mode == 'SPLIT') {
356:     block.getInput("AT").setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("te
xt", Blockly.Blocks.Utilities.INPUT));
357:     block.setFieldValue(Blockly.Msg.LANG_TEXT_SPLIT_INPUT_AT, 'ARG2_NAME');
358:   } else if (mode == 'SPLITATFIRSTOFANY' || mode == 'SPLITATANY') {
359:     block.getInput("AT").setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("li
st", Blockly.Blocks.Utilities.INPUT));
360:     block.setFieldValue(Blockly.Msg.LANG_TEXT_SPLIT_INPUT_AT_LIST, 'ARG2_NAME');
361:   }
362:   ;
363:   block.setTooltip(Blockly.Blocks.text_split.TOOLTIPS()[mode]);
364: };
365:
366: Blockly.Blocks.text_split.dropdown_onchange = function (mode) {
367:   Blockly.Blocks.text_split.adjustToMode(mode, this.sourceBlock_)
368: };
369:
370: // The order here determines the order in the dropdown
371: Blockly.Blocks.text_split.OPERATORS = function () {
372:   return [
373:     [Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT, 'SPLIT'],
374:     [Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_FIRST, 'SPLITATFIRST'],
375:     [Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_ANY, 'SPLITATANY'],
376:     [Blockly.Msg.LANG_TEXT_SPLIT_OPERATOR_SPLIT_AT_FIRST_OF_ANY, 'SPLITATFIRSTOFANY'
]
377:   ]
378: };
379:
380: Blockly.Blocks.text_split.TOOLTIPS = function () {
381:   return {
382:     SPLITATFIRST: Blockly.Msg.LANG_TEXT_SPLIT_TOOLTIP_SPLIT_AT_FIRST,
383:     SPLITATFIRSTOFANY: Blockly.Msg.LANG_TEXT_SPLIT_TOOLTIP_SPLIT_AT_FIRST_OF_ANY,
384:     SPLIT: Blockly.Msg.LANG_TEXT_SPLIT_TOOLTIP_SPLIT,
385:     SPLITATANY: Blockly.Msg.LANG_TEXT_SPLIT_TOOLTIP_SPLIT_AT_ANY
386:   }
387: };
388:
389: Blockly.Blocks.text_split.HELPURLS = function () {
390:   return {
391:     SPLITATFIRST: Blockly.Msg.LANG_TEXT_SPLIT_HELPURL_SPLIT_AT_FIRST,
392:     SPLITATFIRSTOFANY: Blockly.Msg.LANG_TEXT_SPLIT_HELPURL_SPLIT_AT_FIRST_OF_ANY,
393:     SPLIT: Blockly.Msg.LANG_TEXT_SPLIT_HELPURL_SPLIT,
394:     SPLITATANY: Blockly.Msg.LANG_TEXT_SPLIT_HELPURL_SPLIT_AT_ANY
395:   }
396: };
397:
398: Blockly.Blocks['text_split_at_spaces'] = {
399:   // Split at spaces
400:   category: 'Text',
401:   helpUrl: Blockly.Msg.LANG_TEXT_SPLIT_AT_SPACES_HELPURL,
402:   init: function () {
403:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
404:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list", Bloc
kly.Blocks.Utilities.OUTPUT));
405:     this.appendValueInput('TEXT')
406:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Blockly.Blo
cks.Utilities.INPUT))
407:         .appendField(Blockly.Msg.LANG_TEXT_SPLIT_AT_SPACES_TITLE);
408:     this.setTooltip(Blockly.Msg.LANG_TEXT_SPLIT_AT_TOOLTIP);
409:   },
410:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_SPLIT_AT_SPACES_TITLE}]
411: };
412:
413: Blockly.Blocks['text_segment'] = {
414:   // Create text segment
415:   category: 'Text',
416:   helpUrl: Blockly.Msg.LANG_TEXT_SEGMENT_HELPURL,
417:   init: function () {
418:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
419:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
420:     var checkTypeText = Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Block
ly.Blocks.Utilities.INPUT);
421:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", B
lockly.Blocks.Utilities.INPUT);
422:     this.interpolateMsg(Blockly.Msg.LANG_TEXT_SEGMENT_INPUT,
423:         ['TEXT', checkTypeText, Blockly.ALIGN_RIGHT],
424:         ['START', checkTypeNumber, Blockly.ALIGN_RIGHT],
425:         ['LENGTH', checkTypeNumber, Blockly.ALIGN_RIGHT],
426:         Blockly.ALIGN_RIGHT);
427:     this.setTooltip(Blockly.Msg.LANG_TEXT_SEGMENT_AT_TOOLTIP);
428:     this.setInputsInline(false);
429:   },
430:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_SEGMENT_TITLE_SEGMENT}]
431: };
432:
433: Blockly.Blocks['text_replace_all'] = {
434:   // Replace all occurrences of text
435:   category: 'Text',
436:   helpUrl: Blockly.Msg.LANG_TEXT_REPLACE_ALL_HELPURL,
437:   init: function () {
438:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
439:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Bloc
kly.Blocks.Utilities.OUTPUT));
440:     var checkTypeText = Blockly.Blocks.Utilities.YailTypeToBlocklyType("text", Block
ly.Blocks.Utilities.INPUT);
441:     this.interpolateMsg(Blockly.Msg.LANG_TEXT_REPLACE_ALL_INPUT,
442:         ['TEXT', checkTypeText, Blockly.ALIGN_RIGHT],
443:         ['SEGMENT', checkTypeText, Blockly.ALIGN_RIGHT],
444:         ['REPLACEMENT', checkTypeText, Blockly.ALIGN_RIGHT],
445:         Blockly.ALIGN_RIGHT);
446:     this.setTooltip(Blockly.Msg.LANG_TEXT_REPLACE_ALL_TOOLTIP);
447:     this.setInputsInline(false);
448:   },
449:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_REPLACE_ALL_TITLE_REPLACE_ALL}]
450: };
451:
452: Blockly.Blocks['obsufcated_text'] = {
453:   // Text value.
454:   category: 'Text',
455:   helpUrl: Blockly.Msg.LANG_TEXT_TEXT_OBSFUCATE_HELPURL,
456:   init: function () {
```

```
  457:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
  458:     this.appendDummyInput().appendField(Blockly.Msg.LANG_TEXT_TEXT_OBSFUCATE
  459:       + " " + Blockly.Msg.LANG_TEXT_TEXT_LEFT_QUOTE).appendField(
  460:         new Blockly.FieldTextBlockInput(''),
  461:         'TEXT').appendField(Blockly.Msg.LANG_TEXT_TEXT_RIGHT_QUOTE);
  462:     this.setOutput(true, [Blockly.Blocks.text.connectionCheck]);
  463:     this.setTooltip(Blockly.Msg.LANG_TEXT_TEXT_OBSFUCATE_TOOLTIP);
  464:     this.confounder = Math.random().toString(36).replace(/[^a-z]+/g, '').substr(0, 8
);
  465:   },
  466:   domToMutation: function(xmlElement) {
  467:     var confounder = xmlElement.getAttribute('confounder');
  468:     this.confounder = confounder;
  469:   },
  470:   mutationToDom: function() {
  471:     var container = document.createElement('mutation')
  472:     container.setAttribute('confounder', this.confounder);
  473:     return container;
  474:   },
  475:   typeblock: [{translatedName: Blockly.Msg.LANG_TEXT_TEXT_OBSFUCATE}]
  476: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Lists blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Blocks.lists');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['lists_create_with'] = {
  // Create a list with any number of elements of any type.
  category: 'Lists',
  helpUrl: Blockly.Msg.LANG_LISTS_CREATE_WITH_EMPTY_HELPURL,
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendValueInput('ADD0')
        .appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST);
    this.appendValueInput('ADD1');
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
    this.setMutator(new Blockly.Mutator(['lists_create_with_item']));
    this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_TOOLTIP);
    this.itemCount_ = 2;
    this.emptyInputName = 'EMPTY';
    this.repeatingInputName = 'ADD';
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function(workspace){
    return Blockly.decompose(workspace,'lists_create_with_item',this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function(){
    this.appendDummyInput(this.emptyInputName)
      .appendField(Blockly.Msg.LANG_LISTS_CREATE_EMPTY_TITLE);
  },
  addInput: function(inputNum){
    var input = this.appendValueInput(this.repeatingInputName + inputNum);
    if(inputNum === 0){
      input.appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST);
    }
    return input;
  },
  updateContainerBlock: function(containerBlock) {
    containerBlock.setFieldValue(Blockly.Msg.LANG_LISTS_CREATE_WITH_CONTAINER_TITLE_
ADD,"CONTAINER_TEXT");
  },
  // create type blocks for both make a list (two items) and create empty list
  typeblock: [
      { translatedName: Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST,
        mutatorAttributes: { items: 2 } },
      { translatedName: Blockly.Msg.LANG_LISTS_CREATE_EMPTY_TITLE,
        mutatorAttributes: { items: 0 } }]
};

Blockly.Blocks['lists_create_with_item'] = {
  // Add items.
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TITLE);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TOOLTIP);
    this.contextMenu = false;
  }
};


Blockly.Blocks['lists_add_items'] = {
  // Create a list with any number of elements of any type.
  category: 'Lists',
  helpUrl: Blockly.Msg.LANG_LISTS_ADD_ITEMS_HELPURL,
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendValueInput('LIST')
      .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_TITLE_ADD)
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_INPUT_LIST);
    this.appendValueInput('ITEM0')
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_INPUT_ITEM)
      .setAlign(Blockly.ALIGN_RIGHT);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEMS_TOOLTIP);
    this.setMutator(new Blockly.Mutator(['lists_add_items_item']));
    this.itemCount_ = 1;
    this.emptyInputName = null;
    this.repeatingInputName = 'ITEM';
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function(workspace){
    return Blockly.decompose(workspace,'lists_add_items_item',this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function(){},
  addInput: function(inputNum){
    var input = this.appendValueInput(this.repeatingInputName + inputNum);
    input.appendField('item').setAlign(Blockly.ALIGN_RIGHT);
    return input;
  },
  updateContainerBlock: function(containerBlock) {
    containerBlock.setFieldValue(Blockly.Msg.LANG_LISTS_ADD_ITEMS_CONTAINER_TITLE_AD
D,"CONTAINER_TEXT");
    containerBlock.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEMS_CONTAINER_TOOLTIP);
  },
  typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_ADD_ITEMS_TITLE_ADD }]
};

Blockly.Blocks['lists_add_items_item'] = {
  // Add items.
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
```

```
121:     this.appendDummyInput()
122:         .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEM_TITLE);
123:     this.setPreviousStatement(true);
124:     this.setNextStatement(true);
125:     this.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEM_TOOLTIP);
126:     this.contextMenu = false;
127:   }
128: };
129:
130: Blockly.Blocks['lists_is_in'] = {
131:   // Is in list?.
132:   category : 'Lists',
133:   helpUrl : Blockly.Msg.LANG_LISTS_IS_IN_HELPURL,
134:   init : function() {
135:     this.setColour(Blockly.LIST_CATEGORY_HUE);
136:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
137:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
138:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_IS_IN_INPUT,
139:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
140:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
141:             Blockly.ALIGN_RIGHT);
142:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
ockly.Blocks.Utilities.OUTPUT));
143:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_IN_TOOLTIP);
144:     this.setInputsInline(false);
145:   },
146:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_IS_IN_TITLE_IS_IN }]
147: };
148:
149:
150: Blockly.Blocks['lists_length'] = {
151:   // Length of list.
152:   category : 'Lists',
153:   helpUrl : Blockly.Msg.LANG_LISTS_LENGTH_HELPURL,
154:   init : function() {
155:     this.setColour(Blockly.LIST_CATEGORY_HUE);
156:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blo
ckly.Blocks.Utilities.OUTPUT));
157:     this.appendValueInput('LIST')
158:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
159:       .appendField(Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LENGTH)
160:       .appendField(Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LIST);
161:     this.setTooltip(Blockly.Msg.LANG_LISTS_LENGTH_TOOLTIP);
162:   },
163:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LENGTH }]
164: };
165:
166: Blockly.Blocks['lists_is_empty'] = {
167:   // Is the list empty?.
168:   category : 'Lists',
169:   helpUrl : Blockly.Msg.LANG_LISTS_IS_EMPTY_HELPURL,
170:   init : function() {
171:     this.setColour(Blockly.LIST_CATEGORY_HUE);
172:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
ockly.Blocks.Utilities.OUTPUT));
173:     this.appendValueInput('LIST')
174:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
175:       .appendField(Blockly.Msg.LANG_LISTS_TITLE_IS_EMPTY)
176:       .appendField(Blockly.Msg.LANG_LISTS_INPUT_LIST);
177:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_EMPTY_TOOLTIP);
178:   },
179:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_TITLE_IS_EMPTY }]
180: };
181:
182: Blockly.Blocks['lists_pick_random_item'] = {
183:   // Length of list.
184:   category : 'Lists',
185:   helpUrl : Blockly.Msg.LANG_LISTS_PICK_RANDOM_ITEM_HELPURL,
186:   init : function() {
187:     this.setColour(Blockly.LIST_CATEGORY_HUE);
188:     this.setOutput(true, null);
189:     this.appendValueInput('LIST')
190:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
191:       .appendField(Blockly.Msg.LANG_LISTS_PICK_RANDOM_TITLE_PICK_RANDOM)
192:       .appendField(Blockly.Msg.LANG_LISTS_PICK_RANDOM_ITEM_INPUT_LIST);
193:     this.setTooltip(Blockly.Msg.LANG_LISTS_PICK_RANDOM_TOOLTIP);
194:   },
195:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_PICK_RANDOM_TITLE_PICK_RANDOM
 }]
196: };
197:
198: Blockly.Blocks['lists_position_in'] = {
199:   // Postion of item in list.
200:   category : 'Lists',
201:   helpUrl : Blockly.Msg.LANG_LISTS_POSITION_IN_HELPURL,
202:   init : function() {
203:     this.setColour(Blockly.LIST_CATEGORY_HUE);
204:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blo
ckly.Blocks.Utilities.OUTPUT));
205:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
206:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
207:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_POSITION_IN_INPUT,
208:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
209:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
210:             Blockly.ALIGN_RIGHT);
211:     this.setTooltip(Blockly.Msg.LANG_LISTS_POSITION_IN_TOOLTIP);
212:     this.setInputsInline(false);
213:   },
214:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_POSITION_IN_TITLE_POSITION }]
215: };
216:
217:
218: Blockly.Blocks['lists_select_item'] = {
219:   // Select from list an item.
220:   category : 'Lists',
221:   helpUrl : Blockly.Msg.LANG_LISTS_SELECT_ITEM_TITLE_HELPURL,
222:   init : function() {
223:     this.setColour(Blockly.LIST_CATEGORY_HUE);
224:     this.setOutput(true, null);
225:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
226:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
227:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_SELECT_ITEM_INPUT,
228:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
229:             ['NUM', checkTypeNumber, Blockly.ALIGN_RIGHT],
230:             Blockly.ALIGN_RIGHT);
```

```
231:     this.setTooltip(Blockly.Msg.LANG_LISTS_SELECT_ITEM_TOOLTIP);
232:     this.setInputsInline(false);
233:   },
234:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_SELECT_ITEM_TITLE_SELECT }]
235: };
236:
237: Blockly.Blocks['lists_insert_item'] = {
238:   // Insert Item in list.
239:   category : 'Lists',
240:   helpUrl : Blockly.Msg.LANG_LISTS_INSERT_ITEM_HELPURL,
241:   init : function() {
242:     this.setColour(Blockly.LIST_CATEGORY_HUE);
243:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
244:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
245:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
246:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_INSERT_INPUT,
247:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
248:             ['INDEX', checkTypeNumber, Blockly.ALIGN_RIGHT],
249:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
250:             Blockly.ALIGN_RIGHT);
251:     this.setPreviousStatement(true);
252:     this.setNextStatement(true);
253:     this.setTooltip(Blockly.Msg.LANG_LISTS_INSERT_TOOLTIP);
254:     this.setInputsInline(false);
255:   },
256:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_INSERT_TITLE_INSERT_LIST }]
257: };
258:
259: Blockly.Blocks['lists_replace_item'] = {
260:   // Replace Item in list.
261:   category : 'Lists',
262:   helpUrl : Blockly.Msg.LANG_LISTS_REPLACE_ITEM_HELPURL,
263:   init : function() {
264:     this.setColour(Blockly.LIST_CATEGORY_HUE);
265:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
266:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
267:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
268:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_REPLACE_ITEM_INPUT,
269:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
270:             ['NUM', checkTypeNumber, Blockly.ALIGN_RIGHT],
271:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
272:             Blockly.ALIGN_RIGHT);
273:     this.setPreviousStatement(true);
274:     this.setNextStatement(true);
275:     this.setTooltip(Blockly.Msg.LANG_LISTS_REPLACE_ITEM_TOOLTIP);
276:     this.setInputsInline(false);
277:   },
278:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_REPLACE_ITEM_TITLE_REPLACE }]
279: };
280:
281: Blockly.Blocks['lists_remove_item'] = {
282:   // Remove Item in list.
283:   category : 'Lists',
284:   helpUrl : Blockly.Msg.LANG_LISTS_REMOVE_ITEM_HELPURL,
285:   init : function() {
286:     this.setColour(Blockly.LIST_CATEGORY_HUE);
287:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
288:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
289:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_REMOVE_ITEM_INPUT,
290:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
291:             ['INDEX', checkTypeNumber, Blockly.ALIGN_RIGHT],
292:             Blockly.ALIGN_RIGHT);
293:     this.setPreviousStatement(true);
294:     this.setNextStatement(true);
295:     this.setTooltip(Blockly.Msg.LANG_LISTS_REMOVE_ITEM_TOOLTIP);
296:     this.setInputsInline(false);
297:   },
298:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_REMOVE_ITEM_TITLE_REMOVE }]
299: };
300:
301: Blockly.Blocks['lists_append_list'] = {
302:   // Append to list.
303:   category : 'Lists',
304:   helpUrl : Blockly.Msg.LANG_LISTS_APPEND_LIST_HELPURL,
305:   init : function() {
306:     this.setColour(Blockly.LIST_CATEGORY_HUE);
307:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
308:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_APPEND_LIST_INPUT,
309:             ['LIST0', checkTypeList, Blockly.ALIGN_RIGHT],
310:             ['LIST1', checkTypeList, Blockly.ALIGN_RIGHT],
311:             Blockly.ALIGN_RIGHT);
312:     this.setPreviousStatement(true);
313:     this.setNextStatement(true);
314:     this.setTooltip(Blockly.Msg.LANG_LISTS_APPEND_LIST_TOOLTIP);
315:     this.setInputsInline(false);
316:   },
317:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_APPEND_LIST_TITLE_APPEND }]
318: };
319:
320:
321: Blockly.Blocks['lists_copy'] = {
322:   // Make a copy of list.
323:   category : 'Lists',
324:   helpUrl : Blockly.Msg.LANG_LISTS_COPY_HELPURL,
325:   init : function() {
326:     this.setColour(Blockly.LIST_CATEGORY_HUE);
327:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
328:     this.appendValueInput('LIST')
329:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
330:       .appendField(Blockly.Msg.LANG_LISTS_COPY_TITLE_COPY)
331:       .appendField(Blockly.Msg.LANG_LISTS_COPY_INPUT_LIST);
332:     this.setTooltip(Blockly.Msg.LANG_LISTS_COPY_TOOLTIP);
333:   },
334:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_COPY_TITLE_COPY }]
335: };
336:
337: Blockly.Blocks['lists_is_list'] = {
338:   // Is a list?
339:   category : 'Lists',
340:   helpUrl : Blockly.Msg.LANG_LISTS_IS_LIST_HELPURL,
341:   init : function() {
342:     this.setColour(Blockly.LIST_CATEGORY_HUE);
343:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
```

```
ockly.Blocks.Utilities.OUTPUT));
  344:     this.appendValueInput('ITEM')
  345:       .appendField(Blockly.Msg.LANG_LISTS_IS_LIST_TITLE_IS_LIST)
  346:       .appendField(Blockly.Msg.LANG_LISTS_IS_LIST_INPUT_THING);
  347:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_LIST_TOOLTIP);
  348:   },
  349:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_IS_LIST_TITLE_IS_LIST }]
  350: };
  351:
  352: Blockly.Blocks['lists_to_csv_row'] = {
  353:   // Make a csv row from list.
  354:   category : 'Lists',
  355:   helpUrl : Blockly.Msg.LANG_LISTS_TO_CSV_ROW_HELPURL,
  356:   init : function() {
  357:     this.setColour(Blockly.LIST_CATEGORY_HUE);
  358:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
  359:     this.appendValueInput('LIST')
  360:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
  361:       .appendField(Blockly.Msg.LANG_LISTS_TO_CSV_ROW_TITLE_TO_CSV)
  362:       .appendField(Blockly.Msg.LANG_LISTS_TO_CSV_ROW_INPUT_LIST);
  363:     this.setTooltip(Blockly.Msg.LANG_LISTS_TO_CSV_ROW_TOOLTIP);
  364:   },
  365:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_TO_CSV_ROW_TITLE_TO_CSV }]
  366: };
  367:
  368: Blockly.Blocks['lists_to_csv_table'] = {
  369:   // Make a csv table from list.
  370:   category : 'Lists',
  371:   helpUrl : Blockly.Msg.LANG_LISTS_TO_CSV_TABLE_HELPURL,
  372:   init : function() {
  373:     this.setColour(Blockly.LIST_CATEGORY_HUE);
  374:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
  375:     this.appendValueInput('LIST')
  376:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
  377:       .appendField(Blockly.Msg.LANG_LISTS_TO_CSV_TABLE_TITLE_TO_CSV)
  378:       .appendField(Blockly.Msg.LANG_LISTS_TO_CSV_TABLE_INPUT_LIST);
  379:     this.setTooltip(Blockly.Msg.LANG_LISTS_TO_CSV_TABLE_TOOLTIP);
  380:   },
  381:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_TO_CSV_TABLE_TITLE_TO_CSV }]
  382: };
  383:
  384: Blockly.Blocks['lists_from_csv_row'] = {
  385:   // Make list from csv row.
  386:   category : 'Lists',
  387:   helpUrl : Blockly.Msg.LANG_LISTS_FROM_CSV_ROW_HELPURL,
  388:   init : function() {
  389:     this.setColour(Blockly.LIST_CATEGORY_HUE);
  390:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
  391:     this.appendValueInput('TEXT')
  392:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text",Blockly.Blocks
.Utilities.INPUT))
  393:       .appendField(Blockly.Msg.LANG_LISTS_FROM_CSV_ROW_TITLE_FROM_CSV)
  394:       .appendField(Blockly.Msg.LANG_LISTS_FROM_CSV_ROW_INPUT_TEXT);
  395:     this.setTooltip(Blockly.Msg.LANG_LISTS_FROM_CSV_ROW_TOOLTIP);
  396:   },
  397:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_FROM_CSV_ROW_TITLE_FROM_CSV }
]
  398: };
  399:
  400: Blockly.Blocks['lists_from_csv_table'] = {
  401:   // Make list from csv table.
  402:   category : 'Lists',
  403:   helpUrl : Blockly.Msg.LANG_LISTS_FROM_CSV_TABLE_HELPURL,
  404:   init : function() {
  405:     this.setColour(Blockly.LIST_CATEGORY_HUE);
  406:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
  407:     this.appendValueInput('TEXT')
  408:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("text",Blockly.Blocks
.Utilities.INPUT))
  409:       .appendField(Blockly.Msg.LANG_LISTS_FROM_CSV_TABLE_TITLE_FROM_CSV)
  410:       .appendField(Blockly.Msg.LANG_LISTS_FROM_CSV_TABLE_INPUT_TEXT);
  411:     this.setTooltip(Blockly.Msg.LANG_LISTS_FROM_CSV_TABLE_TOOLTIP);
  412:   },
  413:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_FROM_CSV_TABLE_TITLE_FROM_CSV
 }]
  414: };
  415:
  416: Blockly.Blocks['lists_lookup_in_pairs'] = {
  417:   // Look up in a list of pairs (key, value).
  418:   category : 'Lists',
  419:   helpUrl : Blockly.Msg.LANG_LISTS_LOOKUP_IN_PAIRS_HELPURL,
  420:   init : function() {
  421:     this.setColour(Blockly.LIST_CATEGORY_HUE);
  422:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockl
y.Blocks.Utilities.OUTPUT));
  423:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
  424:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
  425:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
  426:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_LOOKUP_IN_PAIRS_INPUT,
  427:             ['KEY', checkTypeAny, Blockly.ALIGN_RIGHT],
  428:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
  429:             ['NOTFOUND', checkTypeAny, Blockly.ALIGN_RIGHT],
  430:             Blockly.ALIGN_RIGHT);
  431:     this.setTooltip(Blockly.Msg.LANG_LISTS_LOOKUP_IN_PAIRS_TOOLTIP);
  432:     this.setInputsInline(false);
  433:   },
  434:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_LOOKUP_IN_PAIRS_TITLE_LOOKUP_
IN_PAIRS }]
  435: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Variables blocks for Blockly, modified for MIT App Inventor.
 * @author fturbak@wellesley.edu (Lyn Turbak)
 */

/**
 * Lyn's History:
 *  * [lyn, written 11/16-17/13, added 07/01/14]
 *   + Added freeVariables, renameFree, and renameBound to local declarations
 *   + Added freeVariables and renameFree to getters and setters
 *   + Added renameVar and renameVars to local declaration
 *   + renamed addDeclarationInputs_ to updatedDeclarationInputs_
 * [lyn, 03/04/13]
 *   + Remove notion of collapsed input from local variable declaration statements/expressions,
 *     which has been eliminated in updated to Blockly v1636.
 *   + Update appendTitle* to appendField*
 * [lyn, 01/18/13] Remove onchange from lexical_variable_get and lexical_variable_set.
 *    This fixes issue 667 (Variable getter/setter names deleted in copied blocks)
 *    and improves laggy drag problem.
 * [lyn, 10/27/13]
 *   + Modified local declaration parameter flydowns so editing the name changes corresponding name in an open mutator.
 *   + Changed local declaration compose() to rebuild inputs only if local names have changed.
 *     (essential for getting param flydown name changes reflected in open mutator).
 *   + Fixed local declaration expression compose() to be the same as that for local declaration statements.
 *   + Modified addDeclarationInputs_ to remove existing declarations, add new ones, and keep
 *     last two declarations (body and collapsed text) rather than recreating them.
 *     This is now used by both domToMutation() and compose(), eliminating duplicated code.
 *   + Eliminated dummy declarations.
 *   + Specify direction of flydowns
 * [lyn, 10/25/13] Made collapsed block labels more sensible.
 * [lyn, 10/10-14/13]
 *   + Installed variable declaration flydowns in global definition and local variable declaration
 *     statements and expressions.
 *   + Abstracted over string labels on all blocks using constants defined in en/_messages.js
 *   + Cleaned up code, including refactoring to increase sharing between
 *     local_declaration_statement and local_declaration_expression.
 *   + Fixed bug: Modified onchange for local declarations to keep localNames_ instance
 *     variable updated when param is edited directly on declaration block.
 *   + In local variable statements/expression, changed both "in do" and "in return"
 *     to "scope" (shape distinguishes them). But maybe these should just be empty string?
 * [lyn, 11/18/12] Renaming for globals (still working on renaming of procedure and loop params)
 * [lyn, 11/17/12] Integration of simple naming into App Inventor
 * [lyn, 11/11/12] More work on onchange event. Allow invalid names for untethered getters/setters
 *                 on workspace, but not when click in to other blocks.
 * [lyn, 11/08-10/12] Get dropdown list of names in scope to work for globals and params
 *                    (including loops) in raw blockly. Pass along to Andrew for integration
 *                    into AI. Initial work on onchange event to change names when getters/setters
 *                    copied and moved.
 * [lyn, 11/05-07/12] Add local variable declaration expressions. Get mutator working for local
 *                    declaration statements and expressions. But these don't save/load properly from XML
 *                    Helpful 10/7 hangout with Andrew and Paul.
 * [lyn, 11/04/12] Created. Add global declarations. Work on local variable declaration statement.
 */

/*
// For debugging only
function myStringify (obj) {
  var seen = [];
  return JSON.stringify(obj, function(key, val) {
   if (typeof val == "object") {
       if (seen.indexOf(val) >= 0) {
           return undefined;
       }
       seen.push(val);
   }
   return val
   });
}
*/

'use strict';

goog.provide('Blockly.Blocks.lexicalvariables');
goog.require('Blockly.Blocks.Utilities');
goog.require('goog.dom');

/**
 * Prototype bindings for a global variable declaration block
 */
Blockly.Blocks['global_declaration'] = {
  // Global var defn
  category: 'Variables',
  helpUrl: Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_HELPURL,
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendValueInput('VALUE')
        .appendField(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TITLE_INIT)
        .appendField(new Blockly.FieldGlobalFlydown(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_NAME,
                                                    Blockly.FieldFlydown.DISPLAY_BELOW), 'NAME')
        .appendField(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TO);
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TOOLTIP);
  },
  getVars: function() {
    return [this.getFieldValue('NAME')];
  },
  renameVar: function(oldName, newName) {
    if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
      this.setFieldValue(newName, 'NAME');
```

```
  104:     }
  105:   },
  106:   typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TITLE_
INIT }]
  107: };
  108:
  109: /**
  110:  * Prototype bindings for a variable getter block
  111:  */
  112: Blockly.Blocks['lexical_variable_get'] = {
  113:   // Variable getter.
  114:   category: 'Variables',
  115:   helpUrl: Blockly.Msg.LANG_VARIABLES_GET_HELPURL,
  116:   init: function() {
  117:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
  118:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
  119:     this.fieldVar_.setBlock(this);
  120:     this.appendDummyInput()
  121:         .appendField(Blockly.Msg.LANG_VARIABLES_GET_TITLE_GET)
  122:         .appendField(this.fieldVar_, 'VAR');
  123:     this.setOutput(true, null);
  124:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_GET_TOOLTIP);
  125:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
  126:   },
  127:   mutationToDom: function() { // Handle getters for event parameters specially (to s
upport i8n)
  128:     return Blockly.LexicalVariable.eventParamMutationToDom(this);
  129:   },
  130:   domToMutation: function(xmlElement) { // Handler getters for event parameters spec
ially (to support i8n)
  131:     Blockly.LexicalVariable.eventParamDomToMutation(this, xmlElement);
  132:   },
  133:   getVars: function() {
  134:     return [this.getFieldValue('VAR')];
  135:   },
  136:   renameLexicalVar: function(oldName, newName) {
  137:     // console.log("Renaming lexical variable from " + oldName + " to " + newName);
  138:     if (oldName === this.getFieldValue('VAR')) {
  139:         this.setFieldValue(newName, 'VAR');
  140:     }
  141:   },
  142:   renameFree: function (freeSubstitution) {
  143:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  144:     var prefix = prefixPair[0];
  145:     // Only rename lexical (nonglobal) names
  146:     if (prefix !== Blockly.globalNamePrefix) {
  147:       var oldName = prefixPair[1];
  148:       var newName = freeSubstitution.apply(oldName);
  149:       if (newName !== oldName) {
  150:         this.renameLexicalVar(oldName, newName);
  151:       }
  152:     }
  153:   },
  154:   freeVariables: function() { // return the free lexical variables of this block
  155:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  156:     var prefix = prefixPair[0];
  157:     // Only return lexical (nonglobal) names
  158:     if (prefix !== Blockly.globalNamePrefix) {
  159:       var oldName = prefixPair[1];
  160:       return new Blockly.NameSet([oldName])
  161:     } else {
  162:       return new Blockly.NameSet();
  163:     }
  164:   },
  165:   typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_GET_TITLE_GET + Blockly.M
sg.LANG_VARIABLES_VARIABLE }]
  166: };
  167:
  168: /**
  169:  * Prototype bindings for a variable setter block
  170:  */
  171: Blockly.Blocks['lexical_variable_set'] = {
  172:   // Variable setter.
  173:   category: 'Variables',
  174:   helpUrl: Blockly.Msg.LANG_VARIABLES_SET_HELPURL, // *** [lyn, 11/10/12] Fix this
  175:   init: function() {
  176:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
  177:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
  178:     this.fieldVar_.setBlock(this);
  179:     this.appendValueInput('VALUE')
  180:         .appendField(Blockly.Msg.LANG_VARIABLES_SET_TITLE_SET)
  181:         .appendField(this.fieldVar_, 'VAR')
  182:         .appendField(Blockly.Msg.LANG_VARIABLES_SET_TITLE_TO);
  183:     this.setPreviousStatement(true);
  184:     this.setNextStatement(true);
  185:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_SET_TOOLTIP);
  186:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
  187:   },
  188:   mutationToDom: function() { // Handle setters for event parameters specially (to s
upport i8n)
  189:     return Blockly.LexicalVariable.eventParamMutationToDom(this);
  190:   },
  191:   domToMutation: function(xmlElement) { // Handler setters for event parameters spec
ially (to support i8n)
  192:     Blockly.LexicalVariable.eventParamDomToMutation(this, xmlElement);
  193:   },
  194:   getVars: function() {
  195:     return [this.getFieldValue('VAR')];
  196:   },
  197:   renameLexicalVar: Blockly.Blocks.lexical_variable_get.renameLexicalVar,
  198:   renameFree: function (freeSubstitution) {
  199:     // potentially rename the set variable
  200:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  201:     var prefix = prefixPair[0];
  202:     // Only rename lexical (nonglobal) names
  203:     if (prefix !== Blockly.globalNamePrefix) {
  204:       var oldName = prefixPair[1];
  205:       var newName = freeSubstitution.apply(oldName);
  206:       if (newName !== oldName) {
  207:         this.renameLexicalVar(oldName, newName);
  208:       }
  209:     }
  210:     // [lyn, 06/26/2014] Don't forget to rename children!
  211:     this.getChildren().map( function(blk) { Blockly.LexicalVariable.renameFree(blk,
freeSubstitution); })
  212:   },
  213:   freeVariables: function() { // return the free lexical variables of this block
  214:     // [lyn, 06/27/2014] Find free vars of *all* children, including subsequent comm
ands in NEXT slot.
  215:     var childrenFreeVars = this.getChildren().map( function(blk) { return Blockly.Le
xicalVariable.freeVariables(blk); } );
  216:     var result = Blockly.NameSet.unionAll(childrenFreeVars);
```

```
    var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
    var prefix = prefixPair[0];
    // Only return lexical (nonglobal) names
    if (prefix !== Blockly.globalNamePrefix) {
      var oldName = prefixPair[1];
      result.insert(oldName);
    }
    return result;
  },
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_SET_TITLE_SET + Blockly.Msg.LANG_VARIABLES_VARIABLE }]
};

/**
 * Prototype bindings for a statement block that declares local names for use in a statement body.
 * [lyn, 10/13/13] Refactored to share more code with Blockly.Blocks.local_declaration_expression
 */
Blockly.Blocks['local_declaration_statement'] = {
  // Define a procedure with no return value.
  // category: null,  // Procedures are handled specially.
  category: 'Variables',  // *** [lyn, 11/07/12] Abstract over this
  helpUrl: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_HELPURL,
  bodyInputName: 'STACK',
  init: function() {
    this.initLocals();
    this.appendStatementInput('STACK')
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_IN_DO);

    // Add notch and nub for vertical statement composition
    this.setPreviousStatement(true);
    this.setNextStatement(true);

    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TOOLTIP);
  },
  initLocals: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.localNames_ = [Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_DEFAULT_NAME];
    var declInput = this.appendValueInput('DECL0');
    declInput.appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TITLE_INIT)
             .appendField(this.parameterFlydown(0), 'VAR0')
             .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_INPUT_TO)
             .setAlign(Blockly.ALIGN_RIGHT);

    // Add mutator for editing local variable names
    this.setMutator(new Blockly.Mutator(['local_mutatorarg']));
  },
  onchange: function () {
     this.localNames_ = this.declaredNames(); // ensure localNames_ is in sync with paramFlydown fields
   },
  mutationToDom: function() { // Store local names in mutation element of XML for block
    var container = document.createElement('mutation');
    for (var i = 0; i< this.localNames_.length; i++) {
      var parameter = document.createElement('localname');
      parameter.setAttribute('name', this.localNames_[i]);
      container.appendChild(parameter);
    }
    return container;
  },
  domToMutation: function(xmlElement) { // Retrieve local names from mutation element of XML for block
                                        // and replace existing declarations
    var children = goog.dom.getChildren(xmlElement);
    if (children.length > 0) { // Ensure xml element is nonempty
      // Else we'll overwrite initial list with "name" for new block
      this.localNames_ = [];
      for (var i = 0, childNode; childNode = children[i]; i++) {
        if (childNode.nodeName.toLowerCase() == 'localname') {
          this.localNames_.push(childNode.getAttribute('name'));
        }
      }
    }
    this.updateDeclarationInputs_(this.localNames_); // add declarations; inits are undefined
  },
  updateDeclarationInputs_: function(names, inits) {
    // Modify this block to replace existing initializers by new declaration inputs created from names and inits.
    // If inits is undefined, treat all initial expressions as undefined.
    // Keep existing body at end of input list.
    // [lyn, 03/04/13] As of change to, Blockly 1636, there is no longer a collapsed input at end.

    // Remember last (= body) input
    var bodyInput = this.inputList[this.inputList.length - 1]; // Body input for local declaration
    var numDecls = this.inputList.length - 1;

    // [lyn, 07/03/14] stop rendering until block is recreated
    var savedRendered = this.rendered;
    this.rendered = false;

    // Modify this local-in-do block according to arrangement of name blocks in mutator editor.
    // Remove all the local declaration inputs ...
    var thisBlock = this; // Grab correct object for use in thunk below
    Blockly.FieldParameterFlydown.withChangeHanderDisabled(
        // [lyn, 07/02/14] Need to disable change handler, else this will try to rename params removed fields.
        function() {
          for (var i = 0; i < numDecls; i++) {
            thisBlock.removeInput('DECL' + i);
          }
        }
    );

    // Empty the inputList and recreate it, building local initializers from mutator
    this.inputList = [];
    this.localNames_ = names;

    for (var i = 0; i < names.length; i++) {
      var declInput = this.appendValueInput('DECL' + i);
      // [lyn, 11/06/12]
      //   This was for case where tried to put "local" keyword on same line with first local name.
      //   But even though alignment set to Blockly.ALIGN_RIGHT, the input was left justified
      //   and covered the plus sign for popping up the mutator. So I put the "local " keyword
      //   on it's own line even though this wastes vertical space. This should be fixed in the future.
```

```
325:       // if (i == 0) {
326:       //  declInput.appendField("local"); // Only put keyword "local" on top line.
327:       // }
328:       declInput.appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TITLE_INIT)
329:                .appendField(this.parameterFlydown(i), 'VAR' + i)
330:                .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_INPUT_TO)
331:                .setAlign(Blockly.ALIGN_RIGHT);
332:       if (inits && inits[i]) { // If there is an initializer, connect it
333:         declInput.connection.connect(inits[i]);
334:       }
335:     }
336:
337:     // Now put back last (= body) input
338:     this.inputList = this.inputList.concat(bodyInput);
339:
340:     this.rendered = savedRendered;
341:     if (this.rendered) {
342:       this.render();
343:     }
344:   },
345:   // [lyn, 10/27/13] Introduced this to correctly handle renaming of mutatorarg in o
pen mutator
346:   // when procedure parameter flydown name is edited.
347:   parameterFlydown: function (paramIndex) { // Return a new local variable parameter
 flydown
348:     var initialParamName = this.localNames_[paramIndex];
349:     var localDecl = this; // Here, "this" is the local decl block. Name it to use in
 function below
350:     var localWorkspace = this.workspace;
351:     var localParameterChangeHandler = function (newParamName) {
352:       // This handler has the same subtleties as procedureParameterChangeHandler in
language/common/procedures.js,
353:       // but is somewhat simpler since doesn't have associated callers to change. Se
e the notes there.
354:
355:       // See Subtleties #1 and #2 in  procedureParameterChangeHandler in language/co
mmon/procedures.js
356:       var newLocals = localDecl.localNames_;
357:       newLocals[paramIndex] = newParamName;
358:
359:       // If there's an open mutator, change the name in the corresponding slot.
360:       if (localDecl.mutator && localDecl.mutator.rootBlock_) {
361:         // Iterate through mutatorarg param blocks and change name of one at paramIn
dex
362:         var mutatorContainer = localDecl.mutator.rootBlock_;
363:         var mutatorargIndex = 0;
364:         var mutatorarg = mutatorContainer.getInputTargetBlock('STACK');
365:         while (mutatorarg && mutatorargIndex < paramIndex) {
366:           mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.target
Block();
367:           mutatorargIndex++;
368:         }
369:         if (mutatorarg && mutatorargIndex == paramIndex) {
370:           // See Subtlety #3 in  procedureParameterChangeHandler in language/common/
procedures.js
371:           Blockly.Field.prototype.setText.call(mutatorarg.getField_("NAME"), newPara
mName);
372:         }
373:       }
374:     }
375:     return new Blockly.FieldParameterFlydown(initialParamName,
376:         true, // name is editable
377:         Blockly.FieldFlydown.DISPLAY_RIGHT,
378:         localParameterChangeHandler);
379:   },
380:   decompose: function(workspace) {
381:     // Create "mutator" editor populated with name blocks with local variable names
382:     var containerBlock = new Blockly.Block.obtain(workspace, 'local_mutatorcontainer
');
383:     containerBlock.initSvg();
384:     containerBlock.setDefBlock(this);
385:     var connection = containerBlock.getInput('STACK').connection;
386:     for (var i = 0; i < this.localNames_.length; i++) {
387:       var localName = this.getFieldValue('VAR' + i);
388:       var nameBlock = new Blockly.Block.obtain(workspace, 'local_mutatorarg');
389:       nameBlock.initSvg();
390:       nameBlock.setFieldValue(localName, 'NAME');
391:       // Store the old location.
392:       nameBlock.oldLocation = i;
393:       connection.connect(nameBlock.previousConnection);
394:       connection = nameBlock.nextConnection;
395:     }
396:     return containerBlock;
397:   },
398:   compose: function(containerBlock) {
399:     // [lyn, 10/27/13] Modified this so that doesn't rebuild block if names haven't
changed.
400:     // This is *essential* to handle Subtlety #3 in localParameterChangeHandler with
in parameterFlydown.
401:
402:     var newLocalNames = [];
403:     var initializers = [];
404:     var mutatorarg = containerBlock.getInputTargetBlock('STACK');
405:     while (mutatorarg) {
406:       newLocalNames.push(mutatorarg.getFieldValue('NAME'));
407:       initializers.push(mutatorarg.valueConnection_); // pushes undefined if doesn't
 exist
408:       mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.targetBloc
k();
409:     }
410:
411:     // Reconstruct inputs only if local list has changed
412:     if (!Blockly.LexicalVariable.stringListsEqual(this.localNames_, newLocalNames))
{
413:
414:       // Switch off rendering while the block is rebuilt.
415:       // var savedRendered = this.rendered;
416:       // this.rendered = false;
417:
418:       this.updateDeclarationInputs_(newLocalNames, initializers);
419:
420:       // Restore rendering and show the changes.
421:       // this.rendered = savedRendered;
422:       // if (this.rendered) {
423:       //  this.render();
424:       // }
425:     }
426:   },
427:   dispose: function() {
428:     // *** [lyn, 11/07/12] Dunno if anything needs to be done here.
429:     // Call parent's destructor.
430:     Blockly.Block.prototype.dispose.apply(this, arguments);
431:     // [lyn, 11/07/12] In above line, don't know where "arguments" param comes from,
432:     // but if it's remove, there's no clicking sound upon deleting the block!
```

```
  433:   },
  434:   saveConnections: function(containerBlock) {
  435:     // Store child initializer blocks for local name declarations with name blocks i
n mutator editor
  436:     var nameBlock = containerBlock.getInputTargetBlock('STACK');
  437:     var i = 0;
  438:     while (nameBlock) {
  439:       var localDecl = this.getInput('DECL' + i);
  440:       nameBlock.valueConnection_ =
  441:         localDecl && localDecl.connection.targetConnection;
  442:       i++;
  443:       nameBlock = nameBlock.nextConnection &&
  444:       nameBlock.nextConnection.targetBlock();
  445:     }
  446:     // Store body statement or expression connection
  447:     var bodyInput = this.getInput(this.bodyInputName); // 'STACK' or 'RETURN'
  448:     if (bodyInput) {
  449:       containerBlock.bodyConnection_ = bodyInput.connection.targetConnection;
  450:     }
  451:   },
  452:   getVars: function() {
  453:     var varList = [];
  454:     for (var i = 0, input; input = this.getFieldValue('VAR' + i); i++) {
  455:       varList.push(input);
  456:     }
  457:     return varList;
  458:   },
  459:   declaredNames: function () { // Interface with Blockly.LexicalVariable.renameParam
  460:     return this.getVars();
  461:   },
  462:   initializerConnections: function() { // [lyn, 11/16/13 ] Return all the initialize
r connections
  463:     var connections = [];
  464:     for (var i = 0, input; input = this.getInput('DECL' + i); i++) {
  465:       connections.push(input.connection && input.connection.targetConnection);
  466:     }
  467:     return connections;
  468:   },
  469:   blocksInScope: function () { // Interface with Blockly.LexicalVariable.renameParam
  470:     var doBody = this.getInputTargetBlock(this.bodyInputName); // *** [lyn, 11/24/12
] This will go away with DO-AND-RETURN block
  471:     var doBodyList = (doBody && [doBody]) || []; // List of non-null doBody or empty
 list for null doBody
  472:     return doBodyList; // List of non-null body elements.
  473:   },
  474:   renameVar: function(oldName, newName) {
  475:     this.renameVars(Blockly.Substitution.simpleSubstitution(oldName,newName));
  476:   },
  477:   renameVars: function(substitution) { // substitution is a dict (i.e., object) mapp
ing old names to new ones
  478:     var localNames = this.declaredNames();
  479:     var renamedLocalNames = substitution.map(localNames);
  480:     if (!Blockly.LexicalVariable.stringListsEqual(renamedLocalNames, localNames)) {
  481:       var initializerConnections = this.initializerConnections();
  482:       this.updateDeclarationInputs_(renamedLocalNames, initializerConnections);
  483:       // Update the mutator's variables if the mutator is open.
  484:       if (this.mutator.isVisible()) {
  485:         var blocks = this.mutator.workspace_.getAllBlocks();
  486:         for (var x = 0, block; block = blocks[x]; x++) {
  487:           if (block.type == 'procedures_mutatorarg') {
  488:             var oldName = block.getFieldValue('NAME');
  489:             var newName = substitution.appy(oldName);
  490:             if (newName !== oldName) {
  491:               block.setFieldValue(newName, 'NAME');
  492:             }
  493:           }
  494:         }
  495:       }
  496:     }
  497:   },
  498:   renameBound: function (boundSubstitution, freeSubstitution) {
  499:     var localNames = this.declaredNames();
  500:     for (var i = 0; i < localNames.length; i++) {
  501:       // This is LET semantics, not LET* semantics, and needs to change!
  502:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DECL'+i), freeSub
stitution);
  503:     }
  504:     var paramSubstitution = boundSubstitution.restrictDomain(localNames);
  505:     this.renameVars(paramSubstitution);
  506:     var newFreeSubstitution = freeSubstitution.remove(localNames).extend(paramSubsti
tution);
  507:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock(this.bodyInputName),
 newFreeSubstitution);
  508:     if (this.nextConnection) {
  509:       var nextBlock = this.nextConnection.targetBlock();
  510:       Blockly.LexicalVariable.renameFree(nextBlock, freeSubstitution);
  511:     }
  512:   },
  513:   renameFree: function (freeSubstitution) {
  514:     // This is LET semantics, not LET* semantics, and needs to change!
  515:     var localNames = this.declaredNames();
  516:     var localNameSet = new Blockly.NameSet(localNames);
  517:     var bodyFreeVars = Blockly.LexicalVariable.freeVariables(this.getInputTargetBloc
k(this.bodyInputName));
  518:     bodyFreeVars.subtract(localNameSet);
  519:     var renamedFreeVars = bodyFreeVars.renamed(freeSubstitution);
  520:     var capturedVars = renamedFreeVars.intersection(localNameSet);
  521:     if (! capturedVars.isEmpty()) { // Case where some names are captured!
  522:       // Must consistently rename declarations and uses of capturedFreeVars with
  523:       // names that do not conflict with renamedFreeVars, localNames, or each other.
  524:       var forbiddenNames = localNameSet.union(renamedFreeVars).toList();
  525:       var boundBindings = {};
  526:       var capturedVarList = capturedVars.toList();
  527:       for (var i= 0, capturedVar; capturedVar = capturedVarList[i]; i++) {
  528:         var newCapturedVar = Blockly.FieldLexicalVariable.nameNotIn(capturedVar, for
biddenNames);
  529:         boundBindings[capturedVar] = newCapturedVar;
  530:         forbiddenNames.push(newCapturedVar);
  531:       }
  532:       this.renameBound(new Blockly.Substitution(boundBindings), freeSubstitution);
  533:     } else {
  534:       this.renameBound(new Blockly.Substitution(), freeSubstitution);
  535:     }
  536:   },
  537:   freeVariables: function() { // return the free lexical variables of this block
  538:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock(this
.bodyInputName));
  539:     var localNames = this.declaredNames();
  540:     result.subtract(new Blockly.NameSet(localNames)); // This is LET semantics, not
LET* semantics, but should be changed!
  541:     var numDecls = localNames.length;
  542:     for (var i = 0; i < numDecls; i++) {
  543:       result.union(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('D
ECL'+i)));
```

```
    }
    if (this.nextConnection) {
      var nextBlock = this.nextConnection.targetBlock();
      result.unite(Blockly.LexicalVariable.freeVariables(nextBlock));
    }
    return result;
  },
  //TODO (user) this has not been internationalized yet
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TRANSLATED_NAME }]
};

/**
 * Prototype bindings for an expression block that declares local names for use in an expression body.
 * [lyn, 10/13/13] Refactored to share more code with Blockly.Blocks.local_declaration_statement
 */
Blockly.Blocks['local_declaration_expression'] = {
  category: 'Variables',  // *** [lyn, 11/07/12] Abstract over this
  helpUrl: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_HELPURL,
  initLocals: Blockly.Blocks.local_declaration_statement.initLocals,
  bodyInputName: 'RETURN',
  init: function() {
    this.initLocals();
    this.appendIndentedValueInput('RETURN')
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_IN_RETURN);
    // Create plug for expression output
    this.setOutput(true, null);
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_TOOLTIP);
  },
  onchange: Blockly.Blocks.local_declaration_statement.onchange,
  mutationToDom: Blockly.Blocks.local_declaration_statement.mutationToDom,
  domToMutation: Blockly.Blocks.local_declaration_statement.domToMutation,
  updateDeclarationInputs_: Blockly.Blocks.local_declaration_statement.updateDeclarationInputs_,
  parameterFlydown: Blockly.Blocks.local_declaration_statement.parameterFlydown,
  blocksInScope: Blockly.Blocks.local_declaration_statement.blocksInScope,
  decompose: Blockly.Blocks.local_declaration_statement.decompose,
  compose: Blockly.Blocks.local_declaration_statement.compose,
  dispose: Blockly.Blocks.local_declaration_statement.dispose,
  saveConnections: Blockly.Blocks.local_declaration_statement.saveConnections,
  getVars: Blockly.Blocks.local_declaration_statement.getVars,
  declaredNames: Blockly.Blocks.local_declaration_statement.declaredNames,
  renameVar: Blockly.Blocks.local_declaration_statement.renameVars,
  renameVars: Blockly.Blocks.local_declaration_statement.renameVar,
  renameBound: Blockly.Blocks.local_declaration_statement.renameBound,
  renameFree: Blockly.Blocks.local_declaration_statement.renameFree,
  freeVariables: Blockly.Blocks.local_declaration_statement.freeVariables,
  //TODO (user) this has not been internationalized yet
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_TRANSLATED_NAME }]
};

Blockly.Blocks['local_mutatorcontainer'] = {
  // Local variable container (for mutator dialog).
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_CONTAINER_TITLE_LOCAL_NAMES);
    this.appendStatementInput('STACK');
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_CONTAINER_TOOLTIP);
    this.contextMenu = false;
  },
  // [lyn. 11/24/12] Set procBlock associated with this container.
  setDefBlock: function (defBlock) {
    this.defBlock_ = defBlock;
  },
  // [lyn. 11/24/12] Set procBlock associated with this container.
  // Invariant: should not be null, since only created as mutator for a particular proc block.
  getDefBlock: function () {
    return this.defBlock_;
  },
  // [lyn. 11/24/12] Return list of param names in this container
  // Invariant: there should be no duplicates!
  declaredNames: function () {
    var paramNames = [];
    var paramBlock = this.getInputTargetBlock('STACK');
    while (paramBlock) {
      paramNames.push(paramBlock.getFieldValue('NAME'));
      paramBlock = paramBlock.nextConnection &&
                   paramBlock.nextConnection.targetBlock();
    }
    return paramNames;
  }
};

Blockly.Blocks['local_mutatorarg'] = {
  // Procedure argument (for mutator dialog).
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_ARG_TITLE_NAME)
        .appendField(new Blockly.FieldTextInput(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_ARG_DEFAULT_VARIABLE,
                                                Blockly.LexicalVariable.renameParam),
                     'NAME');
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip('');
    this.contextMenu = false;
  },
  getContainerBlock: function () {
    var parent = this.getParent();
    while (parent && ! (parent.type === "local_mutatorcontainer")) {
      parent = parent.getParent();
    }
    // [lyn, 11/24/12] Cache most recent container block so can reference it upon removal from mutator arg stack
    this.cachedContainerBlock_ = (parent && (parent.type === "local_mutatorcontainer") && parent) || null;
    return this.cachedContainerBlock_;
  },
  getDefBlock: function () {
    var container = this.getContainerBlock();
    return (container && container.getDefBlock()) || null;
  },
  blocksInScope: function () {
```

```
  655:     var defBlock = this.getDefBlock();
  656:     return (defBlock && defBlock.blocksInScope()) || [];
  657:   },
  658:   declaredNames: function () {
  659:     var container = this.getContainerBlock();
  660:     return (container && container.declaredNames()) || [];
  661:   },
  662:
  663:   // [lyn, 11/24/12] Check for situation in which mutator arg has been removed from
stack,
  664:   onchange: function() {
  665:     var paramName = this.getFieldValue('NAME');
  666:     if (paramName) { // paramName is null when delete from stack
  667:       // console.log("Mutatorarg onchange: " + paramName);
  668:       var cachedContainer = this.cachedContainerBlock_;
  669:       var container = this.getContainerBlock(); // Order is important; this must com
e after cachedContainer
  670:                                                 // since it sets cachedContainerBloc
k_
  671:       // console.log("Mutatorarg onchange: " + paramName
  672:       //           + "; cachedContainer = " + JSON.stringify((cachedContainer && ca
chedContainer.type) || null)
  673:       //           + "; container = " + JSON.stringify((container && container.type
) || null));
  674:       if ((! cachedContainer) && container) {
  675:         // Event: added mutator arg to container stack
  676:         // console.log("Mutatorarg onchange ADDED: " + paramName);
  677:         var declaredNames = this.declaredNames();
  678:         var firstIndex = declaredNames.indexOf(paramName);
  679:         if (firstIndex != -1) {
  680:           // Assertion: we should get here, since paramName should be among names
  681:           var secondIndex = declaredNames.indexOf(paramName, firstIndex+1);
  682:           if (secondIndex != -1) {
  683:             // If we get here, there is a duplicate on insertion that must be resolv
ed
  684:             var newName = Blockly.FieldLexicalVariable.nameNotIn(paramName,declaredN
ames);
  685:             this.setFieldValue(newName, 'NAME');
  686:           }
  687:         }
  688:       }
  689:     }
  690:   }
  691: };
  692:
  693:
  694:
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Procedure blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

/**
 * Lyn's Change History:
 * [lyn, written 11/16-17/13, added 07/01/14]
 *   + Added freeVariables, renameFree, and renameBound to procedure declarations
 *   + Added renameVars for procedure declarations, which allows renaming multiple parameters simultaneously
 *   + Modified updateParams_ to accept optional params argument
 *   + Introduced bodyInputName field for procedure declarations ('STACK' for procedures_defnoreturn;
 *     'RETURN' for procedures_return), and use this to share more methods between the two kinds
 *     of procedure declarations.
 *   + Replaced inlined string list equality tests by new Blockly.LexicalVariable.stringListsEqual
 * [lyn, 10/28/13]
 *   + Fixed a missing change of Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure
 *   + I was wrong about re-rendering not being needed in updatedParams_!
 *     Without it, changing horizontal -> vertical params doesn't handle body slot correctly.
 *     So added it back.
 * [lyn, 10/27/13]
 *   + Fix bug in list of callers in flyout by simplifying domToMutation for procedure callers.
 *     This should never look for associated declaration, but just take arguments from given xml.
 *   + Removed render() call from updateParams_. Seems unnecessary. <== I WAS WRONG. SEE 10/28/13 NOTE
 *   + Specify direction of flydowns
 *   + Replaced Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure in proc decls
 * [lyn, 10/26/13] Modify procedure parameter changeHandler to propagate name changes to caller arg labels
 *     and open mutator labels
 * [lyn, 10/25/13]
 *   + Modified procedures_defnoreturn compose method so that it doesn't call updateParams_
 *     if mutator hasn't changed parameter names. This helps avoid a situation where
 *     an attempt is made to update params of a collapsed declaration.
 *   + Modified collapsed decls to have 'to ' prefix and collapsed callers to have 'call ' prefix.
 * [lyn, 10/24/13] Allowed switching between horizontal and vertical display of arguments
 * [lyn, 10/23/13] Fixed bug in domToMutation for callers that was giving wrong args to caller blocks.
 * [lyn, 10/10-14/13]
 *   + Installed variable declaration flydowns in both types of procedure declarations.
 *   + Fixed bug: Modified onchange for procedure declarations to keep arguments_ instance
 *     variable updated when param is edited directly on declaration block.
 *   + Removed get block (still in Variable drawer; no longer needed with parameter flydowns)
 *   + Removed "do {} then-return []" block since (1) it's in Control drawer and
 *     (2) it will be superseded in the context by Michael Phox's proc_defnoreturn mutator
 *     that allows adding a DO statement.
 *   + TODO: Abstract over string labels on all blocks using constants defined in en/_messages.js
 *   + TODO: Clean up code, including refactoring to increase sharing between
 *     procedures_defnoreturn and procedures_defreturn.
 * [lyn, 11/29/12] Integrated into App Inventor blocks. Known bugs:
 *   + Reordering mutator_args in mutator_container changes references to ??? because it interprets it
 *     as removing and inserting rather than moving.
 * [lyn, 11/24/12] Implemented procedure parameter renaming:
 *   + changing a variable name in mutator_arg for procedure changes it immediately in references in body.
 *   + no duplicate names are allowed in mutator_args; alpha-renaming prevents this.
 *   + no variables can be captured by renaming; alpha-renaming prevents this.
 */
'use strict';

goog.provide('Blockly.Blocks.procedures');

goog.require('Blockly.Blocks.Utilities');
goog.require('goog.dom');

Blockly.Blocks['procedures_defnoreturn'] = {
  // Define a procedure with no return value.
  category: 'Procedures',  // Procedures are handled specially.
  helpUrl: Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_HELPURL,
  bodyInputName: 'STACK',
  init: function() {
    this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
    var name = Blockly.Procedures.findLegalName(
        Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_PROCEDURE, this);
    this.appendDummyInput('HEADER')
        .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DEFINE)
        // [lyn, 10/27/13] Replaced Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure
        .appendField(new Blockly.FieldTextInput(name, Blockly.AIProcedure.renameProcedure), 'NAME');
    this.horizontalParameters = true; // horizontal by default
    this.appendStatementInput('STACK')
        .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DO);
    this.setMutator(new Blockly.Mutator(['procedures_mutatorarg']));
    this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_TOOLTIP);
    this.arguments_ = []; // List of declared local variable names; has one ("name") initially
                          // Other methods guarantee the invariant that this variable contains
                          // the list of names declared in the local declaration block.
    this.warnings = [{name:"checkEmptySockets",sockets:["STACK"]}];
  },
  onchange: function () {
    this.arguments_ = this.declaredNames(); // ensure arguments_ is in sync with paramFlydown fields
  },
  updateParams_: function(opt_params) {  // make rendered block reflect the parameter names currently in this.arguments_
    // [lyn, 11/17/13] Added optional opt_params argument:
    //    If its falsey (null or undefined), use the existing this.arguments_ list
```

```
    //    Otherwise, replace this.arguments_ by opt_params
    // In either case, make rendered block reflect the parameter names in this.argum
ents_
    if (opt_params) {
      this.arguments_ = opt_params;
    }
    // Check for duplicated arguments.
    // [lyn 10/10/13] Note that in blocks edited within AI2, duplicate parameter nam
es should never occur
    //    because parameters are renamed to avoid duplication. But duplicates might
show up
    //    in XML code hand-edited by user.
    // console.log("enter procedures_defnoreturn updateParams_()");
    var badArg = false;
    var hash = {};
    for (var x = 0; x < this.arguments_.length; x++) {
      if (hash['arg_' + this.arguments_[x].toLowerCase()]) {
        badArg = true;
        break;
      }
      hash['arg_' + this.arguments_[x].toLowerCase()] = true;
    }
    if (badArg) {
      this.setWarningText(Blockly.Msg.LANG_PROCEDURES_DEF_DUPLICATE_WARNING);
    } else {
      this.setWarningText(null);
    }

    var procName = this.getFieldValue('NAME');
    //save the first two input lines and the last input line
    //to be re added to the block later
    // var firstInput = this.inputList[0];  // [lyn, 10/24/13] need to reconstruct f
irst input
    var bodyInput = this.inputList[this.inputList.length - 1]; // Body of procedure

    // stop rendering until block is recreated
    var savedRendered = this.rendered;
    this.rendered = false;

    // remove first input
    // console.log("updateParams_: remove input HEADER");
    var thisBlock = this; // Grab correct object for use in thunk below
    Blockly.FieldParameterFlydown.withChangeHanderDisabled(
        // [lyn, 07/02/14] Need to disable change handler, else this will try to ren
ame params for horizontal arg fields!
        function() {thisBlock.removeInput('HEADER');}
    );

    // [lyn, 07/02/14 fixed logic] remove all old argument inputs (if they were vert
ical)
    if (! this.horizontalParameters) {
      var oldArgCount = this.inputList.length - 1; // Only args and body are left
      if (oldArgCount > 0) {
        var paramInput0 = this.getInput('VAR0');
        if (paramInput0) { // Yes, they were vertical
          for (var i = 0; i < oldArgCount; i++)
          {
            try
            {
              Blockly.FieldParameterFlydown.withChangeHanderDisabled(
                  // [lyn, 07/02/14] Need to disable change handler, else this will
try to rename params for vertical arg fields!
                  function() {thisBlock.removeInput('VAR' + i);}
              );
            }
            catch(err)
            {
              console.log(err);
            }
          }
        }
      }
    }

    //empty the inputList then recreate it
    this.inputList = [];

    // console.log("updateParams_: create input HEADER");
    var headerInput =
        this.appendDummyInput('HEADER')
            .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DEFINE)
            // [lyn, 10/28/13] Replaced Blockly.Procedures.rename by Blockly.AIProce
dure.renameProcedure
            .appendField(new Blockly.FieldTextInput(procName, Blockly.AIProcedure.re
nameProcedure), 'NAME');

    //add an input title for each argument
    //name each input after the block and where it appears in the block to reference
 it later
    for (var i = 0; i < this.arguments_.length; i++) {
      if (this.horizontalParameters) { // horizontal case
        headerInput.appendField(' ')
                   .appendField(this.parameterFlydown(i), // [lyn, 10/10/13] Changed
 to param flydown
                                'VAR' + i); // Tag with param tag to make it easy to
 find later.
      } else { // vertical case
        this.appendDummyInput('VAR' + i)
            // .appendField(this.arguments_[i])
            .appendField(this.parameterFlydown(i), 'VAR' + i)
            .setAlign(Blockly.ALIGN_RIGHT);
      }
    }

    //put the last two arguments back
    this.inputList = this.inputList.concat(bodyInput);

    this.rendered = savedRendered;
    // [lyn, 10/28/13] I thought this rerendering was unnecessary. But I was wrong!
    // Without it, get bug noticed by Andrew in which toggling horizontal -> vertica
l params
    // in procedure decl doesn't handle body tag appropriately!
    if (this.rendered) {
       this.render();
    }
    // console.log("exit procedures_defnoreturn updateParams_()");
  },
  // [lyn, 10/26/13] Introduced this to correctly handle renaming of [(1) caller arg
 labels and
  // (2) mutatorarg in open mutator] when procedure parameter flydown name is edited
.
  parameterFlydown: function (paramIndex) { // Return a new procedure parameter flyd
own
    var initialParamName = this.arguments_[paramIndex];
```

```
204:     var procDecl = this; // Here, "this" is the proc decl block. Name it to use in f
unction below
205:     var procWorkspace = this.workspace;
206:     var procedureParameterChangeHandler = function (newParamName) {
207:       // console.log("enter procedureParameterChangeHandler");
208:
209:
210:       // Extra work that needs to be done when procedure param name is changed, in a
ddition
211:       // to renaming lexical variables:
212:       //   1. Change all callers so label reflects new name
213:       //   2. If there's an open mutator, change the corresponding slot.
214:       // Note: this handler is invoked as method on field, so within the handler bod
y,
215:       // "this" will be bound to that field and *not* the procedure declaration obje
ct!
216:
217:       // Subtlety #1: within this changeHandler, procDecl.arguments_ has *not* yet b
een
218:       // updated to include newParamName. This only happens later. But since we know
 newParamName
219:       // *and* paramIndex, we know how to update procDecl.arguments_ ourselves!
220:
221:       // Subtlety #2: I would have thought we would want to create local copy of
222:       // procedure arguments_ list rather than mutate that list, but I'd be wrong!
223:       // Turns out that *not* mutating list here causes trouble below in the line
224:       //
225:       //   Blockly.Field.prototype.setText.call(mutatorarg.getTitle_("NAME"), newPar
amName);
226:       //
227:       // The reason is that this fires a change event in mutator workspace, which ca
uses
228:       // a call to the proc decl compose() method, and when it detects a difference
in
229:       // the arguments it calls proc decl updateParams_. This removes proc decl inpu
ts
230:       // before adding them back, and all hell breaks loose when the procedure name
field
231:       // and previous parameter flydown fields are disposed before an attempt is mad
e to
232:       // disposed this field. At this point, the SVG element associated with the pro
cedure name
233:       // is gone but the field is still in the title list. Attempting to dispose thi
s field
234:       // attempts to hide the open HTML editor widget, which attempts to re-render t
he
235:       // procedure declaration block. But the null SVG for the procedure name field
236:       // raises an exception.
237:       //
238:       // It turns out that by mutating proc decl arguments_, when compose() is calle
d,
239:       // updateParams_() is *not* called, and this prevents the above scenario.
240:       //  So rather than doing
241:       //
242:       //    var newArguments = [].concat(procDecl.arguments_)
243:       //
244:       // we instead do:
245:       var newArguments = procDecl.arguments_;
246:       newArguments[paramIndex] = newParamName;
247:
248:       var procName = procDecl.getFieldValue('NAME');
249:
250:       // 1. Change all callers so label reflects new name
251:       Blockly.Procedures.mutateCallers(procName, procWorkspace, newArguments, procDe
cl.paramIds_);
252:
253:       // 2. If there's an open mutator, change the name in the corresponding slot.
254:       if (procDecl.mutator && procDecl.mutator.rootBlock_) {
255:         // Iterate through mutatorarg param blocks and change name of one at paramIn
dex
256:         var mutatorContainer = procDecl.mutator.rootBlock_;
257:         var mutatorargIndex = 0;
258:         var mutatorarg = mutatorContainer.getInputTargetBlock('STACK');
259:         while (mutatorarg && mutatorargIndex < paramIndex) {
260:             mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.targ
etBlock();
261:             mutatorargIndex++;
262:         }
263:         if (mutatorarg && mutatorargIndex == paramIndex) {
264:           // Subtlety #3: If call mutatorargs's setValue, its change handler will be
 invoked
265:           // several times, and on one of those times, it will find new param name i
n
266:           // the procedures arguments_ instance variable and will try to renumber it
267:           // (e.g. "a" -> "a2"). To avoid this, invoke the setText method of its Fie
ld s
268:           // superclass directly. I.e., can't do this:
269:           //   mutatorarg.getTitle_("NAME").setValue(newParamName);
270:           // so instead do this:
271:             Blockly.Field.prototype.setText.call(mutatorarg.getField_("NAME"), newPa
ramName);
272:         }
273:       }
274:       // console.log("exit procedureParameterChangeHandler");
275:     }
276:     return new Blockly.FieldParameterFlydown(initialParamName,
277:                                              true, // name is editable
278:                                              // [lyn, 10/27/13] flydown location dep
ends on parameter orientation
279:                                              this.horizontalParameters ? Blockly.Fie
ldFlydown.DISPLAY_BELOW
280:                                                                        : Blockly.Fie
ldFlydown.DISPLAY_RIGHT,
281:                                              procedureParameterChangeHandler);
282:   },
283:   setParameterOrientation: function(isHorizontal) {
284:     var params = this.getParameters();
285:     if (params.length != 0 && isHorizontal !== this.horizontalParameters) {
286:       this.horizontalParameters = isHorizontal;
287:       this.updateParams_();
288:     }
289:   },
290:   mutationToDom: function() {
291:     var container = document.createElement('mutation');
292:     if (!this.horizontalParameters) {
293:       container.setAttribute('vertical_parameters', "true"); // Only store an elemen
t for vertical
294:       // The absence of this attribute means horizontal.
295:     }
296:     for (var x = 0; x < this.arguments_.length; x++) {
297:       var parameter = document.createElement('arg');
298:       parameter.setAttribute('name', this.arguments_[x]);
299:       container.appendChild(parameter);
300:     }
```

```
301:     return container;
302:   },
303:   domToMutation: function(xmlElement) {
304:     var params = [];
305:     var children = goog.dom.getChildren(xmlElement);
306:     for (var x = 0, childNode; childNode = children[x]; x++) {
307:       if (childNode.nodeName.toLowerCase() == 'arg') {
308:         params.push(childNode.getAttribute('name'));
309:       }
310:     }
311:     this.horizontalParameters = xmlElement.getAttribute('vertical_parameters') !== "
true";
312:     this.updateParams_(params);
313:   },
314:   decompose: function(workspace) {
315:     var containerBlock = new Blockly.Block.obtain(workspace, 'procedures_mutatorcont
ainer');
316:     containerBlock.initSvg();
317:     // [lyn, 11/24/12] Remember the associated procedure, so can
318:     // appropriately change body when update name in param block.
319:     containerBlock.setProcBlock(this);
320:     this.paramIds_ = [] // [lyn, 10/26/13] Added
321:     var connection = containerBlock.getInput('STACK').connection;
322:     for (var x = 0; x < this.arguments_.length; x++) {
323:       var paramBlock = new Blockly.Block.obtain(workspace, 'procedures_mutatorarg');
324:       this.paramIds_.push(paramBlock.id); // [lyn, 10/26/13] Added
325:       paramBlock.initSvg();
326:       paramBlock.setFieldValue(this.arguments_[x], 'NAME');
327:       // Store the old location.
328:       paramBlock.oldLocation = x;
329:       connection.connect(paramBlock.previousConnection);
330:       connection = paramBlock.nextConnection;
331:     }
332:     // [lyn, 10/26/13] Rather than passing null for paramIds, pass actual paramIds
333:     // and use true flag to initialize tracking.
334:     Blockly.Procedures.mutateCallers(this.getFieldValue('NAME'),
335:                                      this.workspace, this.arguments_, this.paramIds_
, true);
336:     return containerBlock;
337:   },
338:   compose: function(containerBlock) {
339:     var params = [];
340:     this.paramIds_ = [];
341:     var paramBlock = containerBlock.getInputTargetBlock('STACK');
342:     while (paramBlock) {
343:       params.push(paramBlock.getFieldValue('NAME'));
344:       this.paramIds_.push(paramBlock.id);
345:       paramBlock = paramBlock.nextConnection &&
346:           paramBlock.nextConnection.targetBlock();
347:     }
348:     // console.log("enter procedures_defnoreturn compose(); prevArguments = "
349:     //     + prevArguments.join(',')
350:     //     + "; currentAguments = "
351:     //     + this.arguments_.join(',')
352:     //     + ";"
353:     // );
354:     // [lyn, 11/24/12] Note: update params updates param list in proc declaration,
355:     // but renameParam updates procedure body appropriately.
356:     if (!Blockly.LexicalVariable.stringListsEqual(params, this.arguments_)) { // Onl
y need updates if param list has changed
357:       this.updateParams_(params);
358:       Blockly.Procedures.mutateCallers(this.getFieldValue('NAME'),
359:           this.workspace, this.arguments_, this.paramIds_);
360:     }
361:     // console.log("exit procedures_defnoreturn compose()");
362:   },
363:   dispose: function() {
364:     var name = this.getFieldValue('NAME');
365:     var editable = this.editable_;
366:     var workspace = this.workspace;
367:
368:     // Call parent's destructor.
369:     Blockly.Block.prototype.dispose.apply(this, arguments);
370:
371:     if (editable) {
372:       // Dispose of any callers.
373:       //Blockly.Procedures.disposeCallers(name, workspace);
374:       Blockly.AIProcedure.removeProcedureValues(name, workspace);
375:     }
376:
377:   },
378:   getProcedureDef: function() {
379:     // Return the name of the defined procedure,
380:     // a list of all its arguments,
381:     // and that it DOES NOT have a return value.
382:     return [this.getFieldValue('NAME'),
383:             this.arguments_,
384:            this.bodyInputName === 'RETURN']; // true for procedures that return valu
es.
385:   },
386:   getVars: function() {
387:     var names = []
388:     for (var i = 0, param; param = this.getFieldValue('VAR' + i); i++) {
389:       names.push(param);
390:     }
391:     return names;
392:   },
393:   declaredNames: function() { // [lyn, 10/11/13] return the names of all parameters
of this procedure
394:     return this.getVars();
395:   },
396:   renameVar: function(oldName, newName) {
397:     this.renameVars(Blockly.Substitution.simpleSubstitution(oldName,newName));
398:   },
399:   renameVars: function(substitution) { // renaming is a dict (i.e., object) mapping
old names to new ones
400:     var oldParams = this.getParameters();
401:     var newParams = substitution.map(oldParams);
402:     if (!Blockly.LexicalVariable.stringListsEqual(oldParams, newParams)) {
403:       this.updateParams_(newParams);
404:       // Update the mutator's variables if the mutator is open.
405:       if (this.mutator.isVisible()) {
406:         var blocks = this.mutator.workspace_.getAllBlocks();
407:         for (var x = 0, block; block = blocks[x]; x++) {
408:           if (block.type == 'procedures_mutatorarg') {
409:             var oldName = block.getFieldValue('NAME');
410:             var newName = substitution.apply(oldName);
411:             if (newName !== oldName) {
412:               block.setFieldValue(newName, 'NAME');
413:             }
414:           }
415:         }
416:       }
417:     }
```

```
  418:   },
  419:   renameBound: function (boundSubstitution, freeSubstitution) {
  420:     var paramSubstitution = boundSubstitution.restrictDomain(this.declaredNames());
  421:     this.renameVars(paramSubstitution);
  422:     var newFreeSubstitution = freeSubstitution.extend(paramSubstitution);
  423:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock(this.bodyInputName),
 newFreeSubstitution);
  424:   },
  425:   renameFree: function (freeSubstitution) { // Should have no effect since only top-
level procedures.
  426:     var freeVars = this.freeVariables(); // Calculate free variables, which should b
e empty,
  427:                                          // throwing exception if not.
  428:     // There should be no free variables, and so nothing to rename. Do nothing else.
  429:   },
  430:   freeVariables: function() { // return the free lexical variables of this block
  431:                               // Should return the empty set: something is wrong if
it doesn't!
  432:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock(this
.bodyInputName));
  433:     result.subtract(new Blockly.NameSet(this.declaredNames()));
  434:     if (result.isEmpty()) {
  435:       return result;
  436:     } else {
  437:       throw "Violation of invariant: procedure declaration has nonempty free variabl
es: " + result.toString();
  438:     }
  439:   },
  440:   // [lyn, 11/24/12] return list of procedure body (if there is one)
  441:   blocksInScope: function () {
  442:     var body = this.getInputTargetBlock(this.bodyInputName);
  443:     return (body && [body]) || [];
  444:   },
  445:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_PROCEDURE +
  446:       ' ' + Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DO }],
  447:   customContextMenu: function (options) {
  448:     Blockly.FieldParameterFlydown.addHorizontalVerticalOption(this, options);
  449:   },
  450:   getParameters: function() {
  451:     return this.arguments_;
  452:   }
  453: };
  454:
  455: // [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN
)
  456: Blockly.Blocks['procedures_defreturn'] = {
  457:   // Define a procedure with a return value.
  458:   category: 'Procedures',  // Procedures are handled specially.
  459:   helpUrl: Blockly.Msg.LANG_PROCEDURES_DEFRETURN_HELPURL,
  460:   bodyInputName: 'RETURN',
  461:   init: function() {
  462:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  463:     var name = Blockly.Procedures.findLegalName(
  464:         Blockly.Msg.LANG_PROCEDURES_DEFRETURN_PROCEDURE, this);
  465:     this.appendDummyInput('HEADER')
  466:         .appendField(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_DEFINE)
  467:       // [lyn, 10/27/13] Replaced Blockly.Procedures.rename by Blockly.AIProcedure.r
enameProcedure
  468:         .appendField(new Blockly.FieldTextInput(name, Blockly.AIProcedure.renameProc
edure), 'NAME');
  469:     this.horizontalParameters = true; // horizontal by default
  470:     this.appendIndentedValueInput('RETURN')
  471:         .appendField(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_RETURN);
  472:     this.setMutator(new Blockly.Mutator(['procedures_mutatorarg']));
  473:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_TOOLTIP);
  474:     this.arguments_ = [];
  475:     this.warnings = [{name:"checkEmptySockets",sockets:["RETURN"]}];
  476:   },
  477:   onchange: Blockly.Blocks.procedures_defnoreturn.onchange,
  478:   // [lyn, 11/24/12] return list of procedure body (if there is one)
  479:   updateParams_: Blockly.Blocks.procedures_defnoreturn.updateParams_,
  480:   parameterFlydown: Blockly.Blocks.procedures_defnoreturn.parameterFlydown,
  481:   setParameterOrientation: Blockly.Blocks.procedures_defnoreturn.setParameterOrienta
tion,
  482:   mutationToDom: Blockly.Blocks.procedures_defnoreturn.mutationToDom,
  483:   domToMutation: Blockly.Blocks.procedures_defnoreturn.domToMutation,
  484:   decompose: Blockly.Blocks.procedures_defnoreturn.decompose,
  485:   compose: Blockly.Blocks.procedures_defnoreturn.compose,
  486:   dispose: Blockly.Blocks.procedures_defnoreturn.dispose,
  487:   getProcedureDef: Blockly.Blocks.getProcedureDef,
  488:   getVars: Blockly.Blocks.procedures_defnoreturn.getVars,
  489:   declaredNames: Blockly.Blocks.procedures_defnoreturn.declaredNames,
  490:   renameVar: Blockly.Blocks.procedures_defnoreturn.renameVar,
  491:   renameVars: Blockly.Blocks.procedures_defnoreturn.renameVars,
  492:   renameBound: Blockly.Blocks.procedures_defnoreturn.renameBound,
  493:   renameFree: Blockly.Blocks.procedures_defnoreturn.renameFree,
  494:   freeVariables: Blockly.Blocks.procedures_defnoreturn.freeVariables,
  495:   blocksInScope: Blockly.Blocks.procedures_defnoreturn.blocksInScope,
  496:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_DEFRETURN_PROCEDURE +
  497:       ' ' + Blockly.Msg.LANG_PROCEDURES_DEFRETURN_RETURN }],
  498:   customContextMenu: Blockly.Blocks.procedures_defnoreturn.customContextMenu,
  499:   getParameters: Blockly.Blocks.procedures_defnoreturn.getParameters
  500: };
  501:
  502: Blockly.Blocks['procedures_mutatorcontainer'] = {
  503:   // Procedure container (for mutator dialog).
  504:   init: function() {
  505:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  506:     this.appendDummyInput()
  507:         .appendField(Blockly.Msg.LANG_PROCEDURES_MUTATORCONTAINER_TITLE);
  508:     this.appendStatementInput('STACK');
  509:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_MUTATORCONTAINER_TOOLTIP);
  510:     this.contextMenu = false;
  511:   },
  512:   // [lyn. 11/24/12] Set procBlock associated with this container.
  513:   setProcBlock: function (procBlock) {
  514:     this.procBlock_ = procBlock;
  515:   },
  516:   // [lyn. 11/24/12] Set procBlock associated with this container.
  517:   // Invariant: should not be null, since only created as mutator for a particular p
roc block.
  518:   getProcBlock: function () {
  519:     return this.procBlock_;
  520:   },
  521:   // [lyn. 11/24/12] Return list of param names in this container
  522:   // Invariant: there should be no duplicates!
  523:   declaredNames: function () {
  524:     var paramNames = [];
  525:     var paramBlock = this.getInputTargetBlock('STACK');
  526:     while (paramBlock) {
  527:       paramNames.push(paramBlock.getFieldValue('NAME'));
  528:       paramBlock = paramBlock.nextConnection &&
  529:                    paramBlock.nextConnection.targetBlock();
  530:     }
```

```
  531:     return paramNames;
  532:   }
  533: };
  534:
  535: Blockly.Blocks['procedures_mutatorarg'] = {
  536:   // Procedure argument (for mutator dialog).
  537:   init: function() {
  538: //    var mutatorarg = this;
  539: //    var mutatorargChangeHandler = function(newName) {
  540: //      var proc = mutatorarg.getProcBlock();
  541: //      var procArguments = proc ? proc.arguments_ : [];
  542: //      console.log("mutatorargChangeHandler: newName = " + newName
  543: //                  + " and proc argumnets = [" + procArguments.join(',') + "]");
  544: //      return Blockly.LexicalVariable.renameParam.call(this,newName);
  545: //    }
  546:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  547:     this.appendDummyInput()
  548:         .appendField(Blockly.Msg.LANG_PROCEDURES_MUTATORARG_TITLE)
  549:         .appendField(new Blockly.FieldTextInput('x',Blockly.LexicalVariable.renamePa
ram), 'NAME');
  550:     this.setPreviousStatement(true);
  551:     this.setNextStatement(true);
  552:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_MUTATORARG_TOOLTIP);
  553:     this.contextMenu = false;
  554:   },
  555:   // [lyn, 11/24/12] Return the container this mutator arg is in, or null if it's no
t in one.
  556:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
  557:   // be in container stack.
  558:   getContainerBlock: function () {
  559:     var parent = this.getParent();
  560:     while (parent && ! (parent.type === "procedures_mutatorcontainer")) {
  561:       parent = parent.getParent();
  562:     }
  563:     // [lyn, 11/24/12] Cache most recent container block so can reference it upon re
moval from mutator arg stack
  564:     this.cachedContainerBlock_ = (parent && (parent.type === "procedures_mutatorcont
ainer") && parent) || null;
  565:     return this.cachedContainerBlock_;
  566:   },
  567:   // [lyn, 11/24/12] Return the procedure associated with mutator arg is in, or null
 if there isn't one.
  568:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
  569:   // be in container stack.
  570:   getProcBlock: function () {
  571:     var container = this.getContainerBlock();
  572:     return (container && container.getProcBlock()) || null;
  573:   },
  574:   // [lyn, 11/24/12] Return the declared names in the procedure associated with muta
tor arg,
  575:   // or the empty list if there isn't one.
  576:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
  577:   // be in container stack.
  578:   declaredNames: function () {
  579:     var container = this.getContainerBlock();
  580:     return (container && container.declaredNames()) || [];
  581:   },
  582:   // [lyn, 11/24/12] Return the blocks in scope of proc params in the the procedure
associated with mutator arg,
  583:   // or the empty list if there isn't one.
  584:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
  585:   // be in container stack.
  586:   blocksInScope: function () {
  587:     var proc = this.getProcBlock();
  588:     return (proc && proc.blocksInScope()) || [];
  589:   },
  590:   // [lyn, 11/24/12] Check for situation in which mutator arg has been removed from
stack,
  591:   // and change all references to its name to ???.
  592:   onchange: function() {
  593:     var paramName = this.getFieldValue('NAME');
  594:     if (paramName) { // paramName is null when delete from stack
  595:       // console.log("Mutatorarg onchange: " + paramName);
  596:       var cachedContainer = this.cachedContainerBlock_;
  597:       var container = this.getContainerBlock(); // Order is important; this must com
e after cachedContainer
  598:                                                 // since it sets cachedContainerBloc
k_
  599:       // console.log("Mutatorarg onchange: " + paramName
  600:       //             + "; cachedContainer = " + JSON.stringify((cachedContainer && ca
chedContainer.type) || null)
  601:       //             + "; container = " + JSON.stringify((container && container.type
) || null));
  602:       if ((! cachedContainer) && container) {
  603:         // Event: added mutator arg to container stack
  604:         // console.log("Mutatorarg onchange ADDED: " + paramName);
  605:         var declaredNames = this.declaredNames();
  606:         var firstIndex = declaredNames.indexOf(paramName);
  607:         if (firstIndex != -1) {
  608:           // Assertion: we should get here, since paramName should be among names
  609:           var secondIndex = declaredNames.indexOf(paramName, firstIndex+1);
  610:           if (secondIndex != -1) {
  611:             // If we get here, there is a duplicate on insertion that must be resolv
ed
  612:             var newName = Blockly.FieldLexicalVariable.nameNotIn(paramName,declaredN
ames);
  613:             this.setFieldValue(newName, 'NAME');
  614:           }
  615:         }
  616:       } /* else if (cachedContainer && (! container)) {
  617:         // Event: removed mutator arg from container stack
  618:         // [lyn, 11/24/12] Mutator arg has been removed from stack. Change all refer
ences to its name to ???
  619:         // console.log("Mutatorarg onchange REMOVED: " + paramName);
  620:         var proc = cachedContainer.getProcBlock();
  621:         var inScopeBlocks = (proc && proc.blocksInScope()) || [];
  622:         var referenceResults = inScopeBlocks.map( function(blk) { return Blockly.Lex
icalVariable.referenceResult(blk, paramName, []); } );
  623:         var blocksToRename = [];
  624:         for (var r = 0; r < referenceResults.length; r++) {
  625:           blocksToRename = blocksToRename.concat(referenceResults[r][0]);
  626:           // ignore capturables, which are not relevant here.
  627:         }
  628:         // Rename getters and setters
  629:         for (var i = 0; i < blocksToRename.length; i++) {
  630:           var block = blocksToRename[i];
  631:           var renamingFunction = block.renameLexicalVar;
  632:           if (renamingFunction) {
  633:             renamingFunction.call(block, "param " + paramName, "???");
  634:           }
```

```
635:         }
636:       }*/
637:     }
638:   }
639: };
640:
641: Blockly.Blocks.procedures_mutatorarg.validator = function(newVar) {
642:   // Merge runs of whitespace.  Strip leading and trailing whitespace.
643:   // Beyond this, all names are legal.
644:   newVar = newVar.replace(/[\s\xa0]+/g, ' ').replace(/^ | $/g, '');
645:   return newVar || null;
646: };
647:
648: /* [lyn 10/10/13] With parameter flydown changes,
649:  * I don't think a special GET block in the Procedure drawer is necesssary
650: Blockly.Blocks.procedure_lexical_variable_get = {
651:   // Variable getter.
652:   category: 'Procedures',
653:   helpUrl: Blockly.Msg.LANG_PROCEDURES_GET_HELPURL, // *** [lyn, 11/10/12] Fix this
654:   init: function() {
655:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
656:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
657:     this.fieldVar_.setBlock(this);
658:     this.appendDummyInput()
659:         .appendField("get")
660:         .appendField(this.fieldVar_, 'VAR');
661:     this.setOutput(true, null);
662:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_GET_TOOLTIP);
663:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
664:   },
665:   getVars: function() {
666:     return [this.getFieldValue('VAR')];
667:   },
668:   onchange: function() {
669:      // [lyn, 11/10/12] Checks if parent has changed. If so, checks if curent variab
le name
670:      //    is still in scope. If so, keeps it as is; if not, changes to ???
671:      //    *** NEED TO MAKE THIS BEHAVIOR BETTER!
672:     if (this.fieldVar_) {
673:        var currentName = this.fieldVar_.getText();
674:        var nameList = this.fieldVar_.getNamesInScope();
675:        var cachedParent = this.fieldVar_.getCachedParent();
676:        var currentParent = this.fieldVar_.getBlock().getParent();
677:        // [lyn, 11/10/12] Allow current name to stay if block moved to workspace in
"untethered" way.
678:        //   Only changed to ??? if tether an untethered block.
679:        if (currentParent != cachedParent) {
680:          this.fieldVar_.setCachedParent(currentParent);
681:          if  (currentParent !== null) {
682:            for (var i = 0; i < nameList.length; i++ ) {
683:              if (nameList[i] === currentName) {
684:                return; // no change
685:              }
686:            }
687:            // Only get here if name not in list
688:            this.fieldVar_.setText(" ");
689:          }
690:        }
691:     }
692:     Blockly.WarningHandler.checkErrors.call(this);
693:   },
694:   renameLexicalVar: function(oldName, newName) {
695:     // console.log("Renaming lexical variable from " + oldName + " to " + newName);
696:     if (oldName === this.getFieldValue('VAR')) {
697:         this.setFieldValue(newName, 'VAR');
698:     }
699:   }
700: };
701: */
702:
703: /* // [lyn, 10/14/13] This will become unnecessary with Michael Phox's
704:  * mutator on procedure_defreturn that allows adding a DO statement.
705:  */
706: /*
707: Blockly.Blocks.procedures_do_then_return = {
708:   // String length.
709:   category: 'Procedures',
710:   helpUrl: Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_HELPURL,
711:   init: function() {
712:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
713:     this.appendStatementInput('STM')
714:         .appendField(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_DO);
715:     this.appendValueInput('VALUE')
716:         .appendField(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_RETURN)
717:         .setAlign(Blockly.ALIGN_RIGHT);
718:     this.setOutput(true, null);
719:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_TOOLTIP);
720:   },
721:   onchange: Blockly.WarningHandler.checkErrors
722: };
723: */
724:
725: Blockly.Blocks['procedures_callnoreturn'] = {
726:   // Call a procedure with no return value.
727:   category: 'Procedures',  // Procedures are handled specially.
728:   helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_HELPURL,
729:   init: function() {
730:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
731:     this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(fals
e);};
732:
733:     this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldPro
cedure.onChange);
734:     this.procDropDown.block = this;
735:     this.appendDummyInput()
736:         .appendField(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_CALL)
737:         .appendField(this.procDropDown,"PROCNAME");
738:     this.setPreviousStatement(true);
739:     this.setNextStatement(true);
740:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_TOOLTIP);
741:     this.arguments_ = [];
742:     this.quarkConnections_ = null;
743:     this.quarkArguments_ = null;
744:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["PROCNAME"]}];
745:     //Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.procNames
Fxn(false)[0][0]);
746:     Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.getField_("
PROCNAME").getValue());
747:   },
748:   getProcedureCall: function() {
749:     return this.getFieldValue('PROCNAME');
750:   },
```

```
  renameProcedure: function(oldName, newName) {
    if (Blockly.Names.equals(oldName, this.getFieldValue('PROCNAME'))) {
      this.setFieldValue(newName, 'PROCNAME');
    }
  },
  // [lyn, 10/27/13] Renamed "fromChange" parameter to "startTracking", because it should be true in any situation
  // where we want caller to start tracking connections associated with paramIds. This includes when a mutator
  // is opened on a procedure declaration.
  setProcedureParameters: function(paramNames, paramIds, startTracking) {
    // Data structures for parameters on each call block:
    // this.arguments = ['x', 'y']
    //     Existing param names.
    // paramNames = ['x', 'y', 'z']
    //     New param names.
    // paramIds = ['piua', 'f8b_', 'oi.o']
    //     IDs of params (consistent for each parameter through the life of a
    //     mutator, regardless of param renaming).
    // this.quarkConnections_ {piua: null, f8b_: Blockly.Connection}
    //     Look-up of paramIds to connections plugged into the call block.
    // this.quarkArguments_ = ['piua', 'f8b_']
    //     Existing param IDs.
    // Note that quarkConnections_ may include IDs that no longer exist, but
    // which might reappear if a param is reattached in the mutator.

    var input;
    var connection;
    var x;

    //fixed parameter alignment see ticket 465
    if (!paramIds) {
      // Reset the quarks (a mutator is about to open).
      this.quarkConnections_ = {};
      this.quarkArguments_ = null;
      // return;  // [lyn, 10/27/13] No, don't return yet. We still want to add paramNames to block!
      // For now, create dummy list of param ids. This needs to be cleaned up further!
      paramIds = [].concat(paramNames); // create a dummy list that's a copy of paramNames.
    }
    if (paramIds.length != paramNames.length) {
      throw 'Error: paramNames and paramIds must be the same length.';
    }
    var paramIdToParamName = {};
    for(var i=0;i<paramNames.length;i++) {
      paramIdToParamName[paramIds[i]] = paramNames[i];
    }
    if(typeof startTracking == "undefined") {
      startTracking = null;
    }

    if (!this.quarkArguments_ || startTracking) {
      // Initialize tracking for this block.
      this.quarkConnections_ = {};
      if (Blockly.LexicalVariable.stringListsEqual(paramNames, this.arguments_) || startTracking) {
        // No change to the parameters, allow quarkConnections_ to be
        // populated with the existing connections.
        this.quarkArguments_ = paramIds;
      } else {
        this.quarkArguments_ = [];
      }
    }
    // Switch off rendering while the block is rebuilt.
    var savedRendered = this.rendered;
    this.rendered = false;
    // Update the quarkConnections_ with existing connections.
    for (x = 0;this.getInput('ARG' + x); x++) {
      input = this.getInput('ARG' + x);
      if (input) {
        connection = input.connection.targetConnection;
        this.quarkConnections_[this.quarkArguments_[x]] = connection;
        // Disconnect all argument blocks and remove all inputs.
        this.removeInput('ARG' + x);
      }
    }
    // Rebuild the block's arguments.
    this.arguments_ = [].concat(paramNames);
    this.quarkArguments_ = paramIds;
    for (x = 0; x < this.arguments_.length; x++) {
      input = this.appendValueInput('ARG' + x)
          .setAlign(Blockly.ALIGN_RIGHT)
          .appendField(this.arguments_[x]);
      if (this.quarkArguments_) {
        // Reconnect any child blocks.
        var quarkName = this.quarkArguments_[x];
        if (quarkName in this.quarkConnections_) {
          connection = this.quarkConnections_[quarkName];
          if (!connection || connection.targetConnection ||
              connection.sourceBlock_.workspace != this.workspace) {
            // Block no longer exists or has been attached elsewhere.
            delete this.quarkConnections_[quarkName];
          } else {
            input.connection.connect(connection);
          }
        } else if(paramIdToParamName[quarkName]){
          connection = this.quarkConnections_[paramIdToParamName[quarkName]];
          if (connection){
            input.connection.connect(connection);
          }
        }
      }
    }
    // Restore rendering and show the changes.
    this.rendered = savedRendered;
    if (this.rendered) {
      this.render();
    }
  },
  mutationToDom: function() {
    // Save the name and arguments (none of which are editable).
    var container = document.createElement('mutation');
    container.setAttribute('name', this.getFieldValue('PROCNAME'));
    for (var x = 0; this.getInput("ARG" + x); x++) {
      var parameter = document.createElement('arg');
      parameter.setAttribute('name', this.getInput("ARG" + x).fieldRow[0].text_);
      container.appendChild(parameter);
    }
    return container;
  },
  domToMutation: function(xmlElement) {
    // Restore the name and parameters.
```

```
  869:     var name = xmlElement.getAttribute('name');
  870:     this.setFieldValue(name, 'PROCNAME');
  871:     // [lyn, 10/27/13] Significantly cleaned up this code. Always take arg names fro
m xmlElement.
  872:     // Do not attempt to find definition.
  873:     this.arguments_ = [];
  874:     var children = goog.dom.getChildren(xmlElement);
  875:     for (var x = 0, childNode; childNode = children[x]; x++) {
  876:       if (childNode.nodeName.toLowerCase() == 'arg') {
  877:         this.arguments_.push(childNode.getAttribute('name'));
  878:       }
  879:     }
  880:     this.setProcedureParameters(this.arguments_, null, true);
  881:       // [lyn, 10/27/13] Above. set tracking to true in case this is a block with ar
gument subblocks.
  882:       // and there's an open mutator.
  883:   },
  884:   renameVar: function(oldName, newName) {
  885:     for (var x = 0; x < this.arguments_.length; x++) {
  886:       if (Blockly.Names.equals(oldName, this.arguments_[x])) {
  887:         this.arguments_[x] = newName;
  888:         this.getInput('ARG' + x).fieldRow[0].setText(newName);
  889:       }
  890:     }
  891:   },
  892:   procCustomContextMenu: function(options) {
  893:     // Add option to find caller.
  894:     var option = {enabled: true};
  895:     option.text = Blockly.Msg.LANG_PROCEDURES_HIGHLIGHT_DEF;
  896:     var name = this.getFieldValue('PROCNAME');
  897:     var workspace = this.workspace;
  898:     option.callback = function() {
  899:       var def = Blockly.Procedures.getDefinition(name, workspace);
  900:       def && def.select();
  901:     };
  902:     options.push(option);
  903:   },
  904:   removeProcedureValue: function() {
  905:     this.setFieldValue("none", 'PROCNAME');
  906:     for(var i=0;this.getInput('ARG' + i) !== null;i++) {
  907:       this.removeInput('ARG' + i);
  908:     }
  909:   },
  910:   // This generates a single generic call to 'call no return' defaulting its value
  911:   // to the first procedure in the list. Calls for each procedure cannot be done her
e because the
  912:   // blocks have not been loaded yet (they are loaded in typeblock.js)
  913:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_TRANSLATED_
NAME}]
  914: };
  915:
  916:
  917: Blockly.Blocks['procedures_callreturn'] = {
  918:   // Call a procedure with a return value.
  919:   category: 'Procedures',  // Procedures are handled specially.
  920:   helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLRETURN_HELPURL,
  921:   init: function() {
  922:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  923:     this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(true
);};
  924:
  925:     this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldPro
cedure.onChange);
  926:     this.procDropDown.block = this;
  927:     this.appendDummyInput()
  928:         .appendField(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_CALL)
  929:         .appendField(this.procDropDown,"PROCNAME");
  930:     this.setOutput(true, null);
  931:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_TOOLTIP);
  932:     this.arguments_ = [];
  933:     this.quarkConnections_ = null;
  934:     this.quarkArguments_ = null;
  935:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["PROCNAME"]}];
  936:     //Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.procNames
Fxn()[0][0]);
  937:     Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.getField_("
PROCNAME").getValue());
  938:   },
  939:   getProcedureCall: Blockly.Blocks.procedures_callnoreturn.getProcedureCall,
  940:   renameProcedure: Blockly.Blocks.procedures_callnoreturn.renameProcedure,
  941:   setProcedureParameters:
  942:       Blockly.Blocks.procedures_callnoreturn.setProcedureParameters,
  943:   mutationToDom: Blockly.Blocks.procedures_callnoreturn.mutationToDom,
  944:   domToMutation: Blockly.Blocks.procedures_callnoreturn.domToMutation,
  945:   renameVar: Blockly.Blocks.procedures_callnoreturn.renameVar,
  946:   procCustomContextMenu: Blockly.Blocks.procedures_callnoreturn.procCustomContextMen
u,
  947:   removeProcedureValue: Blockly.Blocks.procedures_callnoreturn.removeProcedureValue,
  948:   // This generates a single generic call to 'call return' defaulting its value
  949:   // to the first procedure in the list. Calls for each procedure cannot be done her
e because the
  950:   // blocks have not been loaded yet (they are loaded in typeblock.js)
  951:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_CALLRETURN_TRANSLATED_NA
ME}]
  952: };
  953:
```

```
    1: /**
    2:  * Visual Blocks Language
    3:  *
    4:  * Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    5:  *
    6:  * Licensed under the Apache License, Version 2.0 (the "License");
    7:  * you may not use this file except in compliance with the License.
    8:  * You may obtain a copy of the License at
    9:  *
   10:  *   http://www.apache.org/licenses/LICENSE-2.0
   11:  *
   12:  * Unless required by applicable law or agreed to in writing, software
   13:  * distributed under the License is distributed on an "AS IS" BASIS,
   14:  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   15:  * See the License for the specific language governing permissions and
   16:  * limitations under the License.
   17:  */
   18:
   19: /**
   20:  * @fileoverview List generators for Blockly, modified for App Inventor
   21:  * @author fraser@google.com (Neil Fraser)
   22:  * @author andrew.f.mckinney@gmail.com (Andrew F. McKinney)
   23:  * Due to the frequency of long strings, the 80-column wrap rule need not apply
   24:  * to language files.
   25:  */
   26:
   27: /**
   28:  * Lyn's History:
   29:  * [lyn, 10/27/13] Modified for loop index variables to begin with YAIL_LOCAL_VAR_TAG (currently '$').
   30:  *     At least on Kawa-legal first character is necessary to ensure AI identifiers
   31:  *     satisfy Kawa's identifier rules.
   32:  * [lyn, 01/15/2013] Added do_then_return, eval_but_ignore, and nothing.
   33:  * [lyn, 12/27/2012] Made code generation of forRange and forEach consistent with parameter change.
   34:  */
   35:
   36: 'use strict';
   37:
   38: goog.provide('Blockly.Yail.control');
   39:
   40: Blockly.Yail['controls_if'] = function() {
   41:
   42:   var code = "";
   43:   for(var i=0;i<this.elseifCount_ + 1;i++){
   44:     var argument = Blockly.Yail.valueToCode(this, 'IF'+ i, Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
   45:     var branch = Blockly.Yail.statementToCode(this, 'DO'+ i) || Blockly.Yail.YAIL_FALSE;
   46:     if(i != 0) {
   47:       code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN;
   48:     }
   49:     code += Blockly.Yail.YAIL_IF + argument + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN
   50:       + branch + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   51:   }
   52:   if(this.elseCount_ == 1){
   53:     var branch = Blockly.Yail.statementToCode(this, 'ELSE') || Blockly.Yail.YAIL_FALSE;
   54:     code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN + branch + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   55:   }
   56:
   57:   for(var i=0;i<this.elseifCount_;i++){
   58:     code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   59:   }
   60:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   61:   return code;
   62: };
   63:
   64: // [lyn, 01/15/2013] Edited to make consistent with removal of "THEN-DO" and "ELSE-DO"
   65: Blockly.Yail['controls_choose'] = function() {
   66:   // Choose.
   67:   var test = Blockly.Yail.valueToCode(this, 'TEST', Blockly.Yail.ORDER_NONE)  || Blockly.Yail.YAIL_FALSE;
   68:   var thenReturn = Blockly.Yail.valueToCode(this, 'THENRETURN', Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
   69:   var elseReturn = Blockly.Yail.valueToCode(this, 'ELSERETURN', Blockly.Yail.ORDER_NONE)  || Blockly.Yail.YAIL_FALSE;
   70:   var code = Blockly.Yail.YAIL_IF + test
   71:              + Blockly.Yail.YAIL_SPACER +  thenReturn
   72:              + Blockly.Yail.YAIL_SPACER +  elseReturn
   73:              + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   74:   return [code,Blockly.Yail.ORDER_ATOMIC];
   75: };
   76:
   77: // [lyn, 12/27/2012]
   78: Blockly.Yail['controls_forEach'] = function() {
   79:   // For each loop.
   80:   var emptyListCode = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "make-yail-list" + Blockly.Yail.YAIL_SPACER;
   81:   emptyListCode += Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   82:
   83:   emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   84:   emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   85:   emptyListCode += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QUOTE + "make a list" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   86:
   87:
   88:   var loopIndexName = Blockly.Yail.YAIL_LOCAL_VAR_TAG + this.getFieldValue('VAR');
   89:   var listCode = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) || emptyListCode;
   90:   var bodyCode = Blockly.Yail.statementToCode(this, 'DO', Blockly.Yail.ORDER_NONE) ||  Blockly.Yail.YAIL_FALSE;
   91:   return Blockly.Yail.YAIL_FOREACH + loopIndexName + Blockly.Yail.YAIL_SPACER
   92:          + Blockly.Yail.YAIL_BEGIN + bodyCode + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER
   93:          + listCode + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   94: };
   95:
   96: // [lyn, 12/27/2012]
   97: Blockly.Yail['controls_forRange'] = function() {
   98:   // For range loop.
   99:   var loopIndexName = Blockly.Yail.YAIL_LOCAL_VAR_TAG + this.getFieldValue('VAR');
  100:   var startCode = Blockly.Yail.valueToCode(this, 'START', Blockly.Yail.ORDER_NONE) || 0;
  101:   var endCode = Blockly.Yail.valueToCode(this, 'END', Blockly.Yail.ORDER_NONE) || 0;
  102:   var stepCode = Blockly.Yail.valueToCode(this, 'STEP', Blockly.Yail.ORDER_NONE) || 0;
  103:   var bodyCode = Blockly.Yail.statementToCode(this, 'DO', Blockly.Yail.ORDER_NONE) |
```

```
| Blockly.Yail.YAIL_FALSE;
  104:   return Blockly.Yail.YAIL_FORRANGE + loopIndexName + Blockly.Yail.YAIL_SPACER
  105:          + Blockly.Yail.YAIL_BEGIN + bodyCode + Blockly.Yail.YAIL_CLOSE_COMBINATION
+ Blockly.Yail.YAIL_SPACER
  106:          + startCode + Blockly.Yail.YAIL_SPACER
  107:          + endCode + Blockly.Yail.YAIL_SPACER
  108:          + stepCode + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  109: };
  110:
  111: Blockly.Yail['for_lexical_variable_get'] = function() {
  112:   return Blockly.Yail.lexical_variable_get.call(this);
  113: }
  114:
  115: Blockly.Yail['controls_while'] = function() {
  116:   // While condition.
  117:   var test = Blockly.Yail.valueToCode(this, 'TEST', Blockly.Yail.ORDER_NONE) || Bloc
kly.Yail.YAIL_FALSE;
  118:   var toDo = Blockly.Yail.statementToCode(this, 'DO') || Blockly.Yail.YAIL_FALSE;
  119:   var code = Blockly.Yail.YAIL_WHILE + test + Blockly.Yail.YAIL_SPACER + Blockly.Yai
l.YAIL_BEGIN + toDo + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_CLOSE_COMBINA
TION;
  120:   return code;
  121: };
  122:
  123: // [lyn, 01/15/2013] Added
  124: Blockly.Yail['controls_do_then_return'] = function() {
  125:   var stm = Blockly.Yail.statementToCode(this, 'STM', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
  126:   var value = Blockly.Yail.valueToCode(this, 'VALUE', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
  127:   var code = Blockly.Yail.YAIL_BEGIN + stm + Blockly.Yail.YAIL_SPACER + value + Bloc
kly.Yail.YAIL_CLOSE_COMBINATION;
  128:   return [code, Blockly.Yail.ORDER_ATOMIC];
  129: };
  130:
  131:  // [lyn, 01/15/2013] Added
  132: // adding 'ignored' here is only for the printout in Do-It.  The value will be ignor
ed because the block shape
  133: // has no output
  134: Blockly.Yail['controls_eval_but_ignore'] = function() {
  135:   var toEval = Blockly.Yail.valueToCode(this, 'VALUE', Blockly.Yail.ORDER_NONE) || B
lockly.Yail.YAIL_FALSE;
  136:   var code = Blockly.Yail.YAIL_BEGIN + toEval + Blockly.Yail.YAIL_SPACER + '"ignored
"' + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  137:   return code;
  138: };
  139:
  140: // [lyn, 01/15/2013] Added
  141: Blockly.Yail['controls_nothing'] = function() {
  142:   return ['*the-null-value*', Blockly.Yail.ORDER_NONE];
  143: };
  144:
  145: Blockly.Yail['controls_openAnotherScreen'] = function() {
  146:   // Open another screen
  147:   var argument0 = Blockly.Yail.valueToCode(this, 'SCREEN', Blockly.Yail.ORDER_NONE)
|| null;
  148:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "open-another-screen" + Blockly
.Yail.YAIL_SPACER;
  149:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  150:   code = code + argument0 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  151:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  152:   code = code + "text" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  153:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "open another screen" + Blockly.Yai
l.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  154:   return code;
  155: };
  156:
  157: Blockly.Yail['controls_openAnotherScreenWithStartValue'] = function() {
  158:   // Open another screen with start value
  159:   var argument0 = Blockly.Yail.valueToCode(this, 'SCREENNAME', Blockly.Yail.ORDER_NO
NE) || null;
  160:   var argument1 = Blockly.Yail.valueToCode(this, 'STARTVALUE', Blockly.Yail.ORDER_NO
NE) || null;
  161:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "open-another-screen-with-start
-value" + Blockly.Yail.YAIL_SPACER;
  162:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  163:   code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL
_CLOSE_COMBINATION;
  164:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  165:   code = code + "text any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL
_SPACER;
  166:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "open another screen with start val
ue" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  167:   return code;
  168: };
  169:
  170: Blockly.Yail['controls_getStartValue'] = function() {
  171:   // Get start value
  172:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "get-start-value" + Blockly.Yai
l.YAIL_SPACER;
  173:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  174:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  175:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  176:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  177:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "get start value" + Blockly.Yail.YA
IL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  178:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  179: };
  180:
  181: Blockly.Yail['controls_closeScreen'] = function() {
  182:   // Close screen
  183:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "close-screen" + Blockly.Yail.Y
AIL_SPACER;
  184:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  185:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  186:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  187:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  188:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "close screen" + Blockly.Yail.YAIL_
DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  189:   return code;
  190: };
  191:
  192: Blockly.Yail['controls_closeScreenWithValue'] = function() {
  193:   // Close screen with value
  194:   var argument0 = Blockly.Yail.valueToCode(this, 'SCREEN', Blockly.Yail.ORDER_NONE)
```

```
|| null;
  195:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "close-screen-with-value" + Blo
ckly.Yail.YAIL_SPACER;
  196:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  197:   code = code + argument0 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  198:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  199:   code = code + "any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPAC
ER;
  200:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "close screen with value" + Blockly
.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  201:   return code;
  202: };
  203:
  204: Blockly.Yail['controls_closeApplication'] = function() {
  205:   // Close application
  206:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "close-application" + Blockly.Y
ail.YAIL_SPACER;
  207:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  208:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  209:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  210:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  211:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "close application" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  212:   return code;
  213: };
  214:
  215: Blockly.Yail['controls_getPlainStartText'] = function() {
  216:   // Get plain start text
  217:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "get-plain-start-text" + Blockl
y.Yail.YAIL_SPACER;
  218:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  219:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  220:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  221:   code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  222:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "get plain start text" + Blockly.Ya
il.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  223:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  224: };
  225:
  226: Blockly.Yail['controls_closeScreenWithPlainText'] = function() {
  227:   // Close screen with plain text
  228:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  229:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "close-screen-with-plain-text"
+ Blockly.Yail.YAIL_SPACER;
  230:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  231:   code = code + argument0 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  232:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  233:   code = code + "text" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  234:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "close screen with plain text" + Bl
ockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  235:   return code;
  236: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Logic blocks yail generators for Blockly, modified for MIT App Inve
ntor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.logic');
   13:
   14: Blockly.Yail['logic_boolean'] = function() {
   15:   // Boolean values true and false.
   16:   var code = (this.getFieldValue('BOOL') == 'TRUE') ? Blockly.Yail.YAIL_TRUE
   17:       : Blockly.Yail.YAIL_FALSE;
   18:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   19: };
   20:
   21: Blockly.Yail['logic_false'] = function() {
   22:   return Blockly.Yail.logic_boolean.call(this);
   23: }
   24:
   25: Blockly.Yail['logic_negate'] = function() {
   26:   // negate operation
   27:   var argument = Blockly.Yail
   28:       .valueToCode(this, 'BOOL', Blockly.Yail.ORDER_NONE)
   29:       || Blockly.Yail.YAIL_FALSE;
   30:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-not"
   31:       + Blockly.Yail.YAIL_SPACER;
   32:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   33:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   34:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   35:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   36:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "boolean"
   37:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   38:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "not"
   39:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   40:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   41: };
   42:
   43: Blockly.Yail['logic_operation'] = function() {
   44:   // The and, or logic operations
   45:   // TODO: (Andrew) Make these take multiple arguments.
   46:   var mode = this.getFieldValue('OP');
   47:   var tuple = Blockly.Yail.logic_operation.OPERATORS[mode];
   48:   var operator = tuple[0];
   49:   var order = tuple[1];
   50:   var argument0 = Blockly.Yail.valueToCode(this, 'A', order) || Blockly.Yail.YAIL_FA
LSE;
   51:   var argument1 = Blockly.Yail.valueToCode(this, 'B', order) || Blockly.Yail.YAIL_FA
LSE;
   52:   var code = Blockly.Yail.YAIL_OPEN_COMBINATION + operator
   53:       + Blockly.Yail.YAIL_SPACER + argument0 + Blockly.Yail.YAIL_SPACER
   54:       + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   55:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   56: };
   57:
   58: Blockly.Yail.logic_operation.OPERATORS = {
   59:   AND : [ 'and-delayed', Blockly.Yail.ORDER_NONE ],
   60:   OR : [ 'or-delayed', Blockly.Yail.ORDER_NONE ]
   61: };
   62:
   63: Blockly.Yail['logic_or'] = function() {
   64:   return Blockly.Yail.logic_operation.call(this);
   65: }
   66:
   67: Blockly.Yail['logic_compare'] = function() {
   68:   // Basic logic compare operators
   69:   // // TODO: (Hal) handle any type?
   70:   var argument0 = Blockly.Yail.valueToCode(this, 'A', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
   71:   var argument1 = Blockly.Yail.valueToCode(this, 'B', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
   72:   var yailCommand = (this.getFieldValue('OP') == "NEQ" ? 'yail-not-equal?' : "yail-e
qual?" );
   73:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + yailCommand
   74:       + Blockly.Yail.YAIL_SPACER;
   75:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   76:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   77:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   78:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   79:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   80:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "any" + Blockly.Yail.YAIL_SPACER
   81:       + "any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   82:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "="
   83:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   84:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   85: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Math blocks yail generators for Blockly, modified for MIT App Inven
tor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.math');
   13:
   14: Blockly.Yail['math_number'] = function() {
   15:   // Numeric value.
   16:   var code = window.parseFloat(this.getFieldValue('NUM'));
   17:   return [code, Blockly.Yail.ORDER_ATOMIC];
   18: };
   19:
   20: Blockly.Yail['math_compare'] = function() {
   21:   // Basic compare operators
   22:   var mode = this.getFieldValue('OP');
   23:   var prim = Blockly.Yail.math_compare.OPERATORS[mode];
   24:   var operator1 = prim[0];
   25:   var operator2 = prim[1];
   26:   var order = prim[2];
   27:   var argument0 = Blockly.Yail.valueToCode(this, 'A', order) || 0;
   28:   var argument1 = Blockly.Yail.valueToCode(this, 'B', order) || 0;
   29:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
   30:       + Blockly.Yail.YAIL_SPACER;
   31:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   32:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   33:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   34:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   35:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   36:       + Blockly.Yail.YAIL_OPEN_COMBINATION + (mode == "EQ" || mode == "NEQ" ? "any a
ny" : "number number" )
   37:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   38:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
   39:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   40:   return [code, Blockly.Yail.ORDER_ATOMIC];
   41: };
   42:
   43: Blockly.Yail.math_compare.OPERATORS = {
   44:   EQ: ['yail-equal?', '=', Blockly.Yail.ORDER_NONE],
   45:   NEQ: ['yail-not-equal?', 'not =', Blockly.Yail.ORDER_NONE],
   46:   LT: ['<', '<', Blockly.Yail.ORDER_NONE],
   47:   LTE: ['<=', '<=', Blockly.Yail.ORDER_NONE],
   48:   GT: ['>', '>', Blockly.Yail.ORDER_NONE],
   49:   GTE: ['>=', '>=', Blockly.Yail.ORDER_NONE]
   50: };
   51:
   52: Blockly.Yail['math_arithmetic'] = function(mode,block) {
   53:   // Basic arithmetic operators.
   54:   var tuple = Blockly.Yail.math_arithmetic.OPERATORS[mode];
   55:   var operator = tuple[0];
   56:   var order = tuple[1];
   57:   var argument0 = Blockly.Yail.valueToCode(block, 'A', order) || 0;
   58:   var argument1 = Blockly.Yail.valueToCode(block, 'B', order) || 0;
   59:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
   60:       + Blockly.Yail.YAIL_SPACER;
   61:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   62:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   63:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   64:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   65:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   66:       + Blockly.Yail.YAIL_OPEN_COMBINATION;
   67:   code += (mode == "EQ" || mode == "NEQ" ? "any any" : "number number" )
   68:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   69:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
   70:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   71:   return [code, Blockly.Yail.ORDER_ATOMIC];
   72: };
   73:
   74: Blockly.Yail['math_subtract'] = function() {
   75:   return Blockly.Yail.math_arithmetic("MINUS",this);
   76: };
   77:
   78: Blockly.Yail['math_division'] = function() {
   79:   return Blockly.Yail.math_arithmetic("DIVIDE",this);
   80: };
   81:
   82: Blockly.Yail['math_power'] = function() {
   83:   return Blockly.Yail.math_arithmetic("POWER",this);
   84: };
   85:
   86: Blockly.Yail['math_add'] = function() {
   87:   return Blockly.Yail.math_arithmetic_list("ADD",this);
   88: };
   89:
   90: Blockly.Yail['math_multiply'] = function() {
   91:   return Blockly.Yail.math_arithmetic_list("MULTIPLY",this);
   92: };
   93:
   94: Blockly.Yail['math_arithmetic_list'] = function(mode,block) {
   95:   // Basic arithmetic operators.
   96:   //var mode = this.getFieldValue('OP');
   97:   var tuple = Blockly.Yail.math_arithmetic.OPERATORS[mode];
   98:   var operator = tuple[0];
   99:   var order = tuple[1];
  100:
  101:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
  102:       + Blockly.Yail.YAIL_SPACER;
  103:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  104:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
  105:   for(var i=0;i<block.itemCount_;i++) {
  106:     var argument = Blockly.Yail.valueToCode(block, 'NUM' + i, order) || 0;
  107:     code += argument + Blockly.Yail.YAIL_SPACER;
  108:   }
  109:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
  110:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  111:       + Blockly.Yail.YAIL_OPEN_COMBINATION;
  112:   for(var i=0;i<block.itemCount_;i++) {
  113:     code += "number" + Blockly.Yail.YAIL_SPACER;
  114:   }
  115:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  116:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
  117:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  118:   return [code, Blockly.Yail.ORDER_ATOMIC];
  119: };
  120:
  121: Blockly.Yail.math_arithmetic.OPERATORS = {
  122:   ADD: ['+', Blockly.Yail.ORDER_NONE],
```

```
  MINUS: ['-', Blockly.Yail.ORDER_NONE],
  MULTIPLY: ['*', Blockly.Yail.ORDER_NONE],
  DIVIDE: ['/', Blockly.Yail.ORDER_NONE],
  POWER: ['expt', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_single'] = function() {
  // Basic arithmetic operators.
  var mode = this.getFieldValue('OP');
  var tuple = Blockly.Yail.math_single.OPERATORS[mode];
  var operator1 = tuple[0];
  var operator2 = tuple[1];
  var order = tuple[2];
  var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 1;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail.math_single.OPERATORS = {
  ROOT: ['sqrt', 'sqrt', Blockly.Yail.ORDER_NONE],
  ABS: ['abs', 'abs', Blockly.Yail.ORDER_NONE],
  NEG: ['-', 'negate', Blockly.Yail.ORDER_NONE],
  LN: ['log', 'log', Blockly.Yail.ORDER_NONE],
  EXP: ['exp', 'exp', Blockly.Yail.ORDER_NONE],
  ROUND: ['yail-round', 'round', Blockly.Yail.ORDER_NONE],
  CEILING: ['yail-ceiling', 'ceiling', Blockly.Yail.ORDER_NONE],
  FLOOR: ['yail-floor', 'floor', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_abs'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_neg'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_round'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_ceiling'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_floor'] = function() {
  return Blockly.Yail.math_single.call(this);
};


Blockly.Yail['math_random_int'] = function() {
  // Random integer between [X] and [Y].
  var argument0 = Blockly.Yail.valueToCode(this, 'FROM',
    Blockly.Yail.ORDER_NONE) || 0;
  var argument1 = Blockly.Yail.valueToCode(this, 'TO',
    Blockly.Yail.ORDER_NONE) || 0;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-integer"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random integer"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['math_random_float'] = function() {
  // Random fraction between 0 and 1.
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-fraction"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR +
  Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE +
  Blockly.Yail.YAIL_OPEN_COMBINATION;
  code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random fraction"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['math_random_set_seed'] = function() {
  // Basic is_a_number.
  var argument = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || 0
;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-set-seed"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random set seed"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return code;
};

Blockly.Yail['math_on_list'] = function() {
  // Min and Max operators.
  var mode = this.getFieldValue('OP');
  var tuple = Blockly.Yail.math_on_list.OPERATORS[mode];
  var operator = tuple[0];
  var order = tuple[1];
  var args = "";
  var typeString = "";
  for(var i=0;i<this.itemCount_;i++) {
    args += (Blockly.Yail.valueToCode(this, 'NUM' + i, order) || 0) + Blockly.Yail.Y
AIL_SPACER;
    typeString += "number" + Blockly.Yail.YAIL_SPACER;
  }
```

```
243:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
244:       + Blockly.Yail.YAIL_SPACER;
245:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
246:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
247:       + args//argument0 + Blockly.Yail.YAIL_SPACER + argument1
248:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
249:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
250:       + Blockly.Yail.YAIL_OPEN_COMBINATION + typeString
251:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
252:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
253:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
254:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
255: };
256:
257: Blockly.Yail.math_on_list.OPERATORS = {
258:   MIN: ['min', Blockly.Yail.ORDER_NONE],
259:   MAX: ['max', Blockly.Yail.ORDER_NONE]
260: };
261:
262: Blockly.Yail['math_divide'] = function() {
263:   // divide operators.
264:   var mode = this.getFieldValue('OP');
265:   var tuple = Blockly.Yail.math_divide.OPERATORS[mode];
266:   var operator = tuple[0];
267:   var order = tuple[1];
268:   var argument0 = Blockly.Yail.valueToCode(this, 'DIVIDEND', order) || 0;
269:   var argument1 = Blockly.Yail.valueToCode(this, 'DIVISOR', order) || 1;
270:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
271:       + Blockly.Yail.YAIL_SPACER;
272:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
273:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
274:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
275:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
276:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
277:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
278:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
279:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
280:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
281:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
282: };
283:
284: Blockly.Yail.math_divide.OPERATORS = {
285:   MODULO: ['modulo', Blockly.Yail.ORDER_NONE],
286:   REMAINDER: ['remainder', Blockly.Yail.ORDER_NONE],
287:   QUOTIENT: ['quotient', Blockly.Yail.ORDER_NONE]
288: };
289:
290: Blockly.Yail['math_trig'] = function() {
291:   // Basic trig operators.
292:   var mode = this.getFieldValue('OP');
293:   var tuple = Blockly.Yail.math_trig.OPERATORS[mode];
294:   var operator1 = tuple[1];
295:   var operator2 = tuple[0];
296:   var order = tuple[2];
297:   var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 0;
298:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
299:       + Blockly.Yail.YAIL_SPACER;
300:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
301:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
302:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
303:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
304:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
305:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
306:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
307:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
308:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
309: };
310:
311: Blockly.Yail.math_trig.OPERATORS = {
312:   SIN: ['sin', 'sin-degrees', Blockly.Yail.ORDER_NONE],
313:   COS: ['cos', 'cos-degrees', Blockly.Yail.ORDER_NONE],
314:   TAN: ['tan', 'tan-degrees', Blockly.Yail.ORDER_NONE],
315:   ASIN: ['asin', 'asin-degrees', Blockly.Yail.ORDER_NONE],
316:   ACOS: ['acos', 'acos-degrees', Blockly.Yail.ORDER_NONE],
317:   ATAN: ['atan', 'atan-degrees', Blockly.Yail.ORDER_NONE]
318: };
319:
320: Blockly.Yail['math_cos'] = function() {
321:   return Blockly.Yail.math_trig.call(this);
322: };
323:
324: Blockly.Yail['math_tan'] = function() {
325:   return Blockly.Yail.math_trig.call(this);
326: };
327:
328: Blockly.Yail['math_atan2'] = function() {
329:   // atan2 operators.
330:   var argument0 = Blockly.Yail.valueToCode(this, 'Y', Blockly.Yail.ORDER_NONE) || 1;
331:   var argument1 = Blockly.Yail.valueToCode(this, 'X', Blockly.Yail.ORDER_NONE) || 1;
332:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "atan2-degrees"
333:       + Blockly.Yail.YAIL_SPACER;
334:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
335:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
336:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
337:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
338:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
339:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
340:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
341:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "atan2"
342:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
343:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
344: };
345:
346: Blockly.Yail['math_convert_angles'] = function() {
347:   // Basic arithmetic operators.
348:   var mode = this.getFieldValue('OP');
349:   var tuple = Blockly.Yail.math_convert_angles.OPERATORS[mode];
350:   var operator1 = tuple[0];
351:   var operator2 = tuple[1];
352:   var order = tuple[2];
353:   var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 0;
354:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
355:       + Blockly.Yail.YAIL_SPACER;
356:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
357:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
358:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
359:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
360:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
361:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
362:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
363:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
364:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
365: };
366:
```

```
  367: Blockly.Yail.math_convert_angles.OPERATORS = {
  368:   RADIANS_TO_DEGREES: ['radians->degrees', 'convert radians to degrees', Blockly.Yai
l.ORDER_NONE],
  369:   DEGREES_TO_RADIANS: ['degrees->radians', 'convert degrees to radians', Blockly.Yai
l.ORDER_NONE]
  370: };
  371:
  372: Blockly.Yail['math_format_as_decimal'] = function() {
  373:   // format_as_decimal.
  374:   var argument0 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) ||
0;
  375:   var argument1 = Blockly.Yail.valueToCode(this, 'PLACES', Blockly.Yail.ORDER_NONE)
|| 0;
  376:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "format-as-decimal"
  377:       + Blockly.Yail.YAIL_SPACER;
  378:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  379:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
  380:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
  381:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  382:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  383:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
  384:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  385:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "format as decimal"
  386:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  387:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  388: };
  389:
  390: Blockly.Yail['math_is_a_number'] = function() {
  391:   // Basic is_a_number.
  392:   var argument = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || B
lockly.Yail.YAIL_FALSE;
  393:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "is-number?"
  394:       + Blockly.Yail.YAIL_SPACER;
  395:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  396:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
  397:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  398:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  399:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "any"
  400:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  401:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is a number?"
  402:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  403:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  404: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2012 Massachusetts Institute of Technology. All rights reserved.

/**
 * @license
 * @fileoverview Color blocks yail generators for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Yail.text');

Blockly.Yail['text'] = function() {
  // Text value.
  var code = Blockly.Yail.quote_(this.getFieldValue('TEXT'));
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail['text_join'] = function() {
  // Create a string made up of elements of any type..
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-append"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;

  for(var i=0;i<this.itemCount_;i++) {
    var argument = Blockly.Yail.valueToCode(this, 'ADD' + i, Blockly.Yail.ORDER_NONE) || "\"\"";
    code += argument + Blockly.Yail.YAIL_SPACER;
  }
  code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION;
  for(var i=0;i<this.itemCount_;i++) {
    code += "text" + Blockly.Yail.YAIL_SPACER;
  }
  code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "join"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['text_length'] = function() {
  // // String length
  var argument = Blockly.Yail.valueToCode(this, 'VALUE', Blockly.Yail.ORDER_NONE) || "\"\"";
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-length"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "text"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "length"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['text_isEmpty'] = function() {
  // Is the string null?
  var argument = Blockly.Yail.valueToCode(this, 'VALUE', Blockly.Yail.ORDER_NONE) || "\"\"";
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-empty?"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "text"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is text empty?"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['text_compare'] = function() {
  // Basic compare operators
  var mode = this.getFieldValue('OP');
  var prim = Blockly.Yail.text_compare.OPERATORS[mode];
  var operator1 = prim[0];
  var operator2 = prim[1];
  var order = prim[2];
  var argument0 = Blockly.Yail.valueToCode(this, 'TEXT1', order) || "\"\"";
  var argument1 = Blockly.Yail.valueToCode(this, 'TEXT2', order) || "\"\"";
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "text text"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['text_compare'].OPERATORS = {
  LT: ['string<?', 'text<', Blockly.Yail.ORDER_NONE],
  GT: ['string>?', 'text>', Blockly.Yail.ORDER_NONE],
  EQUAL: ['string=?', 'text=', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['text_trim'] = function() {
  // String trim
  var argument = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) || "\"\"";
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-trim"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "text"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "trim"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

```

```
120: Blockly.Yail['text_changeCase'] = function() {
121:   // String change case.
122:   var mode = this.getFieldValue('OP');
123:   var tuple = Blockly.Yail.text_changeCase.OPERATORS[mode];
124:   var operator1 = tuple[0];
125:   var operator2 = tuple[1];
126:   var order = tuple[2];
127:   var argument = Blockly.Yail.valueToCode(this, 'TEXT', order) || "\"\"";
128:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
129:       + Blockly.Yail.YAIL_SPACER;
130:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
131:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
132:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
133:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
134:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text"
135:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
136:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
137:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
138:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
139: };
140:
141: Blockly.Yail['text_changeCase'].OPERATORS = {
142:   UPCASE: ['string-to-upper-case', 'upcase', Blockly.Yail.ORDER_NONE],
143:   DOWNCASE: ['string-to-lower-case', 'downcase', Blockly.Yail.ORDER_NONE]
144: };
145:
146: Blockly.Yail.text_starts_at = function() {
147:   // String starts at
148:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
149:   var argument1 = Blockly.Yail.valueToCode(this, 'PIECE', Blockly.Yail.ORDER_NONE) |
| "\"\"";
150:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-starts-at"
151:       + Blockly.Yail.YAIL_SPACER;
152:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
153:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
154:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
155:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
156:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
157:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text text"
158:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
159:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "starts at"
160:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
161:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
162: };
163:
164: Blockly.Yail['text_contains'] = function() {
165:   // String contains.
166:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
167:   var argument1 = Blockly.Yail.valueToCode(this, 'PIECE', Blockly.Yail.ORDER_NONE) |
| "\"\"";
168:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-contains"
169:       + Blockly.Yail.YAIL_SPACER;
170:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
171:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
172:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
173:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
174:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
175:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text text"
176:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
177:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "contains"
178:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
179:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
180: };
181:
182: Blockly.Yail['text_split'] = function() {
183:   // String split operations.
184:   // Note that the type of arg2 might be text or list, depending on the dropdown sel
ection
185:   var mode = this.getFieldValue('OP');
186:   var tuple = Blockly.Yail.text_split.OPERATORS[mode];
187:   var operator1 = tuple[0];
188:   var operator2 = tuple[1];
189:   var order = tuple[2];
190:   var arg2Type = tuple[3];
191:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
192:   var argument1 = Blockly.Yail.valueToCode(this, 'AT', Blockly.Yail.ORDER_NONE) || 1
;
193:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
194:       + Blockly.Yail.YAIL_SPACER;
195:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
196:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
197:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
198:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
199:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
200:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text" +  Blockly.Yail.YAIL_SPACER + ar
g2Type
201:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
202:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
203:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
204:   return [ code, order ];
205: };
206:
207: Blockly.Yail['text_split'].OPERATORS = {
208:   SPLITATFIRST : [ 'string-split-at-first', 'split at first',
209:       Blockly.Yail.ORDER_ATOMIC, 'text' ],
210:   SPLITATFIRSTOFANY : [ 'string-split-at-first-of-any',
211:       'split at first of any', Blockly.Yail.ORDER_ATOMIC, 'list' ],
212:   SPLIT : [ 'string-split', 'split', Blockly.Yail.ORDER_ATOMIC, 'text' ],
213:   SPLITATANY : [ 'string-split-at-any', 'split at any', Blockly.Yail.ORDER_ATOMIC, '
list' ]
214: };
215:
216: Blockly.Yail['text_split_at_spaces'] = function() {
217:   // split at spaces
218:   var argument = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
"\"\"";
219:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-split-at-spaces"
220:       + Blockly.Yail.YAIL_SPACER;
221:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
222:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
223:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
224:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
225:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text"
226:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
227:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "split at spaces"
228:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
229:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
230: };
231:
232: Blockly.Yail['text_segment'] = function() {
233:   // Create string segment
```

```
  234:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
  235:   var argument1 = Blockly.Yail.valueToCode(this, 'START', Blockly.Yail.ORDER_NONE) |
| 1;
  236:   var argument2 = Blockly.Yail.valueToCode(this, 'LENGTH', Blockly.Yail.ORDER_NONE)
|| 1;
  237:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-substring"
  238:       + Blockly.Yail.YAIL_SPACER;
  239:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  240:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
  241:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
  242:       + Blockly.Yail.YAIL_SPACER + argument2
  243:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  244:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  245:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text number number"
  246:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  247:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "segment"
  248:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  249:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  250: };
  251:
  252: Blockly.Yail['text_replace_all'] = function() {
  253:   // String replace with segment
  254:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
  255:   var argument1 = Blockly.Yail.valueToCode(this, 'SEGMENT', Blockly.Yail.ORDER_NONE)
 || "\"\"";
  256:   var argument2 = Blockly.Yail.valueToCode(this, 'REPLACEMENT', Blockly.Yail.ORDER_N
ONE) || "\"\"";
  257:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "string-replace-all"
  258:       + Blockly.Yail.YAIL_SPACER;
  259:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  260:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
  261:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
  262:       + Blockly.Yail.YAIL_SPACER + argument2
  263:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  264:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  265:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text text text"
  266:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  267:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "replace all"
  268:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  269:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  270: };
  271:
  272: Blockly.Yail['obsufcated_text'] = function() {
  273:   // Deobsfucate the TEXT input argument
  274:   var setupObsfucation = function(input, confounder) {
  275:     // The algorithm below is also implemented in scheme in runtime.scm
  276:     // If you change it here, you have to change it there!
  277:     // Note: This algorithm is like xor, if applied to its output
  278:     // it regenerates it input.
  279:     var acc = [];
  280:     // First make sure the confounder is long enough...
  281:     while (confounder.length < input.length) {
  282:       confounder += confounder;
  283:     }
  284:     for (var i = 0; i < input.length; i++) {
  285:       var c = (input.charCodeAt(i) ^ confounder.charCodeAt(i)) & 0xFF;
  286:       var b = (c ^ input.length - i) & 0xFF;
  287:       var b2 = ((c >> 8) ^ i) & 0xFF;
  288:       acc.push(String.fromCharCode((b2 << 8 | b) & 0xFF));
  289:     }
  290:     return acc.join('');
  291:   }
  292:   var input = this.getFieldValue('TEXT');
  293:   var argument = Blockly.Yail.quote_(setupObsfucation(input, this.confounder));
  294:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "text-deobsfucate"
  295:       + Blockly.Yail.YAIL_SPACER;
  296:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
  297:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
  298:       + argument + Blockly.Yail.YAIL_SPACER
  299:       + Blockly.Yail.quote_(this.confounder) + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  300:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
  301:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "text text"
  302:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  303:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "deobsfucate text"
  304:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  305:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  306: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Lists blocks yail generators for Blockly, modified for MIT App Inve
ntor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.lists');
   13:
   14: Blockly.Yail.emptyListCode = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "make-yail-list
" + Blockly.Yail.YAIL_SPACER;
   15: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL
_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   16: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAI
L_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   17: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   18: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QU
OTE + "make a list" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   19:
   20:
   21: Blockly.Yail['lists_create_with'] = function() {
   22:
   23:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "make-yail-list" + Blockly.Yail
.YAIL_SPACER;
   24:   code += Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR +
Blockly.Yail.YAIL_SPACER;
   25:   var itemsAdded = 0;
   26:   for(var i=0;i<this.itemCount_;i++) {
   27:     var argument = Blockly.Yail.valueToCode(this, 'ADD' + i, Blockly.Yail.ORDER_NONE
) || null;
   28:     if(argument != null){
   29:       code += argument + Blockly.Yail.YAIL_SPACER;
   30:       itemsAdded++;
   31:     }
   32:   }
   33:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Y
ail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   34:   for(var i=0;i<itemsAdded;i++) {
   35:     code += "any" + Blockly.Yail.YAIL_SPACER;
   36:   }
   37:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   38:   code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QUOTE + "make a list"
+ Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   39:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   40:
   41: };
   42:
   43: Blockly.Yail['lists_select_item'] = function() {
   44:   // Select from list an item.
   45:
   46:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
   47:   var argument1 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) ||
1;
   48:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-get-item" + Blockly.
Yail.YAIL_SPACER;
   49:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
   50:   code = code + argument0;
   51:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
   52:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
   53:   code = code + "list number" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.Y
AIL_SPACER;
   54:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "select list item" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   55:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   56: };
   57:
   58: Blockly.Yail['lists_replace_item'] = function() {
   59:   // Replace Item in list.
   60:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
   61:   var argument1 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) ||
1;
   62:   var argument2 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
   63:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-set-item!" + Blockly
.Yail.YAIL_SPACER;
   64:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
   65:   code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   66:   code = code + Blockly.Yail.YAIL_SPACER + argument2 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
   67:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
   68:   code = code + "list number any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Ya
il.YAIL_SPACER;
   69:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "replace list item" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   70:   return code;
   71: };
   72:
   73: Blockly.Yail['lists_remove_item'] = function() {
   74:   // Remove Item in list.
   75:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
   76:   var argument1 = Blockly.Yail.valueToCode(this, 'INDEX', Blockly.Yail.ORDER_NONE) |
| 1;
   77:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-remove-item!" + Bloc
kly.Yail.YAIL_SPACER;
   78:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
   79:   code = code + argument0;
   80:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
   81:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
   82:   code = code + "list number" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.Y
AIL_SPACER;
   83:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "remove list item" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   84:   return code;
   85: };
   86:
   87: Blockly.Yail['lists_insert_item'] = function() {
   88:   // Insert Item in list.
   89:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
```

```
 Blockly.Yail.emptyListCode;
   90:   var argument1 = Blockly.Yail.valueToCode(this, 'INDEX', Blockly.Yail.ORDER_NONE) |
| 1;
   91:   var argument2 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
   92:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-insert-item!" + Bloc
kly.Yail.YAIL_SPACER;
   93:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
   94:   code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1;
   95:   code = code + Blockly.Yail.YAIL_SPACER + argument2 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
   96:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
   97:   code = code + "list number any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Ya
il.YAIL_SPACER;
   98:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "insert list item" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   99:   return code;
  100: };
  101:
  102: Blockly.Yail['lists_length'] = function() {
  103:   // Length of list.
  104:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  105:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-length" + Blockly.Ya
il.YAIL_SPACER;
  106:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  107:   code = code + argument0;
  108:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  109:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  110:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  111:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "length of list" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  112:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  113: };
  114:
  115: Blockly.Yail['lists_append_list'] = function() {
  116:   // Append to list.
  117:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST0', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.emptyListCode;
  118:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST1', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.emptyListCode;
  119:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-append!" + Blockly.Y
ail.YAIL_SPACER;
  120:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  121:   code = code + argument0;
  122:   code = code + Blockly.Yail.YAIL_SPACER;
  123:   code = code + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  124:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  125:   code = code + "list list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAI
L_SPACER;
  126:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "append to list" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  127:   return code;
  128: };
  129:
  130: Blockly.Yail['lists_add_items'] = function() {
  131:   // Add items to list.
  132:   // TODO: (Andrew) Make this handle multiple items.
  133:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  134:   var argument1 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 1;
  135:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-add-to-list!" + Bloc
kly.Yail.YAIL_SPACER;
  136:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  137:   code = code + argument0 + Blockly.Yail.YAIL_SPACER;
  138:
  139:   for(var i=0;i<this.itemCount_;i++) {
  140:     var argument = Blockly.Yail.valueToCode(this, 'ITEM' + i, Blockly.Yail.ORDER_NON
E) || Blockly.Yail.YAIL_FALSE;
  141:     code += argument + Blockly.Yail.YAIL_SPACER;
  142:   }
  143:
  144:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
  145:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  146:   code = code + "list ";
  147:   for(var i=0;i<this.itemCount_;i++) {
  148:     code += "any" + Blockly.Yail.YAIL_SPACER;
  149:   }
  150:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  151:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "add items to list" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  152:   return code;
  153: };
  154:
  155: Blockly.Yail['lists_is_in'] = function() {
  156:   // Is in list?.
  157:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  158:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  159:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-member?" + Blockly.Y
ail.YAIL_SPACER;
  160:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  161:   code = code + argument0;
  162:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
  163:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  164:   code = code + "any list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL
_SPACER;
  165:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is in list?" + Blockly.Yail.YAIL_D
OUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  166:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  167: };
  168:
  169: Blockly.Yail['lists_position_in'] = function() {
  170:   // Postion of item in list.
  171:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 1;
  172:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  173:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-index" + Blockly.Yai
l.YAIL_SPACER;
```

```
  174:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  175:   code = code + argument0;
  176:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
  177:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  178:   code = code + "any list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL
_SPACER;
  179:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "position in list" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  180:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  181: };
  182:
  183: Blockly.Yail['lists_pick_random_item'] = function() {
  184:   // Pick random item
  185:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  186:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-pick-random" + Block
ly.Yail.YAIL_SPACER;
  187:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  188:   code = code + argument0;
  189:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  190:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  191:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  192:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "pick random item" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  193:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  194: };
  195:
  196: Blockly.Yail['lists_is_empty'] = function() {
  197:   // Is the list empty?.
  198:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  199:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-empty?" + Blockly.Ya
il.YAIL_SPACER;
  200:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  201:   code = code + argument0;
  202:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  203:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  204:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  205:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is list empty?" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  206:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  207: };
  208:
  209: Blockly.Yail['lists_copy'] = function() {
  210:   // Make a copy of list.
  211:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  212:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-copy" + Blockly.Yail
.YAIL_SPACER;
  213:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  214:   code = code + argument0;
  215:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  216:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  217:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  218:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "copy list" + Blockly.Yail.YAIL_DOU
BLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  219:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  220: };
  221:
  222: Blockly.Yail['lists_is_list'] = function() {
  223:   // Create an empty list.
  224:   // TODO:(Andrew) test whether thing is var or text or number etc...
  225:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  226:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list?" + Blockly.Yail.YAI
L_SPACER;
  227:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  228:   code = code + argument0;
  229:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  230:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  231:   code = code + "any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPAC
ER;
  232:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is a list?" + Blockly.Yail.YAIL_DO
UBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  233:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  234: };
  235:
  236: Blockly.Yail['lists_to_csv_row'] = function() {
  237:   // Make a csv row from list.
  238:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  239:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-to-csv-row" + Blockl
y.Yail.YAIL_SPACER;
  240:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  241:   code = code + argument0;
  242:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  243:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  244:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  245:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "list to csv row" + Blockly.Yail.YA
IL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  246:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  247: };
  248:
  249: Blockly.Yail['lists_to_csv_table'] = function() {
  250:   // Make a csv table from list
  251:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  252:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-to-csv-table" + Bloc
kly.Yail.YAIL_SPACER;
  253:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  254:   code = code + argument0;
  255:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  256:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  257:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
```

```
  258:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "list to csv table" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  259:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  260: };
  261:
  262: Blockly.Yail['lists_from_csv_row'] = function() {
  263:   // Make list from csv row.
  264:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
  265:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-from-csv-row" + Bloc
kly.Yail.YAIL_SPACER;
  266:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  267:   code = code + argument0;
  268:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  269:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  270:   code = code + "text" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  271:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "list from csv row" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  272:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  273: };
  274:
  275: Blockly.Yail['lists_from_csv_table'] = function() {
  276:   // Make list from csv table.
  277:   var argument0 = Blockly.Yail.valueToCode(this, 'TEXT', Blockly.Yail.ORDER_NONE) ||
 "\"\"";
  278:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-from-csv-table" + Bl
ockly.Yail.YAIL_SPACER;
  279:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  280:   code = code + argument0;
  281:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  282:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  283:   code = code + "text" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  284:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "list from csv table" + Blockly.Yai
l.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  285:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  286: };
  287:
  288:  Blockly.Yail['lists_lookup_in_pairs'] = function() {
  289:    // Lookup in pairs in list of lists (key, value).
  290:    var argument0 = Blockly.Yail.valueToCode(this, 'KEY', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  291:    var argument1 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.emptyListCode;
  292:    var argument2 = Blockly.Yail.valueToCode(this, 'NOTFOUND', Blockly.Yail.ORDER_NON
E) || Blockly.Yail.YAIL_NULL;
  293:    var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-alist-lookup" + Blockly.
Yail.YAIL_SPACER;
  294:    code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRU
CTOR + Blockly.Yail.YAIL_SPACER;
  295:    code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAI
L_SPACER + argument2 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  296:    code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.Y
AIL_OPEN_COMBINATION;
  297:    code = code + "any list any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail
.YAIL_SPACER;
  298:    code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QUOTE + "lookup
 in pairs" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  299:    return [ code, Blockly.Yail.ORDER_ATOMIC ];
  300:  };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Procedure yail generators for Blockly, modified for MIT App Invento
r.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.procedures');
   13:
   14: /**
   15:  * Lyn's History:
   16:  * [lyn, 10/29/13] Fixed bug in handling parameters of zero-arg procedures.
   17:  * [lyn, 10/27/13] Modified procedure names to begin with YAIL_PROC_TAG (currently '
p$')
   18:  *     and parameters to begin with YAIL_LOCAL_VAR_TAG (currently '$').
   19:  *     At least on Kawa-legal first character is necessary to ensure AI identifiers
   20:  *     satisfy Kawa's identifier rules. And the procedure 'p$' tag is necessary to
   21:  *     distinguish procedures from globals (which use the 'g$' tag).
   22:  * [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN
)
   23:  */
   24:
   25: Blockly.Yail.YAIL_PROC_TAG = 'p$'; // See notes on this in generators/yail/variables
.js
   26:
   27: // Generator code for procedure call with return
   28: // [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN
)
   29: Blockly.Yail['procedures_defreturn'] = function() {
   30:   var argPrefix = Blockly.Yail.YAIL_LOCAL_VAR_TAG
   31:                   + (Blockly.usePrefixInYail && this.arguments_.length != 0 ? "param
_" : "");
   32:   var args = this.arguments_.map(function (arg) {return argPrefix + arg;}).join(' ')
;
   33:   var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('NAME');
   34:   var returnVal = Blockly.Yail.valueToCode(this, 'RETURN', Blockly.Yail.ORDER_NONE)
|| Blockly.Yail.YAIL_FALSE;
   35:   var code = Blockly.Yail.YAIL_DEFINE + Blockly.Yail.YAIL_OPEN_COMBINATION + procNam
e
   36:       + Blockly.Yail.YAIL_SPACER + args + Blockly.Yail.YAIL_CLOSE_COMBINATION
   37:       + Blockly.Yail.YAIL_SPACER + returnVal + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   38:   return code;
   39: };
   40:
   41: // Generator code for procedure call with return
   42: Blockly.Yail['procedures_defnoreturn'] = function() {
   43:   var argPrefix = Blockly.Yail.YAIL_LOCAL_VAR_TAG
   44:                   + (Blockly.usePrefixInYail && this.arguments_.length != 0 ? "param
_" : "");
   45:   var args = this.arguments_.map(function (arg) {return argPrefix + arg;}).join(' ')
;
   46:   var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('NAME');
   47:   var body = Blockly.Yail.statementToCode(this, 'STACK', Blockly.Yail.ORDER_NONE)  |
| Blockly.Yail.YAIL_FALSE;
   48:   var code = Blockly.Yail.YAIL_DEFINE + Blockly.Yail.YAIL_OPEN_COMBINATION + procNam
e
   49:       + Blockly.Yail.YAIL_SPACER + args + Blockly.Yail.YAIL_CLOSE_COMBINATION + body
   50:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   51:   return code;
   52: };
   53:
   54: Blockly.Yail['procedure_lexical_variable_get'] = function() {
   55:   return Blockly.Yail.lexical_variable_get.call(this);
   56: }
   57:
   58: //call the do return in control category
   59: Blockly.Yail['procedures_do_then_return'] = function() {
   60:   return Blockly.Yail.controls_do_then_return.call(this);
   61: }
   62:
   63: // Generator code for procedure call with return
   64: Blockly.Yail['procedures_callnoreturn'] = function() {
   65:   var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
   66:   var argCode = [];
   67:   for ( var x = 0;this.getInput("ARG" + x); x++) {
   68:     argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE)
|| Blockly.Yail.YAIL_FALSE;
   69:   }
   70:   var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + p
rocName
   71:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.joi
n(' ')
   72:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   73:   return code;
   74: };
   75:
   76: // Generator code for procedure call with return
   77: Blockly.Yail['procedures_callreturn'] = function() {
   78:   var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
   79:   var argCode = [];
   80:   for ( var x = 0; this.getInput("ARG" + x); x++) {
   81:     argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE)
|| Blockly.Yail.YAIL_FALSE;
   82:   }
   83:   var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + p
rocName
   84:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.joi
n(' ')
   85:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   86:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   87: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2009-2011 Google, All Rights reserved
    3: // Copyright 2011-2012 MIT, All rights reserved
    4: // Released under the Apache License, Version 2.0
    5: // http://www.apache.org/licenses/LICENSE-2.0
    6:
    7: package com.google.appinventor.client.editor.simple.components;
    8:
    9: import static com.google.appinventor.client.Ode.MESSAGES;
   10:
   11: import java.util.HashMap;
   12: import java.util.HashSet;
   13: import java.util.Map;
   14:
   15: import com.google.appinventor.client.editor.simple.SimpleEditor;
   16: import com.google.appinventor.client.editor.simple.components.utils.PropertiesUtil;
   17: import com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidLeng
thPropertyEditor;
   18: import com.google.appinventor.client.editor.youngandroid.properties.YoungAndroidVert
icalAlignmentChoicePropertyEditor;
   19: import com.google.appinventor.client.output.OdeLog;
   20: import com.google.appinventor.client.properties.BadPropertyEditorException;
   21: import com.google.appinventor.shared.settings.SettingsConstants;
   22: import com.google.gwt.dom.client.DivElement;
   23: import com.google.gwt.dom.client.Document;
   24: import com.google.gwt.dom.client.Element;
   25: import com.google.gwt.dom.client.Style;
   26: import com.google.gwt.user.client.ui.AbsolutePanel;
   27: import com.google.gwt.user.client.ui.Composite;
   28: import com.google.gwt.user.client.ui.DockPanel;
   29: import com.google.gwt.user.client.ui.HorizontalPanel;
   30: import com.google.gwt.user.client.ui.Image;
   31: import com.google.gwt.user.client.ui.Label;
   32: import com.google.gwt.user.client.ui.ScrollPanel;
   33: import com.google.gwt.user.client.ui.TreeItem;
   34:
   35: /**
   36:  * Mock Form component.
   37:  *
   38:  */
   39: public final class MockForm extends MockContainer {
   40:
   41:   /**
   42:    * Component type name.
   43:    */
   44:   public static final String TYPE = "Form";
   45:
   46:   private static final String VISIBLE_TYPE = "Screen";
   47:
   48:   // TODO(lizlooney) 320x480 is the resolution of the G1. Do we want to change this
to the
   49:   // resolution of the Nexus One?
   50:   private static final int PORTRAIT_WIDTH = 160;
   51:   private static final int PORTRAIT_HEIGHT = 160;
   52:   private static final int LANDSCAPE_WIDTH = PORTRAIT_WIDTH;
   53:   private static final int LANDSCAPE_HEIGHT = PORTRAIT_HEIGHT;
   54:
   55:   // Property names
   56:   private static final String PROPERTY_NAME_TITLE = "Title";
   57:   private static final String PROPERTY_NAME_SCREEN_ORIENTATION = "ScreenOrientation"
;
   58:   private static final String PROPERTY_NAME_SCROLLABLE = "Scrollable";
   59:   private static final String PROPERTY_NAME_ICON = "Icon";
   60:   private static final String PROPERTY_NAME_VCODE = "VersionCode";
   61:   private static final String PROPERTY_NAME_VNAME = "VersionName";
   62:   private static final String PROPERTY_NAME_ANAME = "AppName";
   63:
   64:   // Form UI components
   65:   AbsolutePanel formWidget;
   66:   private MockComponent selectedComponent;
   67:
   68:   private int screenWidth;
   69:   private int screenHeight;
   70:   private int usableScreenHeight;
   71:
   72:   // Set of listeners for any changes of the form
   73:   final HashSet<FormChangeListener> formChangeListeners = new HashSet<FormChangeList
ener>();
   74:
   75:   // Don't access the verticalScrollbarWidth field directly. Use getVerticalScrollba
rWidth().
   76:   private static int verticalScrollbarWidth;
   77:
   78:   private MockFormLayout myLayout;
   79:
   80:   // flag to control attempting to enable/disable vertical
   81:   // alignment when scrollable property is changed
   82:   private boolean initialized = false;
   83:
   84:   private YoungAndroidVerticalAlignmentChoicePropertyEditor myVAlignmentPropertyEdit
or;
   85:
   86:   public static final String PROPERTY_NAME_HORIZONTAL_ALIGNMENT = "AlignHorizontal";
   87:   public static final String PROPERTY_NAME_VERTICAL_ALIGNMENT = "AlignVertical";
   88:
   89:   /**
   90:    * Creates a new MockForm component.
   91:    *
   92:    * @param editor  editor of source file the component belongs to
   93:    */
   94:   public MockForm(SimpleEditor editor) {
   95:     // Note(Hal): This helper thing is a kludge because I really want to write:
   96:     // myLayout = new MockHVLayout(orientation);
   97:     // super(editor, type, icon, myLayout);
   98:     // but Java won't let me do that.
   99:
  100:     super(editor, TYPE, images.form(), MockFormHelper.makeLayout());
  101:     // Note(hal): There better not be any calls to MockFormHelper before the
  102:     // next instruction.  Note that the Helper methods are synchronized to avoid pos
sible
  103:     // future problems if we ever have threads creating forms in parallel.
  104:     myLayout = MockFormHelper.getLayout();
  105:
  106:     formWidget = new AbsolutePanel();
  107:     formWidget.setStylePrimaryName("ode-SimpleMockForm");
  108:
  109:     screenWidth = PORTRAIT_WIDTH;
  110:     screenHeight = PORTRAIT_HEIGHT;
  111:     usableScreenHeight = screenHeight;
  112:
  113:     // This is just the initial size of the form. It will be resized in refresh();
  114:     rootPanel.setPixelSize(screenWidth, usableScreenHeight);
  115:     resizePanels();
  116:
```

```
117:     initComponent(formWidget);
118:
119:     // Set up the initial state of the vertical alignment property editor and its
120:     // dropdowns
121:     try {
122:       myVAlignmentPropertyEditor = PropertiesUtil.getVAlignmentEditor(properties);
123:     } catch (BadPropertyEditorException e) {
124:       OdeLog.log(MESSAGES.badAlignmentPropertyEditorForArrangement());
125:       return;
126:     }
127:     enableAndDisableDropdowns();
128:     initialized = true;
129:     // Now that the default for Scrollable is false, we need to force setting the pr
operty when creating the MockForm
130:     setScrollableProperty(getPropertyValue(PROPERTY_NAME_SCROLLABLE));
131:   }
132:
133:   /*
134:    * Resizes the formWidget based on the screen size.
135:    */
136:   private void resizePanels() {
137:     formWidget.setPixelSize(screenWidth, screenHeight);
138:   }
139:
140:   /*
141:    * Returns the width of a vertical scroll bar, calculating it if necessary.
142:    */
143:   private static int getVerticalScrollbarWidth() {
144:     // We only calculate the vertical scroll bar width once, then we store it in the
 static field
145:     // verticalScrollbarWidth. If the field is non-zero, we don't need to calculate
it again.
146:     if (verticalScrollbarWidth == 0) {
147:       // The following code will calculate (on the fly) the width of a vertical scro
ll bar.
148:       // We'll create two divs, one inside the other and add the outer div to the do
cument body,
149:       // but off-screen where the user won't see it.
150:       // We'll measure the width of the inner div twice: (first) when the outer div'
s vertical
151:       // scrollbar is hidden and (second) when the outer div's vertical scrollbar is
 visible.
152:       // The width of inner div will be smaller when outer div's vertical scrollbar
is visible.
153:       // By subtracting the two measurements, we can calculate the width of the vert
ical scrollbar.
154:
155:       // I used code from the following websites as reference material:
156:       // http://jdsharp.us/jQuery/minute/calculate-scrollbar-width.php
157:       // http://www.fleegix.org/articles/2006-05-30-getting-the-scrollbar-width-in-p
ixels
158:
159:       Document document = Document.get();
160:
161:       // Create an outer div.
162:       DivElement outerDiv = document.createDivElement();
163:       Style outerDivStyle = outerDiv.getStyle();
164:       // Use absolute positioning and set the top/left so that it is off-screen.
165:       // We don't want the user to see anything while we do this calculation.
166:       outerDivStyle.setProperty("position", "absolute");
167:       outerDivStyle.setProperty("top", "-1000px");
168:       outerDivStyle.setProperty("left", "-1000px");
169:       // Set the width and height of the outer div to a fixed size in pixels.
170:       outerDivStyle.setProperty("width", "100px");
171:       outerDivStyle.setProperty("height", "50px");
172:       // Hide the outer div's scrollbar by setting the "overflow" property to "hidde
n".
173:       outerDivStyle.setProperty("overflow", "hidden");
174:
175:       // Create an inner div and put it inside the outer div.
176:       DivElement innerDiv = document.createDivElement();
177:       Style innerDivStyle = innerDiv.getStyle();
178:       // Set the height of the inner div to be 4 times the height of the outer div s
o that a
179:       // vertical scrollbar will be necessary (but hidden for now) on the outer div.
180:       innerDivStyle.setProperty("height", "200px");
181:       outerDiv.appendChild(innerDiv);
182:
183:       // Temporarily add the outer div to the document body. It's off-screen so the
user won't
184:       // actually see anything.
185:       Element bodyElement = document.getElementsByTagName("body").getItem(0);
186:       bodyElement.appendChild(outerDiv);
187:
188:       // Get the width of the inner div while the outer div's vertical scrollbar is
hidden.
189:       int widthWithoutScrollbar = innerDiv.getOffsetWidth();
190:       // Show the outer div's vertical scrollbar by setting the "overflow" property
to "auto".
191:       outerDivStyle.setProperty("overflow", "auto");
192:       // Now, get the width of the inner div while the vertical scrollbar is visible
.
193:       int widthWithScrollbar = innerDiv.getOffsetWidth();
194:
195:       // Remove the outer div from the document body.
196:       bodyElement.removeChild(outerDiv);
197:
198:       // Calculate the width of the vertical scrollbar by subtracting the two widths
.
199:       verticalScrollbarWidth = widthWithoutScrollbar - widthWithScrollbar;
200:     }
201:
202:     return verticalScrollbarWidth;
203:   }
204:
205:   @Override
206:   public final MockForm getForm() {
207:     return this;
208:   }
209:
210:   @Override
211:   public boolean isForm() {
212:     return true;
213:   }
214:
215:
216:   @Override
217:   public String getVisibleTypeName() {
218:     return VISIBLE_TYPE;
219:   }
220:
221:   @Override
222:   protected void addWidthHeightProperties() {
223:     addProperty(PROPERTY_NAME_WIDTH, "" + PORTRAIT_WIDTH, null,
```

```
        new YoungAndroidLengthPropertyEditor());
    addProperty(PROPERTY_NAME_HEIGHT, "" + LENGTH_PREFERRED, null,
        new YoungAndroidLengthPropertyEditor());
  }

  @Override
  public boolean isPropertyPersisted(String propertyName) {
    // We use the Width and Height properties to make the form appear correctly in t
he designer,
    // but they aren't actually persisted to the .scm file.
    if (propertyName.equals(PROPERTY_NAME_WIDTH) ||
        propertyName.equals(PROPERTY_NAME_HEIGHT)) {
      return false;
    }
    return super.isPropertyPersisted(propertyName);
  }

  @Override
  protected boolean isPropertyVisible(String propertyName) {
    if (propertyName.equals(PROPERTY_NAME_WIDTH) ||
        propertyName.equals(PROPERTY_NAME_HEIGHT)) {
      return false;
    }

    if (propertyName.equals(PROPERTY_NAME_ICON)) {
      // The Icon property actually applies to the application and is only visible o
n Screen1.
      return editor.isScreen1();
    }

    if (propertyName.equals(PROPERTY_NAME_VNAME)) {
      // The VersionName property actually applies to the application and is only vi
sible on Screen1.
      return editor.isScreen1();
    }

    if (propertyName.equals(PROPERTY_NAME_VCODE)) {
      // The VersionCode property actually applies to the application and is only vi
sible on Screen1.
      return editor.isScreen1();
    }

    if (propertyName.equals(PROPERTY_NAME_ANAME)) {
      // The AppName property actually applies to the application and is only visibl
e on Screen1.
      return editor.isScreen1();
    }

    return super.isPropertyVisible(propertyName);
  }

  /*
   * Sets the form's BackgroundColor property to a new value.
   */
  private void setBackgroundColorProperty(String text) {
    if (MockComponentsUtil.isNoneColor(text)) {
      text = "&HFF000000";  // black
    } else if (MockComponentsUtil.isDefaultColor(text)) {
      text = "&HFFFFFFFF";  // white
    }
    MockComponentsUtil.setWidgetBackgroundColor(rootPanel, text);
  }

  /*
   * Sets the form's BackgroundImage property to a new value.
   */
  private void setBackgroundImageProperty(String text) {
    String url = convertImagePropertyValueToUrl(text);
    if (url == null) {
      // text was not recognized as an asset.
      url = "";
    }
    MockComponentsUtil.setWidgetBackgroundImage(rootPanel, url);
  }

  private void setScreenOrientationProperty(String text) {
    if (hasProperty(PROPERTY_NAME_WIDTH) && hasProperty(PROPERTY_NAME_HEIGHT) &&
        hasProperty(PROPERTY_NAME_SCROLLABLE)) {
      if (text.equalsIgnoreCase("landscape")) {
        screenWidth = LANDSCAPE_WIDTH;
        screenHeight = LANDSCAPE_HEIGHT;
      } else {
        screenWidth = PORTRAIT_WIDTH;
        screenHeight = PORTRAIT_HEIGHT;
      }
      usableScreenHeight = screenHeight;
      resizePanels();

      changeProperty(PROPERTY_NAME_WIDTH, "" + screenWidth);
      boolean scrollable = Boolean.parseBoolean(getPropertyValue(PROPERTY_NAME_SCROL
LABLE));
      if (!scrollable) {
        changeProperty(PROPERTY_NAME_HEIGHT, "" + usableScreenHeight);
      }
    }
  }

  private void setScrollableProperty(String text) {
    if (hasProperty(PROPERTY_NAME_HEIGHT)) {
      final boolean scrollable = Boolean.parseBoolean(text);
      int heightHint = scrollable ? LENGTH_PREFERRED : usableScreenHeight;
      changeProperty(PROPERTY_NAME_HEIGHT, "" + heightHint);
    }
  }

  private void setIconProperty(String icon) {
    // The Icon property actually applies to the application and is only visible on
Screen1.
    // When we load a form that is not Screen1, this method will be called with the
default value
    // for icon (empty string). We need to ignore that.
    if (editor.isScreen1()) {
      editor.getProjectEditor().changeProjectSettingsProperty(
          SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
          SettingsConstants.YOUNG_ANDROID_SETTINGS_ICON, icon);
    }
  }

  private void setVCodeProperty(String vcode) {
    // The VersionCode property actually applies to the application and is only visi
ble on Screen1.
    // When we load a form that is not Screen1, this method will be called with the
default value
    // for VersionCode (1). We need to ignore that.
```

```
338:     if (editor.isScreen1()) {
339:       editor.getProjectEditor().changeProjectSettingsProperty(
340:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
341:           SettingsConstants.YOUNG_ANDROID_SETTINGS_VERSION_CODE, vcode);
342:     }
343:   }
344:
345:   private void setVNameProperty(String vname) {
346:     // The VersionName property actually applies to the application and is only visi
ble on Screen1.
347:     // When we load a form that is not Screen1, this method will be called with the
default value
348:     // for VersionName (1.0). We need to ignore that.
349:     if (editor.isScreen1()) {
350:       editor.getProjectEditor().changeProjectSettingsProperty(
351:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
352:           SettingsConstants.YOUNG_ANDROID_SETTINGS_VERSION_NAME, vname);
353:     }
354:   }
355:
356:   private void setANameProperty(String aname) {
357:     // The AppName property actually applies to the application and is only visible
on Screen1.
358:     // When we load a form that is not Screen1, this method will be called with the
default value
359:     if (editor.isScreen1()) {
360:       editor.getProjectEditor().changeProjectSettingsProperty(
361:           SettingsConstants.PROJECT_YOUNG_ANDROID_SETTINGS,
362:           SettingsConstants.YOUNG_ANDROID_SETTINGS_APP_NAME, aname);
363:     }
364:   }
365:
366:   /**
367:    * Forces a re-layout of the child components of the container.
368:    */
369:   public final void refresh() {
370:     Map<MockComponent, LayoutInfo> layoutInfoMap = new HashMap<MockComponent, Layout
Info>();
371:
372:     collectLayoutInfos(layoutInfoMap, this);
373:
374:     LayoutInfo formLayoutInfo = layoutInfoMap.get(this);
375:     layout.layoutChildren(formLayoutInfo);
376:     rootPanel.setPixelSize(formLayoutInfo.width,
377:         Math.max(formLayoutInfo.height, usableScreenHeight));
378:
379:     for (LayoutInfo layoutInfo : layoutInfoMap.values()) {
380:       layoutInfo.cleanUp();
381:     }
382:     layoutInfoMap.clear();
383:   }
384:
385:   /*
386:    * Collects the LayoutInfo of the given component and, recursively, all of
387:    * its children.
388:    *
389:    * If a component's width/height hint is automatic, the corresponding
390:    * LayoutInfo's width/height will be set to the calculated width/height.
391:    * If a component's width/height hint is fill parent, the corresponding
392:    * LayoutInfo's width/height may be set to fill parent. This will be resolved
393:    * when layoutChildren is called.
394:    */
395:   private static void collectLayoutInfos(Map<MockComponent, LayoutInfo> layoutInfoMa
p,
396:       MockComponent component) {
397:
398:     LayoutInfo layoutInfo = component.createLayoutInfo(layoutInfoMap);
399:
400:     // If this component is a container, collect the LayoutInfos of its children.
401:     if (component instanceof MockContainer) {
402:       if (!layoutInfo.visibleChildren.isEmpty()) {
403:         // We resize the container to be very large so that we get accurate
404:         // results when we ask for a child's size using getOffsetWidth/getOffsetHeig
ht.
405:         // If the container is its normal size (or perhaps the default empty
406:         // size), then the browser won't give us anything bigger than that
407:         // when we ask for a child's size.
408:         if (component.isForm()) {
409:           ((MockForm) component).rootPanel.setPixelSize(1000, 1000);
410:         } else {
411:           component.setPixelSize(1000, 1000);
412:         }
413:
414:         // Show children that should be shown and collect their layoutInfos.
415:         // Note that some MockLayout implementations may hide children that are in t
he
416:         // visibleChildren list. For example, in MockTableLayout, if two or more chi
ldren occupy
417:         // the same cell in the table, all but one of the children are hidden.
418:         for (MockComponent child : layoutInfo.visibleChildren) {
419:           child.setVisible(true);
420:           collectLayoutInfos(layoutInfoMap, child);
421:         }
422:       }
423:
424:       // Hide children that should be hidden.
425:       for (MockComponent child : component.getHiddenVisibleChildren()) {
426:         child.setVisible(false);
427:       }
428:     }
429:
430:     layoutInfo.gatherDimensions();
431:   }
432:
433:   /**
434:    * Adds an {@link FormChangeListener} to the listener set if it isn't already in t
here.
435:    *
436:    * @param listener  the {@code FormChangeListener} to be added
437:    */
438:   public void addFormChangeListener(FormChangeListener listener) {
439:     formChangeListeners.add(listener);
440:   }
441:
442:   /**
443:    * Removes an {@link FormChangeListener} from the listener list.
444:    *
445:    * @param listener  the {@code FormChangeListener} to be removed
446:    */
447:   public void removeFormChangeListener(FormChangeListener listener) {
448:     formChangeListeners.remove(listener);
449:   }
450:
451:   /**
```

```
   * Triggers a component property change event to be sent to the listener on the li
stener list.
   */
  protected void fireComponentPropertyChanged(MockComponent component,
      String propertyName, String propertyValue) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentPropertyChanged(component, propertyName, propertyValue);
    }
  }

  /**
   * Triggers a component removed event to be sent to the listener on the listener l
ist.
   */
  protected void fireComponentRemoved(MockComponent component, boolean permanentlyDe
leted) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentRemoved(component, permanentlyDeleted);
    }
  }

  /**
   * Triggers a component added event to be sent to the listener on the listener lis
t.
   */
  protected void fireComponentAdded(MockComponent component) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentAdded(component);
    }
  }

  /**
   * Triggers a component renamed event to be sent to the listener on the listener l
ist.
   */
  protected void fireComponentRenamed(MockComponent component, String oldName) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentRenamed(component, oldName);
    }
  }

  /**
   * Triggers a component selection change event to be sent to the listener on the l
istener list.
   */
  protected void fireComponentSelectionChange(MockComponent component, boolean selec
ted) {
    for (FormChangeListener listener : formChangeListeners) {
      listener.onComponentSelectionChange(component, selected);
    }
  }

  /**
   * Changes the component that is currently selected in the form.
   * <p>
   * There will always be exactly one component selected in a form
   * at any given time.
   */
  public final void setSelectedComponent(MockComponent newSelectedComponent) {
    MockComponent oldSelectedComponent = selectedComponent;

    if (newSelectedComponent == null) {
      throw new IllegalArgumentException("at least one component must always be sele
cted");
    }
    if (newSelectedComponent == oldSelectedComponent) {
      return;
    }

    selectedComponent = newSelectedComponent;

    if (oldSelectedComponent != null) {     // Can be null initially
      oldSelectedComponent.onSelectedChange(false);
    }
    newSelectedComponent.onSelectedChange(true);
  }

  public final MockComponent getSelectedComponent() {
    return selectedComponent;
  }

  /**
   * Builds a tree of the component hierarchy of the form for display in the
   * {@code SourceStructureExplorer}.
   *
   * @return  tree showing the component hierarchy of the form
   */
  public TreeItem buildComponentsTree() {
    return buildTree();
  }

  // PropertyChangeListener implementation

  @Override
  public void onPropertyChange(String propertyName, String newValue) {
    super.onPropertyChange(propertyName, newValue);

    // Apply changed properties to the mock component
    /*if (propertyName.equals(PROPERTY_NAME_BACKGROUNDCOLOR)) {
      setBackgroundColorProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_BACKGROUNDIMAGE)) {
      setBackgroundImageProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_SCREEN_ORIENTATION)) {
      setScreenOrientationProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_SCROLLABLE)) {
      setScrollableProperty(newValue);
      adjustAlignmentDropdowns();
    } else */if (propertyName.equals(PROPERTY_NAME_ICON)) {
      setIconProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_VCODE)) {
      setVCodeProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_VNAME)) {
      setVNameProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_ANAME)) {
      setANameProperty(newValue);
    } else if (propertyName.equals(PROPERTY_NAME_HORIZONTAL_ALIGNMENT)) {
      myLayout.setHAlignmentFlags(newValue);
      refreshForm();
    } else if (propertyName.equals(PROPERTY_NAME_VERTICAL_ALIGNMENT)) {
    myLayout.setVAlignmentFlags(newValue);
    refreshForm();
    }
  }

```

```
568:   // enableAndDisable It should not be called until the component is initialized.
569:   // Otherwise, we'll get NPEs in trying to use myAlignmentPropertyEditor.
570:   private void adjustAlignmentDropdowns() {
571:     if (initialized) enableAndDisableDropdowns();
572:   }
573:
574:   // Don't forget to call this on initialization!!!
575:   // If scrollable is True, the selector for vertical alignment should be disabled.
576:   private void enableAndDisableDropdowns() {
577:     String scrollable = properties.getProperty(PROPERTY_NAME_SCROLLABLE).getValue();
578:     if (scrollable.equals("True")) {
579:       myVAlignmentPropertyEditor.disable();
580:     } else myVAlignmentPropertyEditor.enable();
581:   }
582: }
```

```
 1: // -*- mode: java; c-basic-offset: 2; -*-
 2: // Released under the Apache License, Version 2.0
 3: // http://www.apache.org/licenses/LICENSE-2.0
 4:
 5: package com.google.appinventor.components.runtime;
 6:
 7: import com.google.appinventor.components.annotations.DesignerComponent;
 8: import com.google.appinventor.components.annotations.DesignerProperty;
 9: import com.google.appinventor.components.annotations.PropertyCategory;
10: import com.google.appinventor.components.annotations.SimpleEvent;
11: import com.google.appinventor.components.annotations.SimpleFunction;
12: import com.google.appinventor.components.annotations.SimpleObject;
13: import com.google.appinventor.components.annotations.SimpleProperty;
14: import com.google.appinventor.components.annotations.UsesPermissions;
15: import com.google.appinventor.components.common.ComponentCategory;
16: import com.google.appinventor.components.common.PropertyTypeConstants;
17: import com.google.appinventor.components.common.YaVersion;
18: import java.io.IOException;
19:
20: @DesignerComponent(version = 1,
21:     description = "<p>Barometer description</p>",
22:     category = ComponentCategory.SENSORS,
23:     nonVisible = true,
24:     iconName = "images/barometer.png")
25: @SimpleObject
26: public class Barometer {
27:
28:         public Barometer() {}
29:
30: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = 1,
17:     description = "<p>Battery description</p>",
18:     category = ComponentCategory.SENSORS,
19:     nonVisible = true,
20:     iconName = "images/battery.png")
21: @SimpleObject
22: public class Battery
23: {
24:   // Battery Level
25:   private int level;
26:
27:   // Constructor
28:   public Battery() {}
29:
30:   /**
31:    * Get Level
32:    */
33:   @SimpleProperty(
34:       category = PropertyCategory.BEHAVIOR)
35:   public int Level() {
36:     return level;
37:   }
38:
39:   /**
40:    * Set Level
41:    */
42:   @SimpleProperty
43:   public void Level(int lev) {
44:     level = lev;
45:   }
46:
47: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.common.ComponentCategory;
11: import com.google.appinventor.components.common.PropertyTypeConstants;
12: import com.google.appinventor.components.common.YaVersion;
13: import java.io.IOException;
14:
15: @DesignerComponent(version = 1,
16:     description = "<p>GPS description</p>",
17:     category = ComponentCategory.SENSORS,
18:     nonVisible = true,
19:     iconName = "images/gps.png")
20: @SimpleObject
21: public class GPS
22: {
23:   // Current position
24:   private double latitude;
25:   private double longitude;
26:   private double altitude;
27:
28:   public GPS() {}
29:
30:   /**
31:    * Get Latitude
32:    */
33:   @SimpleProperty(
34:       category = PropertyCategory.BEHAVIOR)
35:   public double Latitude() {
36:     return latitude;
37:   }
38:
39:   /**
40:    * Get Longitude
41:    */
42:   @SimpleProperty(
43:       category = PropertyCategory.BEHAVIOR)
44:   public double Longitude() {
45:     return longitude;
46:   }
47:
48:   /**
49:    * Get Altitude
50:    */
51:   @SimpleProperty(
52:       category = PropertyCategory.BEHAVIOR)
53:   public double Altitude() {
54:     return altitude;
55:   }
56:
57:   /**
58:    * Set Latitude
59:    */
60:   @SimpleProperty
61:   public void Latitude(double lat) {
62:     latitude = lat;
63:   }
64:
65:   /**
66:    * Set Longitude
67:    */
68:   @SimpleProperty
69:   public void Longitude(double lon) {
70:     longitude = lon;
71:   }
72:
73:   /**
74:    * Set Altitude
75:    */
76:   @SimpleProperty
77:   public void Altitude(double alt) {
78:     altitude = alt;
79:   }
80:
81: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = 1,
17:     description = "<p>GPSController description</p>",
18:     category = ComponentCategory.ACTUATORS,
19:     nonVisible = true,
20:     iconName = "images/gps.png")
21: @SimpleObject
22: public class GPSController
23: {
24:   public GPSController() {}
25:
26:   /*
27:    * Change Home Coordinates
28:    */
29:   @SimpleFunction
30:   public void setHome(double latitude, double longitude, double altitude) {}
31:
32:   /**
33:    * Reset Home Coordinates
34:    */
35:   @SimpleFunction
36:   public void resetHome() {}
37:
38: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = 1,
17:     description = "<p>Gyroscope description</p>",
18:     category = ComponentCategory.SENSORS,
19:     nonVisible = true,
20:     iconName = "images/accelerometersensor.png")
21: @SimpleObject
22: public class Gyroscope {
23:
24:   // Backing for sensor values
25:   private float xGyr;
26:   private float yGyr;
27:   private float zGyr;
28:
29:   public Gyroscope () {}
30:
31:   /**
32:    * Get XGyr
33:    */
34:   @SimpleProperty(
35:       category = PropertyCategory.BEHAVIOR)
36:   public float XGyr() {
37:     return xGyr;
38:   }
39:
40:   /**
41:    * Get YGyr
42:    */
43:   @SimpleProperty(
44:       category = PropertyCategory.BEHAVIOR)
45:   public float YGyr() {
46:     return yGyr;
47:   }
48:
49:   /**
50:    * Get ZGyr
51:    */
52:   @SimpleProperty(
53:       category = PropertyCategory.BEHAVIOR)
54:   public float ZGyr() {
55:     return zGyr;
56:   }
57:
58: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = 1,
17:     description = "<p>Motor description</p>",
18:     category = ComponentCategory.SENSORS,
19:     nonVisible = true,
20:     iconName = "images/motor.png")
21: @SimpleObject
22: public class Motor
23: {
24:   private boolean landed;
25:   private boolean takingOff;
26:   private boolean hovering;
27:   private boolean flying;
28:   private boolean landing;
29:   private boolean emergency;
30:   private double latitude;
31:   private double longitude;
32:   private double altitude;
33:   private double speedx;
34:   private double speedy;
35:   private double speedz;
36:   private int roll;
37:   private int pitch;
38:   private int yaw;
39:
40:   public Motor(){}
41:
42:   /**
43:    * Get isLanded
44:    */
45:   @SimpleProperty(
46:       category = PropertyCategory.BEHAVIOR)
47:   public boolean isLanded() {
48:     return landed;
49:   }
50:
51:   /**
52:    * Get isTakingOff
53:    */
54:   @SimpleProperty(
55:       category = PropertyCategory.BEHAVIOR)
56:   public boolean isTakingOff() {
57:     return takingOff;
58:   }
59:
60:   /**
61:    * Get isHovering
62:    */
63:   @SimpleProperty(
64:       category = PropertyCategory.BEHAVIOR)
65:   public boolean isHovering() {
66:     return hovering;
67:   }
68:
69:   /**
70:    * Get isFlying
71:    */
72:   @SimpleProperty(
73:       category = PropertyCategory.BEHAVIOR)
74:   public boolean isFlying() {
75:     return flying;
76:   }
77:
78:   /**
79:    * Get isLanding
80:    */
81:   @SimpleProperty(
82:       category = PropertyCategory.BEHAVIOR)
83:   public boolean isLanding() {
84:     return landing;
85:   }
86:
87:   /**
88:    * Get inEmergency
89:    */
90:   @SimpleProperty(
91:       category = PropertyCategory.BEHAVIOR)
92:   public boolean inEmergency() {
93:     return emergency;
94:   }
95:
96:   /**
97:    * Set isLanded
98:    */
99:   @SimpleProperty
100:  public void isLanded(boolean landed) {
101:    this.landed = landed;
102:  }
103:
104:  /**
105:   * Set isTakingOff
106:   */
107:  @SimpleProperty
108:  public void isTakingOff(boolean takingOff) {
109:    this.takingOff = takingOff;
110:  }
111:
112:  /**
113:   * Set isHovering
114:   */
115:  @SimpleProperty
116:  public void isHovering(boolean hovering) {
117:    this.hovering = hovering;
118:  }
119:
120:  /**
121:   * Set isFlying
122:   */
123:  @SimpleProperty
124:  public void isFlying(boolean flying) {
```

```
125:     this.flying = flying;
126:   }
127:
128:   /**
129:    * Set isLanding
130:    */
131:   @SimpleProperty
132:   public void isLanding(boolean landing) {
133:     this.landing = landing;
134:   }
135:
136:   /**
137:    * Set inEmergency
138:    */
139:   @SimpleProperty
140:   public void inEmergency(boolean emergency) {
141:     this.emergency = emergency;
142:   }
143:
144:   /**
145:    * Get Latitude
146:    */
147:   @SimpleProperty(
148:       category = PropertyCategory.BEHAVIOR)
149:   public double Latitude() {
150:     return latitude;
151:   }
152:
153:   /**
154:    * Get Longitude
155:    */
156:   @SimpleProperty(
157:       category = PropertyCategory.BEHAVIOR)
158:   public double Longitude() {
159:     return longitude;
160:   }
161:
162:   /**
163:    * Get Altitude
164:    */
165:   @SimpleProperty(
166:       category = PropertyCategory.BEHAVIOR)
167:   public double Altitude() {
168:     return altitude;
169:   }
170:
171:   /**
172:    * Set Latitude
173:    */
174:   @SimpleProperty
175:   public void Latitude(double lat) {
176:     latitude = lat;
177:   }
178:
179:   /**
180:    * Set Longitude
181:    */
182:   @SimpleProperty
183:   public void Longitude(double lon) {
184:     longitude = lon;
185:   }
186:
187:   /**
188:    * Set Altitude
189:    */
190:   @SimpleProperty
191:   public void Altitude(double alt) {
192:     altitude = alt;
193:   }
194:
195:   /**
196:    * Get SpeedX
197:    */
198:   @SimpleProperty(
199:       category = PropertyCategory.BEHAVIOR)
200:   public double SpeedX() {
201:     return speedx;
202:   }
203:
204:   /**
205:    * Get SpeedY
206:    */
207:   @SimpleProperty(
208:       category = PropertyCategory.BEHAVIOR)
209:   public double SpeedY() {
210:     return speedy;
211:   }
212:
213:   /**
214:    * Get SpeedZ
215:    */
216:   @SimpleProperty(
217:       category = PropertyCategory.BEHAVIOR)
218:   public double SpeedZ() {
219:     return speedz;
220:   }
221:
222:   /**
223:    * Set SpeedX
224:    */
225:   @SimpleProperty
226:   public void SpeedX(double spx) {
227:     speedx = spx;
228:   }
229:
230:   /**
231:    * Set SpeedY
232:    */
233:   @SimpleProperty
234:   public void SpeedY(double spy) {
235:     speedy = spy;
236:   }
237:
238:   /**
239:    * Set SpeedZ
240:    */
241:   @SimpleProperty
242:   public void SpeedZ(double spz) {
243:     speedz = spz;
244:   }
245:
246:   /**
247:    * Get Roll
248:    */
```

```
249:   @SimpleProperty(
250:       category = PropertyCategory.BEHAVIOR)
251:   public int Roll() {
252:     return roll;
253:   }
254:
255:   /**
256:    * Get Pitch
257:    */
258:   @SimpleProperty(
259:       category = PropertyCategory.BEHAVIOR)
260:   public int Pitch() {
261:     return pitch;
262:   }
263:
264:   /**
265:    * Get Yaw
266:    */
267:   @SimpleProperty(
268:       category = PropertyCategory.BEHAVIOR)
269:   public int Yaw() {
270:     return yaw;
271:   }
272:
273:   /**
274:    * Set Roll
275:    */
276:   @SimpleProperty
277:   public void Roll(int roll) {
278:     this.roll = roll;
279:   }
280:
281:   /**
282:    * Set Pitch
283:    */
284:   @SimpleProperty
285:   public void Pitch(int pitch) {
286:     this.pitch = pitch;
287:   }
288:
289:   /**
290:    * Set Yaw
291:    */
292:   @SimpleProperty
293:   public void Yaw(int yaw) {
294:     this.yaw = yaw;
295:   }
296:
297: }
```

```java
package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import java.io.IOException;

@DesignerComponent(version = 1,
    description = "<p>Motor description</p>",
    category = ComponentCategory.ACTUATORS,
    nonVisible = true,
    iconName = "images/motor.png")
@SimpleObject
public class MotorController
{
        public MotorController() {}

  /**
   * flatTrim
   */
  @SimpleFunction
  public void flatTrim() {}

  /**
   * takeOff
   */
  @SimpleFunction
  public void takeOff() {}

  /**
   * moveForward
   */
  @SimpleFunction
  public void moveForward(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * moveBackward
   */
  @SimpleFunction
  public void moveBackward(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * moveRight
   */
  @SimpleFunction
  public void moveRight(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * moveLeft
   */
  @SimpleFunction
  public void moveLeft(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * up
   */
  @SimpleFunction
  public void up(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * down
   */
  @SimpleFunction
  public void down(double SpeedX, double SpeedY, double SpeedZ) {}

  /**
   * land
   */
  @SimpleFunction
  public void land() {}

  /**
   * startNavigateHome
   */
  @SimpleFunction
  public void startNavigateHome() {}

  /**
   * stopNavigateHome
   */
  @SimpleFunction
  public void stopNavigateHome() {}

  /**
   * emergencyMode
   */
  @SimpleFunction
  public void emergencyMode() {}

  /**
   * flip
   */
  @SimpleFunction
  public void flip(String direction) {}

}
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = YaVersion.SOUND_COMPONENT_VERSION,
17:     description = "<p>LEDs description</p>",
18:     category = ComponentCategory.ACTUATORS,
19:     nonVisible = true,
20:     iconName = "images/led.png")
21: @SimpleObject
22: public class LEDs
23: {
24:   // Power
25:   private boolean isOn;
26:
27:   public LEDs() {}
28:
29:   /**
30:    * Get On/Off
31:    */
32:   @SimpleProperty(
33:       category = PropertyCategory.BEHAVIOR)
34:   public boolean On() {
35:     return isOn;
36:   }
37:
38:   /**
39:    * Set On/Off
40:    */
41:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
42:       defaultValue = "False")
43:   @SimpleProperty
44:   public void On(boolean status) {
45:     isOn = status;
46:   }
47:
48: }
```

```
 1: package com.google.appinventor.components.runtime;
 2:
 3: import com.google.appinventor.components.annotations.DesignerComponent;
 4: import com.google.appinventor.components.annotations.DesignerProperty;
 5: import com.google.appinventor.components.annotations.PropertyCategory;
 6: import com.google.appinventor.components.annotations.SimpleEvent;
 7: import com.google.appinventor.components.annotations.SimpleFunction;
 8: import com.google.appinventor.components.annotations.SimpleObject;
 9: import com.google.appinventor.components.annotations.SimpleProperty;
10: import com.google.appinventor.components.annotations.UsesPermissions;
11: import com.google.appinventor.components.common.ComponentCategory;
12: import com.google.appinventor.components.common.PropertyTypeConstants;
13: import com.google.appinventor.components.common.YaVersion;
14: import java.io.IOException;
15:
16: @DesignerComponent(version = 1,
17:     description = "<p>UltrasoundsSensor description</p>",
18:     category = ComponentCategory.SENSORS,
19:     nonVisible = true,
20:     iconName = "images/ultrasoundsensor.png")
21: @SimpleObject
22: public class UltrasoundsSensor
23: {
24:   private float distance;
25:
26:   public UltrasoundsSensor() {}
27:
28:   /**
29:    * Distance
30:    */
31:   @SimpleProperty(
32:       category = PropertyCategory.BEHAVIOR)
33:   public float Distance() {
34:     return distance;
35:   }
36:
37: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0


package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;

/**
 * Physical world component that can detect shaking and measure
 * acceleration in three dimensions.  It is implemented using
 * android.hardware.SensorListener
 * (http://developer.android.com/reference/android/hardware/SensorListener.html).
 *
 * <p>From the Android documentation:
 * "Sensor values are acceleration in the X, Y and Z axis, where the X axis
 * has positive direction toward the right side of the device, the Y axis has
 * positive direction toward the top of the device and the Z axis has
 * positive direction toward the front of the device. The direction of the
 * force of gravity is indicated by acceleration values in the X, Y and Z
 * axes. The typical case where the device is flat relative to the surface of
 * the Earth appears as -STANDARD_GRAVITY in the Z axis and X and Y values
 * close to zero. Acceleration values are given in SI units (m/s^2)."
 *
 */
// TODO(user): ideas - event for knocking
@DesignerComponent(version = YaVersion.ACCELEROMETERSENSOR_COMPONENT_VERSION,
    description = "Non-visible component that can detect shaking and " +
    "measure acceleration approximately in three dimensions using SI units " +
    "(m/s<sup>2</sup>).  The components are: <ul>\n" +
    "<li> <strong>xAccel</strong>: 0 when the phone is at rest on a flat " +
    "     surface, positive when the phone is tilted to the right (i.e., " +
    "     its left side is raised), and negative when the phone is tilted " +
    "     to the left (i.e., its right size is raised).</li>\n " +
    "<li> <strong>yAccel</strong>: 0 when the phone is at rest on a flat " +
    "     surface, positive when its bottom is raised, and negative when " +
    "     its top is raised. </li>\n " +
    "<li> <strong>zAccel</strong>: Equal to -9.8 (earth's gravity in meters per " +
    "     second per second when the device is at rest parallel to the ground " +
    "     with the display facing up, " +
    "     0 when perpindicular to the ground, and +9.8 when facing down.  " +
    "     The value can also be affected by accelerating it with or against " +
    "     gravity. </li></ul>",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/accelerometersensor.png")
@SimpleObject
public class AccelerometerSensor
{
  // Backing for sensor values
  private float xAccel;
  private float yAccel;
  private float zAccel;

  private int accuracy;

  private int sensitivity;

  // Indicates whether the accelerometer should generate events
  private boolean enabled;

  private boolean available;

  //Specifies the minimum time interval between calls to Shaking()
  private int minimumInterval;

  /**
   * Creates a new AccelerometerSensor component
   */
  public AccelerometerSensor() {
    enabled = true;
    MinimumInterval(400);
    Sensitivity(2);
  }


  /**
   * Returns the minimum interval required between calls to Shaking(),
   * in milliseconds.
   * Once the phone starts being shaken, all further Shaking() calls will be ignored
   * until the interval has elapsed.
   * @return  minimum interval in ms
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "The minimum interval between phone shakes")
  public int MinimumInterval() {
    return minimumInterval;
  }

  /**
   * Specifies the minimum interval required between calls to Shaking(),
   * in milliseconds.
   * Once the phone starts being shaken, all further Shaking() calls will be ignored
   * until the interval has elapsed.
   * @param interval  minimum interval in ms
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
      defaultValue = "400") //Default value derived by trial of 12 people on 3 different devices
  @SimpleProperty
  public void MinimumInterval(int interval) {
    minimumInterval = interval;
  }

  /**
   * Returns a number that encodes how sensitive the AccelerometerSensor is.
   * The choices are: 1 = weak, 2 = moderate, 3 = strong.
   *
   * @return  one of {@link Component#ACCELEROMETER_SENSITIVITY_WEAK},
   *          {@link Component#ACCELEROMETER_SENSITIVITY_MODERATE} or
   *          {@link Component#ACCELEROMETER_SENSITIVITY_STRONG}
   */
```

```
  123:   @SimpleProperty(
  124:       category = PropertyCategory.APPEARANCE,
  125:       description = "A number that encodes how sensitive the accelerometer is. " +
  126:               "The choices are: 1 = weak, 2 = moderate, " +
  127:               " 3 = strong.")
  128:   public int Sensitivity() {
  129:     return sensitivity;
  130:   }
  131:
  132:   /**
  133:    * Specifies the sensitivity of the accelerometer
  134:    * and checks that the argument is a legal value.
  135:    *
  136:    * @param sensitivity one of {@link Component#ACCELEROMETER_SENSITIVITY_WEAK},
  137:    *          {@link Component#ACCELEROMETER_SENSITIVITY_MODERATE} or
  138:    *          {@link Component#ACCELEROMETER_SENSITIVITY_STRONG}
  139:    *
  140:    */
  141:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ACCELEROMETER_S
ENSITIVITY,
  142:       defaultValue = Component.ACCELEROMETER_SENSITIVITY_MODERATE + "")
  143:   @SimpleProperty
  144:   public void Sensitivity(int sensitivity) {
  145:     if ((sensitivity == 1) || (sensitivity == 2) || (sensitivity == 3)) {
  146:       this.sensitivity = sensitivity;
  147:     } else {
  148:       //form.dispatchErrorOccurredEvent(this, "Sensitivity",
  149:           //ErrorMessages.ERROR_BAD_VALUE_FOR_ACCELEROMETER_SENSITIVITY, sensitivity
);
  150:     }
  151:   }
  152:
  153:   /**
  154:    * Indicates the acceleration changed in the X, Y, and/or Z dimensions.
  155:    */
  156:   //@SimpleEvent
  157:   public void AccelerationChanged(float xAccel, float yAccel, float zAccel) {
  158:     this.xAccel = xAccel;
  159:     this.yAccel = yAccel;
  160:     this.zAccel = zAccel;
  161:   }
  162:
  163:   /**
  164:    * Available property getter method (read-only property)
  165:    */
  166:   @SimpleProperty(
  167:       category = PropertyCategory.BEHAVIOR)
  168:   public boolean Available() {
  169:     return available;
  170:   }
  171:
  172:   /**
  173:    * If true, the sensor will generate events.  Otherwise, no events
  174:    * are generated even if the device is accelerated or shaken.
  175:    *
  176:    * @return {@code true},
  177:    *         {@code false}
  178:    */
  179:   @SimpleProperty(
  180:       category = PropertyCategory.BEHAVIOR)
  181:   public boolean Enabled() {
  182:     return enabled;
  183:   }
  184:
  185:   /**
  186:    * Specifies whether the sensor should generate events.  If true,
  187:    * the sensor will generate events.  Otherwise, no events are
  188:    * generated even if the device is accelerated or shaken.
  189:    *
  190:    * @param enabled  {@code true},
  191:    *                 {@code false}
  192:    */
  193:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
  194:       defaultValue = "True")
  195:   @SimpleProperty
  196:   public void Enabled(boolean enabled) {
  197:     if (this.enabled == enabled) {
  198:       return;
  199:     }
  200:     this.enabled = enabled;
  201:   }
  202:
  203:   /**
  204:    * Returns the acceleration in the X-dimension in SI units (m/s^2).
  205:    * The sensor must be enabled to return meaningful values.
  206:    *
  207:    * @return  X acceleration
  208:    */
  209:   @SimpleProperty(
  210:       category = PropertyCategory.BEHAVIOR)
  211:   public float XAccel() {
  212:     return xAccel;
  213:   }
  214:
  215:   /**
  216:    * Returns the acceleration in the Y-dimension in SI units (m/s^2).
  217:    * The sensor must be enabled to return meaningful values.
  218:    *
  219:    * @return  Y acceleration
  220:    */
  221:   @SimpleProperty(
  222:       category = PropertyCategory.BEHAVIOR)
  223:   public float YAccel() {
  224:     return yAccel;
  225:   }
  226:
  227:   /**
  228:    * Returns the acceleration in the Z-dimension in SI units (m/s^2).
  229:    * The sensor must be enabled to return meaningful values.
  230:    *
  231:    * @return  Z acceleration
  232:    */
  233:   @SimpleProperty(
  234:       category = PropertyCategory.BEHAVIOR)
  235:   public float ZAccel() {
  236:     return zAccel;
  237:   }
  238: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;

@DesignerComponent(version = 1,
    description = "Buzzer description",
    category = ComponentCategory.ACTUATORS,
    nonVisible = true,
    iconName = "images/soundEffect.png")
@SimpleObject
public class Buzzer
{
  // stream id returned from last call to SoundPool.play
  private int minimumInterval;            // minimum interval between Play() calls

  // Power
  private boolean isOn;

  public Buzzer()
  {
    // Default property values
    MinimumInterval(500);
  }

  /**
   * On/Off
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR)
  public boolean On() {
    return isOn;
  }

  /**
   * On/Off
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
      defaultValue = "False")
  @SimpleProperty
  public void On(boolean status) {
    isOn = status;
  }

  /**
   * Returns the minimum interval required between calls to Play(), in
   * milliseconds.
   * Once the sound starts playing, all further Play() calls will be ignored
   * until the interval has elapsed.
   * @return  minimum interval in ms
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "The minimum interval between sounds.  If you play a sound, " +
      "all further Play() calls will be ignored until the interval has elapsed.")
  public int MinimumInterval() {
    return minimumInterval;
  }

  /**
   * Specify the minimum interval required between calls to Play(), in
   * milliseconds.
   * Once the sound starts playing, all further Play() calls will be ignored
   * until the interval has elapsed.
   * @param interval  minimum interval in ms
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_IN
TEGER,
      defaultValue = "500")
  @SimpleProperty
  public void MinimumInterval(int interval) {
    minimumInterval = interval;
  }

  /**
   * Plays the buzzer.
   */
  @SimpleFunction(description = "Plays the buzzer.")
  public void Play(){}

  /**
   * Pauses playing the sound if it is being played.
   */
  @SimpleFunction(description = "Pauses playing the sound if it is being played.")
  public void Pause(){}

  /**
   * Resumes playing the sound after a pause.
   */
  @SimpleFunction(description = "Resumes playing the sound after a pause.")
  public void Resume(){}

  /**
   * Stops playing the sound if it is being played.
   */
  @SimpleFunction(description = "Stops playing the sound if it is being played.")
  public void Stop(){}

}
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.YaVersion;

@DesignerComponent(version = 1,
    description = "Camera description",
    category = ComponentCategory.ACTUATORS,
    nonVisible = true,
    iconName = "images/camera.png")
@SimpleObject
public class Camera
{
  // Vertical move in degrees
  private int tilt;

  // Horizontal move in degrees
  private int pan;

  /**
   * Creates a Camera component
   */
  public Camera(){}

  /**
   * Takes a picture
   */
  @SimpleFunction
  public void takePicture(){}

  /*
   * Camera Orientation
   */
  @SimpleFunction(description = "set Camera orientation")
  public void orientation(int tilt, int pan) {
    if(tilt >= 0 && tilt <= 360 && pan >= 0 && pan <= 360)
    {
        this.tilt = tilt;
        this.pan = pan;
    }
  }

  /**
   * StartVideo
   */
  @SimpleFunction
  public void startVideo() {}

  /**
   * StopVideo
   */
  @SimpleFunction
  public void stopVideo() {}

}
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.errors.YailRuntimeError;
import com.google.appinventor.components.runtime.util.Dates;
import com.google.appinventor.components.runtime.util.TimerInternal;

import java.util.Calendar;

/**
 * Clock provides the phone's clock, a timer, calendar and time calculations.
 * Everything is represented in milliseconds.
 *
 */

@DesignerComponent(version = YaVersion.CLOCK_COMPONENT_VERSION,
    description = "Non-visible component that provides the instant in "
    + "time using the internal clock on the phone. It can fire a "
    + "timer at regularly set intervals and perform time "
    + "calculations, manipulations, and conversions. Methods to "
    + "format the date and time are also available.",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/clock.png")
@SimpleObject
public final class Clock extends AndroidNonvisibleComponent
    implements Component, AlarmHandler, OnStopListener, OnResumeListener, OnDestroyL
istener,
               Deleteable {
  private static final int DEFAULT_INTERVAL = 1000;  // ms
  private static final boolean DEFAULT_ENABLED = true;

  private TimerInternal timerInternal;
  private boolean timerAlwaysFires = true;
  private boolean onScreen = false;

  /**
   * Creates a new Clock component.
   *
   * @param container ignored (because this is a non-visible component)
   */
  public Clock(ComponentContainer container) {
    super(container.$form());
    timerInternal = new TimerInternal(this, DEFAULT_ENABLED, DEFAULT_INTERVAL);

    // Set up listeners
    form.registerForOnResume(this);
    form.registerForOnStop(this);
    form.registerForOnDestroy(this);

    if (form instanceof ReplForm) {
      // In REPL, if this Clock component was added to the project after the onResum
e call occurred,
      // then onScreen would be false, but the REPL app is, in fact, on screen.
      onScreen = true;
    }
  }

  // Only the above constructor should be used in practice.
  public Clock() {
    super(null);
    // To allow testing without Timer
  }

  /**
   * Default Timer event handler.
   */
  /*@SimpleEvent(
      description = "Timer has gone off.")*/
  public void Timer() {
    if (timerAlwaysFires || onScreen) {
      EventDispatcher.dispatchEvent(this, "Timer");
    }
  }

  /**
   * Interval property getter method.
   *
   * @return timer interval in ms
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description ="Interval between timer events in ms")
  public int TimerInterval() {
    return timerInternal.Interval();
  }

  /**
   * Interval property setter method: sets the interval between timer events.
   *
   * @param interval timer interval in ms
   */
  @DesignerProperty(
      editorType = PropertyTypeConstants.PROPERTY_TYPE_NON_NEGATIVE_INTEGER,
      defaultValue = DEFAULT_INTERVAL + "")
  @SimpleProperty
  public void TimerInterval(int interval) {
    timerInternal.Interval(interval);
  }

  /**
   * Enabled property getter method.
   *
   * @return {@code true} indicates a running timer, {@code false} a stopped
   *         timer
   */
  @SimpleProperty(
      category = PropertyCategory.BEHAVIOR,
      description = "Fires timer if true")
```

```
123:   public boolean TimerEnabled() {
124:     return timerInternal.Enabled();
125:   }
126:
127:   /**
128:    * Enabled property setter method: starts or stops the timer.
129:    *
130:    * @param enabled {@code true} starts the timer, {@code false} stops it
131:    */
132:   @DesignerProperty(
133:       editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
134:       defaultValue = DEFAULT_ENABLED ? "True" : "False")
135:   @SimpleProperty
136:   public void TimerEnabled(boolean enabled) {
137:     timerInternal.Enabled(enabled);
138:   }
139:
140:   /**
141:    * TimerAlwaysFires property getter method.
142:    *
143:    *  return {@code true} if the timer event will fire even if the application
144:    *   is not on the screen
145:    */
146:   @SimpleProperty(
147:       category = PropertyCategory.BEHAVIOR,
148:       description = "Will fire even when application is not showing on the "
149:       + "screen if true")
150:   public boolean TimerAlwaysFires() {
151:     return timerAlwaysFires;
152:   }
153:
154:   /**
155:    * TimerAlwaysFires property setter method: instructs when to disable
156:    *
157:    *  @param always {@code true} if the timer event should fire even if the
158:    *  application is not on the screen
159:    */
160:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defaul
tValue = "True")
161:   @SimpleProperty
162:   public void TimerAlwaysFires(boolean always) {
163:     timerAlwaysFires = always;
164:   }
165:
166:   // AlarmHandler implementation
167:
168:   @Override
169:   public void alarm() {
170:     Timer();
171:   }
172:
173:   /**
174:    * Returns the current system time in milliseconds.
175:    *
176:    * @return  current system time in milliseconds
177:    */
178:   @SimpleFunction (description = "The phone's internal time")
179:   public static long SystemTime() {
180:     return Dates.Timer();
181:   }
182:
183:   @SimpleFunction(description = "The current instant in time read from "
184:       + "phone's clock")
185:   public static Calendar Now() {
186:     return Dates.Now();
187:   }
188:
189:   /**
190:    * An instant in time specified by MM/DD/YYYY hh:mm:ss or MM/DD/YYYY or hh:mm
191:    * where MM is the month (01-12), DD the day (01-31), YYYY the year
192:    * (0000-9999), hh the hours (00-23), mm the minutes (00-59) and ss
193:    * the seconds (00-59).
194:    *
195:    * @param from  string to convert
196:    * @return  date
197:    */
198:   @SimpleFunction(
199:       description = "An instant specified by MM/DD/YYYY hh:mm:ss or MM/DD/YYYY or hh
:mm")
200:   public static Calendar MakeInstant(String from) {
201:     try {
202:       return Dates.DateValue(from);
203:     } catch (IllegalArgumentException e) {
204:       throw new YailRuntimeError(
205:           "Argument to MakeInstant should have form MM/DD/YYYY, hh:mm:ss, or MM/DD/Y
YYY or hh:mm",
206:           "Sorry to be so picky.");
207:     }
208:   }
209:
210:   /**
211:    * Create an Calendar from ms since 1/1/1970 00:00:00.0000
212:    * Probably should go in Calendar.
213:    *
214:    * @param millis raw millisecond number.
215:    */
216:   @SimpleFunction(description = "An instant in time specified by the milliseconds si
nce 1970.")
217:   public static Calendar MakeInstantFromMillis(long millis) {
218:     Calendar instant = Dates.Now(); // just to get our hands on an instant
219:     instant.setTimeInMillis(millis);
220:     return instant;
221:   }
222:
223:   /**
224:    * Calendar property getter method: gets the raw millisecond representation of
225:    *  a Calendar.
226:    * @param instant Calendar
227:    * @return milliseconds since 1/1/1970.
228:    */
229:   @SimpleFunction (description = "The instant in time measured as milliseconds since
 1970.")
230:   public static long GetMillis(Calendar instant) {
231:     return instant.getTimeInMillis();
232:   }
233:
234:   @SimpleFunction(description = "An instant in time some seconds after the argument"
)
235:   public static Calendar AddSeconds(Calendar instant, int seconds) {
236:     Calendar newInstant = (Calendar) instant.clone();
237:     Dates.DateAdd(newInstant, Calendar.SECOND, seconds);
238:     return newInstant;
239:   }
240:
```

```
  @SimpleFunction(description = "An instant in time some minutes after the argument"
)
  public static Calendar AddMinutes(Calendar instant, int minutes) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.MINUTE, minutes);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some hours after the argument")
  public static Calendar AddHours(Calendar instant, int hours) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.HOUR_OF_DAY, hours);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some days after the argument")
  public static Calendar AddDays(Calendar instant, int days) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.DATE, days);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some weeks after the argument")
  public static Calendar AddWeeks(Calendar instant, int weeks) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.WEEK_OF_YEAR, weeks);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some months after the argument")
  public static Calendar AddMonths(Calendar instant, int months) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.MONTH, months);
    return newInstant;
  }

  @SimpleFunction(description = "An instant in time some years after the argument")
  public static Calendar AddYears(Calendar instant, int years) {
    Calendar newInstant = (Calendar) instant.clone();
    Dates.DateAdd(newInstant, Calendar.YEAR, years);
    return newInstant;
  }

  /**
   * Returns the milliseconds by which end follows start (+ or -)
   *
   * @param start beginning instant
   * @param end ending instant
   * @return  milliseconds
   */
  @SimpleFunction (description = "Milliseconds elapsed between instants")
  public static long Duration(Calendar start, Calendar end) {
    return end.getTimeInMillis() - start.getTimeInMillis();
  }

  /**
   * Returns the seconds for the given instant.
   *
   * @param instant  instant to use seconds of
   * @return  seconds (range 0 - 59)
   */
  @SimpleFunction (description = "The second of the minute")
  public static int Second(Calendar instant) {
    return Dates.Second(instant);
  }

/**
   * Returns the minutes for the given date.
   *
   * @param instant instant to use minutes of
   * @return  minutes (range 0 - 59)
   */
  @SimpleFunction(description = "The minute of the hour")
  public static int Minute(Calendar instant) {
    return Dates.Minute(instant);
  }

  /**
   * Returns the hours for the given date.
   *
   * @param instant Calendar to use hours of
   * @return  hours (range 0 - 23)
   */
  @SimpleFunction (description = "The hour of the day")
  public static int Hour(Calendar instant) {
    return Dates.Hour(instant);
  }

  /**
   * Returns the day of the month.
   *
   * @param instant  instant to use day of the month of
   * @return  day: [1...31]
   */
  @SimpleFunction (description = "The day of the month")
  public static int DayOfMonth(Calendar instant) {
    return Dates.Day(instant);
  }

  /**
   * Returns the weekday for the given instant.
   *
   * @param instant  instant to use day of week of
   * @return day of week: [1...7] starting with Sunday
   */
  @SimpleFunction (description = "The day of the week represented as a "
      + "number from 1 (Sunday) to 7 (Saturday)")
  public static int Weekday(Calendar instant) {
    return Dates.Weekday(instant);
  }

  /**
   * Returns the name of the weekday for the given instant.
   *
   * @param instant  instant to use weekday of
   * @return  weekday, as a string.
   */
  @SimpleFunction (description = "The name of the day of the week")
  public static String WeekdayName(Calendar instant) {
    return Dates.WeekdayName(instant);
  }

  /**
   * Returns the number of the month for the given instant.
```

```
364:    *
365:    * @param instant  instant to use month of
366:    * @return  number of month
367:    */
368:   @SimpleFunction (description = "The month of the year represented as a "
369:       + "number from 1 to 12)")
370:   public static int Month(Calendar instant) {
371:     return Dates.Month(instant) + 1;
372:   }
373:
374:   /**
375:    * Returns the name of the month for the given instant.
376:    *
377:    * @param instant  instant to use month of
378:    * @return  name of month
379:    */
380:   @SimpleFunction (description = "The name of the month")
381:   public static String MonthName(Calendar instant) {
382:     return Dates.MonthName(instant);
383:   }
384:
385:  /**
386:    * Returns the year of the given instant.
387:    *
388:    * @param instant  instant to use year of
389:    * @return  year
390:    */
391:   @SimpleFunction(description = "The year")
392:   public static int Year(Calendar instant) {
393:     return Dates.Year(instant);
394:   }
395:
396:   /**
397:    * Converts and formats the given instant into a string.   *
398:    *
399:    * @param instant  instant to format
400:    * @return  formatted instant
401:    */
402:   @SimpleFunction (description = "Text representing the date and time of an"
403:       + " instant")
404:   public static String FormatDateTime(Calendar instant) {
405:     return Dates.FormatDateTime(instant);
406:   }
407:
408:   /**
409:    * Converts and formats the given instant into a string.
410:    *
411:    * @param instant  instant to format
412:    * @return  formatted instant
413:    */
414:   @SimpleFunction (description = "Text representing the date of an instant")
415:   public static String FormatDate(Calendar instant) {
416:     return Dates.FormatDate(instant);
417:   }
418:
419:   /**
420:    * Converts and formats the given instant into a string.
421:    *
422:    * @param instant  instant to format
423:    * @return  formatted instant
424:    */
425:   @SimpleFunction (description = "Text representing the time of an instant")
426:   public static String FormatTime(Calendar instant) {
427:     return Dates.FormatTime(instant);
428:   }
429:
430:   @Override
431:   public void onStop() {
432:     onScreen = false;
433:   }
434:
435:   @Override
436:   public void onResume() {
437:     onScreen = true;
438:   }
439:
440:   @Override
441:   public void onDestroy() {
442:     timerInternal.Enabled(false);
443:   }
444:
445:   @Override
446:   public void onDelete() {
447:     timerInternal.Enabled(false);
448:   }
449: }
```

```
    1:  // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2009-2011 Google, All Rights reserved
    3: // Copyright 2011-2012 MIT, All rights reserved
    4: // Released under the Apache License, Version 2.0
    5: // http://www.apache.org/licenses/LICENSE-2.0
    6:
    7: // ********************************************
    8: // If we're not going to go this route with onDestroy, then at least get rid of the
DEBUG flag.
    9:
   10: package com.google.appinventor.components.runtime;
   11:
   12: import java.io.IOException;
   13: import java.lang.reflect.InvocationTargetException;
   14: import java.lang.reflect.Method;
   15: import java.util.HashMap;
   16: import java.util.List;
   17: import java.util.Map;
   18: import java.util.Set;
   19:
   20: import org.json.JSONException;
   21:
   22: import android.app.Activity;
   23: import android.app.Dialog;
   24: import android.content.ActivityNotFoundException;
   25: import android.content.Intent;
   26: import android.content.pm.ActivityInfo;
   27: import android.content.res.Configuration;
   28: import android.graphics.drawable.Drawable;
   29: import android.os.Bundle;
   30: import android.os.Handler;
   31: import android.text.Html;
   32: import android.util.Log;
   33: import android.view.KeyEvent;
   34: import android.view.Menu;
   35: import android.view.MenuItem;
   36: import android.view.MenuItem.OnMenuItemClickListener;
   37: import android.view.ViewGroup;
   38: import android.view.WindowManager;
   39: import android.widget.FrameLayout;
   40: import android.widget.ScrollView;
   41:
   42: import com.google.appinventor.components.annotations.DesignerComponent;
   43: import com.google.appinventor.components.annotations.DesignerProperty;
   44: import com.google.appinventor.components.annotations.PropertyCategory;
   45: import com.google.appinventor.components.annotations.SimpleEvent;
   46: import com.google.appinventor.components.annotations.SimpleObject;
   47: import com.google.appinventor.components.annotations.SimpleProperty;
   48: import com.google.appinventor.components.annotations.UsesPermissions;
   49: import com.google.appinventor.components.common.ComponentCategory;
   50: import com.google.appinventor.components.common.ComponentConstants;
   51: import com.google.appinventor.components.common.PropertyTypeConstants;
   52: import com.google.appinventor.components.common.YaVersion;
   53: import com.google.appinventor.components.runtime.collect.Lists;
   54: import com.google.appinventor.components.runtime.collect.Maps;
   55: import com.google.appinventor.components.runtime.collect.Sets;
   56: import com.google.appinventor.components.runtime.util.AlignmentUtil;
   57: import com.google.appinventor.components.runtime.util.AnimationUtil;
   58: import com.google.appinventor.components.runtime.util.ErrorMessages;
   59: import com.google.appinventor.components.runtime.util.FullScreenVideoUtil;
   60: import com.google.appinventor.components.runtime.util.JsonUtil;
   61: import com.google.appinventor.components.runtime.util.MediaUtil;
   62: import com.google.appinventor.components.runtime.util.OnInitializeListener;
   63: import com.google.appinventor.components.runtime.util.SdkLevel;
   64: import com.google.appinventor.components.runtime.util.ViewUtil;
   65:
   66: /**
   67:  * Component underlying activities and UI apps, not directly accessible to Simple pr
ogrammers.
   68:  *
   69:  * <p>This is the root container of any Android activity and also the
   70:  * superclass for for Simple/Android UI applications.
   71:  *
   72:  * The main form is always named "Screen1".
   73:  *
   74:  */
   75: @DesignerComponent(version = YaVersion.FORM_COMPONENT_VERSION,
   76:     //category = ComponentCategory.LAYOUT,
   77:     description = "Top-level component containing all other components in the progra
m",
   78:     showOnPalette = false)
   79: @SimpleObject
   80: @UsesPermissions(permissionNames = "android.permission.INTERNET,android.permission.A
CCESS_WIFI_STATE,android.permission.ACCESS_NETWORK_STATE")
   81: public class Form extends Activity
   82:     implements Component, ComponentContainer, HandlesEventDispatching {
   83:   private static final String LOG_TAG = "Form";
   84:
   85:   private static final String RESULT_NAME = "APP_INVENTOR_RESULT";
   86:
   87:   private static final String ARGUMENT_NAME = "APP_INVENTOR_START";
   88:
   89:   public static final String APPINVENTOR_URL_SCHEME = "appinventor";
   90:
   91:   // Keep track of the current form object.
   92:   // activeForm always holds the Form that is currently handling event dispatching s
o runtime.scm
   93:   // can lookup symbols in the correct environment.
   94:   // There is at least one case where an event can be fired when the activity is not
 the foreground
   95:   // activity: if a Clock component's TimerAlwaysFires property is true, the Clock c
omponent's
   96:   // Timer event will still fire, even when the activity is no longer in the foregro
und. For this
   97:   // reason, we cannot assume that the activeForm is the foreground activity.
   98:   protected static Form activeForm;
   99:
  100:   // applicationIsBeingClosed is set to true during closeApplication.
  101:   private static boolean applicationIsBeingClosed;
  102:
  103:   private final Handler androidUIHandler = new Handler();
  104:
  105:   protected String formName;
  106:
  107:   private boolean screenInitialized;
  108:
  109:   private static final int SWITCH_FORM_REQUEST_CODE = 1;
  110:   private static int nextRequestCode = SWITCH_FORM_REQUEST_CODE + 1;
  111:
  112:   // Backing for background color
  113:   private int backgroundColor;
  114:
  115:   // Information string the app creator can set.  It will be shown when
  116:   // "about this application" menu item is selected.
```

```
  117:   private String aboutScreen;
  118:
  119:   private String backgroundImagePath = "";
  120:   private Drawable backgroundDrawable;
  121:
  122:   // Layout
  123:   private LinearLayout viewLayout;
  124:
  125:   // translates App Inventor alignment codes to Android gravity
  126:   private AlignmentUtil alignmentSetter;
  127:
  128:   // the alignment for this component's LinearLayout
  129:   private int horizontalAlignment;
  130:   private int verticalAlignment;
  131:
  132:   // String representing the transition animation type
  133:   private String openAnimType;
  134:   private String closeAnimType;
  135:
  136:   private FrameLayout frameLayout;
  137:   private boolean scrollable;
  138:
  139:   // Application lifecycle related fields
  140:   private final HashMap<Integer, ActivityResultListener> activityResultMap = Maps.ne
wHashMap();
  141:   private final Set<OnStopListener> onStopListeners = Sets.newHashSet();
  142:   private final Set<OnNewIntentListener> onNewIntentListeners = Sets.newHashSet();
  143:   private final Set<OnResumeListener> onResumeListeners = Sets.newHashSet();
  144:   private final Set<OnPauseListener> onPauseListeners = Sets.newHashSet();
  145:   private final Set<OnDestroyListener> onDestroyListeners = Sets.newHashSet();
  146:
  147:   // AppInventor lifecycle: listeners for the Initialize Event
  148:   private final Set<OnInitializeListener> onInitializeListeners = Sets.newHashSet();
  149:
  150:   // Set to the optional String-valued Extra passed in via an Intent on startup.
  151:   // This is passed directly in the Repl.
  152:   protected String startupValue = "";
  153:
  154:   // To control volume of error complaints
  155:   private static long minimumToastWait = 10000000000L; // 10 seconds
  156:   private long lastToastTime = System.nanoTime() - minimumToastWait;
  157:
  158:   // In a multiple screen application, when a secondary screen is opened, nextFormNa
me is set to
  159:   // the name of the secondary screen. It is saved so that it can be passed to the O
therScreenClosed
  160:   // event.
  161:   private String nextFormName;
  162:
  163:   private FullScreenVideoUtil fullScreenVideoUtil;
  164:
  165:   @Override
  166:   public void onCreate(Bundle icicle) {
  167:     // Called when the activity is first created
  168:     super.onCreate(icicle);
  169:
  170:     // Figure out the name of this form.
  171:     String className = getClass().getName();
  172:     int lastDot = className.lastIndexOf('.');
  173:     formName = className.substring(lastDot + 1);
  174:     Log.d(LOG_TAG, "Form " + formName + " got onCreate");
  175:
  176:     activeForm = this;
  177:     Log.i(LOG_TAG, "activeForm is now " + activeForm.formName);
  178:
  179:     viewLayout = new LinearLayout(this, ComponentConstants.LAYOUT_ORIENTATION_VERTIC
AL);
  180:     alignmentSetter = new AlignmentUtil(viewLayout);
  181:
  182:     defaultPropertyValues();
  183:
  184:     // Get startup text if any before adding components
  185:     Intent startIntent = getIntent();
  186:     if (startIntent != null && startIntent.hasExtra(ARGUMENT_NAME)) {
  187:       startupValue = startIntent.getStringExtra(ARGUMENT_NAME);
  188:     }
  189:
  190:     fullScreenVideoUtil = new FullScreenVideoUtil(this, androidUIHandler);
  191:
  192:     // Set soft keyboard to not cover the focused UI element, e.g., when you are typ
ing
  193:     // into a textbox near the bottom of the screen.
  194:     WindowManager.LayoutParams params = getWindow().getAttributes();
  195:     int softInputMode = params.softInputMode;
  196:     getWindow().setSoftInputMode(
  197:         softInputMode | WindowManager.LayoutParams.SOFT_INPUT_ADJUST_RESIZE);
  198:
  199:     // Add application components to the form
  200:     $define();
  201:
  202:     // Special case for Event.Initialize(): all other initialize events are triggere
d after
  203:     // completing the constructor. This doesn't work for Android apps though because
 this method
  204:     // is called after the constructor completes and therefore the Initialize event
would run
  205:     // before initialization finishes. Instead the compiler suppresses the invocatio
n of the
  206:     // event and leaves it up to the library implementation.
  207:     Initialize();
  208:   }
  209:
  210:   private void defaultPropertyValues() {
  211:     Scrollable(false); // frameLayout is created in Scrollable()
  212:     BackgroundImage("");
  213:     AboutScreen("");
  214:     AlignVertical(ComponentConstants.GRAVITY_TOP);
  215:     Title("");
  216:   }
  217:
  218:   @Override
  219:   public void onConfigurationChanged(Configuration newConfig) {
  220:     super.onConfigurationChanged(newConfig);
  221:
  222:     final int newOrientation = newConfig.orientation;
  223:     if (newOrientation == Configuration.ORIENTATION_LANDSCAPE ||
  224:         newOrientation == Configuration.ORIENTATION_PORTRAIT) {
  225:       // At this point, the screen has not be resized to match the new orientation.
  226:       // We use Handler.post so that we'll dispatch the ScreenOrientationChanged eve
nt after the
  227:       // screen has been resized to match the new orientation.
  228:
  229:       androidUIHandler.post(new Runnable() {
  230:         public void run() {
```

```
231:           boolean dispatchEventNow = false;
232:           if (frameLayout != null) {
233:             if (newOrientation == Configuration.ORIENTATION_LANDSCAPE) {
234:               if (frameLayout.getWidth() >= frameLayout.getHeight()) {
235:                 dispatchEventNow = true;
236:               }
237:             } else { // Portrait
238:               if (frameLayout.getHeight() >= frameLayout.getWidth()) {
239:                 dispatchEventNow = true;
240:               }
241:             }
242:           }
243:           if (dispatchEventNow) {
244:             ScreenOrientationChanged();
245:           } else {
246:             // Try again later.
247:             androidUIHandler.post(this);
248:           }
249:         }
250:       });
251:     }
252:   }
253:
254:   /*
255:    * Here we override the hardware back button, just to make sure
256:    * that the closing screen animation is applied. (In API level
257:    * 5, we can simply override the onBackPressed method rather
258:    * than bothering with onKeyDown)
259:    */
260:   @Override
261:   public boolean onKeyDown(int keyCode, KeyEvent event) {
262:     if (keyCode == KeyEvent.KEYCODE_BACK) {
263:       if (!BackPressed()) {
264:         boolean handled = super.onKeyDown(keyCode, event);
265:         AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
266:         return handled;
267:       } else {
268:         return true;
269:       }
270:     }
271:     return super.onKeyDown(keyCode, event);
272:   }
273:
274:   //@SimpleEvent(description = "Device back button pressed.")
275:   public boolean BackPressed() {
276:     return EventDispatcher.dispatchEvent(this, "BackPressed");
277:   }
278:
279:   // onActivityResult should be triggered in only two cases:
280:   // (1) The result is for some other component in the app, not this Form itself
281:   // (2) This page started another page, and that page is closing, and passing
282:   // its value back as a JSON-encoded string in the intent.
283:
284:   @Override
285:   protected void onActivityResult(int requestCode, int resultCode, Intent data) {
286:     Log.i(LOG_TAG, "Form " + formName + " got onActivityResult, requestCode = " +
287:         requestCode + ", resultCode = " + resultCode);
288:     if (requestCode == SWITCH_FORM_REQUEST_CODE) {
289:       // Assume this is a multiple screen application, and a secondary
290:       // screen has closed.  Process the result as a JSON-encoded string.
291:       // This can also happen if the user presses the back button, in which case
292:       // there's no data.
293:       String resultString;
294:       if (data != null && data.hasExtra(RESULT_NAME)) {
295:         resultString = data.getStringExtra(RESULT_NAME);
296:       } else {
297:         resultString = "";
298:       }
299:       Object decodedResult = decodeJSONStringForForm(resultString, "other screen clo
sed");
300:       // nextFormName was set when this screen opened the secondary screen
301:       OtherScreenClosed(nextFormName, decodedResult);
302:     } else {
303:       // Another component (such as a ListPicker, ActivityStarter, etc) is expecting
 this result.
304:       ActivityResultListener component = activityResultMap.get(requestCode);
305:       if (component != null) {
306:         component.resultReturned(requestCode, resultCode, data);
307:       }
308:     }
309:   }
310:
311:   // functionName is a string to include in the error message that will be shown
312:   // if the JSON decoding fails
313:   private  static Object decodeJSONStringForForm(String jsonString, String functionN
ame) {
314:     Log.i(LOG_TAG, "decodeJSONStringForForm -- decoding JSON representation:" + json
String);
315:     Object valueFromJSON = "";
316:     try {
317:       valueFromJSON = JsonUtil.getObjectFromJson(jsonString);
318:       Log.i(LOG_TAG, "decodeJSONStringForForm -- got decoded JSON:" + valueFromJSON.
toString());
319:     } catch (JSONException e) {
320:       activeForm.dispatchErrorOccurredEvent(activeForm, functionName,
321:           // showing the start value here will produce an ugly error on the phone, b
ut it's
322:           // more useful than not showing the value
323:           ErrorMessages.ERROR_SCREEN_BAD_VALUE_RECEIVED, jsonString);
324:     }
325:     return valueFromJSON;
326:   }
327:
328:   public int registerForActivityResult(ActivityResultListener listener) {
329:     int requestCode = generateNewRequestCode();
330:     activityResultMap.put(requestCode, listener);
331:     return requestCode;
332:   }
333:
334:   public void unregisterForActivityResult(ActivityResultListener listener) {
335:     List<Integer> keysToDelete = Lists.newArrayList();
336:     for (Map.Entry<Integer, ActivityResultListener> mapEntry : activityResultMap.ent
rySet()) {
337:       if (listener.equals(mapEntry.getValue())) {
338:         keysToDelete.add(mapEntry.getKey());
339:       }
340:     }
341:     for (Integer key : keysToDelete) {
342:       activityResultMap.remove(key);
343:     }
344:   }
345:
346:   private static int generateNewRequestCode() {
347:     return nextRequestCode++;
```

```
348:   }
349:
350:   @Override
351:   protected void onResume() {
352:     super.onResume();
353:     Log.i(LOG_TAG, "Form " + formName + " got onResume");
354:     activeForm = this;
355:
356:     // If applicationIsBeingClosed is true, call closeApplication() immediately to c
ontinue
357:     // unwinding through all forms of a multi-screen application.
358:     if (applicationIsBeingClosed) {
359:       closeApplication();
360:       return;
361:     }
362:
363:     for (OnResumeListener onResumeListener : onResumeListeners) {
364:       onResumeListener.onResume();
365:     }
366:   }
367:
368:   public void registerForOnResume(OnResumeListener component) {
369:     onResumeListeners.add(component);
370:   }
371:
372:   /**
373:    * An app can register to be notified when App Inventor's Initialize
374:    * block has fired.  They will be called in Initialize().
375:    *
376:    * @param component
377:    */
378:   public void registerForOnInitialize(OnInitializeListener component) {
379:     onInitializeListeners.add(component);
380:   }
381:
382:   @Override
383:   protected void onNewIntent(Intent intent) {
384:     super.onNewIntent(intent);
385:     Log.d(LOG_TAG, "Form " + formName + " got onNewIntent " + intent);
386:     for (OnNewIntentListener onNewIntentListener : onNewIntentListeners) {
387:       onNewIntentListener.onNewIntent(intent);
388:     }
389:   }
390:
391:   public void registerForOnNewIntent(OnNewIntentListener component) {
392:     onNewIntentListeners.add(component);
393:   }
394:
395:   @Override
396:   protected void onPause() {
397:     super.onPause();
398:     Log.i(LOG_TAG, "Form " + formName + " got onPause");
399:     for (OnPauseListener onPauseListener : onPauseListeners) {
400:       onPauseListener.onPause();
401:     }
402:   }
403:
404:   public void registerForOnPause(OnPauseListener component) {
405:     onPauseListeners.add(component);
406:   }
407:
408:   @Override
409:   protected void onStop() {
410:     super.onStop();
411:     Log.i(LOG_TAG, "Form " + formName + " got onStop");
412:     for (OnStopListener onStopListener : onStopListeners) {
413:       onStopListener.onStop();
414:     }
415:   }
416:
417:   public void registerForOnStop(OnStopListener component) {
418:     onStopListeners.add(component);
419:   }
420:
421:   @Override
422:   protected void onDestroy() {
423:     super.onDestroy();
424:     // for debugging and future growth
425:     Log.i(LOG_TAG, "Form " + formName + " got onDestroy");
426:
427:     // Unregister events for components in this form.
428:     EventDispatcher.removeDispatchDelegate(this);
429:
430:     for (OnDestroyListener onDestroyListener : onDestroyListeners) {
431:       onDestroyListener.onDestroy();
432:     }
433:   }
434:
435:   public void registerForOnDestroy(OnDestroyListener component) {
436:     onDestroyListeners.add(component);
437:   }
438:
439:   public Dialog onCreateDialog(int id) {
440:     switch(id) {
441:     case FullScreenVideoUtil.FULLSCREEN_VIDEO_DIALOG_FLAG:
442:       return fullScreenVideoUtil.createFullScreenVideoDialog();
443:     default:
444:       return super.onCreateDialog(id);
445:     }
446:   }
447:
448:   public void onPrepareDialog(int id, Dialog dialog) {
449:     switch(id) {
450:     case FullScreenVideoUtil.FULLSCREEN_VIDEO_DIALOG_FLAG:
451:       fullScreenVideoUtil.prepareFullScreenVideoDialog(dialog);
452:       break;
453:     default:
454:       super.onPrepareDialog(id, dialog);
455:     }
456:   }
457:
458:   /**
459:    * Compiler-generated method to initialize and add application components to
460:    * the form.  We just provide an implementation here to artificially make
461:    * this class concrete so that it is included in the documentation and
462:    * Codeblocks language definition file generated by
463:    * {@link com.google.appinventor.components.scripts.DocumentationGenerator} and
464:    * {@link com.google.appinventor.components.scripts.LangDefXmlGenerator},
465:    * respectively.  The actual implementation appears in {@code runtime.scm}.
466:    */
467:   protected void $define() {    // This must be declared protected because we are ca
lled from Screen1 which subclasses
468:                                 // us and isn't in our package.
469:     throw new UnsupportedOperationException();
```

```
  470:   }
  471:
  472:   @Override
  473:   public boolean canDispatchEvent(Component component, String eventName) {
  474:     // Events can only be dispatched after the screen initialized event has complete
d.
  475:     boolean canDispatch = screenInitialized ||
  476:         (component == this && eventName.equals("Initialize"));
  477:
  478:     if (canDispatch) {
  479:       // Set activeForm to this before the event is dispatched.
  480:       // runtime.scm will call getActiveForm() when the event handler executes.
  481:       activeForm = this;
  482:     }
  483:
  484:     return canDispatch;
  485:   }
  486:
  487:   /**
  488:    * A trivial implementation to artificially make this class concrete so
  489:    * that it is included in the documentation and
  490:    * Codeblocks language definition file generated by
  491:    * {@link com.google.appinventor.components.scripts.DocumentationGenerator} and
  492:    * {@link com.google.appinventor.components.scripts.LangDefXmlGenerator},
  493:    * respectively.  The actual implementation appears in {@code runtime.scm}.
  494:    */
  495:   @Override
  496:   public boolean dispatchEvent(Component component, String componentName, String eve
ntName,
  497:       Object[] args) {
  498:     throw new UnsupportedOperationException();
  499:   }
  500:
  501:
  502:   /**
  503:    * Initialize event handler.
  504:    */
  505:   //@SimpleEvent(description = "Screen starting")
  506:   public void Initialize() {
  507:     // Dispatch the Initialize event only after the screen's width and height are no
 longer zero.
  508:     androidUIHandler.post(new Runnable() {
  509:       public void run() {
  510:         if (frameLayout != null && frameLayout.getWidth() != 0 && frameLayout.getHei
ght() != 0) {
  511:           EventDispatcher.dispatchEvent(Form.this, "Initialize");
  512:           screenInitialized = true;
  513:
  514:           //  Call all apps registered to be notified when Initialize Event is dispa
tched
  515:           for (OnInitializeListener onInitializeListener : onInitializeListeners) {
  516:             onInitializeListener.onInitialize();
  517:           }
  518:           if (activeForm instanceof ReplForm) { // We are the Companion
  519:             ((ReplForm)activeForm).HandleReturnValues();
  520:           }
  521:         } else {
  522:           // Try again later.
  523:           androidUIHandler.post(this);
  524:         }
  525:       }
  526:     });
  527:   }
  528:
  529:   //@SimpleEvent(description = "Screen orientation changed")
  530:   public void ScreenOrientationChanged() {
  531:     EventDispatcher.dispatchEvent(this, "ScreenOrientationChanged");
  532:   }
  533:
  534:   /**
  535:    * ErrorOccurred event handler.
  536:    */
  537:   /*@SimpleEvent(
  538:       description = "Event raised when an error occurs. Only some errors will " +
  539:       "raise this condition.  For those errors, the system will show a notification
" +
  540:       "by default.  You can use this event handler to prescribe an error " +
  541:       "behavior different than the default.")*/
  542:   public void ErrorOccurred(Component component, String functionName, int errorNumbe
r,
  543:       String message) {
  544:     String componentType = component.getClass().getName();
  545:     componentType = componentType.substring(componentType.lastIndexOf(".") + 1);
  546:     Log.e(LOG_TAG, "Form " + formName + " ErrorOccurred, errorNumber = " + errorNumb
er +
  547:         ", componentType = " + componentType + ", functionName = " + functionName +
  548:         ", messages = " + message);
  549:     if ((!(EventDispatcher.dispatchEvent(
  550:         this, "ErrorOccurred", component, functionName, errorNumber, message)))
  551:         && screenInitialized)  {
  552:       // If dispatchEvent returned false, then no user-supplied error handler was ru
n.
  553:       // If in addition, the screen initializer was run, then we assume that the
  554:       // user did not provide an error handler.   In this case, we run a default
  555:       // error handler, namely, showing a notification to the end user of the app.
  556:       // The app writer can override this by providing an error handler.
  557:       //new Notifier(this).ShowAlert("Error " + errorNumber + ": " + message);
  558:     }
  559:   }
  560:
  561:
  562:   public void ErrorOccurredDialog(Component component, String functionName, int erro
rNumber,
  563:       String message, String title, String buttonText) {
  564:     String componentType = component.getClass().getName();
  565:     componentType = componentType.substring(componentType.lastIndexOf(".") + 1);
  566:     Log.e(LOG_TAG, "Form " + formName + " ErrorOccurred, errorNumber = " + errorNumb
er +
  567:         ", componentType = " + componentType + ", functionName = " + functionName +
  568:         ", messages = " + message);
  569:     if ((!(EventDispatcher.dispatchEvent(
  570:         this, "ErrorOccurred", component, functionName, errorNumber, message)))
  571:         && screenInitialized)  {
  572:       // If dispatchEvent returned false, then no user-supplied error handler was ru
n.
  573:       // If in addition, the screen initializer was run, then we assume that the
  574:       // user did not provide an error handler.   In this case, we run a default
  575:       // error handler, namely, showing a message dialog to the end user of the app.
  576:       // The app writer can override this by providing an error handler.
  577:       //new Notifier(this).ShowMessageDialog("Error " + errorNumber + ": " + message
, title, buttonText);
  578:     }
  579:   }
  580:
```

```
  581:
  582:   public void dispatchErrorOccurredEvent(final Component component, final String fun
ctionName,
  583:       final int errorNumber, final Object... messageArgs) {
  584:     runOnUiThread(new Runnable() {
  585:       public void run() {
  586:         String message = ErrorMessages.formatMessage(errorNumber, messageArgs);
  587:         ErrorOccurred(component, functionName, errorNumber, message);
  588:       }
  589:     });
  590:   }
  591:
  592:   // This is like dispatchErrorOccurred, except that it defaults to showing
  593:   // a message dialog rather than an alert.   The app writer can override either of
these behaviors,
  594:   // but using the event dialog version frees the app writer of the need to explicit
ly override
  595:   // the alert behavior in the case
  596:   // where a message dialog is what's generally needed.
  597:   public void dispatchErrorOccurredEventDialog(final Component component, final Stri
ng functionName,
  598:       final int errorNumber, final Object... messageArgs) {
  599:     runOnUiThread(new Runnable() {
  600:       public void run() {
  601:         String message = ErrorMessages.formatMessage(errorNumber, messageArgs);
  602:         ErrorOccurredDialog(
  603:             component,
  604:             functionName,
  605:             errorNumber,
  606:             message,
  607:             "Error in " + functionName,
  608:             "Dismiss");
  609:       }
  610:     });
  611:   }
  612:
  613:
  614:
  615:   /**
  616:    * Scrollable property getter method.
  617:    *
  618:    * @return  true if the screen is vertically scrollable
  619:    */
  620:   @SimpleProperty(category = PropertyCategory.APPEARANCE,
  621:     description = "When checked, there will be a vertical scrollbar on the "
  622:     + "screen, and the height of the application can exceed the physical "
  623:     + "height of the device. When unchecked, the application height is "
  624:     + "constrained to the height of the device.")
  625:   public boolean Scrollable() {
  626:     return scrollable;
  627:   }
  628:
  629:   /**
  630:    * Scrollable property setter method.
  631:    *
  632:    * @param scrollable  true if the screen should be vertically scrollable
  633:    */
  634:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
  635:     defaultValue = "False")
  636:   @SimpleProperty
  637:   public void Scrollable(boolean scrollable) {
  638:     if (this.scrollable == scrollable && frameLayout != null) {
  639:       return;
  640:     }
  641:
  642:     // Remove our view from the current frameLayout.
  643:     if (frameLayout != null) {
  644:       frameLayout.removeAllViews();
  645:     }
  646:
  647:     this.scrollable = scrollable;
  648:
  649:     frameLayout = scrollable ? new ScrollView(this) : new FrameLayout(this);
  650:     frameLayout.addView(viewLayout.getLayoutManager(), new ViewGroup.LayoutParams(
  651:         ViewGroup.LayoutParams.MATCH_PARENT,
  652:         ViewGroup.LayoutParams.MATCH_PARENT));
  653:
  654:     frameLayout.setBackgroundColor(backgroundColor);
  655:     if (backgroundDrawable != null) {
  656:       ViewUtil.setBackgroundImage(frameLayout, backgroundDrawable);
  657:     }
  658:
  659:     setContentView(frameLayout);
  660:     frameLayout.requestLayout();
  661:   }
  662:
  663:   /**
  664:    * Returns the path of the background image.
  665:    *
  666:    * @return  the path of the background image
  667:    */
  668:   /*@SimpleProperty(
  669:       category = PropertyCategory.APPEARANCE,
  670:       description = "The screen background image.")*/
  671:   public String BackgroundImage() {
  672:     return backgroundImagePath;
  673:   }
  674:
  675:
  676:   /**
  677:    * Specifies the path of the background image.
  678:    *
  679:    * <p/>See {@link MediaUtil#determineMediaSource} for information about what
  680:    * a path can be.
  681:    *
  682:    * @param path the path of the background image
  683:    */
  684:   /*@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_ASSET,
  685:       defaultValue = "")
  686:   @SimpleProperty(
  687:       category = PropertyCategory.APPEARANCE,
  688:       description = "The screen background image.")*/
  689:   public void BackgroundImage(String path) {
  690:     backgroundImagePath = (path == null) ? "" : path;
  691:
  692:     try {
  693:       backgroundDrawable = MediaUtil.getBitmapDrawable(this, backgroundImagePath);
  694:     } catch (IOException ioe) {
  695:       Log.e(LOG_TAG, "Unable to load " + backgroundImagePath);
  696:       backgroundDrawable = null;
  697:     }
  698:
  699:     ViewUtil.setBackgroundImage(frameLayout, backgroundDrawable);
  700:     frameLayout.invalidate();
```

```
  }

  /**
   * Title property getter method.
   *
   * @return  form caption
   */
  /*@SimpleProperty(category = PropertyCategory.APPEARANCE,
      description = "The caption for the form, which apears in the title bar")*/
  public String Title() {
    return getTitle().toString();
  }

  /**
   * Title property setter method: sets a new caption for the form in the
   * form's title bar.
   *
   * @param title  new form caption
   */
  /*@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
      defaultValue = "")
  @SimpleProperty*/
  public void Title(String title) {
    setTitle(title);
  }


  /**
   * AboutScreen property getter method.
   *
   * @return  AboutScreen string
   */
  /*@SimpleProperty(category = PropertyCategory.APPEARANCE,
      description = "Information about the screen.  It appears when \"About this App
lication\" "
      + "is selected from the system menu. Use it to inform people about your app.
In multiple "
      + "screen apps, each screen has its own AboutScreen info.")*/
  public String AboutScreen() {
    return aboutScreen;
  }

  /**
   * AboutScreen property setter method: sets a new aboutApp string for the form in
the
   * form's "About this application" menu.
   *
   * @param title  new form caption
   */
  /*@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_TEXTAREA,
      defaultValue = "")
  @SimpleProperty*/
  public void AboutScreen(String aboutScreen) {
    this.aboutScreen = aboutScreen;
  }

  /**
   * The requested screen orientation. Commonly used values are
      unspecified (-1), landscape (0), portrait (1), sensor (4), and user (2).  " +
      "See the Android developer documentation for ActivityInfo.Screen_Orientation f
or the " +
      "complete list of possible settings.
   *
   * ScreenOrientation property getter method.
   *
   * @return  screen orientation
   */
  /*@SimpleProperty(category = PropertyCategory.APPEARANCE,
      description = "The requested screen orientation, specified as a text value.  "
+
      "Commonly used values are " +
      "landscape, portrait, sensor, user and unspecified.  " +
      "See the Android developer documentation for ActivityInfo.Screen_Orientation f
or the " +
      "complete list of possible settings.")*/
  public String ScreenOrientation() {
    switch (getRequestedOrientation()) {
      case ActivityInfo.SCREEN_ORIENTATION_BEHIND:
        return "behind";
      case ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE:
        return "landscape";
      case ActivityInfo.SCREEN_ORIENTATION_NOSENSOR:
        return "nosensor";
      case ActivityInfo.SCREEN_ORIENTATION_PORTRAIT:
        return "portrait";
      case ActivityInfo.SCREEN_ORIENTATION_SENSOR:
        return "sensor";
      case ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED:
        return "unspecified";
    }

    return "unspecified";
  }

  /**
   * ScreenOrientation property setter method: sets the screen orientation for
   * the form.
   *
   * @param screenOrientation  the screen orientation as a string
   */
  /*@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SCREEN_ORIENT
ATION,
      defaultValue = "unspecified")
  @SimpleProperty(category = PropertyCategory.APPEARANCE)*/
  public void ScreenOrientation(String screenOrientation) {
    if (screenOrientation.equalsIgnoreCase("behind")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_BEHIND);
    } else if (screenOrientation.equalsIgnoreCase("landscape")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    } else if (screenOrientation.equalsIgnoreCase("nosensor")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_NOSENSOR);
    } else if (screenOrientation.equalsIgnoreCase("portrait")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    } else if (screenOrientation.equalsIgnoreCase("sensor")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_SENSOR);
    } else if (screenOrientation.equalsIgnoreCase("unspecified")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED);
    } else if (screenOrientation.equalsIgnoreCase("user")) {
      setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_USER);
    } else if (SdkLevel.getLevel() >= SdkLevel.LEVEL_GINGERBREAD) {
      if (screenOrientation.equalsIgnoreCase("fullSensor")) {
        setRequestedOrientation(10); // ActivityInfo.SCREEN_ORIENTATION_FULL_SENSOR
      } else if (screenOrientation.equalsIgnoreCase("reverseLandscape")) {
        setRequestedOrientation(8); // ActivityInfo.SCREEN_ORIENTATION_REVERSE_LANDS
```

```
CAPE
  818:       } else if (screenOrientation.equalsIgnoreCase("reversePortrait")) {
  819:         setRequestedOrientation(9); // ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTR
AIT
  820:       } else if (screenOrientation.equalsIgnoreCase("sensorLandscape")) {
  821:         setRequestedOrientation(6); // ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSC
APE
  822:       } else if (screenOrientation.equalsIgnoreCase("sensorPortrait")) {
  823:         setRequestedOrientation(7); // ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRA
IT
  824:       } else {
  825:         dispatchErrorOccurredEvent(this, "ScreenOrientation",
  826:             ErrorMessages.ERROR_INVALID_SCREEN_ORIENTATION, screenOrientation);
  827:       }
  828:     } else {
  829:       dispatchErrorOccurredEvent(this, "ScreenOrientation",
  830:           ErrorMessages.ERROR_INVALID_SCREEN_ORIENTATION, screenOrientation);
  831:     }
  832:   }
  833:
  834:
  835:   // Note(halabelson): This section on centering is duplicated between Form and HVAr
rangement
  836:   // I did not see a clean way to abstract it.  Someone should have a look.
  837:
  838:   // Note(halabelson): The numeric encodings of the alignment specifications are spe
cified
  839:   // in ComponentConstants
  840:
  841:   /**
  842:    * Returns a number that encodes how contents of the arrangement are aligned vertic
ally.
  843:    * The choices are: 1 = top, 2 = vertically centered, 3 = aligned at the bottom.
  844:    * Vertical alignment has no effect if the screen is scrollable.
  845:    */
  846:   @SimpleProperty(
  847:       category = PropertyCategory.APPEARANCE,
  848:       description = "A number that encodes how the contents of the arrangement are al
igned " +
  849:       "vertically. The choices are: 1 = aligned at the top, 2 = vertically centered,
" +
  850:       "3 = aligned at the bottom. Vertical alignment has no effect if the screen is s
crollable.")
  851:   public int AlignVertical() {
  852:     return verticalAlignment;
  853:   }
  854:
  855:   /**
  856:    * Sets the vertical alignment for contents of the screen
  857:    *
  858:    * @param alignment
  859:    */
  860:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_VERTICAL_ALIGNME
NT,
  861:       defaultValue = ComponentConstants.VERTICAL_ALIGNMENT_DEFAULT + "")
  862:   @SimpleProperty
  863:   public void AlignVertical(int alignment) {
  864:     try {
  865:       // notice that the throw will prevent the alignment from being changed
  866:       // if the argument is illegal
  867:       alignmentSetter.setVerticalAlignment(alignment);
  868:       verticalAlignment = alignment;
  869:     } catch (IllegalArgumentException e) {
  870:       this.dispatchErrorOccurredEvent(this, "VerticalAlignment",
  871:           ErrorMessages.ERROR_BAD_VALUE_FOR_VERTICAL_ALIGNMENT, alignment);
  872:     }
  873:   }
  874:
  875:   /*
  876:    * Used by ListPicker, and ActivityStarter to get this Form's current opening tran
sition
  877:    * animation
  878:    */
  879:   public String getOpenAnimType() {
  880:     return openAnimType;
  881:   }
  882:   /**
  883:    * Specifies the App Name.
  884:    *
  885:    * @param aName the display name of the installed application in the phone
  886:    */
  887:   /*@DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_STRING,
  888:     defaultValue = "")
  889:   @SimpleProperty(userVisible = false,
  890:     description = "This is the display name of the installed application in the phon
e." +
  891:                   "If the AppName is blank, it will be set to the name of the project
when the project is built.")*/
  892:   public void AppName(String aName) {
  893:     // We don't actually need to do anything.
  894:   }
  895:
  896:   /**
  897:    * Width property getter method.
  898:    *
  899:    * @return  width property used by the layout
  900:    */
  901:   /*@SimpleProperty(category = PropertyCategory.APPEARANCE,
  902:     description = "Screen width (x-size).")*/
  903:   public int Width() {
  904:     return frameLayout.getWidth();
  905:   }
  906:
  907:   /**
  908:    * Height property getter method.
  909:    *
  910:    * @return  height property used by the layout
  911:    */
  912:   /*@SimpleProperty(category = PropertyCategory.APPEARANCE,
  913:     description = "Screen height (y-size).")*/
  914:   public int Height() {
  915:     return frameLayout.getHeight();
  916:   }
  917:
  918:   /**
  919:    * Display a new form.
  920:    *
  921:    * @param nextFormName the name of the new form to display
  922:    */
  923:   // This is called from runtime.scm when a "open another screen" block is executed.
  924:   public static void switchForm(String nextFormName) {
  925:     if (activeForm != null) {
  926:       activeForm.startNewForm(nextFormName, null);
  927:     } else {
```

```
  928:       throw new IllegalStateException("activeForm is null");
  929:     }
  930:   }
  931:
  932:   /**
  933:    * Display a new form and pass a startup value to the new form.
  934:    *
  935:    * @param nextFormName the name of the new form to display
  936:    * @param startValue the start value to pass to the new form
  937:    */
  938:   // This is called from runtime.scm when a "open another screen with start value" b
lock is
  939:   // executed.  Note that startNewForm will JSON encode the start value
  940:   public static void switchFormWithStartValue(String nextFormName, Object startValue
) {
  941:     Log.i(LOG_TAG, "Open another screen with start value:" + nextFormName);
  942:     if (activeForm != null) {
  943:       activeForm.startNewForm(nextFormName, startValue);
  944:       } else {
  945:         throw new IllegalStateException("activeForm is null");
  946:     }
  947:   }
  948:
  949:   // This JSON encodes the startup value
  950:   protected void startNewForm(String nextFormName, Object startupValue) {
  951:     Log.i(LOG_TAG, "startNewForm:" + nextFormName);
  952:     Intent activityIntent = new Intent();
  953:     // Note that the following is dependent on form generated class names being the
same as
  954:     // their form names and all forms being in the same package.
  955:     activityIntent.setClassName(this, getPackageName() + "." + nextFormName);
  956:     String functionName = (startupValue == null) ? "open another screen" :
  957:       "open another screen with start value";
  958:     String jValue;
  959:     if (startupValue != null) {
  960:       Log.i(LOG_TAG, "StartNewForm about to JSON encode:" + startupValue);
  961:       jValue = jsonEncodeForForm(startupValue, functionName);
  962:       Log.i(LOG_TAG, "StartNewForm got JSON encoding:" + jValue);
  963:     } else{
  964:       jValue = "";
  965:     }
  966:     activityIntent.putExtra(ARGUMENT_NAME, jValue);
  967:     // Save the nextFormName so that it can be passed to the OtherScreenClosed event
 in the
  968:     // future.
  969:     this.nextFormName = nextFormName;
  970:     Log.i(LOG_TAG, "about to start new form" + nextFormName);
  971:     try {
  972:       Log.i(LOG_TAG, "startNewForm starting activity:" + activityIntent);
  973:       startActivityForResult(activityIntent, SWITCH_FORM_REQUEST_CODE);
  974:       AnimationUtil.ApplyOpenScreenAnimation(this, openAnimType);
  975:     } catch (ActivityNotFoundException e) {
  976:       dispatchErrorOccurredEvent(this, functionName,
  977:           ErrorMessages.ERROR_SCREEN_NOT_FOUND, nextFormName);
  978:     }
  979:   }
  980:
  981:   // functionName is used for including in the error message to be shown
  982:   // if the JSON encoding fails
  983:   protected static String jsonEncodeForForm(Object value, String functionName) {
  984:     String jsonResult = "";
  985:     Log.i(LOG_TAG, "jsonEncodeForForm -- creating JSON representation:" + value.toSt
ring());
  986:     try {
  987:       // TODO(hal): check that this is OK for raw strings
  988:       jsonResult = JsonUtil.getJsonRepresentation(value);
  989:       Log.i(LOG_TAG, "jsonEncodeForForm -- got JSON representation:" + jsonResult);
  990:     } catch (JSONException e) {
  991:       activeForm.dispatchErrorOccurredEvent(activeForm, functionName,
  992:           // showing the bad value here will produce an ugly error on the phone, but
 it's
  993:           // more useful than not showing the value
  994:           ErrorMessages.ERROR_SCREEN_BAD_VALUE_FOR_SENDING, value.toString());
  995:     }
  996:     return jsonResult;
  997:   }
  998:
  999:   /*@SimpleEvent(description = "Event raised when another screen has closed and cont
rol has " +
 1000:       "returned to this screen.")*/
 1001:   public void OtherScreenClosed(String otherScreenName, Object result) {
 1002:     Log.i(LOG_TAG, "Form " + formName + " OtherScreenClosed, otherScreenName = " +
 1003:         otherScreenName + ", result = " + result.toString());
 1004:     EventDispatcher.dispatchEvent(this, "OtherScreenClosed", otherScreenName, result
);
 1005:   }
 1006:
 1007:
 1008:   // Component implementation
 1009:
 1010:   @Override
 1011:   public HandlesEventDispatching getDispatchDelegate() {
 1012:     return this;
 1013:   }
 1014:
 1015:   // ComponentContainer implementation
 1016:
 1017:   @Override
 1018:   public Activity $context() {
 1019:     return this;
 1020:   }
 1021:
 1022:   @Override
 1023:   public Form $form() {
 1024:     return this;
 1025:   }
 1026:
 1027:   @Override
 1028:   public void $add(AndroidViewComponent component) {
 1029:     viewLayout.add(component);
 1030:   }
 1031:
 1032:   @Override
 1033:   public void setChildWidth(AndroidViewComponent component, int width) {
 1034:     // A form is a vertical layout.
 1035:     ViewUtil.setChildWidthForVerticalLayout(component.getView(), width);
 1036:   }
 1037:
 1038:   @Override
 1039:   public void setChildHeight(AndroidViewComponent component, int height) {
 1040:     // A form is a vertical layout.
 1041:     ViewUtil.setChildHeightForVerticalLayout(component.getView(), height);
 1042:   }
 1043:
```

```
1044:   /*
1045:    * This is called from runtime.scm at the beginning of each event handler.
1046:    * It allows runtime.scm to know which form environment should be used for
1047:    * looking up symbols. The active form is the form that is currently
1048:    * (or was most recently) dispatching an event.
1049:    */
1050:   public static Form getActiveForm() {
1051:     return activeForm;
1052:   }
1053:
1054:
1055:   /**
1056:    * Returns the string that was passed to this screen when it was opened
1057:    *
1058:    * @return StartupText
1059:    */
1060:   // This is called from runtime.scm when a "get plain start text" block is executed
.
1061:   public static String getStartText() {
1062:     if (activeForm != null) {
1063:       return activeForm.startupValue;
1064:     } else {
1065:       throw new IllegalStateException("activeForm is null");
1066:     }
1067:   }
1068:
1069:   /**
1070:    * Returns the value that was passed to this screen when it was opened
1071:    *
1072:    * @return StartValue
1073:    */
1074:   // TODO(hal): cache this?
1075:   // Note: This is called as a primitive from runtime.scm and it returns an arbitrar
y Java object.
1076:   // Therefore it must be explicitly sanitized by runtime, unlike methods, which
1077:   // are sanitized via call-component-method.
1078:   public static Object getStartValue() {
1079:     if (activeForm != null) {
1080:       return decodeJSONStringForForm(activeForm.startupValue, "get start value");
1081:     } else {
1082:       throw new IllegalStateException("activeForm is null");
1083:     }
1084:   }
1085:
1086:
1087:   /**
1088:    * Closes the current screen, as opposed to finishApplication, which
1089:    * exits the entire application.
1090:    */
1091:   // This is called from runtime.scm when a "close screen" block is executed.
1092:   public static void finishActivity() {
1093:     if (activeForm != null) {
1094:       activeForm.closeForm(null);
1095:     } else {
1096:       throw new IllegalStateException("activeForm is null");
1097:     }
1098:   }
1099:
1100:   // This is called from runtime.scm when a "close screen with value" block is execu
ted.
1101:   public static void finishActivityWithResult(Object result) {
1102:     if (activeForm != null) {
1103:       if (activeForm instanceof ReplForm) {
1104:         ((ReplForm)activeForm).setResult(result);
1105:         activeForm.closeForm(null);        // This will call RetValManager.popScreen
()
1106:       } else {
1107:         String jString = jsonEncodeForForm(result, "close screen with value");
1108:         Intent resultIntent = new Intent();
1109:         resultIntent.putExtra(RESULT_NAME, jString);
1110:         activeForm.closeForm(resultIntent);
1111:       }
1112:     } else {
1113:       throw new IllegalStateException("activeForm is null");
1114:     }
1115:   }
1116:
1117:   // This is called from runtime.scm when a "close screen with plain text" block is
executed.
1118:   public static void finishActivityWithTextResult(String result) {
1119:     if (activeForm != null) {
1120:       Intent resultIntent = new Intent();
1121:       resultIntent.putExtra(RESULT_NAME, result);
1122:       activeForm.closeForm(resultIntent);
1123:     } else {
1124:       throw new IllegalStateException("activeForm is null");
1125:     }
1126:   }
1127:
1128:
1129:   protected void closeForm(Intent resultIntent) {
1130:     if (resultIntent != null) {
1131:       setResult(Activity.RESULT_OK, resultIntent);
1132:     }
1133:     finish();
1134:     AnimationUtil.ApplyCloseScreenAnimation(this, closeAnimType);
1135:   }
1136:
1137:   // This is called from runtime.scm when a "close application" block is executed.
1138:   public static void finishApplication() {
1139:     if (activeForm != null) {
1140:       activeForm.closeApplicationFromBlocks();
1141:     } else {
1142:       throw new IllegalStateException("activeForm is null");
1143:     }
1144:   }
1145:
1146:   protected void closeApplicationFromBlocks() {
1147:     closeApplication();
1148:   }
1149:
1150:   private void closeApplicationFromMenu() {
1151:     closeApplication();
1152:   }
1153:
1154:   private void closeApplication() {
1155:     // In a multi-screen application, only Screen1 can successfully call System.exit
(0). Here, we
1156:     // set applicationIsBeingClosed to true. If this is not Screen1, when we call fi
nish() below,
1157:     // the previous form's onResume method will be called. In onResume, we check
1158:     // applicationIsBeingClosed and call closeApplication again. The stack of forms
will unwind
1159:     // until we get back to Screen1; then we'll call System.exit(0) below.
```

```
1160:     applicationIsBeingClosed = true;
1161:
1162:     finish();
1163:
1164:     if (formName.equals("Screen1")) {
1165:       // I know that this is frowned upon in Android circles but I really think that it's
1166:       // confusing to users if the exit button doesn't really stop everything, including other
1167:       // forms in the app (when we support them), non-UI threads, etc.  We might need to be
1168:       // careful about this is we ever support services that start up on boot (since it might
1169:       // mean that the only way to restart that service) is to reboot but that's a long way off.
1170:       System.exit(0);
1171:     }
1172:   }
1173:
1174:   // Configure the system menu to include items to kill the application and to show "about"
1175:   // information
1176:
1177:   @Override
1178:   public boolean onCreateOptionsMenu(Menu menu) {
1179:     // This procedure is called only once.  To change the items dynamically
1180:     // we would use onPrepareOptionsMenu.
1181:     super.onCreateOptionsMenu(menu);
1182:     // add the menu items
1183:     // Comment out the next line if we don't want the exit button
1184:     addExitButtonToMenu(menu);
1185:     addAboutInfoToMenu(menu);
1186:     return true;
1187:   }
1188:
1189:   public void addExitButtonToMenu(Menu menu) {
1190:     MenuItem stopApplicationItem = menu.add(Menu.NONE, Menu.NONE, Menu.FIRST,
1191:     "Stop this application")
1192:     .setOnMenuItemClickListener(new OnMenuItemClickListener() {
1193:       public boolean onMenuItemClick(MenuItem item) {
1194:         showExitApplicationNotification();
1195:         return true;
1196:       }
1197:     });
1198:     stopApplicationItem.setIcon(android.R.drawable.ic_notification_clear_all);
1199:   }
1200:
1201:   public void addAboutInfoToMenu(Menu menu) {
1202:     MenuItem aboutAppItem = menu.add(Menu.NONE, Menu.NONE, 2,
1203:     "About this application")
1204:     .setOnMenuItemClickListener(new OnMenuItemClickListener() {
1205:       public boolean onMenuItemClick(MenuItem item) {
1206:         showAboutApplicationNotification();
1207:         return true;
1208:       }
1209:     });
1210:     aboutAppItem.setIcon(android.R.drawable.sym_def_app_icon);
1211:   }
1212:
1213:   private void showExitApplicationNotification() {
1214:     String title = "Stop application?";
1215:     String message = "Stop this application and exit? You'll need to relaunch " +
1216:         "the application to use it again.";
1217:     String positiveButton = "Stop and exit";
1218:     String negativeButton = "Don't stop";
1219:     // These runnables are passed to twoButtonAlert.  They perform the corresponding actions
1220:     // when the button is pressed.   Here there's nothing to do for "don't stop" and cancel
1221:     Runnable stopApplication = new Runnable() {public void run () {closeApplicationFromMenu();}};
1222:     Runnable doNothing = new Runnable () {public void run() {}};
1223:     /*Notifier.twoButtonDialog(
1224:         this,
1225:         message,
1226:         title,
1227:         positiveButton,
1228:         negativeButton,
1229:         false, // cancelable is false
1230:         stopApplication,
1231:         doNothing,
1232:         doNothing);*/
1233:   }
1234:
1235:   private String yandexTranslateTagline = "";
1236:
1237:   void setYandexTranslateTagline(){
1238:     yandexTranslateTagline = "<p><small>Language translation powered by Yandex.Translate</small></p>";
1239:   }
1240:
1241:   private void showAboutApplicationNotification() {
1242:     String title = "About this app";
1243:     String MITtagline = "<p><small><em>Invented with MIT App Inventor<br>appinventor.mit.edu</em></small></p>";
1244:     // Users can hide the taglines by including an HTML open comment <!-- in the about screen message
1245:     String message = aboutScreen + MITtagline + yandexTranslateTagline;
1246:     message = message.replaceAll("\\n", "<br>"); // Allow for line breaks in the string.
1247:     String buttonText ="Got it";
1248:     //Notifier.oneButtonAlert(this, message, title, buttonText);
1249:   }
1250:
1251:   // This is called from clear-current-form in runtime.scm.
1252:   public void clear() {
1253:     viewLayout.getLayoutManager().removeAllViews();
1254:     // Set all screen properties to default values.
1255:     defaultPropertyValues();
1256:     screenInitialized = false;
1257:   }
1258:
1259:   public void deleteComponent(Object component) {
1260:     if (component instanceof OnStopListener) {
1261:       OnStopListener onStopListener = (OnStopListener) component;
1262:       if (onStopListeners.contains(onStopListener)) {
1263:         onStopListeners.remove(onStopListener);
1264:       }
1265:     }
1266:     if (component instanceof OnResumeListener) {
1267:       OnResumeListener onResumeListener = (OnResumeListener) component;
1268:       if (onResumeListeners.contains(onResumeListener)) {
1269:         onResumeListeners.remove(onResumeListener);
1270:       }
```

```
1271:     }
1272:     if (component instanceof OnPauseListener) {
1273:       OnPauseListener onPauseListener = (OnPauseListener) component;
1274:       if (onPauseListeners.contains(onPauseListener)) {
1275:         onPauseListeners.remove(onPauseListener);
1276:       }
1277:     }
1278:     if (component instanceof OnDestroyListener) {
1279:       OnDestroyListener onDestroyListener = (OnDestroyListener) component;
1280:       if (onDestroyListeners.contains(onDestroyListener)) {
1281:         onDestroyListeners.remove(onDestroyListener);
1282:       }
1283:     }
1284:     if (component instanceof Deleteable) {
1285:       ((Deleteable) component).onDelete();
1286:     }
1287:   }
1288:
1289:   public void dontGrabTouchEventsForComponent() {
1290:     // The following call results in the Form not grabbing our events and
1291:     // handling dragging on its own, which it wants to do to handle scrolling.
1292:     // Its effect only lasts long as the current set of motion events
1293:     // generated during this touch and drag sequence.  Consequently, if a
1294:     // component wants to handle dragging it needs to call this in the
1295:     // onTouchEvent of its View.
1296:     frameLayout.requestDisallowInterceptTouchEvent(true);
1297:   }
1298:
1299:
1300:   // This is used by Repl to throttle error messages which can get out of
1301:   // hand, e.g. if triggered by Accelerometer.
1302:   protected boolean toastAllowed() {
1303:     long now = System.nanoTime();
1304:     if (now > lastToastTime + minimumToastWait) {
1305:       lastToastTime = now;
1306:       return true;
1307:     }
1308:     return false;
1309:   }
1310:
1311:   // This is used by runtime.scm to call the Initialize of a component.
1312:   public void callInitialize(Object component) throws Throwable {
1313:     Method method;
1314:     try {
1315:       method = component.getClass().getMethod("Initialize", (Class<?>[]) null);
1316:     } catch (SecurityException e) {
1317:       Log.i(LOG_TAG, "Security exception " + e.getMessage());
1318:       return;
1319:     } catch (NoSuchMethodException e) {
1320:       //This is OK.
1321:       return;
1322:     }
1323:     try {
1324:       Log.i(LOG_TAG, "calling Initialize method for Object " + component.toString())
;
1325:       method.invoke(component, (Object[]) null);
1326:     } catch (InvocationTargetException e){
1327:       Log.i(LOG_TAG, "invoke exception: " + e.getMessage());
1328:       throw e.getTargetException();
1329:     }
1330:   }
1331:
1332:   /**
1333:    * Perform some action related to fullscreen video display.
1334:    * @param action
1335:    *          Can be any of the following:
1336:    *          <ul>
1337:    *          <li>
1338:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_DURATION}
1339:    *          </li>
1340:    *          <li>
1341:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_FULLSCREEN}
1342:    *          </li>
1343:    *          <li>
1344:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_PAUSE}
1345:    *          </li>
1346:    *          <li>
1347:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_PLAY}
1348:    *          </li>
1349:    *          <li>
1350:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_SEEK}
1351:    *          </li>
1352:    *          <li>
1353:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_SOURCE}
1354:    *          </li>
1355:    *          <li>
1356:    *          {@link com.google.appinventor.components.runtime.util.FullScreenVideoU
til#FULLSCREEN_VIDEO_ACTION_STOP}
1357:    *          </li>
1358:    *          </ul>
1359:    * @param source
1360:    *          The VideoPlayer to use in some actions.
1361:    * @param data
1362:    *          Used by the method. This object varies depending on the action.
1363:    * @return Varies depending on what action was passed in.
1364:    */
1365:   public synchronized Bundle fullScreenVideoAction(int action, VideoPlayer source, O
bject data) {
1366:     return fullScreenVideoUtil.performAction(action, source, data);
1367:   }
1368: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleFunction;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.annotations.UsesPermissions;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.ErrorMessages;

import android.content.Context;
import android.location.Address;
import android.location.Criteria;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.location.LocationProvider;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

/**
 * Sensor that can provide information on longitude, latitude, and altitude.
 *
 */
@DesignerComponent(version = YaVersion.LOCATIONSENSOR_COMPONENT_VERSION,
    description = "Non-visible component providing location information, " +
    "including longitude, latitude, altitude (if supported by the device), " +
    "and address.  This can also perform \"geocoding\", converting a given " +
    "address (not necessarily the current one) to a latitude (with the " +
    "<code>LatitudeFromAddress</code> method) and a longitude (with the " +
    "<code>LongitudeFromAddress</code> method).</p>\n" +
    "<p>In order to function, the component must have its " +
    "<code>Enabled</code> property set to True, and the device must have " +
    "location sensing enabled through wireless networks or GPS " +
    "satellites (if outdoors).</p>\n" +
    "Location information might not be immediately available when an app starts.  Yo
u'll have to wait a short time for " +
    "a location provider to be found and used, or wait for the OnLocationChanged eve
nt",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/locationSensor.png")
@SimpleObject
@UsesPermissions(permissionNames =
                 "android.permission.ACCESS_FINE_LOCATION," +
                 "android.permission.ACCESS_COARSE_LOCATION," +
                 "android.permission.ACCESS_MOCK_LOCATION," +
                 "android.permission.ACCESS_LOCATION_EXTRA_COMMANDS")
public class LocationSensor extends AndroidNonvisibleComponent
    implements Component, OnStopListener, OnResumeListener, Deleteable {

  /**
   * Class that listens for changes in location, raises appropriate events,
   * and provides properties.
   *
   */
  private class MyLocationListener implements LocationListener {
    @Override
    // This sets fields longitude, latitude, altitude, hasLocationData, and
    // hasAltitude, then calls LocationSensor.LocationChanged(), alll in the
    // enclosing class LocationSensor.
    public void onLocationChanged(Location location) {
      lastLocation = location;
      longitude = location.getLongitude();
      latitude = location.getLatitude();
      // If the current location doesn't have altitude information, the prior
      // altitude reading is retained.
      if (location.hasAltitude()) {
        hasAltitude = true;
        altitude = location.getAltitude();
      }
      hasLocationData = true;
      LocationChanged(latitude, longitude, altitude);
    }

    @Override
    public void onProviderDisabled(String provider) {
      StatusChanged(provider, "Disabled");
      stopListening();
      if (enabled) {
        RefreshProvider();
      }
    }

    @Override
    public void onProviderEnabled(String provider) {
      StatusChanged(provider, "Enabled");
      RefreshProvider();
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {
      switch (status) {
        // Ignore TEMPORARILY_UNAVAILABLE, because service usually returns quickly.
        case LocationProvider.TEMPORARILY_UNAVAILABLE:
          StatusChanged(provider, "TEMPORARILY_UNAVAILABLE");
          break;
        case LocationProvider.OUT_OF_SERVICE:
          // If the provider we were listening to is no longer available,
          // find another.
          StatusChanged(provider, "OUT_OF_SERVICE");

          if (provider.equals(providerName)) {
            stopListening();
            RefreshProvider();
          }
          break;
```

```
123:         case LocationProvider.AVAILABLE:
124:           // If another provider becomes available and is one we hadn't known
125:           // about see if it is better than the one we're currently using.
126:           StatusChanged(provider, "AVAILABLE");
127:           if (!provider.equals(providerName) &&
128:               !allProviders.contains(provider)) {
129:             RefreshProvider();
130:           }
131:           break;
132:       }
133:     }
134:   }
135:
136:   /**
137:    * Constant returned by {@link #Longitude()}, {@link #Latitude()}, and
138:    * {@link #Altitude()} if no value could be obtained for them.  The client
139:    * can find this out directly by calling {@link #HasLongitudeLatitude()} or
140:    * {@link #HasAltitude()}.
141:    */
142:   public static final int UNKNOWN_VALUE = 0;
143:
144:   // These variables contain information related to the LocationProvider.
145:   private final Criteria locationCriteria;
146:   private final Handler handler;
147:   private final LocationManager locationManager;
148:
149:   private boolean providerLocked = false; // if true we can't change providerName
150:   private String providerName;
151:   // Invariant: providerLocked => providerName is non-empty
152:
153:   private int timeInterval;
154:   private int distanceInterval;
155:
156:   private MyLocationListener myLocationListener;
157:
158:   private LocationProvider locationProvider;
159:   private boolean listening = false;
160:     // Invariant: listening <=> a myLocationListener is registered with locationMana
ger
161:     // Invariant: !listening <=> locationProvider == null
162:
163:   //This holds all the providers available when we last chose providerName.
164:   //The reported best provider is first, possibly duplicated.
165:   private List<String> allProviders;
166:
167:   // These location-related values are set in MyLocationListener.onLocationChanged()
.
168:   private Location lastLocation;
169:   private double longitude = UNKNOWN_VALUE;
170:   private double latitude = UNKNOWN_VALUE;
171:   private double altitude = UNKNOWN_VALUE;
172:   private boolean hasLocationData = false;
173:   private boolean hasAltitude = false;
174:
175:   // This is used in reverse geocoding.
176:   private Geocoder geocoder;
177:
178:   // User-settable properties
179:   private boolean enabled = true;  // the default value is true
180:
181:   /**
182:    * Creates a new LocationSensor component.
183:    *
184:    * @param container  ignored (because this is a non-visible component)
185:    */
186:   public LocationSensor(ComponentContainer container) {
187:     super(container.$form());
188:     handler = new Handler();
189:     // Set up listener
190:     form.registerForOnResume(this);
191:     form.registerForOnStop(this);
192:
193:     // Initialize sensor properties (60 seconds; 5 meters)
194:     timeInterval = 60000;
195:     distanceInterval = 5;
196:
197:     // Initialize location-related fields
198:     Context context = container.$context();
199:     geocoder = new Geocoder(context);
200:     locationManager = (LocationManager) context.getSystemService(Context.LOCATION_SE
RVICE);
201:     locationCriteria = new Criteria();
202:     myLocationListener = new MyLocationListener();
203:     allProviders = new ArrayList<String>();
204:     // Do some initialization depending on the initial enabled state
205:     Enabled(enabled);
206:   }
207:
208:   // Events
209:
210:   /**
211:    * Indicates that a new location has been detected.
212:    */
213:   //@SimpleEvent
214:   public void LocationChanged(double latitude, double longitude, double altitude) {
215:     if (enabled) {
216:       EventDispatcher.dispatchEvent(this, "LocationChanged", latitude, longitude, al
titude);
217:     }
218:   }
219:
220:   /**
221:    * Indicates that the status of the location provider service has changed, such as
 when a
222:    * provider is lost or a new provider starts being used.
223:    */
224:   //@SimpleEvent
225:   public void StatusChanged(String provider, String status) {
226:     if (enabled) {
227:       EventDispatcher.dispatchEvent(this, "StatusChanged", provider, status);
228:     }
229:   }
230:
231:   // Properties
232:
233:   /**
234:    * Indicates the source of the location information.  If there is no provider, the
235:    * string "NO PROVIDER" is returned.  This is useful primarily for debugging.
236:    */
237:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
238:   public String ProviderName() {
239:     if (providerName == null) {
240:       return "NO PROVIDER";
241:     } else {
```

```
  242:       return providerName;
  243:     }
  244:   }
  245:
  246:   /**
  247:    * Change the location provider.
  248:    * If the blocks program changes the name, try to change the provider.
  249:    * Whatever happens now, the provider and the reported name may be switched to
  250:    * Android's preferred provider later. This is primarily for debugging.
  251:    */
  252:   @SimpleProperty
  253:   public void ProviderName(String providerName) {
  254:     this.providerName = providerName;
  255:     if (!empty(providerName) && startProvider(providerName)) {
  256:       return;
  257:     } else {
  258:       RefreshProvider();
  259:     }
  260:   }
  261:
  262:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  263:   public boolean ProviderLocked() {
  264:     return providerLocked;
  265:   }
  266:
  267:   /**
  268:    * Indicates whether the sensor should allow the developer to
  269:    * manually change the provider (GPS, GSM, Wifi, etc.)
  270:    * from which location updates are received.
  271:    */
  272:   @SimpleProperty
  273:   public void ProviderLocked(boolean lock) {
  274:       providerLocked = lock;
  275:   }
  276:
  277:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SENSOR_TIME_INT
ERVAL,
  278:       defaultValue = "60000")
  279:   @SimpleProperty
  280:   public void TimeInterval(int interval) {
  281:
  282:       // make sure that the provided value is a valid one.
  283:       // choose 1000000 miliseconds to be the upper limit
  284:       if (interval < 0 || interval > 1000000)
  285:           return;
  286:
  287:       timeInterval = interval;
  288:
  289:       // restart listening for location updates, using the new time interval
  290:       if (enabled) {
  291:           RefreshProvider();
  292:       }
  293:   }
  294:
  295:   @SimpleProperty(
  296:       description = "Determines the minimum time interval, in milliseconds, that the
 sensor will try " +
  297:           "to use for sending out location updates. However, location updates will o
nly be received " +
  298:           "when the location of the phone actually changes, and use of the specified
 time interval " +
  299:           "is not guaranteed. For example, if 1000 is used as the time interval, loc
ation updates will " +
  300:           "never be fired sooner than 1000ms, but they may be fired anytime after.",
  301:           category = PropertyCategory.BEHAVIOR)
  302:   public int TimeInterval() {
  303:       return timeInterval;
  304:   }
  305:
  306:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_SENSOR_DIST_INT
ERVAL,
  307:       defaultValue = "5")
  308:   @SimpleProperty
  309:   public void DistanceInterval(int interval) {
  310:
  311:       // make sure that the provided value is a valid one.
  312:       // choose 1000 meters to be the upper limit
  313:       if (interval < 0 || interval > 1000)
  314:           return;
  315:
  316:       distanceInterval = interval;
  317:
  318:       // restart listening for location updates, using the new distance interval
  319:       if (enabled) {
  320:           RefreshProvider();
  321:       }
  322:   }
  323:
  324:   @SimpleProperty(
  325:       description = "Determines the minimum distance interval, in meters, that the s
ensor will try " +
  326:       "to use for sending out location updates. For example, if this is set to 5, th
en the sensor will " +
  327:       "fire a LocationChanged event only after 5 meters have been traversed. However
, the sensor does " +
  328:       "not guarantee that an update will be received at exactly the distance interva
l. It may take more " +
  329:       "than 5 meters to fire an event, for instance.",
  330:       category = PropertyCategory.BEHAVIOR)
  331:   public int DistanceInterval() {
  332:       return distanceInterval;
  333:   }
  334:
  335:   /**
  336:    * Indicates whether longitude and latitude information is available.  (It is
  337:    * always the case that either both or neither are.)
  338:    */
  339:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  340:   public boolean HasLongitudeLatitude() {
  341:     return hasLocationData && enabled;
  342:   }
  343:
  344:   /**
  345:    * Indicates whether altitude information is available.
  346:    */
  347:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  348:   public boolean HasAltitude() {
  349:     return hasAltitude && enabled;
  350:   }
  351:
  352:   /**
  353:    * Indicates whether information about location accuracy is available.
  354:    */
  355:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
```

```
356:   public boolean HasAccuracy() {
357:     return Accuracy() != UNKNOWN_VALUE && enabled;
358:   }
359:
360:   /**
361:    * The most recent available longitude value.  If no value is available,
362:    * 0 will be returned.
363:    */
364:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
365:   public double Longitude() {
366:     return longitude;
367:   }
368:
369:   /**
370:    * The most recently available latitude value.  If no value is available,
371:    * 0 will be returned.
372:    */
373:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
374:   public double Latitude() {
375:       return latitude;
376:   }
377:
378:   /**
379:    * The most recently available altitude value, in meters.  If no value is
380:    * available, 0 will be returned.
381:    */
382:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
383:   public double Altitude() {
384:     return altitude;
385:   }
386:
387:   /**
388:    * The most recent measure of accuracy, in meters.  If no value is available,
389:    * 0 will be returned.
390:    */
391:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
392:   public double Accuracy() {
393:     if (lastLocation != null && lastLocation.hasAccuracy()) {
394:       return lastLocation.getAccuracy();
395:     } else if (locationProvider != null) {
396:       return locationProvider.getAccuracy();
397:     } else {
398:       return UNKNOWN_VALUE;
399:     }
400:   }
401:
402:   /**
403:    * Indicates whether the user has specified that the sensor should
404:    * listen for location changes and raise the corresponding events.
405:    */
406:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
407:   public boolean Enabled() {
408:     return enabled;
409:   }
410:
411:   /**
412:    * Indicates whether the sensor should listen for location chagnes
413:    * and raise the corresponding events.
414:    */
415:   @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
416:       defaultValue = "True")
417:   @SimpleProperty
418:   public void Enabled(boolean enabled) {
419:     this.enabled = enabled;
420:     if (!enabled) {
421:       stopListening();
422:     } else {
423:       RefreshProvider();
424:     }
425:   }
426:
427:   /**
428:    * Provides a textual representation of the current address or
429:    * "No address available".
430:    */
431:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
432:   public String CurrentAddress() {
433:     if (hasLocationData &&
434:         latitude <= 90 && latitude >= -90 &&
435:         longitude <= 180 || longitude >= -180) {
436:       try {
437:         List<Address> addresses = geocoder.getFromLocation(latitude, longitude, 1);
438:         if (addresses != null && addresses.size() == 1) {
439:           Address address = addresses.get(0);
440:           if (address != null) {
441:             StringBuilder sb = new StringBuilder();
442:             for (int i = 0; i <= address.getMaxAddressLineIndex(); i++) {
443:               sb.append(address.getAddressLine(i));
444:               sb.append("\n");
445:             }
446:             return sb.toString();
447:           }
448:         }
449:
450:       } catch (Exception e) {
451:         // getFromLocation can throw an IOException or an IllegalArgumentException
452:         // a bad result can give an indexOutOfBoundsException
453:         // are there others?
454:         if (e instanceof IllegalArgumentException
455:             || e instanceof IOException
456:             || e instanceof IndexOutOfBoundsException ) {
457:           Log.e("LocationSensor", "Exception thrown by getting current address " + e
.getMessage());
458:         } else {
459:           // what other exceptions can happen here?
460:           Log.e("LocationSensor",
461:               "Unexpected exception thrown by getting current address " + e.getMessa
ge());
462:         }
463:       }
464:     }
465:     return "No address available";
466:   }
467:
468:   /**
469:    * Derives Latitude from Address
470:    *
471:    * @param locationName  human-readable address
472:    *
473:    * @return latitude in degrees, 0 if not found.
474:    */
475:   @SimpleFunction(description = "Derives latitude of given address")
476:   public double LatitudeFromAddress(String locationName) {
477:     try {
```

```
478:       List<Address> addressObjs = geocoder.getFromLocationName(locationName, 1);
479:       Log.i("LocationSensor", "latitude addressObjs size is " + addressObjs.size() +
 " for " + locationName);
480:       if ( (addressObjs == null) || (addressObjs.size() == 0) ){
481:         throw new IOException("");
482:       }
483:       return addressObjs.get(0).getLatitude();
484:     } catch (IOException e) {
485:       form.dispatchErrorOccurredEvent(this, "LatitudeFromAddress",
486:           ErrorMessages.ERROR_LOCATION_SENSOR_LATITUDE_NOT_FOUND, locationName);
487:       return 0;
488:     }
489:   }
490:
491:   /**
492:    * Derives Longitude from Address
493:    * @param locationName  human-readable address
494:    *
495:    * @return longitude in degrees, 0 if not found.
496:    */
497:   @SimpleFunction(description = "Derives longitude of given address")
498:   public double LongitudeFromAddress(String locationName) {
499:     try {
500:       List<Address> addressObjs = geocoder.getFromLocationName(locationName, 1);
501:       Log.i("LocationSensor", "longitude addressObjs size is " + addressObjs.size()
+ " for " + locationName);
502:       if ( (addressObjs == null) || (addressObjs.size() == 0) ){
503:         throw new IOException("");
504:       }
505:       return addressObjs.get(0).getLongitude();
506:     } catch (IOException e) {
507:       form.dispatchErrorOccurredEvent(this, "LongitudeFromAddress",
508:           ErrorMessages.ERROR_LOCATION_SENSOR_LONGITUDE_NOT_FOUND, locationName);
509:       return 0;
510:     }
511:   }
512:
513:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
514:   public List<String> AvailableProviders () {
515:     return allProviders;
516:   }
517:
518:   // Methods to stop and start listening to LocationProviders
519:
520:   /**
521:    * Refresh provider attempts to choose and start the best provider unless
522:    * someone has set and locked the provider. Currently, blocks programmers
523:    * cannot do that because the relevant methods are not declared as properties.
524:    *
525:    */
526:
527:   // @SimpleFunction(description = "Find and start listening to a location provider.
")
528:   public void RefreshProvider() {
529:     stopListening();             // In case another provider is active.
530:     if (providerLocked && !empty(providerName)) {
531:       listening = startProvider(providerName);
532:       return;
533:     }
534:     allProviders = locationManager.getProviders(true);  // Typically it's ("network"
 "gps")
535:     String bProviderName = locationManager.getBestProvider(locationCriteria, true);
536:     if (bProviderName != null && !bProviderName.equals(allProviders.get(0))) {
537:       allProviders.add(0, bProviderName);
538:     }
539:     // We'll now try the best first and stop as soon as one successfully starts.
540:     for (String providerN : allProviders) {
541:       listening = startProvider(providerN);
542:       if (listening) {
543:         if (!providerLocked) {
544:           providerName = providerN;
545:         }
546:         return;
547:       }
548:     }
549:   }
550:
551:   /* Start listening to ProviderName.
552:    * Return true iff successful.
553:    */
554:   private boolean startProvider(String providerName) {
555:     this.providerName = providerName;
556:     LocationProvider tLocationProvider = locationManager.getProvider(providerName);
557:     if (tLocationProvider == null) {
558:       Log.d("LocationSensor", "getProvider(" + providerName + ") returned null");
559:       return false;
560:     }
561:     stopListening();
562:     locationProvider = tLocationProvider;
563:     locationManager.requestLocationUpdates(providerName, timeInterval,
564:         distanceInterval, myLocationListener);
565:     listening = true;
566:     return true;
567:   }
568:
569:   /**
570:    * This unregisters {@link #myLocationListener} as a listener to location
571:    * updates.  It is safe to call this even if no listener had been registered,
572:    * in which case it has no effect.  This also sets the value of
573:    * {@link #locationProvider} to {@code null} and sets {@link #listening}
574:    * to {@code false}.
575:    */
576:   private void stopListening() {
577:     if (listening) {
578:       locationManager.removeUpdates(myLocationListener);
579:       locationProvider = null;
580:       listening = false;
581:     }
582:   }
583:
584:
585:   // OnResumeListener implementation
586:
587:   @Override
588:   public void onResume() {
589:     if (enabled) {
590:       RefreshProvider();
591:     }
592:   }
593:
594:   // OnStopListener implementation
595:
596:   @Override
597:   public void onStop() {
```

```
598:     stopListening();
599:   }
600:
601:   // Deleteable implementation
602:
603:   @Override
604:   public void onDelete() {
605:     stopListening();
606:   }
607:
608:   private boolean empty(String s) {
609:     return s == null || s.length() == 0;
610:   }
611: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2012 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import com.google.appinventor.components.annotations.DesignerComponent;
import com.google.appinventor.components.annotations.DesignerProperty;
import com.google.appinventor.components.annotations.PropertyCategory;
import com.google.appinventor.components.annotations.SimpleEvent;
import com.google.appinventor.components.annotations.SimpleObject;
import com.google.appinventor.components.annotations.SimpleProperty;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import com.google.appinventor.components.runtime.util.FroyoUtil;
import com.google.appinventor.components.runtime.util.OrientationSensorUtil;
import com.google.appinventor.components.runtime.util.SdkLevel;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.util.Log;
import android.view.Display;
import android.view.Surface;
import android.view.WindowManager;

/**
 * Sensor that can measure absolute orientation in 3 dimensions.
 *
 */
@DesignerComponent(version = YaVersion.ORIENTATIONSENSOR_COMPONENT_VERSION,
    description = "<p>Non-visible component providing information about the " +
    "device's physical orientation in three dimensions: <ul> " +
    "<li> <strong>Roll</strong>: 0 degrees when the device is level, increases to "
+
    "     90 degrees as the device is tilted up on its left side, and " +
    "     decreases to -90 degrees when the device is tilted up on its right side. "
 +
    "     </li> " +
    "<li> <strong>Pitch</strong>: 0 degrees when the device is level, up to " +
    "     90 degrees as the device is tilted so its top is pointing down, " +
    "     up to 180 degrees as it gets turned over.  Similarly, as the device " +
    "     is tilted so its bottom points down, pitch decreases to -90 " +
    "     degrees, then further decreases to -180 degrees as it gets turned all the
way " +
    "     over.</li> " +
    "<li> <strong>Azimuth</strong>: 0 degrees when the top of the device is " +
    "     pointing north, 90 degrees when it is pointing east, 180 degrees " +
    "     when it is pointing south, 270 degrees when it is pointing west, " +
    "     etc.</li></ul>" +
    "     These measurements assume that the device itself is not moving.</p>",
    category = ComponentCategory.SENSORS,
    nonVisible = true,
    iconName = "images/orientationsensor.png")

@SimpleObject
public class OrientationSensor extends AndroidNonvisibleComponent
    implements SensorEventListener, Deleteable, OnPauseListener, OnResumeListener {
  // Constants
  private static final String LOG_TAG = "OrientationSensor";
  // offsets in array returned by SensorManager.getOrientation()
  private static final int AZIMUTH = 0;
  private static final int PITCH = 1;
  private static final int ROLL = 2;
  private static final int DIMENSIONS = 3;  // Warning: specific to our universe

  // Properties
  private boolean enabled;
  private float azimuth;    // degrees
  private float pitch;  // degrees
  private float roll;   // degrees
  private int accuracy;

  // Sensor information
  private final SensorManager sensorManager;
  private final Sensor accelerometerSensor;
  private final Sensor magneticFieldSensor;
  private boolean listening;

  // Pre-allocated arrays to hold sensor data so that we don't cause so many garbage
collections
  // while processing sensor events. All are used only in onSensorChanged.
  private final float[] accels = new float[DIMENSIONS];  // acceleration vector
  private final float[] mags = new float[DIMENSIONS];    // magnetic field vector

  // Flags to tell whether the above arrays are filled. They are set in onSensorChan
ged and cleared
  // in stopListening.
  private boolean accelsFilled;
  private boolean magsFilled;

  // Pre-allocated matrixes used to compute orientation values from acceleration and
magnetic
  // field data.
  private final float[] rotationMatrix = new float[DIMENSIONS * DIMENSIONS];
  private final float[] inclinationMatrix = new float[DIMENSIONS * DIMENSIONS];
  private final float[] values = new float[DIMENSIONS];

  /**
   * Creates a new OrientationSensor component.
   *
   * @param container  ignored (because this is a non-visible component)
   */
  public OrientationSensor(ComponentContainer container) {
    super(container.$form());

    // Get sensors, and start listening.
    sensorManager =
      (SensorManager) container.$context().getSystemService(Context.SENSOR_SERVICE);
    accelerometerSensor = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    magneticFieldSensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD)
;

    // Begin listening in onResume() and stop listening in onPause().
    form.registerForOnResume(this);
    form.registerForOnPause(this);

    // Set default property values.
    Enabled(true);
```

```
  }

  private void startListening() {
    if (!listening) {
      sensorManager.registerListener(this, accelerometerSensor,
                                     SensorManager.SENSOR_DELAY_NORMAL);
      sensorManager.registerListener(this, magneticFieldSensor,
                                     SensorManager.SENSOR_DELAY_NORMAL);
      listening = true;
    }
  }

  private void stopListening() {
    if (listening) {
      sensorManager.unregisterListener(this);
      listening = false;

      // Throw out sensor information that will go stale.
      accelsFilled = false;
      magsFilled = false;
    }
  }

  // Events

  /**
   * Default OrientationChanged event handler.
   *
   * <p>This event is signalled when the device's orientation has changed.  It
   * reports the new values of azimuth, pich, and roll, and it also sets the Azimuth
, Pitch,
   * and roll properties.</p>
   * <p>Azimuth is the compass heading in degrees, pitch indicates how the device
   * is tilted from top to bottom, and roll indicates how much the device is tilted
from
   * side to side.</p>
   */
  //@SimpleEvent
  public void OrientationChanged(float azimuth, float pitch, float roll) {
    EventDispatcher.dispatchEvent(this, "OrientationChanged", azimuth, pitch, roll);
  }

  // Properties

  /**
   * Available property getter method (read-only property).
   *
   * @return {@code true} indicates that an orientation sensor is available,
   *         {@code false} that it isn't
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public boolean Available() {
    return sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER).size() > 0
        && sensorManager.getSensorList(Sensor.TYPE_MAGNETIC_FIELD).size() > 0;
  }

  /**
   * Enabled property getter method.
   *
   * @return {@code true} indicates that the sensor generates events,
   *         {@code false} that it doesn't
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public boolean Enabled() {
    return enabled;
  }

  /**
   * Enabled property setter method.
   *
   * @param enabled  {@code true} enables sensor event generation,
   *                 {@code false} disables it
   */
  @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN,
      defaultValue = "True")
  @SimpleProperty
  public void Enabled(boolean enabled) {
    if (this.enabled != enabled) {
      this.enabled = enabled;
      if (enabled) {
        startListening();
      } else {
        stopListening();
      }
    }
  }

  /**
   * Pitch property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current pitch
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Pitch() {
    return pitch;
  }

  /**
   * Roll property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current roll
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Roll() {
    return roll;
  }

  /**
   * Azimuth property getter method (read-only property).
   *
   * <p>To return meaningful values the sensor must be enabled.</p>
   *
   * @return  current azimuth
   */
  @SimpleProperty(category = PropertyCategory.BEHAVIOR)
  public float Azimuth() {
    return azimuth;
  }

  /**
```

```
240:    * <p>Angle property getter method (read-only property).  Specifically, this
241:    * provides the angle in which the orientation sensor is tilted, treating
242:    * -{@link #Roll()} as the x-coordinate and {@link #Pitch()} as the
243:    * y-coordinate.  For the amount of the tilt, use {@link #Magnitude()}.</p>
244:    *
245:    * <p>To return meaningful values the sensor must be enabled.</p>
246:    *
247:    * @return the angle in degrees
248:    */
249:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
250:   public float Angle() {
251:     return OrientationSensor.computeAngle(pitch, roll);
252:   }
253:
254:   /**
255:    * Computes the angle the phone is tilted.  This has been lifted out
256:    * of {@link #Angle()} for ease of testing.
257:    *
258:    * @param pitch an angle indicating how far the device is tilted vertically,
259:    *              with a value of +90 degrees if the top is pointing straight
260:    *              down, +180 degrees if the phone is entirely turned over,
261:    *              -90/+270 degrees if the top is pointing straight up, etc.
262:    * @param roll an angle indicating how far the device is tilted horizontally,
263:    *             with a value of +90 degrees if it is tilted entirely on its
264:    *             left side, -90 degrees if it is tilted entirely on its right
265:    *             side; the maximum absolute value of roll is 90 degrees, after
266:    *             which it decreases back toward 0 (flat face-up or face-down).
267:    *
268:    * @returns the corresonding angle in the range [-180, +180] degrees
269:    */
270:   static float computeAngle(float pitch, float roll) {
271:     return (float) Math.toDegrees(Math.atan2(Math.toRadians(pitch),
272:                                              // invert roll to correct sign
273:                                              -Math.toRadians(roll)));
274:   }
275:
276:   /**
277:    * Magnitude property getter method (read-only property).  Specifically, this
278:    * returns a number between 0 and 1, indicating how much the device
279:    * is tilted.  For the angle of tilt, use {@link #Angle()}.
280:    *
281:    * <p>To return meaningful values the sensor must be enabled.</p>
282:    *
283:    * @return the magnitude of the tilt, from 0 to 1
284:    */
285:   @SimpleProperty(category = PropertyCategory.BEHAVIOR)
286:   public float Magnitude() {
287:     // Limit pitch and roll to 90; otherwise, the phone is upside down.
288:     // The official documentation falsely claims that the range of pitch and
289:     // roll is [-90, 90].  If the device is upside-down, it can range from
290:     // -180 to 180.  We restrict it to the range [-90, 90].
291:     // With that restriction, if the pitch and roll angles are P and R, then
292:     // the force is given by 1 - cos(P)cos(R).  I have found a truly wonderful
293:     // proof of this theorem, but the margin enforced by Lint is too small to
294:     // contain it.
295:     final int MAX_VALUE = 90;
296:     double npitch = Math.toRadians(Math.min(MAX_VALUE, Math.abs(pitch)));
297:     double nroll = Math.toRadians(Math.min(MAX_VALUE, Math.abs(roll)));
298:     return (float) (1.0 - Math.cos(npitch) * Math.cos(nroll));
299:   }
300:
301:   // SensorListener implementation
302:
303:   /*
304:    * Returns the rotation of the screen from its "natural" orientation.
305:    * Note that this is the angle of rotation of the drawn graphics on the
306:    * screen, which is the opposite direction of the physical rotation of the
307:    * device. For example, if the device is rotated 90 degrees counter-clockwise,
308:    * to compensate rendering will be rotated by 90 degrees clockwise and thus
309:    * the returned value here will be Surface.ROTATION_90.  Return values will
310:    * be in the set Surface.ROTATION_{0,90,180,270}.
311:    */
312:   private int getScreenRotation() {
313:     Display display =
314:         ((WindowManager) form.getSystemService(Context.WINDOW_SERVICE)).
315:         getDefaultDisplay();
316:     if (SdkLevel.getLevel() >= SdkLevel.LEVEL_FROYO) {
317:       return FroyoUtil.getRotation(display);
318:     } else {
319:       return display.getOrientation();
320:     }
321:   }
322:
323:   /**
324:    * Responds to changes in the accelerometer or magnetic field sensors to
325:    * recompute orientation.  This only updates azimuth, pitch, and roll and
326:    * raises the OrientationChanged event if both sensors have reported in
327:    * at least once.
328:    *
329:    * @param sensorEvent an event from the accelerometer or magnetic field sensor
330:    */
331:   @Override
332:   public void onSensorChanged(SensorEvent sensorEvent) {
333:     if (enabled) {
334:       int eventType = sensorEvent.sensor.getType();
335:
336:       // Save the new sensor information about acceleration or the magnetic field.
337:       switch (eventType) {
338:         case Sensor.TYPE_ACCELEROMETER:
339:           // Update acceleration array.
340:           System.arraycopy(sensorEvent.values, 0, accels, 0, DIMENSIONS);
341:           accelsFilled = true;
342:           // Only update the accuracy property for the accelerometer.
343:           accuracy = sensorEvent.accuracy;
344:           break;
345:
346:         case Sensor.TYPE_MAGNETIC_FIELD:
347:           // Update magnetic field array.
348:           System.arraycopy(sensorEvent.values, 0, mags, 0, DIMENSIONS);
349:           magsFilled = true;
350:           break;
351:
352:         default:
353:           Log.e(LOG_TAG, "Unexpected sensor type: " + eventType);
354:           return;
355:       }
356:
357:       // If we have both acceleration and magnetic information, recompute values.
358:       if (accelsFilled && magsFilled) {
359:         SensorManager.getRotationMatrix(rotationMatrix,    // output
360:                                         inclinationMatrix, // output
361:                                         accels,
362:                                         mags);
363:         SensorManager.getOrientation(rotationMatrix, values);
```

```
364:
365:         // Make sure values are in expected range.
366:         azimuth = OrientationSensorUtil.normalizeAzimuth(
367:             (float) Math.toDegrees(values[AZIMUTH]));
368:         pitch = OrientationSensorUtil.normalizePitch(
369:             (float) Math.toDegrees(values[PITCH]));
370:         // Sign change for roll is for compatibility with earlier versions
371:         // of App Inventor that got orientation sensor information differently.
372:         roll = OrientationSensorUtil.normalizeRoll(
373:             (float) -Math.toDegrees(values[ROLL]));
374:
375:         // Adjust pitch and roll for phone rotation (e.g., landscape)
376:         int rotation = getScreenRotation();
377:         switch(rotation) {
378:           case Surface.ROTATION_0:  // normal rotation
379:             break;
380:           case Surface.ROTATION_90:  // phone is turned 90 degrees counter-clockwise
381:             float temp = -pitch;
382:             pitch = -roll;
383:             roll = temp;
384:             break;
385:           case Surface.ROTATION_180: // phone is rotated 180 degrees
386:             roll = -roll;
387:             break;
388:           case Surface.ROTATION_270:  // phone is turned 90 degrees clockwise
389:             temp = pitch;
390:             pitch = roll;
391:             roll = temp;
392:             break;
393:           default:
394:             Log.e(LOG_TAG, "Illegal value for getScreenRotation(): " +
395:                   rotation);
396:             break;
397:         }
398:
399:         // Raise event.
400:         OrientationChanged(azimuth, pitch, roll);
401:       }
402:     }
403:   }
404:
405:   @Override
406:   public void onAccuracyChanged(Sensor sensor, int accuracy) {
407:     // TODO(markf): Figure out if we actually need to do something here.
408:   }
409:
410:   // Deleteable implementation
411:
412:   @Override
413:   public void onDelete() {
414:     stopListening();
415:   }
416:
417:   // OnPauseListener implementation
418:
419:   public void onPause() {
420:     stopListening();
421:   }
422:
423:   // OnResumeListener implementation
424:
425:   public void onResume() {
426:     if (enabled) {
427:       startListening();
428:     }
429:   }
430: }
```

```java
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2009-2011 Google, All Rights reserved
// Copyright 2011-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0

package com.google.appinventor.components.runtime;

import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import com.google.appinventor.components.annotations.*;
import com.google.appinventor.components.common.ComponentCategory;
import com.google.appinventor.components.common.PropertyTypeConstants;
import com.google.appinventor.components.common.YaVersion;
import java.util.List;

@DesignerComponent(version = YaVersion.PROXIMITYSENSOR_COMPONENT_VERSION,
        description = "<p>Non-visible component that can measures the proximity of a
n object in cm " +
                "relative to the view screen of a device. This sensor is typically u
sed to determine " +
                "whether a handset is being held up to a persons ear; " +
                "i.e. lets you determine how far away an object is from a device. "
+
                "Many devices return the absolute distance, in cm, but some return o
nly near and far values. " +
                "In this case, the sensor usually reports its maximum range value in
 the far state " +
                "and a lesser value in the near state.</p>",
        category = ComponentCategory.SENSORS,
        nonVisible = true,
        iconName = "images/proximitysensor.png")
@SimpleObject
public class ProximitySensor extends AndroidNonvisibleComponent
        implements OnStopListener, OnResumeListener, SensorComponent, OnPauseListene
r,
        SensorEventListener, Deleteable {

    private Sensor proximitySensor;

    private final SensorManager sensorManager;

    // Indicates whether the sensor should generate events
    private boolean enabled;
    private float distance=0f;

    // Indicates if the sensor should be running when screen is off (on pause)
    private boolean keepRunningWhenOnPause;

    /**
     * Creates a new ProximitySensor component.
     *
     * @param container  ignored (because this is a non-visible component)
     */
    public ProximitySensor(ComponentContainer container) {
        super(container.$form());
        form.registerForOnResume(this);
        form.registerForOnStop(this);
        form.registerForOnPause(this);

        enabled = true;
        sensorManager = (SensorManager) container.$context().getSystemService(Contex
t.SENSOR_SERVICE);
        proximitySensor = sensorManager.getDefaultSensor(Sensor.TYPE_PROXIMITY);
        startListening();
    }

    /**
     * Used to determine if the device has ProximitySensor
     *
     * @return {@code true} indicates that an proximity sensor is available,
     *         {@code false} that it isn't
     */
    @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Reports whe
ther or not the device has a proximity sensor")
    public boolean Available() {
        List<Sensor> sensors = sensorManager.getSensorList(Sensor.TYPE_PROXIMITY);
        return (sensors.size() > 0);
    }

    @Override
    public void onResume() {
        if (enabled) {
            startListening();
        }
    }

    @Override
    public void onStop() {
        if (enabled) {
            stopListening();
        }
    }

    @Override
    public void onDelete() {
        if (enabled) {
            stopListening();
        }
    }

    @Override
    public void onPause() {
        if (enabled && !keepRunningWhenOnPause) {
            stopListening();
        }
    }


    /**
     * Registers the sensor to start listening for proximity changes
     */
    private void startListening() {
        sensorManager.registerListener(this, proximitySensor, SensorManager.SENSOR_D
ELAY_NORMAL);
    }

    /**
     * Stops the sensors from listening to the proximity changes
     */
    private void stopListening() {
```

```
116:         sensorManager.unregisterListener(this);
117:     }
118:
119:     /**
120:      * Called when sensor values have changed
121:      * @param sensorEvent holds information such as the sensor's type,
122:      *                    the time-stamp, accuracy and sensor's data
123:      */
124:     @Override
125:     public void onSensorChanged(SensorEvent sensorEvent) {
126:         if (enabled) {
127:             final float[] values = sensorEvent.values.clone();
128:             distance = values[0];
129:             ProximityChanged(distance);
130:         }
131:     }
132:
133:     /**
134:      * Determines a sensor's maximum range. Some proximity sensors return binary val
ues
135:      * that represent "near" or "far." In this case, the sensor usually reports
136:      * its maximum range value in the far state and a lesser value in the near state
.
137:      * Typically, the far value is a value > 5 cm, but this can vary from sensor to
sensor.
138:      *
139:      * @return Sensor's maximum range.
140:      */
141:     @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Reports the
 Maximum Range of the device's ProximitySensor")
142:     public float MaximumRange() {
143:         return proximitySensor.getMaximumRange();
144:     }
145:
146:     /**
147:      * If true, the sensor will generate events.  Otherwise, no events
148:      * are generated .
149:      *
150:      * @return {@code true} indicates that the sensor generates events,
151:      *         {@code false} that it doesn't
152:      */
153:     @SimpleProperty(category = PropertyCategory.BEHAVIOR)
154:     public boolean Enabled() {
155:         return enabled;
156:     }
157:
158:     /**
159:      * Specifies whether the sensor should generate events.  If true,
160:      * the sensor will generate events.  Otherwise, no events are generated.
161:      *
162:      * @param enabled  {@code true} enables sensor event generation,
163:      *                 {@code false} disables it
164:      */
165:     @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defa
ultValue = "True")
166:     @SimpleProperty (description = "If enabled, then device will listen for changes
in proximity")
167:     public void Enabled(boolean enabled) {
168:         if (this.enabled == enabled) {
169:             return;
170:         }
171:
172:         this.enabled = enabled;
173:         if (enabled) {
174:             startListening();
175:         } else {
176:             stopListening();
177:         }
178:     }
179:
180:     /**
181:      * Returns value of keepRunningWhenOnPause
182:      */
183:     @SimpleProperty(category = PropertyCategory.BEHAVIOR)
184:     public boolean KeepRunningWhenOnPause() {
185:         return keepRunningWhenOnPause;
186:     }
187:
188:     /**
189:      * Specifies if sensor should still be listening when activity is not active
190:      *
191:      * @param enabled
192:      */
193:     @DesignerProperty(editorType = PropertyTypeConstants.PROPERTY_TYPE_BOOLEAN, defa
ultValue = "False")
194:     @SimpleProperty (description = "If set to true, it will keep sensing for proximi
ty changes even when the app is not visible")
195:     public void KeepRunningWhenOnPause(boolean enabled) {
196:
197:         this.keepRunningWhenOnPause = enabled;
198:     }
199:
200:     //@SimpleEvent(description = "Triggered when distance (in cm) of the object to t
he device changes. ")
201:     public void ProximityChanged(float distance) {
202:         this.distance = distance;
203:         EventDispatcher.dispatchEvent(this, "ProximityChanged", this.distance);
204:     }
205:
206:     /**
207:      * Returns the distance.
208:      * The sensor must be enabled to return meaningful values.
209:      *
210:      * @return  distance
211:      */
212:     @SimpleProperty(category = PropertyCategory.BEHAVIOR, description = "Returns the
 distance from the object to the device")
213:     public float Distance() {
214:         return distance;
215:     }
216:
217:     /**
218:      * Called when the accuracy of the registered sensor has changed
219:      * @param sensor Sensor
220:      * @param accuracy the new accuracy of this sensor
221:      */
222:     @Override
223:     public void onAccuracyChanged(Sensor sensor, int accuracy) {
224:         //To change body of implemented methods use File | Settings | File Templates
.
225:     }
226: }
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @fileoverview Control blocks for Blockly, modified for App Inventor
 * @author fraser@google.com (Neil Fraser)
 * @author andrew.f.mckinney@gmail.com (Andrew F. McKinney)
 * Due to the frequency of long strings, the 80-column wrap rule need not apply
 * to language files.
 */

/**
 * Lyn's History:
 * [lyn, written 11/16-17/13, added 07/01/14] Added freeVariables, renameFree, and renameBound to forRange and forEach loops
 * [lyn, 10/27/13] Specify direction of flydowns
 * [lyn, 10/25/13] Made collapsed block labels more sensible.
 * [lyn, 10/10-14/13]
 *   + Installed flydown index variable declarations in forRange and forEach loops
 *   + Abstracted over string labels on all blocks using constants defined in en/_messages.js
 *   + Renamed "for <i> start [] end [] step []" block to "for each <number> from [] to [] by []"
 *   + Renamed "for each <i> in list []" block to "for each <item> in list []"
 *   + Renamed "choose test [] then-return [] else-return []" to "if [] then [] else []"
 *     (TODO: still needs to have a mutator like  the "if" statement blocks).
 *   + Renamed "evaluate" block to "evaluate but ignore result"
 *   + Renamed "do {} then-return []" block to "do {} result []" and re-added this block
 *     to the Control drawer (who removed it?)
 *   + Removed get block (still in Variable drawer; no longer needed with parameter flydowns)
 * [lyn, 11/29-30/12]
 *   + Change forEach and forRange loops to take name as input text rather than via plug.
 *   + For these blocks, add extra methods to support renaming.
 */

'use strict';

goog.provide('Blockly.Blocks.control');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['controls_if'] = {
  // If/elseif/else condition.
  category: 'Control',
  helpUrl: Blockly.Msg.LANG_CONTROLS_IF_HELPURL,
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendValueInput('IF0')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_IF);
    this.appendStatementInput('DO0')
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setMutator(new Blockly.Mutator(['controls_if_elseif',
      'controls_if_else']));
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      if (!thisBlock.elseifCount_ && !thisBlock.elseCount_) {
        return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_1;
      } else if (!thisBlock.elseifCount_ && thisBlock.elseCount_) {
        return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_2;
      } else if (thisBlock.elseifCount_ && !thisBlock.elseCount_) {
        return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_3;
      } else if (thisBlock.elseifCount_ && thisBlock.elseCount_) {
        return Blockly.Msg.LANG_CONTROLS_IF_TOOLTIP_4;
      }
      return '';
    });
    this.elseifCount_ = 0;
    this.elseCount_ = 0;
    this.warnings = [{name: "checkEmptySockets", sockets: [{baseName: "IF"}, {baseName: "DO"}]}];
  },
  mutationToDom: function () {
    if (!this.elseifCount_ && !this.elseCount_) {
      return null;
    }
    var container = document.createElement('mutation');
    if (this.elseifCount_) {
      container.setAttribute('elseif', this.elseifCount_);
    }
    if (this.elseCount_) {
      container.setAttribute('else', 1);
    }
    return container;
  },
  domToMutation: function (xmlElement) {
    if (xmlElement.getAttribute('elseif') === null) {
      this.elseifCount_ = 0;
    } else {
      this.elseifCount_ = window.parseInt(xmlElement.getAttribute('elseif'), 10);
    }

    this.elseCount_ = window.parseInt(xmlElement.getAttribute('else'), 10);
    for (var x = 1; x <= this.elseifCount_; x++) {
      this.appendValueInput('IF' + x)
          .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
          .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSEIF);
      this.appendStatementInput('DO' + x)
          .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
    }
    if (this.elseCount_) {
      this.appendStatementInput('ELSE')
          .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSE);
    }
  },
  decompose: function (workspace) {
    var containerBlock = new Blockly.Block.obtain(workspace, 'controls_if_if');
    containerBlock.initSvg();
    var connection = containerBlock.getInput('STACK').connection;
    for (var x = 1; x <= this.elseifCount_; x++) {
      var elseifBlock = new Blockly.Block.obtain(workspace, 'controls_if_elseif');
      elseifBlock.initSvg();
      connection.connect(elseifBlock.previousConnection);
      connection = elseifBlock.nextConnection;
```

```
    }
    if (this.elseCount_) {
      var elseBlock = new Blockly.Block.obtain(workspace, 'controls_if_else');
      elseBlock.initSvg();
      connection.connect(elseBlock.previousConnection);
    }
    return containerBlock;
  },
  compose: function (containerBlock) {
    // Disconnect the else input blocks and destroy the inputs.
    if (this.elseCount_) {
      this.removeInput('ELSE');
    }
    this.elseCount_ = 0;
    // Disconnect all the elseif input blocks and destroy the inputs.
    for (var x = this.elseifCount_; x > 0; x--) {
      this.removeInput('IF' + x);
      this.removeInput('DO' + x);
    }
    this.elseifCount_ = 0;
    // Rebuild the block's optional inputs.
    var clauseBlock = containerBlock.getInputTargetBlock('STACK');
    while (clauseBlock) {
      switch (clauseBlock.type) {
        case 'controls_if_elseif':
          this.elseifCount_++;
          var ifInput = this.appendValueInput('IF' + this.elseifCount_)
              .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Bl
ockly.Blocks.Utilities.INPUT))
              .appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSEIF);
          var doInput = this.appendStatementInput('DO' + this.elseifCount_);
          doInput.appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_THEN);
          // Reconnect any child blocks.
          if (clauseBlock.valueConnection_) {
            ifInput.connection.connect(clauseBlock.valueConnection_);
          }
          if (clauseBlock.statementConnection_) {
            doInput.connection.connect(clauseBlock.statementConnection_);
          }
          break;
        case 'controls_if_else':
          this.elseCount_++;
          var elseInput = this.appendStatementInput('ELSE');
          elseInput.appendField(Blockly.Msg.LANG_CONTROLS_IF_MSG_ELSE);
          // Reconnect any child blocks.
          if (clauseBlock.statementConnection_) {
            elseInput.connection.connect(clauseBlock.statementConnection_);
          }
          break;
        default:
          throw 'Unknown block type.';
      }
      clauseBlock = clauseBlock.nextConnection &&
      clauseBlock.nextConnection.targetBlock();
    }
  },
  saveConnections: function (containerBlock) {
    // Store a pointer to any connected child blocks.
    var inputDo;
    var clauseBlock = containerBlock.getInputTargetBlock('STACK');
    var x = 1;
    while (clauseBlock) {
      switch (clauseBlock.type) {
        case 'controls_if_elseif':
          var inputIf = this.getInput('IF' + x);
          inputDo = this.getInput('DO' + x);
          clauseBlock.valueConnection_ =
              inputIf && inputIf.connection.targetConnection;
          clauseBlock.statementConnection_ =
              inputDo && inputDo.connection.targetConnection;
          x++;
          break;
        case 'controls_if_else':
          inputDo = this.getInput('ELSE');
          clauseBlock.statementConnection_ =
              inputDo && inputDo.connection.targetConnection;
          break;
        default:
          throw 'Unknown block type.';
      }
      clauseBlock = clauseBlock.nextConnection &&
      clauseBlock.nextConnection.targetBlock();
    }
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_IF_IF_TITLE_IF}]
};

Blockly.Blocks['controls_if_if'] = {
  // If condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_IF_TITLE_IF);
    this.appendStatementInput('STACK');
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_IF_TOOLTIP);
    this.contextMenu = false;
  }
};

Blockly.Blocks['controls_if_elseif'] = {
  // Else-If condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_ELSEIF_TITLE_ELSEIF);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_ELSEIF_TOOLTIP);
    this.contextMenu = false;
  }
};

Blockly.Blocks['controls_if_else'] = {
  // Else condition.
  init: function () {
    this.setColour(Blockly.CONTROL_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_CONTROLS_IF_ELSE_TITLE_ELSE);
    this.setPreviousStatement(true);
    this.setTooltip(Blockly.Msg.LANG_CONTROLS_IF_ELSE_TOOLTIP);
    this.contextMenu = false;
  }
};
```

```
238: Blockly.Blocks['controls_forRange'] = {
239:   // For range.
240:   category: 'Control',
241:   helpUrl: Blockly.Msg.LANG_CONTROLS_FORRANGE_HELPURL,
242:   init: function () {
243:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
244:     //this.setOutput(true, null);
245:     // Need to deal with variables here
246:     // [lyn, 11/30/12] Changed variable to be text input box that does renaming righ
t (i.e., avoids variable capture)
247:     // Old code:
248:     // this.appendValueInput('VAR').appendField('for range').appendField('variable')
.setAlign(Blockly.ALIGN_RIGHT);
249:     // this.appendValueInput('START').setCheck(Number).appendField('start').setAlign
(Blockly.ALIGN_RIGHT);
250:     this.appendValueInput('START')
251:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
252:         .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_ITEM)
253:         .appendField(new Blockly.FieldParameterFlydown(Blockly.Msg.LANG_CONTROLS_FOR
RANGE_INPUT_VAR, true, Blockly.FieldFlydown.DISPLAY_BELOW), 'VAR')
254:         .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_START)
255:         .setAlign(Blockly.ALIGN_RIGHT);
256:     this.appendValueInput('END')
257:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
258:         .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_END)
259:         .setAlign(Blockly.ALIGN_RIGHT);
260:     this.appendValueInput('STEP')
261:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
262:         .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_STEP)
263:         .setAlign(Blockly.ALIGN_RIGHT);
264:     this.appendStatementInput('DO')
265:         .appendField(Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_DO)
266:         .setAlign(Blockly.ALIGN_RIGHT);
267:     this.setPreviousStatement(true);
268:     this.setNextStatement(true);
269:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_FORRANGE_TOOLTIP);
270:   },
271:   getVars: function () {
272:     return [this.getFieldValue('VAR')];
273:   },
274:   blocksInScope: function () {
275:     var doBlock = this.getInputTargetBlock('DO');
276:     if (doBlock) {
277:       return [doBlock];
278:     } else {
279:       return [];
280:     }
281:   },
282:   declaredNames: function () {
283:     return [this.getFieldValue('VAR')];
284:   },
285:   renameVar: function (oldName, newName) {
286:     if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
287:       this.setFieldValue(newName, 'VAR');
288:     }
289:   },
290:   renameBound: function (boundSubstitution, freeSubstitution) {
291:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('START'), freeSubsti
tution);
292:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('END'), freeSubstitu
tion);
293:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('STEP'), freeSubstit
ution);
294:     var oldIndexVar = this.getFieldValue('VAR');
295:     var newIndexVar = boundSubstitution.apply(oldIndexVar);
296:     if (newIndexVar !== oldIndexVar) {
297:       this.renameVar(oldIndexVar, newIndexVar);
298:       var indexSubstitution = Blockly.Substitution.simpleSubstitution(oldIndexVar, n
ewIndexVar);
299:       var extendedFreeSubstitution = freeSubstitution.extend(indexSubstitution);
300:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), extendedFre
eSubstitution);
301:     } else {
302:       var removedFreeSubstitution = freeSubstitution.remove([oldIndexVar]);
303:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DO'), removedFree
Substitution);
304:     }
305:     if (this.nextConnection) {
306:       var nextBlock = this.nextConnection.targetBlock();
307:       Blockly.LexicalVariable.renameFree(nextBlock, freeSubstitution);
308:     }
309:   },
310:   renameFree: function (freeSubstitution) {
311:     var indexVar = this.getFieldValue('VAR');
312:     var bodyFreeVars = Blockly.LexicalVariable.freeVariables(this.getInputTargetBloc
k('DO'));
313:     bodyFreeVars.deleteName(indexVar);
314:     var renamedBodyFreeVars = bodyFreeVars.renamed(freeSubstitution);
315:     if (renamedBodyFreeVars.isMember(indexVar)) { // Variable capture!
316:       var newIndexVar = Blockly.FieldLexicalVariable.nameNotIn(indexVar, renamedBody
FreeVars.toList());
317:       var boundSubstitution = Blockly.Substitution.simpleSubstitution(indexVar, newI
ndexVar);
318:       this.renameBound(boundSubstitution, freeSubstitution);
319:     } else {
320:       this.renameBound(new Blockly.Substitution(), freeSubstitution);
321:     }
322:   },
323:   freeVariables: function () { // return the free variables of this block
324:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('DO'
));
325:     result.deleteName(this.getFieldValue('VAR')); // Remove bound index variable fro
m body free vars
326:     result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('STA
RT')));
327:     result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('END
')));
328:     result.unite(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('STE
P')));
329:     if (this.nextConnection) {
330:       var nextBlock = this.nextConnection.targetBlock();
331:       result.unite(Blockly.LexicalVariable.freeVariables(nextBlock));
332:     }
333:     return result;
334:   },
335:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_FORRANGE_INPUT_ITEM}]
336: };
337:
338: Blockly.Blocks['controls_while'] = {
339:   // While condition.
340:   category: 'Control',
```

```
  341:   helpUrl: Blockly.Msg.LANG_CONTROLS_WHILE_HELPURL,
  342:   init: function () {
  343:     this.setColour(Blockly.CONTROL_CATEGORY_HUE);
  344:     this.appendValueInput('TEST')
  345:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
  346:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_TITLE)
  347:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_TEST)
  348:         .setAlign(Blockly.ALIGN_RIGHT);
  349:     this.appendStatementInput('DO')
  350:         .appendField(Blockly.Msg.LANG_CONTROLS_WHILE_INPUT_DO)
  351:         .setAlign(Blockly.ALIGN_RIGHT);
  352:     this.setPreviousStatement(true);
  353:     this.setNextStatement(true);
  354:     this.setTooltip(Blockly.Msg.LANG_CONTROLS_WHILE_TOOLTIP);
  355:   },
  356:   typeblock: [{translatedName: Blockly.Msg.LANG_CONTROLS_WHILE_TITLE}]
  357: };
  358:
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Logic blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Blocks.logic');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['logic_boolean'] = {
  // Boolean data type: true and false.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'BOOL');
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('BOOL');
      return Blockly.Blocks.logic_boolean.TOOLTIPS()[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('BOOL');
    return Blockly.Blocks.logic_boolean.HELPURLS()[op];
  },
  typeblock: [{
    translatedName: Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE,
    dropDown: {
      titleName: 'BOOL',
      value: 'TRUE'
    }
  }, {
    translatedName: Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE,
    dropDown: {
      titleName: 'BOOL',
      value: 'FALSE'
    }
  }]
};

Blockly.Blocks.logic_boolean.OPERATORS = function () {
  return [
    [Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE, 'TRUE'],
    [Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE, 'FALSE']
  ];
};

Blockly.Blocks.logic_boolean.TOOLTIPS = function () {
  return {
    TRUE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TOOLTIP_TRUE,
    FALSE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TOOLTIP_FALSE
  }
};

Blockly.Blocks.logic_boolean.HELPURLS = function () {
  return {
    TRUE: Blockly.Msg.LANG_LOGIC_BOOLEAN_TRUE_HELPURL,
    FALSE: Blockly.Msg.LANG_LOGIC_BOOLEAN_FALSE_HELPURL
  }
};

Blockly.Blocks['logic_false'] = {
  // Boolean data type: true and false.
  category: 'Logic',
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendDummyInput()
        .appendField(new Blockly.FieldDropdown(Blockly.Blocks.logic_boolean.OPERATORS), 'BOOL');
    this.setFieldValue('FALSE', 'BOOL');
    var thisBlock = this;
    this.setTooltip(function () {
      var op = thisBlock.getFieldValue('BOOL');
      return Blockly.Blocks.logic_boolean.TOOLTIPS()[op];
    });
  },
  helpUrl: function () {
    var op = this.getFieldValue('BOOL');
    return Blockly.Blocks.logic_boolean.HELPURLS()[op];
  }
};

Blockly.Blocks['logic_negate'] = {
  // Negation.
  category: 'Logic',
  helpUrl: Blockly.Msg.LANG_LOGIC_NEGATE_HELPURL,
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('BOOL')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_LOGIC_NEGATE_INPUT_NOT);
    this.setTooltip(Blockly.Msg.LANG_LOGIC_NEGATE_TOOLTIP);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_LOGIC_NEGATE_INPUT_NOT}]
};

Blockly.Blocks['logic_compare'] = {
  // Comparison operator.
  category: 'Logic',
  helpUrl: function () {
    var mode = this.getFieldValue('OP');
    return Blockly.Blocks.logic_compare.HELPURLS()[mode];
  },
  init: function () {
    this.setColour(Blockly.LOGIC_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A');
    this.appendValueInput('B')
```

```
119:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
120:     this.setInputsInline(true);
121:     // Assign 'this' to a variable for use in the tooltip closure below.
122:     var thisBlock = this;
123:     this.setTooltip(function () {
124:       var mode = thisBlock.getFieldValue('OP');
125:       return Blockly.Blocks.logic_compare.TOOLTIPS()[mode];
126:     });
127:   },
128:   // Potential clash with Math =, so using 'logic equal' for now
129:   typeblock: [{translatedName: Blockly.Msg.LANG_LOGIC_COMPARE_TRANSLATED_NAME}]
130: };
131:
132: Blockly.Blocks.logic_compare.TOOLTIPS = function () {
133:   return {
134:     EQ: Blockly.Msg.LANG_LOGIC_COMPARE_TOOLTIP_EQ,
135:     NEQ: Blockly.Msg.LANG_LOGIC_COMPARE_TOOLTIP_NEQ
136:   }
137: };
138:
139: Blockly.Blocks.logic_compare.HELPURLS = function () {
140:   return {
141:     EQ: Blockly.Msg.LANG_LOGIC_COMPARE_HELPURL_EQ,
142:     NEQ: Blockly.Msg.LANG_LOGIC_COMPARE_HELPURL_NEQ
143:   }
144: };
145:
146: Blockly.Blocks.logic_compare.OPERATORS = function () {
147:   return [
148:     [Blockly.Msg.LANG_LOGIC_COMPARE_EQ, 'EQ'],
149:     [Blockly.Msg.LANG_LOGIC_COMPARE_NEQ, 'NEQ']
150:   ];
151: };
152:
153: Blockly.Blocks['logic_operation'] = {
154:   // Logical operations: 'and', 'or'.
155:   category: 'Logic',
156:   init: function () {
157:     this.setColour(Blockly.LOGIC_CATEGORY_HUE);
158:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
159:     this.appendValueInput('A')
160:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT));
161:     this.appendValueInput('B')
162:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", Blockly.
Blocks.Utilities.INPUT))
163:         .appendField(new Blockly.FieldDropdown(this.OPERATORS), 'OP');
164:     this.setInputsInline(true);
165:     // Assign 'this' to a variable for use in the tooltip closure below.
166:     var thisBlock = this;
167:     this.setTooltip(function () {
168:       var op = thisBlock.getFieldValue('OP');
169:       return Blockly.Blocks.logic_operation.TOOLTIPS()[op];
170:     });
171:   },
172:   helpUrl: function () {
173:     var op = this.getFieldValue('OP');
174:     return Blockly.Blocks.logic_operation.HELPURLS()[op];
175:   },
176:   typeblock: [{
177:     translatedName: Blockly.Msg.LANG_LOGIC_OPERATION_AND,
178:     dropDown: {
179:       titleName: 'OP',
180:       value: 'AND'
181:     }
182:   }]
183: };
184:
185: Blockly.Blocks.logic_operation.OPERATORS = function () {
186:   return [
187:     [Blockly.Msg.LANG_LOGIC_OPERATION_AND, 'AND']
188:   ]
189: };
190:
191: Blockly.Blocks.logic_operation.HELPURLS = function () {
192:   return {
193:     AND: Blockly.Msg.LANG_LOGIC_OPERATION_HELPURL_AND
194:   }
195: };
196: Blockly.Blocks.logic_operation.TOOLTIPS = function () {
197:   return {
198:     AND: Blockly.Msg.LANG_LOGIC_OPERATION_TOOLTIP_AND
199:   }
200: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @license
    7:  * @fileoverview Math blocks for Blockly, modified for MIT App Inventor.
    8:  * @author mckinney@mit.edu (Andrew F. McKinney)
    9:  */
   10:
   11: 'use strict';
   12:
   13: goog.provide('Blockly.Blocks.math');
   14:
   15: goog.require('Blockly.Blocks.Utilities');
   16:
   17: Blockly.Blocks['math_number'] = {
   18:   // Numeric value.
   19:   category: 'Math',
   20:   helpUrl: Blockly.Msg.LANG_MATH_NUMBER_HELPURL,
   21:   init: function () {
   22:     this.setColour(Blockly.MATH_CATEGORY_HUE);
   23:     this.appendDummyInput().appendField(
   24:         new Blockly.FieldTextInput('0', Blockly.Blocks.math_number.validator), 'NUM'
);
   25:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
   26:     this.setTooltip(Blockly.Msg.LANG_MATH_NUMBER_TOOLTIP);
   27:   },
   28:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_MUTATOR_ITEM_INPUT_NUMBER}]
   29: };
   30:
   31: Blockly.Blocks.math_number.validator = function (text) {
   32:   // Ensure that only a number may be entered.
   33:   // TODO: Handle cases like 'o', 'ten', '1,234', '3,14', etc.
   34:   var n = window.parseFloat(text || 0);
   35:   return window.isNaN(n) ? null : String(n);
   36: };
   37:
   38: Blockly.Blocks['math_compare'] = {
   39:   // Basic arithmetic operator.
   40:   // TODO(Andrew): equality block needs to have any on the sockets.
   41:   category: 'Math',
   42:   helpUrl: function () {
   43:     var mode = this.getFieldValue('OP');
   44:     return Blockly.Blocks.math_compare.HELPURLS()[mode];
   45:   },
   46:   init: function () {
   47:     this.setColour(Blockly.MATH_CATEGORY_HUE);
   48:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean", B
lockly.Blocks.Utilities.OUTPUT));
   49:     this.appendValueInput('A')
   50:         .setCheck(null);
   51:     this.appendValueInput('B')
   52:         .setCheck(null)
   53:         .appendField(new Blockly.FieldDropdown(this.OPERATORS, Blockly.Blocks.math_c
ompare.onchange), 'OP');
   54:     this.setInputsInline(true);
   55:     // Assign 'this' to a variable for use in the tooltip closure below.
   56:     var thisBlock = this;
   57:     this.setTooltip(function () {
   58:       var mode = thisBlock.getFieldValue('OP');
   59:       return Blockly.Blocks.math_compare.TOOLTIPS()[mode];
   60:     });
   61:   },
   62:   // Potential clash with logic equal, using '=' for now
   63:   typeblock: [{
   64:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_EQ,
   65:     dropDown: {
   66:       titleName: 'OP',
   67:       value: 'EQ'
   68:     }
   69:   }, {
   70:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_NEQ,
   71:     dropDown: {
   72:       titleName: 'OP',
   73:       value: 'NEQ'
   74:     }
   75:   }, {
   76:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_LT,
   77:     dropDown: {
   78:       titleName: 'OP',
   79:       value: 'LT'
   80:     }
   81:   }, {
   82:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_LTE,
   83:     dropDown: {
   84:       titleName: 'OP',
   85:       value: 'LTE'
   86:     }
   87:   }, {
   88:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_GT,
   89:     dropDown: {
   90:       titleName: 'OP',
   91:       value: 'GT'
   92:     }
   93:   }, {
   94:     translatedName: Blockly.Msg.LANG_MATH_COMPARE_GTE,
   95:     dropDown: {
   96:       titleName: 'OP',
   97:       value: 'GTE'
   98:     }
   99:   }]
  100: };
  101:
  102: Blockly.Blocks.math_compare.onchange = function (value) {
  103:   if (!this.sourceBlock_) {
  104:     return;
  105:   }
  106:   if (value == "EQ" || value == "NEQ") {
  107:     this.sourceBlock_.getInput("A").setCheck(null);
  108:     this.sourceBlock_.getInput("B").setCheck(null);
  109:   } else {
  110:     this.sourceBlock_.getInput("A")
  111:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  112:     this.sourceBlock_.getInput("B")
  113:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  114:   }
  115: };
  116:
  117: Blockly.Blocks.math_compare.OPERATORS = function () {
  118:   return [[Blockly.Msg.LANG_MATH_COMPARE_EQ, 'EQ'],
```

```
  119:     [Blockly.Msg.LANG_MATH_COMPARE_NEQ, 'NEQ'],
  120:     [Blockly.Msg.LANG_MATH_COMPARE_LT, 'LT'],
  121:     [Blockly.Msg.LANG_MATH_COMPARE_LTE, 'LTE'],
  122:     [Blockly.Msg.LANG_MATH_COMPARE_GT, 'GT'],
  123:     [Blockly.Msg.LANG_MATH_COMPARE_GTE, 'GTE']];
  124: };
  125:
  126: Blockly.Blocks.math_compare.TOOLTIPS = function () {
  127:   return {
  128:     EQ: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_EQ,
  129:     NEQ: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_NEQ,
  130:     LT: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_LT,
  131:     LTE: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_LTE,
  132:     GT: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_GT,
  133:     GTE: Blockly.Msg.LANG_MATH_COMPARE_TOOLTIP_GTE
  134:   }
  135: };
  136:
  137: Blockly.Blocks.math_compare.HELPURLS = function () {
  138:   return {
  139:     EQ: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_EQ,
  140:     NEQ: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_NEQ,
  141:     LT: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_LT,
  142:     LTE: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_LTE,
  143:     GT: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_GT,
  144:     GTE: Blockly.Msg.LANG_MATH_COMPARE_HELPURL_GTE
  145:   }
  146: };
  147:
  148: Blockly.Blocks['math_add'] = {
  149:   // Basic arithmetic operator.
  150:   category: 'Math',
  151:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_ADD,
  152:   init: function () {
  153:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  154:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  155:     this.appendValueInput('NUM0')
  156:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  157:     // append the title on a separate line to avoid overly long lines
  158:     this.appendValueInput('NUM1')
  159:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  160:         .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_ADD);
  161:     this.setInputsInline(true);
  162:     // Assign 'this' to a variable for use in the tooltip closure below.
  163:     var thisBlock = this;
  164:     this.setTooltip(function () {
  165:       return Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_ADD;
  166:     });
  167:     this.setMutator(new Blockly.Mutator(['math_mutator_item']));
  168:     this.emptyInputName = 'EMPTY';
  169:     this.repeatingInputName = 'NUM';
  170:     this.itemCount_ = 2;
  171:   },
  172:   mutationToDom: Blockly.mutationToDom,
  173:   domToMutation: Blockly.domToMutation,
  174:   decompose: function (workspace) {
  175:     return Blockly.decompose(workspace, 'math_mutator_item', this);
  176:   },
  177:   compose: Blockly.compose,
  178:   saveConnections: Blockly.saveConnections,
  179:   addEmptyInput: function () {
  180:     var input = this.appendDummyInput(this.emptyInputName);
  181:   },
  182:   addInput: function (inputNum) {
  183:     var input = this.appendValueInput(this.repeatingInputName + inputNum)
  184:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  185:     if (inputNum !== 0) {
  186:       input.appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_ADD);
  187:     }
  188:     return input;
  189:   },
  190:   updateContainerBlock: function (containerBlock) {
  191:     containerBlock.setFieldValue("+", "CONTAINER_TEXT");
  192:   },
  193:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_ADD}]
  194: };
  195:
  196: Blockly.Blocks['math_mutator_item'] = {
  197:   // Add items.
  198:   init: function () {
  199:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  200:     this.appendDummyInput()
  201:       //.appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TITLE);
  202:         .appendField("number");
  203:     this.setPreviousStatement(true);
  204:     this.setNextStatement(true);
  205:     //this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TOOLTIP_1);
  206:     this.contextMenu = false;
  207:   }
  208: };
  209:
  210: Blockly.Blocks['math_subtract'] = {
  211:   // Basic arithmetic operator.
  212:   category: 'Math',
  213:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_MINUS,
  214:   init: function () {
  215:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  216:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
  217:     this.appendValueInput('A')
  218:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
  219:     this.appendValueInput('B')
  220:         .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
  221:         .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_MINUS);
  222:     this.setInputsInline(true);
  223:     // Assign 'this' to a variable for use in the tooltip closure below.
  224:     this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_MINUS);
  225:   },
  226:   typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_MINUS}]
  227: };
  228:
  229: Blockly.Blocks['math_multiply'] = {
  230:   // Basic arithmetic operator.
  231:   category: 'Math',
  232:   helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_MULTIPLY,
  233:   init: function () {
  234:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  235:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
```

```
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('NUM0')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('NUM1')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Blocks.Utilities.times_symbol);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(function () {
      return Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_MULTIPLY;
    });
    this.setMutator(new Blockly.Mutator(['math_mutator_item']));
    this.emptyInputName = 'EMPTY';
    this.repeatingInputName = 'NUM';
    this.itemCount_ = 2;
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function (workspace) {
    return Blockly.decompose(workspace, 'math_mutator_item', this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function () {
    var input = this.appendDummyInput(this.emptyInputName);
  },
  addInput: function (inputNum) {
    var input = this.appendValueInput(this.repeatingInputName + inputNum)
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    if (inputNum !== 0) {
      input.appendField(Blockly.Blocks.Utilities.times_symbol);
    }
    return input;
  },
  updateContainerBlock: function (containerBlock) {
    containerBlock.setFieldValue(Blockly.Blocks.Utilities.times_symbol, "CONTAINER_T
EXT");
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_MULTIPLY}]
};

Blockly.Blocks['math_division'] = {
  // Basic arithmetic operator.
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_DIVIDE,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_DIVIDE);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_DIVIDE);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_DIVIDE}]
};

Blockly.Blocks['math_power'] = {
  // Basic arithmetic operator.
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_ARITHMETIC_HELPURL_POWER,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));
    this.appendValueInput('A')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT));
    this.appendValueInput('B')
        .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Blockly.B
locks.Utilities.INPUT))
        .appendField(Blockly.Msg.LANG_MATH_ARITHMETIC_POWER);
    this.setInputsInline(true);
    // Assign 'this' to a variable for use in the tooltip closure below.
    var thisBlock = this;
    this.setTooltip(Blockly.Msg.LANG_MATH_ARITHMETIC_TOOLTIP_POWER);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_ARITHMETIC_POWER}]
};

Blockly.Blocks['math_random_int'] = {
  // Random integer between [X] and [Y].
  category: 'Math',
  helpUrl: Blockly.Msg.LANG_MATH_RANDOM_INT_HELPURL,
  init: function () {
    this.setColour(Blockly.MATH_CATEGORY_HUE);
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", Bl
ockly.Blocks.Utilities.OUTPUT));

    var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number", B
lockly.Blocks.Utilities.INPUT);
    this.interpolateMsg(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT,
        ['FROM', checkTypeNumber, Blockly.ALIGN_RIGHT],
        ['TO', checkTypeNumber, Blockly.ALIGN_RIGHT],
        Blockly.ALIGN_RIGHT)
    /*this.appendValueInput('FROM')
     .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Block
s.Utilities.INPUT))
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_TITLE_RANDOM)
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT_FROM);
     this.appendValueInput('TO')
     .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blockly.Block
s.Utilities.INPUT))
     .appendField(Blockly.Msg.LANG_MATH_RANDOM_INT_INPUT_TO);*/
    this.setInputsInline(true);
    this.setTooltip(Blockly.Msg.LANG_MATH_RANDOM_INT_TOOLTIP);
  },
  typeblock: [{translatedName: Blockly.Msg.LANG_MATH_RANDOM_INT_TITLE_RANDOM}]
};

/*---------------------------------------*/

Blockly.Blocks['now_time'] = {
  category: 'Math',
```

```
  345:   init: function() {
  346:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  347:     this.setHelpUrl('http://www.example.com/');
  348:     this.appendDummyInput()
  349:         .appendField(new Blockly.FieldDropdown([["now", "NOW_T"], ["exec_time", "EXE
C_T"], ["start_time", "START_T"]]), "FUNC");
  350:     this.setOutput(true, "Number");
  351:     this.setTooltip('');
  352:   }
  353: };
  354:
  355: Blockly.Blocks['exec_time'] = {
  356:   category: 'Math',
  357:   init: function() {
  358:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  359:     this.setHelpUrl('http://www.example.com/');
  360:     this.appendDummyInput()
  361:         .appendField(new Blockly.FieldDropdown([["exec_time", "EXEC_T"], ["now", "NO
W_T"], ["start_time", "START_T"]]), "FUNC");
  362:     this.setOutput(true, "Number");
  363:     this.setTooltip('');
  364:   }
  365: };
  366:
  367: Blockly.Blocks['start_time'] = {
  368:   category: 'Math',
  369:   init: function() {
  370:     this.setColour(Blockly.MATH_CATEGORY_HUE);
  371:     this.setHelpUrl('http://www.example.com/');
  372:     this.appendDummyInput()
  373:         .appendField(new Blockly.FieldDropdown([["start_time", "START_T"], ["now", "
NOW_T"], ["exec_time", "EXEC_T"]]), "FUNC");
  374:     this.setOutput(true, "Number");
  375:     this.setTooltip('');
  376:   }
  377: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @license
    7:  * @fileoverview Text blocks for Blockly, modified for MIT App Inventor.
    8:  * @author mckinney@mit.edu (Andrew F. McKinney)
    9:  */
   10:
   11: 'use strict';
   12:
   13: goog.provide('Blockly.Blocks.text');
   14:
   15: goog.require('Blockly.Blocks.Utilities');
   16:
   17: Blockly.Blocks['text'] = {
   18:   // Text value.
   19:   category: 'Text',
   20:   helpUrl: Blockly.Msg.LANG_TEXT_TEXT_HELPURL,
   21:   init: function () {
   22:     this.setColour(Blockly.TEXT_CATEGORY_HUE);
   23:     this.appendDummyInput().appendField(Blockly.Msg.LANG_TEXT_TEXT_LEFT_QUOTE).appen
dField(
   24:         new Blockly.FieldTextBlockInput(''),
   25:         'TEXT').appendField(Blockly.Msg.LANG_TEXT_TEXT_RIGHT_QUOTE);
   26:     this.setOutput(true, [Blockly.Blocks.text.connectionCheck]);
   27:     this.setTooltip(Blockly.Msg.LANG_TEXT_TEXT_TOOLTIP);
   28:   },
   29:   typeblock: [{translatedName: Blockly.Msg.LANG_CATEGORY_TEXT}]
   30: };
   31:
   32: Blockly.Blocks.text.connectionCheck = function (myConnection, otherConnection) {
   33:   var block = myConnection.sourceBlock_;
   34:   var otherTypeArray = otherConnection.check_;
   35:   for (var i = 0; i < otherTypeArray.length; i++) {
   36:     if (otherTypeArray[i] == "String") {
   37:       return true;
   38:     } else if (otherTypeArray[i] == "Number" && !isNaN(parseFloat(block.getFieldValu
e('TEXT')))) {
   39:       return true;
   40:     }
   41:   }
   42:   return false;
   43: };
   44:
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Lists blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Blocks.lists');

goog.require('Blockly.Blocks.Utilities');

Blockly.Blocks['lists_create_with'] = {
  // Create a list with any number of elements of any type.
  category: 'Lists',
  helpUrl: Blockly.Msg.LANG_LISTS_CREATE_WITH_EMPTY_HELPURL,
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendValueInput('ADD0')
        .appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST);
    this.appendValueInput('ADD1');
    this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
    this.setMutator(new Blockly.Mutator(['lists_create_with_item']));
    this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_TOOLTIP);
    this.itemCount_ = 2;
    this.emptyInputName = 'EMPTY';
    this.repeatingInputName = 'ADD';
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function(workspace){
    return Blockly.decompose(workspace,'lists_create_with_item',this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function(){
    this.appendDummyInput(this.emptyInputName)
      .appendField(Blockly.Msg.LANG_LISTS_CREATE_EMPTY_TITLE);
  },
  addInput: function(inputNum){
    var input = this.appendValueInput(this.repeatingInputName + inputNum);
    if(inputNum === 0){
      input.appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST);
    }
    return input;
  },
  updateContainerBlock: function(containerBlock) {
    containerBlock.setFieldValue(Blockly.Msg.LANG_LISTS_CREATE_WITH_CONTAINER_TITLE_
ADD,"CONTAINER_TEXT");
  },
  // create type blocks for both make a list (two items) and create empty list
  typeblock: [
      { translatedName: Blockly.Msg.LANG_LISTS_CREATE_WITH_TITLE_MAKE_LIST,
        mutatorAttributes: { items: 2 } },
      { translatedName: Blockly.Msg.LANG_LISTS_CREATE_EMPTY_TITLE,
        mutatorAttributes: { items: 0 } }]
};

Blockly.Blocks['lists_create_with_item'] = {
  // Add items.
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TITLE);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_LISTS_CREATE_WITH_ITEM_TOOLTIP);
    this.contextMenu = false;
  }
};

Blockly.Blocks['lists_add_items'] = {
  // Create a list with any number of elements of any type.
  category: 'Lists',
  helpUrl: Blockly.Msg.LANG_LISTS_ADD_ITEMS_HELPURL,
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
    this.appendValueInput('LIST')
      .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_TITLE_ADD)
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_INPUT_LIST);
    this.appendValueInput('ITEM0')
      .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEMS_INPUT_ITEM)
      .setAlign(Blockly.ALIGN_RIGHT);
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEMS_TOOLTIP);
    this.setMutator(new Blockly.Mutator(['lists_add_items_item']));
    this.itemCount_ = 1;
    this.emptyInputName = null;
    this.repeatingInputName = 'ITEM';
  },
  mutationToDom: Blockly.mutationToDom,
  domToMutation: Blockly.domToMutation,
  decompose: function(workspace){
    return Blockly.decompose(workspace,'lists_add_items_item',this);
  },
  compose: Blockly.compose,
  saveConnections: Blockly.saveConnections,
  addEmptyInput: function(){},
  addInput: function(inputNum){
    var input = this.appendValueInput(this.repeatingInputName + inputNum);
    input.appendField('item').setAlign(Blockly.ALIGN_RIGHT);
    return input;
  },
  updateContainerBlock: function(containerBlock) {
    containerBlock.setFieldValue(Blockly.Msg.LANG_LISTS_ADD_ITEMS_CONTAINER_TITLE_AD
D,"CONTAINER_TEXT");
    containerBlock.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEMS_CONTAINER_TOOLTIP);
  },
  typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_ADD_ITEMS_TITLE_ADD }]
};

Blockly.Blocks['lists_add_items_item'] = {
  // Add items.
  init: function() {
    this.setColour(Blockly.LIST_CATEGORY_HUE);
```

```
121:     this.appendDummyInput()
122:         .appendField(Blockly.Msg.LANG_LISTS_ADD_ITEM_TITLE);
123:     this.setPreviousStatement(true);
124:     this.setNextStatement(true);
125:     this.setTooltip(Blockly.Msg.LANG_LISTS_ADD_ITEM_TOOLTIP);
126:     this.contextMenu = false;
127:   }
128: };
129:
130: Blockly.Blocks['lists_is_in'] = {
131:   // Is in list?.
132:   category : 'Lists',
133:   helpUrl : Blockly.Msg.LANG_LISTS_IS_IN_HELPURL,
134:   init : function() {
135:     this.setColour(Blockly.LIST_CATEGORY_HUE);
136:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
137:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
138:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_IS_IN_INPUT,
139:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
140:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
141:             Blockly.ALIGN_RIGHT);
142:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
ockly.Blocks.Utilities.OUTPUT));
143:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_IN_TOOLTIP);
144:     this.setInputsInline(false);
145:   },
146:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_IS_IN_TITLE_IS_IN }]
147: };
148:
149:
150: Blockly.Blocks['lists_length'] = {
151:   // Length of list.
152:   category : 'Lists',
153:   helpUrl : Blockly.Msg.LANG_LISTS_LENGTH_HELPURL,
154:   init : function() {
155:     this.setColour(Blockly.LIST_CATEGORY_HUE);
156:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blo
ckly.Blocks.Utilities.OUTPUT));
157:     this.appendValueInput('LIST')
158:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
159:       .appendField(Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LENGTH)
160:       .appendField(Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LIST);
161:     this.setTooltip(Blockly.Msg.LANG_LISTS_LENGTH_TOOLTIP);
162:   },
163:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_LENGTH_INPUT_LENGTH }]
164: };
165:
166: Blockly.Blocks['lists_is_empty'] = {
167:   // Is the list empty?.
168:   category : 'Lists',
169:   helpUrl : Blockly.Msg.LANG_LISTS_IS_EMPTY_HELPURL,
170:   init : function() {
171:     this.setColour(Blockly.LIST_CATEGORY_HUE);
172:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
ockly.Blocks.Utilities.OUTPUT));
173:     this.appendValueInput('LIST')
174:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
175:       .appendField(Blockly.Msg.LANG_LISTS_TITLE_IS_EMPTY)
176:       .appendField(Blockly.Msg.LANG_LISTS_INPUT_LIST);
177:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_EMPTY_TOOLTIP);
178:   },
179:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_TITLE_IS_EMPTY }]
180: };
181:
182: Blockly.Blocks['lists_position_in'] = {
183:   // Postion of item in list.
184:   category : 'Lists',
185:   helpUrl : Blockly.Msg.LANG_LISTS_POSITION_IN_HELPURL,
186:   init : function() {
187:     this.setColour(Blockly.LIST_CATEGORY_HUE);
188:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Blo
ckly.Blocks.Utilities.OUTPUT));
189:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
190:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
191:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_POSITION_IN_INPUT,
192:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
193:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
194:             Blockly.ALIGN_RIGHT);
195:     this.setTooltip(Blockly.Msg.LANG_LISTS_POSITION_IN_TOOLTIP);
196:     this.setInputsInline(false);
197:   },
198:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_POSITION_IN_TITLE_POSITION }]
199: };
200:
201:
202: Blockly.Blocks['lists_select_item'] = {
203:   // Select from list an item.
204:   category : 'Lists',
205:   helpUrl : Blockly.Msg.LANG_LISTS_SELECT_ITEM_TITLE_HELPURL,
206:   init : function() {
207:     this.setColour(Blockly.LIST_CATEGORY_HUE);
208:     this.setOutput(true, null);
209:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
210:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
211:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_SELECT_ITEM_INPUT,
212:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
213:             ['NUM', checkTypeNumber, Blockly.ALIGN_RIGHT],
214:             Blockly.ALIGN_RIGHT);
215:     this.setTooltip(Blockly.Msg.LANG_LISTS_SELECT_ITEM_TOOLTIP);
216:     this.setInputsInline(false);
217:   },
218:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_SELECT_ITEM_TITLE_SELECT }]
219: };
220:
221: Blockly.Blocks['lists_insert_item'] = {
222:   // Insert Item in list.
223:   category : 'Lists',
224:   helpUrl : Blockly.Msg.LANG_LISTS_INSERT_ITEM_HELPURL,
225:   init : function() {
226:     this.setColour(Blockly.LIST_CATEGORY_HUE);
227:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
228:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
229:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
```

```
230:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_INSERT_INPUT,
231:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
232:             ['INDEX', checkTypeNumber, Blockly.ALIGN_RIGHT],
233:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
234:             Blockly.ALIGN_RIGHT);
235:     this.setPreviousStatement(true);
236:     this.setNextStatement(true);
237:     this.setTooltip(Blockly.Msg.LANG_LISTS_INSERT_TOOLTIP);
238:     this.setInputsInline(false);
239:   },
240:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_INSERT_TITLE_INSERT_LIST }]
241: };
242:
243: Blockly.Blocks['lists_replace_item'] = {
244:   // Replace Item in list.
245:   category : 'Lists',
246:   helpUrl : Blockly.Msg.LANG_LISTS_REPLACE_ITEM_HELPURL,
247:   init : function() {
248:     this.setColour(Blockly.LIST_CATEGORY_HUE);
249:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
250:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
251:     var checkTypeAny = Blockly.Blocks.Utilities.YailTypeToBlocklyType("any",Blockly.
Blocks.Utilities.INPUT);
252:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_REPLACE_ITEM_INPUT,
253:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
254:             ['NUM', checkTypeNumber, Blockly.ALIGN_RIGHT],
255:             ['ITEM', checkTypeAny, Blockly.ALIGN_RIGHT],
256:             Blockly.ALIGN_RIGHT);
257:     this.setPreviousStatement(true);
258:     this.setNextStatement(true);
259:     this.setTooltip(Blockly.Msg.LANG_LISTS_REPLACE_ITEM_TOOLTIP);
260:     this.setInputsInline(false);
261:   },
262:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_REPLACE_ITEM_TITLE_REPLACE }]
263: };
264:
265: Blockly.Blocks['lists_remove_item'] = {
266:   // Remove Item in list.
267:   category : 'Lists',
268:   helpUrl : Blockly.Msg.LANG_LISTS_REMOVE_ITEM_HELPURL,
269:   init : function() {
270:     this.setColour(Blockly.LIST_CATEGORY_HUE);
271:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
272:     var checkTypeNumber = Blockly.Blocks.Utilities.YailTypeToBlocklyType("number",Bl
ockly.Blocks.Utilities.INPUT);
273:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_REMOVE_ITEM_INPUT,
274:             ['LIST', checkTypeList, Blockly.ALIGN_RIGHT],
275:             ['INDEX', checkTypeNumber, Blockly.ALIGN_RIGHT],
276:             Blockly.ALIGN_RIGHT);
277:     this.setPreviousStatement(true);
278:     this.setNextStatement(true);
279:     this.setTooltip(Blockly.Msg.LANG_LISTS_REMOVE_ITEM_TOOLTIP);
280:     this.setInputsInline(false);
281:   },
282:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_REMOVE_ITEM_TITLE_REMOVE }]
283: };
284:
285: Blockly.Blocks['lists_append_list'] = {
286:   // Append to list.
287:   category : 'Lists',
288:   helpUrl : Blockly.Msg.LANG_LISTS_APPEND_LIST_HELPURL,
289:   init : function() {
290:     this.setColour(Blockly.LIST_CATEGORY_HUE);
291:     var checkTypeList = Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockl
y.Blocks.Utilities.INPUT);
292:     this.interpolateMsg(Blockly.Msg.LANG_LISTS_APPEND_LIST_INPUT,
293:             ['LIST0', checkTypeList, Blockly.ALIGN_RIGHT],
294:             ['LIST1', checkTypeList, Blockly.ALIGN_RIGHT],
295:             Blockly.ALIGN_RIGHT);
296:     this.setPreviousStatement(true);
297:     this.setNextStatement(true);
298:     this.setTooltip(Blockly.Msg.LANG_LISTS_APPEND_LIST_TOOLTIP);
299:     this.setInputsInline(false);
300:   },
301:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_APPEND_LIST_TITLE_APPEND }]
302: };
303:
304:
305: Blockly.Blocks['lists_copy'] = {
306:   // Make a copy of list.
307:   category : 'Lists',
308:   helpUrl : Blockly.Msg.LANG_LISTS_COPY_HELPURL,
309:   init : function() {
310:     this.setColour(Blockly.LIST_CATEGORY_HUE);
311:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Block
ly.Blocks.Utilities.OUTPUT));
312:     this.appendValueInput('LIST')
313:       .setCheck(Blockly.Blocks.Utilities.YailTypeToBlocklyType("list",Blockly.Blocks
.Utilities.INPUT))
314:       .appendField(Blockly.Msg.LANG_LISTS_COPY_TITLE_COPY)
315:       .appendField(Blockly.Msg.LANG_LISTS_COPY_INPUT_LIST);
316:     this.setTooltip(Blockly.Msg.LANG_LISTS_COPY_TOOLTIP);
317:   },
318:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_COPY_TITLE_COPY }]
319: };
320:
321: Blockly.Blocks['lists_is_list'] = {
322:   // Is a list?
323:   category : 'Lists',
324:   helpUrl : Blockly.Msg.LANG_LISTS_IS_LIST_HELPURL,
325:   init : function() {
326:     this.setColour(Blockly.LIST_CATEGORY_HUE);
327:     this.setOutput(true, Blockly.Blocks.Utilities.YailTypeToBlocklyType("boolean",Bl
ockly.Blocks.Utilities.OUTPUT));
328:     this.appendValueInput('ITEM')
329:       .appendField(Blockly.Msg.LANG_LISTS_IS_LIST_TITLE_IS_LIST)
330:       .appendField(Blockly.Msg.LANG_LISTS_IS_LIST_INPUT_THING);
331:     this.setTooltip(Blockly.Msg.LANG_LISTS_IS_LIST_TOOLTIP);
332:   },
333:   typeblock: [{ translatedName: Blockly.Msg.LANG_LISTS_IS_LIST_TITLE_IS_LIST }]
334: };
335:
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Variables blocks for Blockly, modified for MIT App Inventor.
 * @author fturbak@wellesley.edu (Lyn Turbak)
 */

/**
 * Lyn's History:
 *  * [lyn, written 11/16-17/13, added 07/01/14]
 *   + Added freeVariables, renameFree, and renameBound to local declarations
 *   + Added freeVariables and renameFree to getters and setters
 *   + Added renameVar and renameVars to local declaration
 *   + renamed addDeclarationInputs_ to updatedDeclarationInputs_
 * [lyn, 03/04/13]
 *   + Remove notion of collapsed input from local variable declaration statements/expressions,
 *     which has been eliminated in updated to Blockly v1636.
 *   + Update appendTitle* to appendField*
 * [lyn, 01/18/13] Remove onchange from lexical_variable_get and lexical_variable_set.
 *    This fixes issue 667 (Variable getter/setter names deleted in copied blocks)
 *    and improves laggy drag problem.
 * [lyn, 10/27/13]
 *   + Modified local declaration parameter flydowns so editing the name changes corresponding name in an open mutator.
 *   + Changed local declaration compose() to rebuild inputs only if local names have changed.
 *     (essential for getting param flydown name changes reflected in open mutator).
 *   + Fixed local declaration expression compose() to be the same as that for local declaration statements.
 *   + Modified addDeclarationInputs_ to remove existing declarations, add new ones, and keep
 *     last two declarations (body and collapsed text) rather than recreating them.
 *     This is now used by both domToMutation() and compose(), eliminating duplicated code.
 *   + Eliminated dummy declarations.
 *   + Specify direction of flydowns
 * [lyn, 10/25/13] Made collapsed block labels more sensible.
 * [lyn, 10/10-14/13]
 *   + Installed variable declaration flydowns in global definition and local variable declaration
 *     statements and expressions.
 *   + Abstracted over string labels on all blocks using constants defined in en/_messages.js
 *   + Cleaned up code, including refactoring to increase sharing between
 *     local_declaration_statement and local_declaration_expression.
 *   + Fixed bug: Modified onchange for local declarations to keep localNames_ instance
 *     variable updated when param is edited directly on declaration block.
 *   + In local variable statements/expression, changed both "in do" and "in return"
 *     to "scope" (shape distinguishes them). But maybe these should just be empty string?
 * [lyn, 11/18/12] Renaming for globals (still working on renaming of procedure and loop params)
 * [lyn, 11/17/12] Integration of simple naming into App Inventor
 * [lyn, 11/11/12] More work on onchange event. Allow invalid names for untethered getters/setters
 *                 on workspace, but not when click in to other blocks.
 * [lyn, 11/08-10/12] Get dropdown list of names in scope to work for globals and params
 *                    (including loops) in raw blockly. Pass along to Andrew for integration
 *                    into AI. Initial work on onchange event to change names when getters/setters
 *                    copied and moved.
 * [lyn, 11/05-07/12] Add local variable declaration expressions. Get mutator working for local
 *                    declaration statements and expressions. But these don't save/load properly from XML
 *                    Helpful 10/7 hangout with Andrew and Paul.
 * [lyn, 11/04/12] Created. Add global declarations. Work on local variable declaration statement.
 */

/*
// For debugging only
function myStringify (obj) {
  var seen = [];
  return JSON.stringify(obj, function(key, val) {
   if (typeof val == "object") {
       if (seen.indexOf(val) >= 0) {
           return undefined;
       }
       seen.push(val);
   }
   return val
   });
}
*/

'use strict';

goog.provide('Blockly.Blocks.lexicalvariables');
goog.require('Blockly.Blocks.Utilities');
goog.require('goog.dom');

/**
 * Prototype bindings for a global variable declaration block
 */
Blockly.Blocks['global_declaration'] = {
  // Global var defn
  category: 'Variables',
  helpUrl: Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_HELPURL,
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendValueInput('VALUE')
        .appendField(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TITLE_INIT)
        .appendField(new Blockly.FieldGlobalFlydown(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_NAME,
                                                    Blockly.FieldFlydown.DISPLAY_BELOW), 'NAME')
        .appendField(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TO);
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TOOLTIP);
  },
  getVars: function() {
    return [this.getFieldValue('NAME')];
  },
  renameVar: function(oldName, newName) {
    if (Blockly.Names.equals(oldName, this.getFieldValue('VAR'))) {
      this.setFieldValue(newName, 'NAME');
```

```
  104:     }
  105:   },
  106:   typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_GLOBAL_DECLARATION_TITLE_
INIT }]
  107: };
  108:
  109: /**
  110:  * Prototype bindings for a variable getter block
  111:  */
  112: Blockly.Blocks['lexical_variable_get'] = {
  113:   // Variable getter.
  114:   category: 'Variables',
  115:   helpUrl: Blockly.Msg.LANG_VARIABLES_GET_HELPURL,
  116:   init: function() {
  117:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
  118:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
  119:     this.fieldVar_.setBlock(this);
  120:     this.appendDummyInput()
  121:         .appendField(Blockly.Msg.LANG_VARIABLES_GET_TITLE_GET)
  122:         .appendField(this.fieldVar_, 'VAR');
  123:     this.setOutput(true, null);
  124:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_GET_TOOLTIP);
  125:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
  126:   },
  127:   mutationToDom: function() { // Handle getters for event parameters specially (to s
upport i8n)
  128:     return Blockly.LexicalVariable.eventParamMutationToDom(this);
  129:   },
  130:   domToMutation: function(xmlElement) { // Handler getters for event parameters spec
ially (to support i8n)
  131:     Blockly.LexicalVariable.eventParamDomToMutation(this, xmlElement);
  132:   },
  133:   getVars: function() {
  134:     return [this.getFieldValue('VAR')];
  135:   },
  136:   renameLexicalVar: function(oldName, newName) {
  137:     // console.log("Renaming lexical variable from " + oldName + " to " + newName);
  138:     if (oldName === this.getFieldValue('VAR')) {
  139:         this.setFieldValue(newName, 'VAR');
  140:     }
  141:   },
  142:   renameFree: function (freeSubstitution) {
  143:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  144:     var prefix = prefixPair[0];
  145:     // Only rename lexical (nonglobal) names
  146:     if (prefix !== Blockly.globalNamePrefix) {
  147:       var oldName = prefixPair[1];
  148:       var newName = freeSubstitution.apply(oldName);
  149:       if (newName !== oldName) {
  150:         this.renameLexicalVar(oldName, newName);
  151:       }
  152:     }
  153:   },
  154:   freeVariables: function() { // return the free lexical variables of this block
  155:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  156:     var prefix = prefixPair[0];
  157:     // Only return lexical (nonglobal) names
  158:     if (prefix !== Blockly.globalNamePrefix) {
  159:       var oldName = prefixPair[1];
  160:       return new Blockly.NameSet([oldName])
  161:     } else {
  162:       return new Blockly.NameSet();
  163:     }
  164:   },
  165:   typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_GET_TITLE_GET + Blockly.M
sg.LANG_VARIABLES_VARIABLE }]
  166: };
  167:
  168: /**
  169:  * Prototype bindings for a variable setter block
  170:  */
  171: Blockly.Blocks['lexical_variable_set'] = {
  172:   // Variable setter.
  173:   category: 'Variables',
  174:   helpUrl: Blockly.Msg.LANG_VARIABLES_SET_HELPURL, // *** [lyn, 11/10/12] Fix this
  175:   init: function() {
  176:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
  177:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
  178:     this.fieldVar_.setBlock(this);
  179:     this.appendValueInput('VALUE')
  180:         .appendField(Blockly.Msg.LANG_VARIABLES_SET_TITLE_SET)
  181:         .appendField(this.fieldVar_, 'VAR')
  182:         .appendField(Blockly.Msg.LANG_VARIABLES_SET_TITLE_TO);
  183:     this.setPreviousStatement(true);
  184:     this.setNextStatement(true);
  185:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_SET_TOOLTIP);
  186:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
  187:   },
  188:   mutationToDom: function() { // Handle setters for event parameters specially (to s
upport i8n)
  189:     return Blockly.LexicalVariable.eventParamMutationToDom(this);
  190:   },
  191:   domToMutation: function(xmlElement) { // Handler setters for event parameters spec
ially (to support i8n)
  192:     Blockly.LexicalVariable.eventParamDomToMutation(this, xmlElement);
  193:   },
  194:   getVars: function() {
  195:     return [this.getFieldValue('VAR')];
  196:   },
  197:   renameLexicalVar: Blockly.Blocks.lexical_variable_get.renameLexicalVar,
  198:   renameFree: function (freeSubstitution) {
  199:     // potentially rename the set variable
  200:     var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
  201:     var prefix = prefixPair[0];
  202:     // Only rename lexical (nonglobal) names
  203:     if (prefix !== Blockly.globalNamePrefix) {
  204:       var oldName = prefixPair[1];
  205:       var newName = freeSubstitution.apply(oldName);
  206:       if (newName !== oldName) {
  207:         this.renameLexicalVar(oldName, newName);
  208:       }
  209:     }
  210:     // [lyn, 06/26/2014] Don't forget to rename children!
  211:     this.getChildren().map( function(blk) { Blockly.LexicalVariable.renameFree(blk,
freeSubstitution); })
  212:   },
  213:   freeVariables: function() { // return the free lexical variables of this block
  214:     // [lyn, 06/27/2014] Find free vars of *all* children, including subsequent comm
ands in NEXT slot.
  215:     var childrenFreeVars = this.getChildren().map( function(blk) { return Blockly.Le
xicalVariable.freeVariables(blk); } );
  216:     var result = Blockly.NameSet.unionAll(childrenFreeVars);
```

```
    var prefixPair = Blockly.unprefixName(this.getFieldValue('VAR'));
    var prefix = prefixPair[0];
    // Only return lexical (nonglobal) names
    if (prefix !== Blockly.globalNamePrefix) {
      var oldName = prefixPair[1];
      result.insert(oldName);
    }
    return result;
  },
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_SET_TITLE_SET + Blockly.Msg.LANG_VARIABLES_VARIABLE }]
};

/**
 * Prototype bindings for a statement block that declares local names for use in a statement body.
 * [lyn, 10/13/13] Refactored to share more code with Blockly.Blocks.local_declaration_expression
 */
Blockly.Blocks['local_declaration_statement'] = {
  // Define a procedure with no return value.
  // category: null,  // Procedures are handled specially.
  category: 'Variables',  // *** [lyn, 11/07/12] Abstract over this
  helpUrl: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_HELPURL,
  bodyInputName: 'STACK',
  init: function() {
    this.initLocals();
    this.appendStatementInput('STACK')
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_IN_DO);

    // Add notch and nub for vertical statement composition
    this.setPreviousStatement(true);
    this.setNextStatement(true);

    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TOOLTIP);
  },
  initLocals: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.localNames_ = [Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_DEFAULT_NAME];
    var declInput = this.appendValueInput('DECL0');
    declInput.appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TITLE_INIT)
             .appendField(this.parameterFlydown(0), 'VAR0')
             .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_INPUT_TO)
             .setAlign(Blockly.ALIGN_RIGHT);

    // Add mutator for editing local variable names
    this.setMutator(new Blockly.Mutator(['local_mutatorarg']));
  },
  onchange: function () {
     this.localNames_ = this.declaredNames(); // ensure localNames_ is in sync with paramFlydown fields
   },
  mutationToDom: function() { // Store local names in mutation element of XML for block
    var container = document.createElement('mutation');
    for (var i = 0; i< this.localNames_.length; i++) {
      var parameter = document.createElement('localname');
      parameter.setAttribute('name', this.localNames_[i]);
      container.appendChild(parameter);
    }
    return container;
  },
  domToMutation: function(xmlElement) { // Retrieve local names from mutation element of XML for block
                                        // and replace existing declarations
    var children = goog.dom.getChildren(xmlElement);
    if (children.length > 0) { // Ensure xml element is nonempty
      // Else we'll overwrite initial list with "name" for new block
      this.localNames_ = [];
      for (var i = 0, childNode; childNode = children[i]; i++) {
        if (childNode.nodeName.toLowerCase() == 'localname') {
          this.localNames_.push(childNode.getAttribute('name'));
        }
      }
    }
    this.updateDeclarationInputs_(this.localNames_); // add declarations; inits are undefined
  },
  updateDeclarationInputs_: function(names, inits) {
    // Modify this block to replace existing initializers by new declaration inputs created from names and inits.
    // If inits is undefined, treat all initial expressions as undefined.
    // Keep existing body at end of input list.
    // [lyn, 03/04/13] As of change to, Blockly 1636, there is no longer a collapsed input at end.

    // Remember last (= body) input
    var bodyInput = this.inputList[this.inputList.length - 1]; // Body input for local declaration
    var numDecls = this.inputList.length - 1;

    // [lyn, 07/03/14] stop rendering until block is recreated
    var savedRendered = this.rendered;
    this.rendered = false;

    // Modify this local-in-do block according to arrangement of name blocks in mutator editor.
    // Remove all the local declaration inputs ...
    var thisBlock = this; // Grab correct object for use in thunk below
    Blockly.FieldParameterFlydown.withChangeHanderDisabled(
        // [lyn, 07/02/14] Need to disable change handler, else this will try to rename params removed fields.
        function() {
          for (var i = 0; i < numDecls; i++) {
            thisBlock.removeInput('DECL' + i);
          }
        }
    );

    // Empty the inputList and recreate it, building local initializers from mutator
    this.inputList = [];
    this.localNames_ = names;

    for (var i = 0; i < names.length; i++) {
      var declInput = this.appendValueInput('DECL' + i);
      // [lyn, 11/06/12]
      //   This was for case where tried to put "local" keyword on same line with first local name.
      //   But even though alignment set to Blockly.ALIGN_RIGHT, the input was left justified
      //   and covered the plus sign for popping up the mutator. So I put the "local " keyword
      //   on it's own line even though this wastes vertical space. This should be fixed in the future.
```

```
325:       // if (i == 0) {
326:       //  declInput.appendField("local"); // Only put keyword "local" on top line.
327:       // }
328:       declInput.appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TITLE_INIT)
329:                .appendField(this.parameterFlydown(i), 'VAR' + i)
330:                .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_INPUT_TO)
331:                .setAlign(Blockly.ALIGN_RIGHT);
332:       if (inits && inits[i]) { // If there is an initializer, connect it
333:         declInput.connection.connect(inits[i]);
334:       }
335:     }
336:
337:     // Now put back last (= body) input
338:     this.inputList = this.inputList.concat(bodyInput);
339:
340:     this.rendered = savedRendered;
341:     if (this.rendered) {
342:       this.render();
343:     }
344:   },
345:   // [lyn, 10/27/13] Introduced this to correctly handle renaming of mutatorarg in o
pen mutator
346:   // when procedure parameter flydown name is edited.
347:   parameterFlydown: function (paramIndex) { // Return a new local variable parameter
 flydown
348:     var initialParamName = this.localNames_[paramIndex];
349:     var localDecl = this; // Here, "this" is the local decl block. Name it to use in
 function below
350:     var localWorkspace = this.workspace;
351:     var localParameterChangeHandler = function (newParamName) {
352:       // This handler has the same subtleties as procedureParameterChangeHandler in
language/common/procedures.js,
353:       // but is somewhat simpler since doesn't have associated callers to change. Se
e the notes there.
354:
355:       // See Subtleties #1 and #2 in  procedureParameterChangeHandler in language/co
mmon/procedures.js
356:       var newLocals = localDecl.localNames_;
357:       newLocals[paramIndex] = newParamName;
358:
359:       // If there's an open mutator, change the name in the corresponding slot.
360:       if (localDecl.mutator && localDecl.mutator.rootBlock_) {
361:         // Iterate through mutatorarg param blocks and change name of one at paramIn
dex
362:         var mutatorContainer = localDecl.mutator.rootBlock_;
363:         var mutatorargIndex = 0;
364:         var mutatorarg = mutatorContainer.getInputTargetBlock('STACK');
365:         while (mutatorarg && mutatorargIndex < paramIndex) {
366:           mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.target
Block();
367:           mutatorargIndex++;
368:         }
369:         if (mutatorarg && mutatorargIndex == paramIndex) {
370:           // See Subtlety #3 in  procedureParameterChangeHandler in language/common/
procedures.js
371:           Blockly.Field.prototype.setText.call(mutatorarg.getField_("NAME"), newPara
mName);
372:         }
373:       }
374:     }
375:     return new Blockly.FieldParameterFlydown(initialParamName,
376:         true, // name is editable
377:         Blockly.FieldFlydown.DISPLAY_RIGHT,
378:         localParameterChangeHandler);
379:   },
380:   decompose: function(workspace) {
381:     // Create "mutator" editor populated with name blocks with local variable names
382:     var containerBlock = new Blockly.Block.obtain(workspace, 'local_mutatorcontainer
');
383:     containerBlock.initSvg();
384:     containerBlock.setDefBlock(this);
385:     var connection = containerBlock.getInput('STACK').connection;
386:     for (var i = 0; i < this.localNames_.length; i++) {
387:       var localName = this.getFieldValue('VAR' + i);
388:       var nameBlock = new Blockly.Block.obtain(workspace, 'local_mutatorarg');
389:       nameBlock.initSvg();
390:       nameBlock.setFieldValue(localName, 'NAME');
391:       // Store the old location.
392:       nameBlock.oldLocation = i;
393:       connection.connect(nameBlock.previousConnection);
394:       connection = nameBlock.nextConnection;
395:     }
396:     return containerBlock;
397:   },
398:   compose: function(containerBlock) {
399:     // [lyn, 10/27/13] Modified this so that doesn't rebuild block if names haven't
changed.
400:     // This is *essential* to handle Subtlety #3 in localParameterChangeHandler with
in parameterFlydown.
401:
402:     var newLocalNames = [];
403:     var initializers = [];
404:     var mutatorarg = containerBlock.getInputTargetBlock('STACK');
405:     while (mutatorarg) {
406:       newLocalNames.push(mutatorarg.getFieldValue('NAME'));
407:       initializers.push(mutatorarg.valueConnection_); // pushes undefined if doesn't
 exist
408:       mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.targetBloc
k();
409:     }
410:
411:     // Reconstruct inputs only if local list has changed
412:     if (!Blockly.LexicalVariable.stringListsEqual(this.localNames_, newLocalNames))
{
413:
414:       // Switch off rendering while the block is rebuilt.
415:       // var savedRendered = this.rendered;
416:       // this.rendered = false;
417:
418:       this.updateDeclarationInputs_(newLocalNames, initializers);
419:
420:       // Restore rendering and show the changes.
421:       // this.rendered = savedRendered;
422:       // if (this.rendered) {
423:       //  this.render();
424:       // }
425:     }
426:   },
427:   dispose: function() {
428:     // *** [lyn, 11/07/12] Dunno if anything needs to be done here.
429:     // Call parent's destructor.
430:     Blockly.Block.prototype.dispose.apply(this, arguments);
431:     // [lyn, 11/07/12] In above line, don't know where "arguments" param comes from,
432:     // but if it's remove, there's no clicking sound upon deleting the block!
```

```
  433:   },
  434:   saveConnections: function(containerBlock) {
  435:     // Store child initializer blocks for local name declarations with name blocks i
n mutator editor
  436:     var nameBlock = containerBlock.getInputTargetBlock('STACK');
  437:     var i = 0;
  438:     while (nameBlock) {
  439:       var localDecl = this.getInput('DECL' + i);
  440:       nameBlock.valueConnection_ =
  441:         localDecl && localDecl.connection.targetConnection;
  442:       i++;
  443:       nameBlock = nameBlock.nextConnection &&
  444:       nameBlock.nextConnection.targetBlock();
  445:     }
  446:     // Store body statement or expression connection
  447:     var bodyInput = this.getInput(this.bodyInputName); // 'STACK' or 'RETURN'
  448:     if (bodyInput) {
  449:       containerBlock.bodyConnection_ = bodyInput.connection.targetConnection;
  450:     }
  451:   },
  452:   getVars: function() {
  453:     var varList = [];
  454:     for (var i = 0, input; input = this.getFieldValue('VAR' + i); i++) {
  455:       varList.push(input);
  456:     }
  457:     return varList;
  458:   },
  459:   declaredNames: function () { // Interface with Blockly.LexicalVariable.renameParam
  460:     return this.getVars();
  461:   },
  462:   initializerConnections: function() { // [lyn, 11/16/13 ] Return all the initialize
r connections
  463:     var connections = [];
  464:     for (var i = 0, input; input = this.getInput('DECL' + i); i++) {
  465:       connections.push(input.connection && input.connection.targetConnection);
  466:     }
  467:     return connections;
  468:   },
  469:   blocksInScope: function () { // Interface with Blockly.LexicalVariable.renameParam
  470:     var doBody = this.getInputTargetBlock(this.bodyInputName); // *** [lyn, 11/24/12
] This will go away with DO-AND-RETURN block
  471:     var doBodyList = (doBody && [doBody]) || []; // List of non-null doBody or empty
 list for null doBody
  472:     return doBodyList; // List of non-null body elements.
  473:   },
  474:   renameVar: function(oldName, newName) {
  475:     this.renameVars(Blockly.Substitution.simpleSubstitution(oldName,newName));
  476:   },
  477:   renameVars: function(substitution) { // substitution is a dict (i.e., object) mapp
ing old names to new ones
  478:     var localNames = this.declaredNames();
  479:     var renamedLocalNames = substitution.map(localNames);
  480:     if (!Blockly.LexicalVariable.stringListsEqual(renamedLocalNames, localNames)) {
  481:       var initializerConnections = this.initializerConnections();
  482:       this.updateDeclarationInputs_(renamedLocalNames, initializerConnections);
  483:       // Update the mutator's variables if the mutator is open.
  484:       if (this.mutator.isVisible()) {
  485:         var blocks = this.mutator.workspace_.getAllBlocks();
  486:         for (var x = 0, block; block = blocks[x]; x++) {
  487:           if (block.type == 'procedures_mutatorarg') {
  488:             var oldName = block.getFieldValue('NAME');
  489:             var newName = substitution.appy(oldName);
  490:             if (newName !== oldName) {
  491:               block.setFieldValue(newName, 'NAME');
  492:             }
  493:           }
  494:         }
  495:       }
  496:     }
  497:   },
  498:   renameBound: function (boundSubstitution, freeSubstitution) {
  499:     var localNames = this.declaredNames();
  500:     for (var i = 0; i < localNames.length; i++) {
  501:       // This is LET semantics, not LET* semantics, and needs to change!
  502:       Blockly.LexicalVariable.renameFree(this.getInputTargetBlock('DECL'+i), freeSub
stitution);
  503:     }
  504:     var paramSubstitution = boundSubstitution.restrictDomain(localNames);
  505:     this.renameVars(paramSubstitution);
  506:     var newFreeSubstitution = freeSubstitution.remove(localNames).extend(paramSubsti
tution);
  507:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock(this.bodyInputName),
 newFreeSubstitution);
  508:     if (this.nextConnection) {
  509:       var nextBlock = this.nextConnection.targetBlock();
  510:       Blockly.LexicalVariable.renameFree(nextBlock, freeSubstitution);
  511:     }
  512:   },
  513:   renameFree: function (freeSubstitution) {
  514:     // This is LET semantics, not LET* semantics, and needs to change!
  515:     var localNames = this.declaredNames();
  516:     var localNameSet = new Blockly.NameSet(localNames);
  517:     var bodyFreeVars = Blockly.LexicalVariable.freeVariables(this.getInputTargetBloc
k(this.bodyInputName));
  518:     bodyFreeVars.subtract(localNameSet);
  519:     var renamedFreeVars = bodyFreeVars.renamed(freeSubstitution);
  520:     var capturedVars = renamedFreeVars.intersection(localNameSet);
  521:     if (! capturedVars.isEmpty()) { // Case where some names are captured!
  522:       // Must consistently rename declarations and uses of capturedFreeVars with
  523:       // names that do not conflict with renamedFreeVars, localNames, or each other.
  524:       var forbiddenNames = localNameSet.union(renamedFreeVars).toList();
  525:       var boundBindings = {};
  526:       var capturedVarList = capturedVars.toList();
  527:       for (var i= 0, capturedVar; capturedVar = capturedVarList[i]; i++) {
  528:         var newCapturedVar = Blockly.FieldLexicalVariable.nameNotIn(capturedVar, for
biddenNames);
  529:         boundBindings[capturedVar] = newCapturedVar;
  530:         forbiddenNames.push(newCapturedVar);
  531:       }
  532:       this.renameBound(new Blockly.Substitution(boundBindings), freeSubstitution);
  533:     } else {
  534:       this.renameBound(new Blockly.Substitution(), freeSubstitution);
  535:     }
  536:   },
  537:   freeVariables: function() { // return the free lexical variables of this block
  538:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock(this
.bodyInputName));
  539:     var localNames = this.declaredNames();
  540:     result.subtract(new Blockly.NameSet(localNames)); // This is LET semantics, not
LET* semantics, but should be changed!
  541:     var numDecls = localNames.length;
  542:     for (var i = 0; i < numDecls; i++) {
  543:       result.union(Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock('D
ECL'+i)));
```

```
    }
    if (this.nextConnection) {
      var nextBlock = this.nextConnection.targetBlock();
      result.unite(Blockly.LexicalVariable.freeVariables(nextBlock));
    }
    return result;
  },
  //TODO (user) this has not been internationalized yet
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_TRANSLATED_NAME }]
};

/**
 * Prototype bindings for an expression block that declares local names for use in an expression body.
 * [lyn, 10/13/13] Refactored to share more code with Blockly.Blocks.local_declaration_statement
 */
Blockly.Blocks['local_declaration_expression'] = {
  category: 'Variables',  // *** [lyn, 11/07/12] Abstract over this
  helpUrl: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_HELPURL,
  initLocals: Blockly.Blocks.local_declaration_statement.initLocals,
  bodyInputName: 'RETURN',
  init: function() {
    this.initLocals();
    this.appendIndentedValueInput('RETURN')
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_IN_RETURN);
    // Create plug for expression output
    this.setOutput(true, null);
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_TOOLTIP);
  },
  onchange: Blockly.Blocks.local_declaration_statement.onchange,
  mutationToDom: Blockly.Blocks.local_declaration_statement.mutationToDom,
  domToMutation: Blockly.Blocks.local_declaration_statement.domToMutation,
  updateDeclarationInputs_: Blockly.Blocks.local_declaration_statement.updateDeclarationInputs_,
  parameterFlydown: Blockly.Blocks.local_declaration_statement.parameterFlydown,
  blocksInScope: Blockly.Blocks.local_declaration_statement.blocksInScope,
  decompose: Blockly.Blocks.local_declaration_statement.decompose,
  compose: Blockly.Blocks.local_declaration_statement.compose,
  dispose: Blockly.Blocks.local_declaration_statement.dispose,
  saveConnections: Blockly.Blocks.local_declaration_statement.saveConnections,
  getVars: Blockly.Blocks.local_declaration_statement.getVars,
  declaredNames: Blockly.Blocks.local_declaration_statement.declaredNames,
  renameVar: Blockly.Blocks.local_declaration_statement.renameVars,
  renameVars: Blockly.Blocks.local_declaration_statement.renameVar,
  renameBound: Blockly.Blocks.local_declaration_statement.renameBound,
  renameFree: Blockly.Blocks.local_declaration_statement.renameFree,
  freeVariables: Blockly.Blocks.local_declaration_statement.freeVariables,
  //TODO (user) this has not been internationalized yet
  typeblock: [{ translatedName: Blockly.Msg.LANG_VARIABLES_LOCAL_DECLARATION_EXPRESSION_TRANSLATED_NAME }]
};

Blockly.Blocks['local_mutatorcontainer'] = {
  // Local variable container (for mutator dialog).
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_CONTAINER_TITLE_LOCAL_NAMES);
    this.appendStatementInput('STACK');
    this.setTooltip(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_CONTAINER_TOOLTIP);
    this.contextMenu = false;
  },
  // [lyn. 11/24/12] Set procBlock associated with this container.
  setDefBlock: function (defBlock) {
    this.defBlock_ = defBlock;
  },
  // [lyn. 11/24/12] Set procBlock associated with this container.
  // Invariant: should not be null, since only created as mutator for a particular proc block.
  getDefBlock: function () {
    return this.defBlock_;
  },
  // [lyn. 11/24/12] Return list of param names in this container
  // Invariant: there should be no duplicates!
  declaredNames: function () {
    var paramNames = [];
    var paramBlock = this.getInputTargetBlock('STACK');
    while (paramBlock) {
      paramNames.push(paramBlock.getFieldValue('NAME'));
      paramBlock = paramBlock.nextConnection &&
                   paramBlock.nextConnection.targetBlock();
    }
    return paramNames;
  }
};

Blockly.Blocks['local_mutatorarg'] = {
  // Procedure argument (for mutator dialog).
  init: function() {
    this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
    this.appendDummyInput()
        .appendField(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_ARG_TITLE_NAME)
        .appendField(new Blockly.FieldTextInput(Blockly.Msg.LANG_VARIABLES_LOCAL_MUTATOR_ARG_DEFAULT_VARIABLE,
                                                Blockly.LexicalVariable.renameParam),
                     'NAME');
    this.setPreviousStatement(true);
    this.setNextStatement(true);
    this.setTooltip('');
    this.contextMenu = false;
  },
  getContainerBlock: function () {
    var parent = this.getParent();
    while (parent && ! (parent.type === "local_mutatorcontainer")) {
      parent = parent.getParent();
    }
    // [lyn, 11/24/12] Cache most recent container block so can reference it upon removal from mutator arg stack
    this.cachedContainerBlock_ = (parent && (parent.type === "local_mutatorcontainer") && parent) || null;
    return this.cachedContainerBlock_;
  },
  getDefBlock: function () {
    var container = this.getContainerBlock();
    return (container && container.getDefBlock()) || null;
  },
  blocksInScope: function () {
```

```
655:     var defBlock = this.getDefBlock();
656:     return (defBlock && defBlock.blocksInScope()) || [];
657:   },
658:   declaredNames: function () {
659:     var container = this.getContainerBlock();
660:     return (container && container.declaredNames()) || [];
661:   },
662:
663:   // [lyn, 11/24/12] Check for situation in which mutator arg has been removed from
stack,
664:   onchange: function() {
665:     var paramName = this.getFieldValue('NAME');
666:     if (paramName) { // paramName is null when delete from stack
667:       // console.log("Mutatorarg onchange: " + paramName);
668:       var cachedContainer = this.cachedContainerBlock_;
669:       var container = this.getContainerBlock(); // Order is important; this must com
e after cachedContainer
670:                                                 // since it sets cachedContainerBloc
k_
671:       // console.log("Mutatorarg onchange: " + paramName
672:       //           + "; cachedContainer = " + JSON.stringify((cachedContainer && ca
chedContainer.type) || null)
673:       //           + "; container = " + JSON.stringify((container && container.type
) || null));
674:       if ((! cachedContainer) && container) {
675:         // Event: added mutator arg to container stack
676:         // console.log("Mutatorarg onchange ADDED: " + paramName);
677:         var declaredNames = this.declaredNames();
678:         var firstIndex = declaredNames.indexOf(paramName);
679:         if (firstIndex != -1) {
680:           // Assertion: we should get here, since paramName should be among names
681:           var secondIndex = declaredNames.indexOf(paramName, firstIndex+1);
682:           if (secondIndex != -1) {
683:             // If we get here, there is a duplicate on insertion that must be resolv
ed
684:             var newName = Blockly.FieldLexicalVariable.nameNotIn(paramName,declaredN
ames);
685:             this.setFieldValue(newName, 'NAME');
686:           }
687:         }
688:       }
689:     }
690:   }
691: };
692:
693: /*---------------------------------------*/
694:
695: Blockly.Blocks['type_def'] = {
696:   category: 'Variables',
697:   init: function() {
698:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
699:     this.appendValueInput("DEF")
700:         .setCheck("Array")
701:         .appendField("type definition")
702:         .appendField(new Blockly.FieldTextInput("type"), "TYPE");
703:         //.appendField(new Blockly.FieldGlobalFlydown('type', Blockly.FieldFlydown.D
ISPLAY_BELOW), 'TYPE');
704:     this.setTooltip('');
705:   }
706: };
707:
708: Blockly.Blocks['atom_def'] = {
709:   category: 'Variables',
710:   init: function() {
711:     this.setColour(Blockly.VARIABLE_CATEGORY_HUE);
712:     this.appendDummyInput()
713:         .appendField("atom:")
714:         .appendField(new Blockly.FieldTextInput(""), "VALUE");
715:     this.setOutput(true, "Array");
716:     this.setTooltip('');
717:   }
718: };
719:
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2013-2014 MIT, All rights reserved
// Released under the Apache License, Version 2.0
// http://www.apache.org/licenses/LICENSE-2.0
/**
 * @license
 * @fileoverview Procedure blocks for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

/**
 * Lyn's Change History:
 * [lyn, written 11/16-17/13, added 07/01/14]
 *   + Added freeVariables, renameFree, and renameBound to procedure declarations
 *   + Added renameVars for procedure declarations, which allows renaming multiple parameters simultaneously
 *   + Modified updateParams_ to accept optional params argument
 *   + Introduced bodyInputName field for procedure declarations ('STACK' for procedures_defnoreturn;
 *     'RETURN' for procedures_return), and use this to share more methods between the two kinds
 *     of procedure declarations.
 *   + Replaced inlined string list equality tests by new Blockly.LexicalVariable.stringListsEqual
 * [lyn, 10/28/13]
 *   + Fixed a missing change of Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure
 *   + I was wrong about re-rendering not being needed in updatedParams_!
 *     Without it, changing horizontal -> vertical params doesn't handle body slot correctly.
 *     So added it back.
 * [lyn, 10/27/13]
 *   + Fix bug in list of callers in flyout by simplifying domToMutation for procedure callers.
 *     This should never look for associated declaration, but just take arguments from given xml.
 *   + Removed render() call from updateParams_. Seems unnecessary. <== I WAS WRONG. SEE 10/28/13 NOTE
 *   + Specify direction of flydowns
 *   + Replaced Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure in proc decls
 * [lyn, 10/26/13] Modify procedure parameter changeHandler to propagate name changes to caller arg labels
 *     and open mutator labels
 * [lyn, 10/25/13]
 *   + Modified procedures_defnoreturn compose method so that it doesn't call updateParams_
 *     if mutator hasn't changed parameter names. This helps avoid a situation where
 *     an attempt is made to update params of a collapsed declaration.
 *   + Modified collapsed decls to have 'to ' prefix and collapsed callers to have 'call ' prefix.
 * [lyn, 10/24/13] Allowed switching between horizontal and vertical display of arguments
 * [lyn, 10/23/13] Fixed bug in domToMutation for callers that was giving wrong args to caller blocks.
 * [lyn, 10/10-14/13]
 *   + Installed variable declaration flydowns in both types of procedure declarations.
 *   + Fixed bug: Modified onchange for procedure declarations to keep arguments_ instance
 *     variable updated when param is edited directly on declaration block.
 *   + Removed get block (still in Variable drawer; no longer needed with parameter flydowns)
 *   + Removed "do {} then-return []" block since (1) it's in Control drawer and
 *     (2) it will be superseded in the context by Michael Phox's proc_defnoreturn mutator
 *     that allows adding a DO statement.
 *   + TODO: Abstract over string labels on all blocks using constants defined in en/_messages.js
 *   + TODO: Clean up code, including refactoring to increase sharing between
 *     procedures_defnoreturn and procedures_defreturn.
 * [lyn, 11/29/12] Integrated into App Inventor blocks. Known bugs:
 *   + Reordering mutator_args in mutator_container changes references to ??? because it interprets it
 *     as removing and inserting rather than moving.
 * [lyn, 11/24/12] Implemented procedure parameter renaming:
 *   + changing a variable name in mutator_arg for procedure changes it immediately in references in body.
 *   + no duplicate names are allowed in mutator_args; alpha-renaming prevents this.
 *   + no variables can be captured by renaming; alpha-renaming prevents this.
 */
'use strict';

goog.provide('Blockly.Blocks.procedures');

goog.require('Blockly.Blocks.Utilities');
goog.require('goog.dom');

Blockly.Blocks['procedures_defnoreturn'] = {
  // Define a procedure with no return value.
  category: 'Procedures',  // Procedures are handled specially.
  helpUrl: Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_HELPURL,
  bodyInputName: 'STACK',
  init: function() {
    this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
    var name = Blockly.Procedures.findLegalName(
        Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_PROCEDURE, this);
    this.appendDummyInput('HEADER')
        .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DEFINE)
        // [lyn, 10/27/13] Replaced Blockly.Procedures.rename by Blockly.AIProcedure.renameProcedure
        .appendField(new Blockly.FieldTextInput(name, Blockly.AIProcedure.renameProcedure), 'NAME');
    this.horizontalParameters = true; // horizontal by default
    this.appendStatementInput('STACK')
        .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DO);
    this.setMutator(new Blockly.Mutator(['procedures_mutatorarg']));
    this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_TOOLTIP);
    this.arguments_ = []; // List of declared local variable names; has one ("name") initially
                          // Other methods guarantee the invariant that this variable contains
                          // the list of names declared in the local declaration block.
    this.warnings = [{name:"checkEmptySockets",sockets:["STACK"]}];
  },
  onchange: function () {
    this.arguments_ = this.declaredNames(); // ensure arguments_ is in sync with paramFlydown fields
  },
  updateParams_: function(opt_params) {  // make rendered block reflect the parameter names currently in this.arguments_
    // [lyn, 11/17/13] Added optional opt_params argument:
    //    If its falsey (null or undefined), use the existing this.arguments_ list
```

```
    //    Otherwise, replace this.arguments_ by opt_params
    // In either case, make rendered block reflect the parameter names in this.argum
ents_
    if (opt_params) {
      this.arguments_ = opt_params;
    }
    // Check for duplicated arguments.
    // [lyn 10/10/13] Note that in blocks edited within AI2, duplicate parameter nam
es should never occur
    //    because parameters are renamed to avoid duplication. But duplicates might
show up
    //    in XML code hand-edited by user.
    // console.log("enter procedures_defnoreturn updateParams_()");
    var badArg = false;
    var hash = {};
    for (var x = 0; x < this.arguments_.length; x++) {
      if (hash['arg_' + this.arguments_[x].toLowerCase()]) {
        badArg = true;
        break;
      }
      hash['arg_' + this.arguments_[x].toLowerCase()] = true;
    }
    if (badArg) {
      this.setWarningText(Blockly.Msg.LANG_PROCEDURES_DEF_DUPLICATE_WARNING);
    } else {
      this.setWarningText(null);
    }

    var procName = this.getFieldValue('NAME');
    //save the first two input lines and the last input line
    //to be re added to the block later
    // var firstInput = this.inputList[0];  // [lyn, 10/24/13] need to reconstruct f
irst input
    var bodyInput = this.inputList[this.inputList.length - 1]; // Body of procedure

    // stop rendering until block is recreated
    var savedRendered = this.rendered;
    this.rendered = false;

    // remove first input
    // console.log("updateParams_: remove input HEADER");
    var thisBlock = this; // Grab correct object for use in thunk below
    Blockly.FieldParameterFlydown.withChangeHanderDisabled(
        // [lyn, 07/02/14] Need to disable change handler, else this will try to ren
ame params for horizontal arg fields!
        function() {thisBlock.removeInput('HEADER');}
    );

    // [lyn, 07/02/14 fixed logic] remove all old argument inputs (if they were vert
ical)
    if (! this.horizontalParameters) {
      var oldArgCount = this.inputList.length - 1; // Only args and body are left
      if (oldArgCount > 0) {
        var paramInput0 = this.getInput('VAR0');
        if (paramInput0) { // Yes, they were vertical
          for (var i = 0; i < oldArgCount; i++)
          {
            try
            {
              Blockly.FieldParameterFlydown.withChangeHanderDisabled(
                  // [lyn, 07/02/14] Need to disable change handler, else this will
try to rename params for vertical arg fields!
                  function() {thisBlock.removeInput('VAR' + i);}
              );
            }
            catch(err)
            {
              console.log(err);
            }
          }
        }
      }
    }

    //empty the inputList then recreate it
    this.inputList = [];

    // console.log("updateParams_: create input HEADER");
    var headerInput =
        this.appendDummyInput('HEADER')
            .appendField(Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DEFINE)
            // [lyn, 10/28/13] Replaced Blockly.Procedures.rename by Blockly.AIProce
dure.renameProcedure
            .appendField(new Blockly.FieldTextInput(procName, Blockly.AIProcedure.re
nameProcedure), 'NAME');

    //add an input title for each argument
    //name each input after the block and where it appears in the block to reference
 it later
    for (var i = 0; i < this.arguments_.length; i++) {
      if (this.horizontalParameters) { // horizontal case
        headerInput.appendField(' ')
                   .appendField(this.parameterFlydown(i), // [lyn, 10/10/13] Changed
 to param flydown
                                'VAR' + i); // Tag with param tag to make it easy to
 find later.
      } else { // vertical case
        this.appendDummyInput('VAR' + i)
            // .appendField(this.arguments_[i])
            .appendField(this.parameterFlydown(i), 'VAR' + i)
            .setAlign(Blockly.ALIGN_RIGHT);
      }
    }

    //put the last two arguments back
    this.inputList = this.inputList.concat(bodyInput);

    this.rendered = savedRendered;
    // [lyn, 10/28/13] I thought this rerendering was unnecessary. But I was wrong!
    // Without it, get bug noticed by Andrew in which toggling horizontal -> vertica
l params
    // in procedure decl doesn't handle body tag appropriately!
    if (this.rendered) {
       this.render();
    }
    // console.log("exit procedures_defnoreturn updateParams_()");
  },
  // [lyn, 10/26/13] Introduced this to correctly handle renaming of [(1) caller arg
 labels and
  // (2) mutatorarg in open mutator] when procedure parameter flydown name is edited
.
  parameterFlydown: function (paramIndex) { // Return a new procedure parameter flyd
own
    var initialParamName = this.arguments_[paramIndex];
```

```
204:     var procDecl = this; // Here, "this" is the proc decl block. Name it to use in f
unction below
205:     var procWorkspace = this.workspace;
206:     var procedureParameterChangeHandler = function (newParamName) {
207:       // console.log("enter procedureParameterChangeHandler");
208:
209:
210:       // Extra work that needs to be done when procedure param name is changed, in a
ddition
211:       // to renaming lexical variables:
212:       //   1. Change all callers so label reflects new name
213:       //   2. If there's an open mutator, change the corresponding slot.
214:       // Note: this handler is invoked as method on field, so within the handler bod
y,
215:       // "this" will be bound to that field and *not* the procedure declaration obje
ct!
216:
217:       // Subtlety #1: within this changeHandler, procDecl.arguments_ has *not* yet b
een
218:       // updated to include newParamName. This only happens later. But since we know
 newParamName
219:       // *and* paramIndex, we know how to update procDecl.arguments_ ourselves!
220:
221:       // Subtlety #2: I would have thought we would want to create local copy of
222:       // procedure arguments_ list rather than mutate that list, but I'd be wrong!
223:       // Turns out that *not* mutating list here causes trouble below in the line
224:       //
225:       //   Blockly.Field.prototype.setText.call(mutatorarg.getTitle_("NAME"), newPar
amName);
226:       //
227:       // The reason is that this fires a change event in mutator workspace, which ca
uses
228:       // a call to the proc decl compose() method, and when it detects a difference
in
229:       // the arguments it calls proc decl updateParams_. This removes proc decl inpu
ts
230:       // before adding them back, and all hell breaks loose when the procedure name
field
231:       // and previous parameter flydown fields are disposed before an attempt is mad
e to
232:       // disposed this field. At this point, the SVG element associated with the pro
cedure name
233:       // is gone but the field is still in the title list. Attempting to dispose thi
s field
234:       // attempts to hide the open HTML editor widget, which attempts to re-render t
he
235:       // procedure declaration block. But the null SVG for the procedure name field
236:       // raises an exception.
237:       //
238:       // It turns out that by mutating proc decl arguments_, when compose() is calle
d,
239:       // updateParams_() is *not* called, and this prevents the above scenario.
240:       //  So rather than doing
241:       //
242:       //    var newArguments = [].concat(procDecl.arguments_)
243:       //
244:       // we instead do:
245:       var newArguments = procDecl.arguments_;
246:       newArguments[paramIndex] = newParamName;
247:
248:       var procName = procDecl.getFieldValue('NAME');
249:
250:       // 1. Change all callers so label reflects new name
251:       Blockly.Procedures.mutateCallers(procName, procWorkspace, newArguments, procDe
cl.paramIds_);
252:
253:       // 2. If there's an open mutator, change the name in the corresponding slot.
254:       if (procDecl.mutator && procDecl.mutator.rootBlock_) {
255:         // Iterate through mutatorarg param blocks and change name of one at paramIn
dex
256:         var mutatorContainer = procDecl.mutator.rootBlock_;
257:         var mutatorargIndex = 0;
258:         var mutatorarg = mutatorContainer.getInputTargetBlock('STACK');
259:         while (mutatorarg && mutatorargIndex < paramIndex) {
260:             mutatorarg = mutatorarg.nextConnection && mutatorarg.nextConnection.targ
etBlock();
261:             mutatorargIndex++;
262:         }
263:         if (mutatorarg && mutatorargIndex == paramIndex) {
264:           // Subtlety #3: If call mutatorargs's setValue, its change handler will be
 invoked
265:           // several times, and on one of those times, it will find new param name i
n
266:           // the procedures arguments_ instance variable and will try to renumber it
267:           // (e.g. "a" -> "a2"). To avoid this, invoke the setText method of its Fie
ld s
268:           // superclass directly. I.e., can't do this:
269:           //   mutatorarg.getTitle_("NAME").setValue(newParamName);
270:           // so instead do this:
271:             Blockly.Field.prototype.setText.call(mutatorarg.getField_("NAME"), newPa
ramName);
272:         }
273:       }
274:       // console.log("exit procedureParameterChangeHandler");
275:     }
276:     return new Blockly.FieldParameterFlydown(initialParamName,
277:                                              true, // name is editable
278:                                              // [lyn, 10/27/13] flydown location dep
ends on parameter orientation
279:                                              this.horizontalParameters ? Blockly.Fie
ldFlydown.DISPLAY_BELOW
280:                                                                        : Blockly.Fie
ldFlydown.DISPLAY_RIGHT,
281:                                              procedureParameterChangeHandler);
282:   },
283:   setParameterOrientation: function(isHorizontal) {
284:     var params = this.getParameters();
285:     if (params.length != 0 && isHorizontal !== this.horizontalParameters) {
286:       this.horizontalParameters = isHorizontal;
287:       this.updateParams_();
288:     }
289:   },
290:   mutationToDom: function() {
291:     var container = document.createElement('mutation');
292:     if (!this.horizontalParameters) {
293:       container.setAttribute('vertical_parameters', "true"); // Only store an elemen
t for vertical
294:       // The absence of this attribute means horizontal.
295:     }
296:     for (var x = 0; x < this.arguments_.length; x++) {
297:       var parameter = document.createElement('arg');
298:       parameter.setAttribute('name', this.arguments_[x]);
299:       container.appendChild(parameter);
300:     }
```

```
301:     return container;
302:   },
303:   domToMutation: function(xmlElement) {
304:     var params = [];
305:     var children = goog.dom.getChildren(xmlElement);
306:     for (var x = 0, childNode; childNode = children[x]; x++) {
307:       if (childNode.nodeName.toLowerCase() == 'arg') {
308:         params.push(childNode.getAttribute('name'));
309:       }
310:     }
311:     this.horizontalParameters = xmlElement.getAttribute('vertical_parameters') !== "
true";
312:     this.updateParams_(params);
313:   },
314:   decompose: function(workspace) {
315:     var containerBlock = new Blockly.Block.obtain(workspace, 'procedures_mutatorcont
ainer');
316:     containerBlock.initSvg();
317:     // [lyn, 11/24/12] Remember the associated procedure, so can
318:     // appropriately change body when update name in param block.
319:     containerBlock.setProcBlock(this);
320:     this.paramIds_ = [] // [lyn, 10/26/13] Added
321:     var connection = containerBlock.getInput('STACK').connection;
322:     for (var x = 0; x < this.arguments_.length; x++) {
323:       var paramBlock = new Blockly.Block.obtain(workspace, 'procedures_mutatorarg');
324:       this.paramIds_.push(paramBlock.id); // [lyn, 10/26/13] Added
325:       paramBlock.initSvg();
326:       paramBlock.setFieldValue(this.arguments_[x], 'NAME');
327:       // Store the old location.
328:       paramBlock.oldLocation = x;
329:       connection.connect(paramBlock.previousConnection);
330:       connection = paramBlock.nextConnection;
331:     }
332:     // [lyn, 10/26/13] Rather than passing null for paramIds, pass actual paramIds
333:     // and use true flag to initialize tracking.
334:     Blockly.Procedures.mutateCallers(this.getFieldValue('NAME'),
335:                                      this.workspace, this.arguments_, this.paramIds_
, true);
336:     return containerBlock;
337:   },
338:   compose: function(containerBlock) {
339:     var params = [];
340:     this.paramIds_ = [];
341:     var paramBlock = containerBlock.getInputTargetBlock('STACK');
342:     while (paramBlock) {
343:       params.push(paramBlock.getFieldValue('NAME'));
344:       this.paramIds_.push(paramBlock.id);
345:       paramBlock = paramBlock.nextConnection &&
346:           paramBlock.nextConnection.targetBlock();
347:     }
348:     // console.log("enter procedures_defnoreturn compose(); prevArguments = "
349:     //     + prevArguments.join(',')
350:     //     + "; currentAguments = "
351:     //     + this.arguments_.join(',')
352:     //     + ";"
353:     // );
354:     // [lyn, 11/24/12] Note: update params updates param list in proc declaration,
355:     // but renameParam updates procedure body appropriately.
356:     if (!Blockly.LexicalVariable.stringListsEqual(params, this.arguments_)) { // Onl
y need updates if param list has changed
357:       this.updateParams_(params);
358:       Blockly.Procedures.mutateCallers(this.getFieldValue('NAME'),
359:           this.workspace, this.arguments_, this.paramIds_);
360:     }
361:     // console.log("exit procedures_defnoreturn compose()");
362:   },
363:   dispose: function() {
364:     var name = this.getFieldValue('NAME');
365:     var editable = this.editable_;
366:     var workspace = this.workspace;
367:
368:     // Call parent's destructor.
369:     Blockly.Block.prototype.dispose.apply(this, arguments);
370:
371:     if (editable) {
372:       // Dispose of any callers.
373:       //Blockly.Procedures.disposeCallers(name, workspace);
374:       Blockly.AIProcedure.removeProcedureValues(name, workspace);
375:     }
376:
377:   },
378:   getProcedureDef: function() {
379:     // Return the name of the defined procedure,
380:     // a list of all its arguments,
381:     // and that it DOES NOT have a return value.
382:     return [this.getFieldValue('NAME'),
383:             this.arguments_,
384:            this.bodyInputName === 'RETURN']; // true for procedures that return valu
es.
385:   },
386:   getVars: function() {
387:     var names = []
388:     for (var i = 0, param; param = this.getFieldValue('VAR' + i); i++) {
389:       names.push(param);
390:     }
391:     return names;
392:   },
393:   declaredNames: function() { // [lyn, 10/11/13] return the names of all parameters
of this procedure
394:     return this.getVars();
395:   },
396:   renameVar: function(oldName, newName) {
397:     this.renameVars(Blockly.Substitution.simpleSubstitution(oldName,newName));
398:   },
399:   renameVars: function(substitution) { // renaming is a dict (i.e., object) mapping
old names to new ones
400:     var oldParams = this.getParameters();
401:     var newParams = substitution.map(oldParams);
402:     if (!Blockly.LexicalVariable.stringListsEqual(oldParams, newParams)) {
403:       this.updateParams_(newParams);
404:       // Update the mutator's variables if the mutator is open.
405:       if (this.mutator.isVisible()) {
406:         var blocks = this.mutator.workspace_.getAllBlocks();
407:         for (var x = 0, block; block = blocks[x]; x++) {
408:           if (block.type == 'procedures_mutatorarg') {
409:             var oldName = block.getFieldValue('NAME');
410:             var newName = substitution.apply(oldName);
411:             if (newName !== oldName) {
412:               block.setFieldValue(newName, 'NAME');
413:             }
414:           }
415:         }
416:       }
417:     }
```

```
  418:   },
  419:   renameBound: function (boundSubstitution, freeSubstitution) {
  420:     var paramSubstitution = boundSubstitution.restrictDomain(this.declaredNames());
  421:     this.renameVars(paramSubstitution);
  422:     var newFreeSubstitution = freeSubstitution.extend(paramSubstitution);
  423:     Blockly.LexicalVariable.renameFree(this.getInputTargetBlock(this.bodyInputName),
 newFreeSubstitution);
  424:   },
  425:   renameFree: function (freeSubstitution) { // Should have no effect since only top-
level procedures.
  426:     var freeVars = this.freeVariables(); // Calculate free variables, which should b
e empty,
  427:                                          // throwing exception if not.
  428:     // There should be no free variables, and so nothing to rename. Do nothing else.
  429:   },
  430:   freeVariables: function() { // return the free lexical variables of this block
  431:                               // Should return the empty set: something is wrong if
it doesn't!
  432:     var result = Blockly.LexicalVariable.freeVariables(this.getInputTargetBlock(this
.bodyInputName));
  433:     result.subtract(new Blockly.NameSet(this.declaredNames()));
  434:     if (result.isEmpty()) {
  435:       return result;
  436:     } else {
  437:       throw "Violation of invariant: procedure declaration has nonempty free variabl
es: " + result.toString();
  438:     }
  439:   },
  440:   // [lyn, 11/24/12] return list of procedure body (if there is one)
  441:   blocksInScope: function () {
  442:     var body = this.getInputTargetBlock(this.bodyInputName);
  443:     return (body && [body]) || [];
  444:   },
  445:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_PROCEDURE +
  446:       ' ' + Blockly.Msg.LANG_PROCEDURES_DEFNORETURN_DO }],
  447:   customContextMenu: function (options) {
  448:     Blockly.FieldParameterFlydown.addHorizontalVerticalOption(this, options);
  449:   },
  450:   getParameters: function() {
  451:     return this.arguments_;
  452:   }
  453: };
  454:
  455: // [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN
)
  456: Blockly.Blocks['procedures_defreturn'] = {
  457:   // Define a procedure with a return value.
  458:   category: 'Procedures',  // Procedures are handled specially.
  459:   helpUrl: Blockly.Msg.LANG_PROCEDURES_DEFRETURN_HELPURL,
  460:   bodyInputName: 'RETURN',
  461:   init: function() {
  462:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  463:     var name = Blockly.Procedures.findLegalName(
  464:         Blockly.Msg.LANG_PROCEDURES_DEFRETURN_PROCEDURE, this);
  465:     this.appendDummyInput('HEADER')
  466:         .appendField(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_DEFINE)
  467:       // [lyn, 10/27/13] Replaced Blockly.Procedures.rename by Blockly.AIProcedure.r
enameProcedure
  468:         .appendField(new Blockly.FieldTextInput(name, Blockly.AIProcedure.renameProc
edure), 'NAME');
  469:     this.horizontalParameters = true; // horizontal by default
  470:     this.appendIndentedValueInput('RETURN')
  471:         .appendField(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_RETURN);
  472:     this.setMutator(new Blockly.Mutator(['procedures_mutatorarg']));
  473:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DEFRETURN_TOOLTIP);
  474:     this.arguments_ = [];
  475:     this.warnings = [{name:"checkEmptySockets",sockets:["RETURN"]}];
  476:   },
  477:   onchange: Blockly.Blocks.procedures_defnoreturn.onchange,
  478:   // [lyn, 11/24/12] return list of procedure body (if there is one)
  479:   updateParams_: Blockly.Blocks.procedures_defnoreturn.updateParams_,
  480:   parameterFlydown: Blockly.Blocks.procedures_defnoreturn.parameterFlydown,
  481:   setParameterOrientation: Blockly.Blocks.procedures_defnoreturn.setParameterOrienta
tion,
  482:   mutationToDom: Blockly.Blocks.procedures_defnoreturn.mutationToDom,
  483:   domToMutation: Blockly.Blocks.procedures_defnoreturn.domToMutation,
  484:   decompose: Blockly.Blocks.procedures_defnoreturn.decompose,
  485:   compose: Blockly.Blocks.procedures_defnoreturn.compose,
  486:   dispose: Blockly.Blocks.procedures_defnoreturn.dispose,
  487:   getProcedureDef: Blockly.Blocks.getProcedureDef,
  488:   getVars: Blockly.Blocks.procedures_defnoreturn.getVars,
  489:   declaredNames: Blockly.Blocks.procedures_defnoreturn.declaredNames,
  490:   renameVar: Blockly.Blocks.procedures_defnoreturn.renameVar,
  491:   renameVars: Blockly.Blocks.procedures_defnoreturn.renameVars,
  492:   renameBound: Blockly.Blocks.procedures_defnoreturn.renameBound,
  493:   renameFree: Blockly.Blocks.procedures_defnoreturn.renameFree,
  494:   freeVariables: Blockly.Blocks.procedures_defnoreturn.freeVariables,
  495:   blocksInScope: Blockly.Blocks.procedures_defnoreturn.blocksInScope,
  496:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_DEFRETURN_PROCEDURE +
  497:       ' ' + Blockly.Msg.LANG_PROCEDURES_DEFRETURN_RETURN }],
  498:   customContextMenu: Blockly.Blocks.procedures_defnoreturn.customContextMenu,
  499:   getParameters: Blockly.Blocks.procedures_defnoreturn.getParameters
  500: };
  501:
  502: Blockly.Blocks['procedures_mutatorcontainer'] = {
  503:   // Procedure container (for mutator dialog).
  504:   init: function() {
  505:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
  506:     this.appendDummyInput()
  507:         .appendField(Blockly.Msg.LANG_PROCEDURES_MUTATORCONTAINER_TITLE);
  508:     this.appendStatementInput('STACK');
  509:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_MUTATORCONTAINER_TOOLTIP);
  510:     this.contextMenu = false;
  511:   },
  512:   // [lyn. 11/24/12] Set procBlock associated with this container.
  513:   setProcBlock: function (procBlock) {
  514:     this.procBlock_ = procBlock;
  515:   },
  516:   // [lyn. 11/24/12] Set procBlock associated with this container.
  517:   // Invariant: should not be null, since only created as mutator for a particular p
roc block.
  518:   getProcBlock: function () {
  519:     return this.procBlock_;
  520:   },
  521:   // [lyn. 11/24/12] Return list of param names in this container
  522:   // Invariant: there should be no duplicates!
  523:   declaredNames: function () {
  524:     var paramNames = [];
  525:     var paramBlock = this.getInputTargetBlock('STACK');
  526:     while (paramBlock) {
  527:       paramNames.push(paramBlock.getFieldValue('NAME'));
  528:       paramBlock = paramBlock.nextConnection &&
  529:                    paramBlock.nextConnection.targetBlock();
  530:     }
```

```
531:     return paramNames;
532:   }
533: };
534:
535: Blockly.Blocks['procedures_mutatorarg'] = {
536:   // Procedure argument (for mutator dialog).
537:   init: function() {
538: //    var mutatorarg = this;
539: //    var mutatorargChangeHandler = function(newName) {
540: //      var proc = mutatorarg.getProcBlock();
541: //      var procArguments = proc ? proc.arguments_ : [];
542: //      console.log("mutatorargChangeHandler: newName = " + newName
543: //                  + " and proc argumnets = [" + procArguments.join(',') + "]");
544: //      return Blockly.LexicalVariable.renameParam.call(this,newName);
545: //    }
546:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
547:     this.appendDummyInput()
548:         .appendField(Blockly.Msg.LANG_PROCEDURES_MUTATORARG_TITLE)
549:         .appendField(new Blockly.FieldTextInput('x',Blockly.LexicalVariable.renamePa
ram), 'NAME');
550:     this.setPreviousStatement(true);
551:     this.setNextStatement(true);
552:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_MUTATORARG_TOOLTIP);
553:     this.contextMenu = false;
554:   },
555:   // [lyn, 11/24/12] Return the container this mutator arg is in, or null if it's no
t in one.
556:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
557:   // be in container stack.
558:   getContainerBlock: function () {
559:     var parent = this.getParent();
560:     while (parent && ! (parent.type === "procedures_mutatorcontainer")) {
561:       parent = parent.getParent();
562:     }
563:     // [lyn, 11/24/12] Cache most recent container block so can reference it upon re
moval from mutator arg stack
564:     this.cachedContainerBlock_ = (parent && (parent.type === "procedures_mutatorcont
ainer") && parent) || null;
565:     return this.cachedContainerBlock_;
566:   },
567:   // [lyn, 11/24/12] Return the procedure associated with mutator arg is in, or null
 if there isn't one.
568:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
569:   // be in container stack.
570:   getProcBlock: function () {
571:     var container = this.getContainerBlock();
572:     return (container && container.getProcBlock()) || null;
573:   },
574:   // [lyn, 11/24/12] Return the declared names in the procedure associated with muta
tor arg,
575:   // or the empty list if there isn't one.
576:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
577:   // be in container stack.
578:   declaredNames: function () {
579:     var container = this.getContainerBlock();
580:     return (container && container.declaredNames()) || [];
581:   },
582:   // [lyn, 11/24/12] Return the blocks in scope of proc params in the the procedure
associated with mutator arg,
583:   // or the empty list if there isn't one.
584:   // Dynamically calculate this by walking up chain, because mutator arg might or mi
ght not
585:   // be in container stack.
586:   blocksInScope: function () {
587:     var proc = this.getProcBlock();
588:     return (proc && proc.blocksInScope()) || [];
589:   },
590:   // [lyn, 11/24/12] Check for situation in which mutator arg has been removed from
stack,
591:   // and change all references to its name to ???.
592:   onchange: function() {
593:     var paramName = this.getFieldValue('NAME');
594:     if (paramName) { // paramName is null when delete from stack
595:       // console.log("Mutatorarg onchange: " + paramName);
596:       var cachedContainer = this.cachedContainerBlock_;
597:       var container = this.getContainerBlock(); // Order is important; this must com
e after cachedContainer
598:                                                 // since it sets cachedContainerBloc
k_
599:       // console.log("Mutatorarg onchange: " + paramName
600:       //             + "; cachedContainer = " + JSON.stringify((cachedContainer && ca
chedContainer.type) || null)
601:       //             + "; container = " + JSON.stringify((container && container.type
) || null));
602:       if ((! cachedContainer) && container) {
603:         // Event: added mutator arg to container stack
604:         // console.log("Mutatorarg onchange ADDED: " + paramName);
605:         var declaredNames = this.declaredNames();
606:         var firstIndex = declaredNames.indexOf(paramName);
607:         if (firstIndex != -1) {
608:           // Assertion: we should get here, since paramName should be among names
609:           var secondIndex = declaredNames.indexOf(paramName, firstIndex+1);
610:           if (secondIndex != -1) {
611:             // If we get here, there is a duplicate on insertion that must be resolv
ed
612:             var newName = Blockly.FieldLexicalVariable.nameNotIn(paramName,declaredN
ames);
613:             this.setFieldValue(newName, 'NAME');
614:           }
615:         }
616:       } /* else if (cachedContainer && (! container)) {
617:         // Event: removed mutator arg from container stack
618:         // [lyn, 11/24/12] Mutator arg has been removed from stack. Change all refer
ences to its name to ???
619:         // console.log("Mutatorarg onchange REMOVED: " + paramName);
620:         var proc = cachedContainer.getProcBlock();
621:         var inScopeBlocks = (proc && proc.blocksInScope()) || [];
622:         var referenceResults = inScopeBlocks.map( function(blk) { return Blockly.Lex
icalVariable.referenceResult(blk, paramName, []); } );
623:         var blocksToRename = [];
624:         for (var r = 0; r < referenceResults.length; r++) {
625:           blocksToRename = blocksToRename.concat(referenceResults[r][0]);
626:           // ignore capturables, which are not relevant here.
627:         }
628:         // Rename getters and setters
629:         for (var i = 0; i < blocksToRename.length; i++) {
630:           var block = blocksToRename[i];
631:           var renamingFunction = block.renameLexicalVar;
632:           if (renamingFunction) {
633:             renamingFunction.call(block, "param " + paramName, "???");
634:           }
```

```
635:         }
636:       }*/
637:     }
638:   }
639: };
640:
641: Blockly.Blocks.procedures_mutatorarg.validator = function(newVar) {
642:   // Merge runs of whitespace.  Strip leading and trailing whitespace.
643:   // Beyond this, all names are legal.
644:   newVar = newVar.replace(/[\s\xa0]+/g, ' ').replace(/^ | $/g, '');
645:   return newVar || null;
646: };
647:
648: /* [lyn 10/10/13] With parameter flydown changes,
649:  * I don't think a special GET block in the Procedure drawer is necesssary
650: Blockly.Blocks.procedure_lexical_variable_get = {
651:   // Variable getter.
652:   category: 'Procedures',
653:   helpUrl: Blockly.Msg.LANG_PROCEDURES_GET_HELPURL, // *** [lyn, 11/10/12] Fix this
654:   init: function() {
655:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
656:     this.fieldVar_ = new Blockly.FieldLexicalVariable(" ");
657:     this.fieldVar_.setBlock(this);
658:     this.appendDummyInput()
659:         .appendField("get")
660:         .appendField(this.fieldVar_, 'VAR');
661:     this.setOutput(true, null);
662:     this.setTooltip(Blockly.Msg.LANG_VARIABLES_GET_TOOLTIP);
663:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["VAR"]}];
664:   },
665:   getVars: function() {
666:     return [this.getFieldValue('VAR')];
667:   },
668:   onchange: function() {
669:      // [lyn, 11/10/12] Checks if parent has changed. If so, checks if curent variab
le name
670:      //    is still in scope. If so, keeps it as is; if not, changes to ???
671:      //    *** NEED TO MAKE THIS BEHAVIOR BETTER!
672:     if (this.fieldVar_) {
673:        var currentName = this.fieldVar_.getText();
674:        var nameList = this.fieldVar_.getNamesInScope();
675:        var cachedParent = this.fieldVar_.getCachedParent();
676:        var currentParent = this.fieldVar_.getBlock().getParent();
677:        // [lyn, 11/10/12] Allow current name to stay if block moved to workspace in
"untethered" way.
678:        //   Only changed to ??? if tether an untethered block.
679:        if (currentParent != cachedParent) {
680:          this.fieldVar_.setCachedParent(currentParent);
681:          if  (currentParent !== null) {
682:            for (var i = 0; i < nameList.length; i++ ) {
683:              if (nameList[i] === currentName) {
684:                return; // no change
685:              }
686:            }
687:            // Only get here if name not in list
688:            this.fieldVar_.setText(" ");
689:          }
690:        }
691:     }
692:     Blockly.WarningHandler.checkErrors.call(this);
693:   },
694:   renameLexicalVar: function(oldName, newName) {
695:     // console.log("Renaming lexical variable from " + oldName + " to " + newName);
696:     if (oldName === this.getFieldValue('VAR')) {
697:         this.setFieldValue(newName, 'VAR');
698:     }
699:   }
700: };
701: */
702:
703: /* // [lyn, 10/14/13] This will become unnecessary with Michael Phox's
704:  * mutator on procedure_defreturn that allows adding a DO statement.
705:  */
706: /*
707: Blockly.Blocks.procedures_do_then_return = {
708:   // String length.
709:   category: 'Procedures',
710:   helpUrl: Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_HELPURL,
711:   init: function() {
712:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
713:     this.appendStatementInput('STM')
714:         .appendField(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_DO);
715:     this.appendValueInput('VALUE')
716:         .appendField(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_RETURN)
717:         .setAlign(Blockly.ALIGN_RIGHT);
718:     this.setOutput(true, null);
719:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_DOTHENRETURN_TOOLTIP);
720:   },
721:   onchange: Blockly.WarningHandler.checkErrors
722: };
723: */
724:
725: Blockly.Blocks['procedures_callnoreturn'] = {
726:   // Call a procedure with no return value.
727:   category: 'Procedures',  // Procedures are handled specially.
728:   helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_HELPURL,
729:   init: function() {
730:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
731:     this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(fals
e);};
732:
733:     this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldPro
cedure.onChange);
734:     this.procDropDown.block = this;
735:     this.appendDummyInput()
736:         .appendField(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_CALL)
737:         .appendField(this.procDropDown,"PROCNAME");
738:     this.setPreviousStatement(true);
739:     this.setNextStatement(true);
740:     this.appendValueInput("wait_time")
741:         .setCheck("Number")
742:         .setAlign(Blockly.ALIGN_RIGHT)
743:         .appendField("wait_time");
744:     this.appendValueInput("wait_trials")
745:         .setCheck("Number")
746:         .setAlign(Blockly.ALIGN_RIGHT)
747:         .appendField("wait_trials");
748:     this.appendValueInput("exec_time")
749:         .setCheck("Number")
750:         .setAlign(Blockly.ALIGN_RIGHT)
751:         .appendField("exec_time");
752:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_TOOLTIP);
753:     this.arguments_ = [];
```

```
754:     this.quarkConnections_ = null;
755:     this.quarkArguments_ = null;
756:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["PROCNAME"]}];
757:     //Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.procNames
Fxn(false)[0][0]);
758:     Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.getField_("
PROCNAME").getValue());
759:   },
760:   getProcedureCall: function() {
761:     return this.getFieldValue('PROCNAME');
762:   },
763:   renameProcedure: function(oldName, newName) {
764:     if (Blockly.Names.equals(oldName, this.getFieldValue('PROCNAME'))) {
765:       this.setFieldValue(newName, 'PROCNAME');
766:     }
767:   },
768:   // [lyn, 10/27/13] Renamed "fromChange" parameter to "startTracking", because it s
hould be true in any situation
769:   // where we want caller to start tracking connections associated with paramIds. Th
is includes when a mutator
770:   // is opened on a procedure declaration.
771:   setProcedureParameters: function(paramNames, paramIds, startTracking) {
772:     // Data structures for parameters on each call block:
773:     // this.arguments = ['x', 'y']
774:     //     Existing param names.
775:     // paramNames = ['x', 'y', 'z']
776:     //     New param names.
777:     // paramIds = ['piua', 'f8b_', 'oi.o']
778:     //     IDs of params (consistent for each parameter through the life of a
779:     //     mutator, regardless of param renaming).
780:     // this.quarkConnections_ {piua: null, f8b_: Blockly.Connection}
781:     //     Look-up of paramIds to connections plugged into the call block.
782:     // this.quarkArguments_ = ['piua', 'f8b_']
783:     //     Existing param IDs.
784:     // Note that quarkConnections_ may include IDs that no longer exist, but
785:     // which might reappear if a param is reattached in the mutator.
786:
787:     var input;
788:     var connection;
789:     var x;
790:
791:     //fixed parameter alignment see ticket 465
792:     if (!paramIds) {
793:       // Reset the quarks (a mutator is about to open).
794:       this.quarkConnections_ = {};
795:       this.quarkArguments_ = null;
796:       // return;  // [lyn, 10/27/13] No, don't return yet. We still want to add para
mNames to block!
797:       // For now, create dummy list of param ids. This needs to be cleaned up furthe
r!
798:       paramIds = [].concat(paramNames); // create a dummy list that's a copy of para
mNames.
799:     }
800:     if (paramIds.length != paramNames.length) {
801:       throw 'Error: paramNames and paramIds must be the same length.';
802:     }
803:     var paramIdToParamName = {};
804:     for(var i=0;i<paramNames.length;i++) {
805:       paramIdToParamName[paramIds[i]] = paramNames[i];
806:     }
807:     if(typeof startTracking == "undefined") {
808:       startTracking = null;
809:     }
810:
811:     if (!this.quarkArguments_ || startTracking) {
812:       // Initialize tracking for this block.
813:       this.quarkConnections_ = {};
814:       if (Blockly.LexicalVariable.stringListsEqual(paramNames, this.arguments_) || s
tartTracking) {
815:         // No change to the parameters, allow quarkConnections_ to be
816:         // populated with the existing connections.
817:         this.quarkArguments_ = paramIds;
818:       } else {
819:         this.quarkArguments_ = [];
820:       }
821:     }
822:     // Switch off rendering while the block is rebuilt.
823:     var savedRendered = this.rendered;
824:     this.rendered = false;
825:     // Update the quarkConnections_ with existing connections.
826:     for (x = 0;this.getInput('ARG' + x); x++) {
827:       input = this.getInput('ARG' + x);
828:       if (input) {
829:         connection = input.connection.targetConnection;
830:         this.quarkConnections_[this.quarkArguments_[x]] = connection;
831:         // Disconnect all argument blocks and remove all inputs.
832:         this.removeInput('ARG' + x);
833:       }
834:     }
835:     // Rebuild the block's arguments.
836:     this.arguments_ = [].concat(paramNames);
837:     this.quarkArguments_ = paramIds;
838:     for (x = 0; x < this.arguments_.length; x++) {
839:       input = this.appendValueInput('ARG' + x)
840:           .setAlign(Blockly.ALIGN_RIGHT)
841:           .appendField(this.arguments_[x]);
842:       if (this.quarkArguments_) {
843:         // Reconnect any child blocks.
844:         var quarkName = this.quarkArguments_[x];
845:         if (quarkName in this.quarkConnections_) {
846:           connection = this.quarkConnections_[quarkName];
847:           if (!connection || connection.targetConnection ||
848:               connection.sourceBlock_.workspace != this.workspace) {
849:             // Block no longer exists or has been attached elsewhere.
850:             delete this.quarkConnections_[quarkName];
851:           } else {
852:             input.connection.connect(connection);
853:           }
854:         } else if(paramIdToParamName[quarkName]){
855:           connection = this.quarkConnections_[paramIdToParamName[quarkName]];
856:           if (connection){
857:             input.connection.connect(connection);
858:           }
859:         }
860:       }
861:     }
862:     // Restore rendering and show the changes.
863:     this.rendered = savedRendered;
864:     if (this.rendered) {
865:       this.render();
866:     }
867:   },
868:   mutationToDom: function() {
```

```
869:     // Save the name and arguments (none of which are editable).
870:     var container = document.createElement('mutation');
871:     container.setAttribute('name', this.getFieldValue('PROCNAME'));
872:     for (var x = 0; this.getInput("ARG" + x); x++) {
873:       var parameter = document.createElement('arg');
874:       parameter.setAttribute('name', this.getInput("ARG" + x).fieldRow[0].text_);
875:       container.appendChild(parameter);
876:     }
877:     return container;
878:   },
879:   domToMutation: function(xmlElement) {
880:     // Restore the name and parameters.
881:     var name = xmlElement.getAttribute('name');
882:     this.setFieldValue(name, 'PROCNAME');
883:     // [lyn, 10/27/13] Significantly cleaned up this code. Always take arg names fro
m xmlElement.
884:     // Do not attempt to find definition.
885:     this.arguments_ = [];
886:     var children = goog.dom.getChildren(xmlElement);
887:     for (var x = 0, childNode; childNode = children[x]; x++) {
888:       if (childNode.nodeName.toLowerCase() == 'arg') {
889:         this.arguments_.push(childNode.getAttribute('name'));
890:       }
891:     }
892:     this.setProcedureParameters(this.arguments_, null, true);
893:       // [lyn, 10/27/13] Above. set tracking to true in case this is a block with ar
gument subblocks.
894:       // and there's an open mutator.
895:   },
896:   renameVar: function(oldName, newName) {
897:     for (var x = 0; x < this.arguments_.length; x++) {
898:       if (Blockly.Names.equals(oldName, this.arguments_[x])) {
899:         this.arguments_[x] = newName;
900:         this.getInput('ARG' + x).fieldRow[0].setText(newName);
901:       }
902:     }
903:   },
904:   procCustomContextMenu: function(options) {
905:     // Add option to find caller.
906:     var option = {enabled: true};
907:     option.text = Blockly.Msg.LANG_PROCEDURES_HIGHLIGHT_DEF;
908:     var name = this.getFieldValue('PROCNAME');
909:     var workspace = this.workspace;
910:     option.callback = function() {
911:       var def = Blockly.Procedures.getDefinition(name, workspace);
912:       def && def.select();
913:     };
914:     options.push(option);
915:   },
916:   removeProcedureValue: function() {
917:     this.setFieldValue("none", 'PROCNAME');
918:     for(var i=0;this.getInput('ARG' + i) !== null;i++) {
919:       this.removeInput('ARG' + i);
920:     }
921:   },
922:   // This generates a single generic call to 'call no return' defaulting its value
923:   // to the first procedure in the list. Calls for each procedure cannot be done her
e because the
924:   // blocks have not been loaded yet (they are loaded in typeblock.js)
925:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_CALLNORETURN_TRANSLATED_
NAME}]
926: };
927:
928:
929: Blockly.Blocks['procedures_callreturn'] = {
930:   // Call a procedure with a return value.
931:   category: 'Procedures',  // Procedures are handled specially.
932:   helpUrl: Blockly.Msg.LANG_PROCEDURES_CALLRETURN_HELPURL,
933:   init: function() {
934:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
935:     this.procNamesFxn = function(){return Blockly.AIProcedure.getProcedureNames(true
);};
936:
937:     this.procDropDown = new Blockly.FieldDropdown(this.procNamesFxn,Blockly.FieldPro
cedure.onChange);
938:     this.procDropDown.block = this;
939:     this.appendDummyInput()
940:         .appendField(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_CALL)
941:         .appendField(this.procDropDown,"PROCNAME");
942:     this.appendValueInput("wait_time")
943:         .setCheck("Number")
944:         .setAlign(Blockly.ALIGN_RIGHT)
945:         .appendField("wait_time");
946:     this.appendValueInput("wait_trials")
947:         .setCheck("Number")
948:         .setAlign(Blockly.ALIGN_RIGHT)
949:         .appendField("wait_trials");
950:     this.appendValueInput("exec_time")
951:         .setCheck("Number")
952:         .setAlign(Blockly.ALIGN_RIGHT)
953:         .appendField("exec_time");
954:     this.setOutput(true, null);
955:     this.setTooltip(Blockly.Msg.LANG_PROCEDURES_CALLRETURN_TOOLTIP);
956:     this.arguments_ = [];
957:     this.quarkConnections_ = null;
958:     this.quarkArguments_ = null;
959:     this.errors = [{name:"checkIsInDefinition"},{name:"checkDropDownContainsValidVal
ue",dropDowns:["PROCNAME"]}];
960:     //Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.procNames
Fxn()[0][0]);
961:     Blockly.FieldProcedure.onChange.call(this.getField_("PROCNAME"),this.getField_("
PROCNAME").getValue());
962:   },
963:   getProcedureCall: Blockly.Blocks.procedures_callnoreturn.getProcedureCall,
964:   renameProcedure: Blockly.Blocks.procedures_callnoreturn.renameProcedure,
965:   setProcedureParameters:
966:       Blockly.Blocks.procedures_callnoreturn.setProcedureParameters,
967:   mutationToDom: Blockly.Blocks.procedures_callnoreturn.mutationToDom,
968:   domToMutation: Blockly.Blocks.procedures_callnoreturn.domToMutation,
969:   renameVar: Blockly.Blocks.procedures_callnoreturn.renameVar,
970:   procCustomContextMenu: Blockly.Blocks.procedures_callnoreturn.procCustomContextMen
u,
971:   removeProcedureValue: Blockly.Blocks.procedures_callnoreturn.removeProcedureValue,
972:   // This generates a single generic call to 'call return' defaulting its value
973:   // to the first procedure in the list. Calls for each procedure cannot be done her
e because the
974:   // blocks have not been loaded yet (they are loaded in typeblock.js)
975:   typeblock: [{ translatedName: Blockly.Msg.LANG_PROCEDURES_CALLRETURN_TRANSLATED_NA
ME}]
976: };
977:
978: Blockly.Blocks['nil_proc'] = {
979:   category: 'Procedures',
980:   init: function() {
```

```
981:     this.setColour(Blockly.PROCEDURE_CATEGORY_HUE);
982:     this.setHelpUrl('http://www.example.com/');
983:     this.appendDummyInput()
984:         .appendField("nil");
985:     this.setPreviousStatement(true);
986:     this.setNextStatement(true);
987:     this.setTooltip('');
988:   }
989: };
990:
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2013-2014 MIT, All rights reserved
    3: // Released under the Apache License, Version 2.0
    4: // http://www.apache.org/licenses/LICENSE-2.0
    5: /**
    6:  * @license
    7:  * @fileoverview Color blocks for Blockly, modified for MIT App Inventor.
    8:  * @author mckinney@mit.edu (Andrew F. McKinney)
    9:  */
   10:
   11: 'use strict';
   12:
   13: goog.provide('Blockly.Blocks.qulog');
   14:
   15: goog.require('Blockly.Blocks.Utilities');
   16:
   17: Blockly.Blocks['remember_call'] = {
   18:   category: 'Qulog',
   19:   init: function() {
   20:     this.setColour(Blockly.QULOG_CATEGORY_HUE);
   21:     this.appendDummyInput()
   22:         .appendField("call")
   23:         .appendField(new Blockly.FieldDropdown([["remember", "REMEMBER"], ["forget",
 "FORGET"]]), "DD");
   24:     this.appendValueInput("belief")
   25:         .setCheck("String")
   26:         .setAlign(Blockly.ALIGN_RIGHT)
   27:         .appendField("belief");
   28:     this.appendValueInput("time")
   29:         .setCheck("Number")
   30:         .setAlign(Blockly.ALIGN_RIGHT)
   31:         .appendField("time");
   32:     this.setPreviousStatement(true);
   33:     this.setNextStatement(true);
   34:     this.setInputsInline(false);
   35:     this.setTooltip('');
   36:   }
   37: };
   38:
   39: Blockly.Blocks['forget_call'] = {
   40:   category: 'Qulog',
   41:   init: function() {
   42:     this.setColour(Blockly.QULOG_CATEGORY_HUE);
   43:     this.appendDummyInput()
   44:         .appendField("call")
   45:         .appendField(new Blockly.FieldDropdown([["forget", "FORGET"], ["remember", "
REMEMBER"]]), "DD");
   46:     this.appendValueInput("belief")
   47:         .setCheck("String")
   48:         .setAlign(Blockly.ALIGN_RIGHT)
   49:         .appendField("belief");
   50:     this.appendValueInput("time")
   51:         .setCheck("Number")
   52:         .setAlign(Blockly.ALIGN_RIGHT)
   53:         .appendField("time");
   54:     this.setPreviousStatement(true);
   55:     this.setNextStatement(true);
   56:     this.setTooltip('');
   57:   }
   58: };
```

```
    1: /**
    2:  * Visual Blocks Language
    3:  *
    4:  * Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    5:  *
    6:  * Licensed under the Apache License, Version 2.0 (the "License");
    7:  * you may not use this file except in compliance with the License.
    8:  * You may obtain a copy of the License at
    9:  *
   10:  *   http://www.apache.org/licenses/LICENSE-2.0
   11:  *
   12:  * Unless required by applicable law or agreed to in writing, software
   13:  * distributed under the License is distributed on an "AS IS" BASIS,
   14:  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
   15:  * See the License for the specific language governing permissions and
   16:  * limitations under the License.
   17:  */
   18:
   19: /**
   20:  * @fileoverview List generators for Blockly, modified for App Inventor
   21:  * @author fraser@google.com (Neil Fraser)
   22:  * @author andrew.f.mckinney@gmail.com (Andrew F. McKinney)
   23:  * Due to the frequency of long strings, the 80-column wrap rule need not apply
   24:  * to language files.
   25:  */
   26:
   27: /**
   28:  * Lyn's History:
   29:  * [lyn, 10/27/13] Modified for loop index variables to begin with YAIL_LOCAL_VAR_TA
G (currently '$').
   30:  *     At least on Kawa-legal first character is necessary to ensure AI identifiers
   31:  *     satisfy Kawa's identifier rules.
   32:  * [lyn, 01/15/2013] Added do_then_return, eval_but_ignore, and nothing.
   33:  * [lyn, 12/27/2012] Made code generation of forRange and forEach consistent with pa
rameter change.
   34:  */
   35:
   36: 'use strict';
   37:
   38: goog.provide('Blockly.Yail.control');
   39:
   40: Blockly.Yail['controls_if'] = function() {
   41:
   42:   var code = "";
   43:   for(var i=0;i<this.elseifCount_ + 1;i++){
   44:     var argument = Blockly.Yail.valueToCode(this, 'IF'+ i, Blockly.Yail.ORDER_NONE)
|| Blockly.Yail.YAIL_FALSE;
   45:     var branch = Blockly.Yail.statementToCode(this, 'DO'+ i) || Blockly.Yail.YAIL_FA
LSE;
   46:     if(i != 0) {
   47:       code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN;
   48:     }
   49:     code += Blockly.Yail.YAIL_IF + argument + Blockly.Yail.YAIL_SPACER + Blockly.Yai
l.YAIL_BEGIN
   50:       + branch + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   51:   }
   52:   if(this.elseCount_ == 1){
   53:     var branch = Blockly.Yail.statementToCode(this, 'ELSE') || Blockly.Yail.YAIL_FAL
SE;
   54:     code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_BEGIN + branch + Blockly.Ya
il.YAIL_CLOSE_COMBINATION;
   55:   }
   56:
   57:   for(var i=0;i<this.elseifCount_;i++){
   58:     code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_CLOSE_COMBINATIO
N;
   59:   }
   60:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   61:   return code;
   62: };
   63:
   64: // [lyn, 12/27/2012]
   65: Blockly.Yail['controls_forRange'] = function() {
   66:   // For range loop.
   67:   var loopIndexName = Blockly.Yail.YAIL_LOCAL_VAR_TAG + this.getFieldValue('VAR');
   68:   var startCode = Blockly.Yail.valueToCode(this, 'START', Blockly.Yail.ORDER_NONE) |
| 0;
   69:   var endCode = Blockly.Yail.valueToCode(this, 'END', Blockly.Yail.ORDER_NONE) || 0;
   70:   var stepCode = Blockly.Yail.valueToCode(this, 'STEP', Blockly.Yail.ORDER_NONE) ||
0;
   71:   var bodyCode = Blockly.Yail.statementToCode(this, 'DO', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.YAIL_FALSE;
   72:   return Blockly.Yail.YAIL_FORRANGE + loopIndexName + Blockly.Yail.YAIL_SPACER
   73:          + Blockly.Yail.YAIL_BEGIN + bodyCode + Blockly.Yail.YAIL_CLOSE_COMBINATION
+ Blockly.Yail.YAIL_SPACER
   74:          + startCode + Blockly.Yail.YAIL_SPACER
   75:          + endCode + Blockly.Yail.YAIL_SPACER
   76:          + stepCode + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   77: };
   78:
   79: Blockly.Yail['controls_while'] = function() {
   80:   // While condition.
   81:   var test = Blockly.Yail.valueToCode(this, 'TEST', Blockly.Yail.ORDER_NONE) || Bloc
kly.Yail.YAIL_FALSE;
   82:   var toDo = Blockly.Yail.statementToCode(this, 'DO') || Blockly.Yail.YAIL_FALSE;
   83:   var code = Blockly.Yail.YAIL_WHILE + test + Blockly.Yail.YAIL_SPACER + Blockly.Yai
l.YAIL_BEGIN + toDo + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_CLOSE_COMBINA
TION;
   84:   return code;
   85: };
   86:
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Logic blocks yail generators for Blockly, modified for MIT App Inve
ntor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.logic');
   13:
   14: Blockly.Yail['logic_boolean'] = function() {
   15:   // Boolean values true and false.
   16:   var code = (this.getFieldValue('BOOL') == 'TRUE') ? Blockly.Yail.YAIL_TRUE
   17:       : Blockly.Yail.YAIL_FALSE;
   18:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   19: };
   20:
   21: Blockly.Yail['logic_false'] = function() {
   22:   return Blockly.Yail.logic_boolean.call(this);
   23: }
   24:
   25: Blockly.Yail['logic_negate'] = function() {
   26:   // negate operation
   27:   var argument = Blockly.Yail
   28:       .valueToCode(this, 'BOOL', Blockly.Yail.ORDER_NONE)
   29:       || Blockly.Yail.YAIL_FALSE;
   30:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-not"
   31:       + Blockly.Yail.YAIL_SPACER;
   32:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   33:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   34:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   35:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   36:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "boolean"
   37:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   38:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "not"
   39:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   40:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   41: };
   42:
   43: Blockly.Yail['logic_operation'] = function() {
   44:   // The and, or logic operations
   45:   // TODO: (Andrew) Make these take multiple arguments.
   46:   var mode = this.getFieldValue('OP');
   47:   var tuple = Blockly.Yail.logic_operation.OPERATORS[mode];
   48:   var operator = tuple[0];
   49:   var order = tuple[1];
   50:   var argument0 = Blockly.Yail.valueToCode(this, 'A', order) || Blockly.Yail.YAIL_FA
LSE;
   51:   var argument1 = Blockly.Yail.valueToCode(this, 'B', order) || Blockly.Yail.YAIL_FA
LSE;
   52:   var code = Blockly.Yail.YAIL_OPEN_COMBINATION + operator
   53:       + Blockly.Yail.YAIL_SPACER + argument0 + Blockly.Yail.YAIL_SPACER
   54:       + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   55:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   56: };
   57:
   58: Blockly.Yail.logic_operation.OPERATORS = {
   59:   AND : [ 'and-delayed', Blockly.Yail.ORDER_NONE ],
   60:   OR : [ 'or-delayed', Blockly.Yail.ORDER_NONE ]
   61: };
   62:
   63: Blockly.Yail['logic_or'] = function() {
   64:   return Blockly.Yail.logic_operation.call(this);
   65: }
   66:
   67: Blockly.Yail['logic_compare'] = function() {
   68:   // Basic logic compare operators
   69:   // // TODO: (Hal) handle any type?
   70:   var argument0 = Blockly.Yail.valueToCode(this, 'A', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
   71:   var argument1 = Blockly.Yail.valueToCode(this, 'B', Blockly.Yail.ORDER_NONE) || Bl
ockly.Yail.YAIL_FALSE;
   72:   var yailCommand = (this.getFieldValue('OP') == "NEQ" ? 'yail-not-equal?' : "yail-e
qual?" );
   73:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + yailCommand
   74:       + Blockly.Yail.YAIL_SPACER;
   75:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
   76:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
   77:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   78:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   79:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
   80:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "any" + Blockly.Yail.YAIL_SPACER
   81:       + "any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   82:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "="
   83:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   84:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   85: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2012 Massachusetts Institute of Technology. All rights reserved.

/**
 * @license
 * @fileoverview Math blocks yail generators for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Yail.math');

Blockly.Yail['math_number'] = function() {
  // Numeric value.
  var code = window.parseFloat(this.getFieldValue('NUM'));
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail['math_compare'] = function() {
  // Basic compare operators
  var mode = this.getFieldValue('OP');
  var prim = Blockly.Yail.math_compare.OPERATORS[mode];
  var operator1 = prim[0];
  var operator2 = prim[1];
  var order = prim[2];
  var argument0 = Blockly.Yail.valueToCode(this, 'A', order) || 0;
  var argument1 = Blockly.Yail.valueToCode(this, 'B', order) || 0;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + (mode == "EQ" || mode == "NEQ" ? "any any" : "number number" )
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail.math_compare.OPERATORS = {
  EQ: ['yail-equal?', '=', Blockly.Yail.ORDER_NONE],
  NEQ: ['yail-not-equal?', 'not =', Blockly.Yail.ORDER_NONE],
  LT: ['<', '<', Blockly.Yail.ORDER_NONE],
  LTE: ['<=', '<=', Blockly.Yail.ORDER_NONE],
  GT: ['>', '>', Blockly.Yail.ORDER_NONE],
  GTE: ['>=', '>=', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_arithmetic'] = function(mode,block) {
  // Basic arithmetic operators.
  var tuple = Blockly.Yail.math_arithmetic.OPERATORS[mode];
  var operator = tuple[0];
  var order = tuple[1];
  var argument0 = Blockly.Yail.valueToCode(block, 'A', order) || 0;
  var argument1 = Blockly.Yail.valueToCode(block, 'B', order) || 0;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION;
  code += (mode == "EQ" || mode == "NEQ" ? "any any" : "number number" )
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail['math_subtract'] = function() {
  return Blockly.Yail.math_arithmetic("MINUS",this);
};

Blockly.Yail['math_division'] = function() {
  return Blockly.Yail.math_arithmetic("DIVIDE",this);
};

Blockly.Yail['math_power'] = function() {
  return Blockly.Yail.math_arithmetic("POWER",this);
};

Blockly.Yail['math_add'] = function() {
  return Blockly.Yail.math_arithmetic_list("ADD",this);
};

Blockly.Yail['math_multiply'] = function() {
  return Blockly.Yail.math_arithmetic_list("MULTIPLY",this);
};

Blockly.Yail['math_arithmetic_list'] = function(mode,block) {
  // Basic arithmetic operators.
  //var mode = this.getFieldValue('OP');
  var tuple = Blockly.Yail.math_arithmetic.OPERATORS[mode];
  var operator = tuple[0];
  var order = tuple[1];

  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
  for(var i=0;i<block.itemCount_;i++) {
    var argument = Blockly.Yail.valueToCode(block, 'NUM' + i, order) || 0;
    code += argument + Blockly.Yail.YAIL_SPACER;
  }
  code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION;
  for(var i=0;i<block.itemCount_;i++) {
    code += "number" + Blockly.Yail.YAIL_SPACER;
  }
  code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail.math_arithmetic.OPERATORS = {
  ADD: ['+', Blockly.Yail.ORDER_NONE],
```

```
  MINUS: ['-', Blockly.Yail.ORDER_NONE],
  MULTIPLY: ['*', Blockly.Yail.ORDER_NONE],
  DIVIDE: ['/', Blockly.Yail.ORDER_NONE],
  POWER: ['expt', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_single'] = function() {
  // Basic arithmetic operators.
  var mode = this.getFieldValue('OP');
  var tuple = Blockly.Yail.math_single.OPERATORS[mode];
  var operator1 = tuple[0];
  var operator2 = tuple[1];
  var order = tuple[2];
  var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 1;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [code, Blockly.Yail.ORDER_ATOMIC];
};

Blockly.Yail.math_single.OPERATORS = {
  ROOT: ['sqrt', 'sqrt', Blockly.Yail.ORDER_NONE],
  ABS: ['abs', 'abs', Blockly.Yail.ORDER_NONE],
  NEG: ['-', 'negate', Blockly.Yail.ORDER_NONE],
  LN: ['log', 'log', Blockly.Yail.ORDER_NONE],
  EXP: ['exp', 'exp', Blockly.Yail.ORDER_NONE],
  ROUND: ['yail-round', 'round', Blockly.Yail.ORDER_NONE],
  CEILING: ['yail-ceiling', 'ceiling', Blockly.Yail.ORDER_NONE],
  FLOOR: ['yail-floor', 'floor', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_abs'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_neg'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_round'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_ceiling'] = function() {
  return Blockly.Yail.math_single.call(this);
};

Blockly.Yail['math_floor'] = function() {
  return Blockly.Yail.math_single.call(this);
};


Blockly.Yail['math_random_int'] = function() {
  // Random integer between [X] and [Y].
  var argument0 = Blockly.Yail.valueToCode(this, 'FROM',
    Blockly.Yail.ORDER_NONE) || 0;
  var argument1 = Blockly.Yail.valueToCode(this, 'TO',
    Blockly.Yail.ORDER_NONE) || 0;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-integer"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random integer"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['math_random_float'] = function() {
  // Random fraction between 0 and 1.
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-fraction"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR +
  Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE +
  Blockly.Yail.YAIL_OPEN_COMBINATION;
  code = code + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random fraction"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['math_random_set_seed'] = function() {
  // Basic is_a_number.
  var argument = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || 0
;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "random-set-seed"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "random set seed"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return code;
};

Blockly.Yail['math_on_list'] = function() {
  // Min and Max operators.
  var mode = this.getFieldValue('OP');
  var tuple = Blockly.Yail.math_on_list.OPERATORS[mode];
  var operator = tuple[0];
  var order = tuple[1];
  var args = "";
  var typeString = "";
  for(var i=0;i<this.itemCount_;i++) {
    args += (Blockly.Yail.valueToCode(this, 'NUM' + i, order) || 0) + Blockly.Yail.Y
AIL_SPACER;
    typeString += "number" + Blockly.Yail.YAIL_SPACER;
  }
```

```
243:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
244:       + Blockly.Yail.YAIL_SPACER;
245:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
246:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
247:       + args//argument0 + Blockly.Yail.YAIL_SPACER + argument1
248:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
249:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
250:       + Blockly.Yail.YAIL_OPEN_COMBINATION + typeString
251:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
252:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
253:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
254:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
255: };
256:
257: Blockly.Yail.math_on_list.OPERATORS = {
258:   MIN: ['min', Blockly.Yail.ORDER_NONE],
259:   MAX: ['max', Blockly.Yail.ORDER_NONE]
260: };
261:
262: Blockly.Yail['math_divide'] = function() {
263:   // divide operators.
264:   var mode = this.getFieldValue('OP');
265:   var tuple = Blockly.Yail.math_divide.OPERATORS[mode];
266:   var operator = tuple[0];
267:   var order = tuple[1];
268:   var argument0 = Blockly.Yail.valueToCode(this, 'DIVIDEND', order) || 0;
269:   var argument1 = Blockly.Yail.valueToCode(this, 'DIVISOR', order) || 1;
270:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator
271:       + Blockly.Yail.YAIL_SPACER;
272:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
273:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
274:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
275:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
276:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
277:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
278:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
279:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator
280:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
281:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
282: };
283:
284: Blockly.Yail.math_divide.OPERATORS = {
285:   MODULO: ['modulo', Blockly.Yail.ORDER_NONE],
286:   REMAINDER: ['remainder', Blockly.Yail.ORDER_NONE],
287:   QUOTIENT: ['quotient', Blockly.Yail.ORDER_NONE]
288: };
289:
290: Blockly.Yail['math_trig'] = function() {
291:   // Basic trig operators.
292:   var mode = this.getFieldValue('OP');
293:   var tuple = Blockly.Yail.math_trig.OPERATORS[mode];
294:   var operator1 = tuple[1];
295:   var operator2 = tuple[0];
296:   var order = tuple[2];
297:   var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 0;
298:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
299:       + Blockly.Yail.YAIL_SPACER;
300:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
301:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
302:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
303:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
304:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
305:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
306:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
307:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
308:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
309: };
310:
311: Blockly.Yail.math_trig.OPERATORS = {
312:   SIN: ['sin', 'sin-degrees', Blockly.Yail.ORDER_NONE],
313:   COS: ['cos', 'cos-degrees', Blockly.Yail.ORDER_NONE],
314:   TAN: ['tan', 'tan-degrees', Blockly.Yail.ORDER_NONE],
315:   ASIN: ['asin', 'asin-degrees', Blockly.Yail.ORDER_NONE],
316:   ACOS: ['acos', 'acos-degrees', Blockly.Yail.ORDER_NONE],
317:   ATAN: ['atan', 'atan-degrees', Blockly.Yail.ORDER_NONE]
318: };
319:
320: Blockly.Yail['math_cos'] = function() {
321:   return Blockly.Yail.math_trig.call(this);
322: };
323:
324: Blockly.Yail['math_tan'] = function() {
325:   return Blockly.Yail.math_trig.call(this);
326: };
327:
328: Blockly.Yail['math_atan2'] = function() {
329:   // atan2 operators.
330:   var argument0 = Blockly.Yail.valueToCode(this, 'Y', Blockly.Yail.ORDER_NONE) || 1;
331:   var argument1 = Blockly.Yail.valueToCode(this, 'X', Blockly.Yail.ORDER_NONE) || 1;
332:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "atan2-degrees"
333:       + Blockly.Yail.YAIL_SPACER;
334:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
335:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
336:       + argument0 + Blockly.Yail.YAIL_SPACER + argument1
337:       + Blockly.Yail.YAIL_CLOSE_COMBINATION;
338:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
339:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
340:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
341:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "atan2"
342:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
343:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
344: };
345:
346: Blockly.Yail['math_convert_angles'] = function() {
347:   // Basic arithmetic operators.
348:   var mode = this.getFieldValue('OP');
349:   var tuple = Blockly.Yail.math_convert_angles.OPERATORS[mode];
350:   var operator1 = tuple[0];
351:   var operator2 = tuple[1];
352:   var order = tuple[2];
353:   var argument = Blockly.Yail.valueToCode(this, 'NUM', order) || 0;
354:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + operator1
355:       + Blockly.Yail.YAIL_SPACER;
356:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
357:       + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
358:       + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
359:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
360:       + Blockly.Yail.YAIL_OPEN_COMBINATION + "number"
361:       + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
362:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + operator2
363:       + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
364:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
365: };
366:
```

```
Blockly.Yail.math_convert_angles.OPERATORS = {
  RADIANS_TO_DEGREES: ['radians->degrees', 'convert radians to degrees', Blockly.Yail.ORDER_NONE],
  DEGREES_TO_RADIANS: ['degrees->radians', 'convert degrees to radians', Blockly.Yail.ORDER_NONE]
};

Blockly.Yail['math_format_as_decimal'] = function() {
  // format_as_decimal.
  var argument0 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || 0;
  var argument1 = Blockly.Yail.valueToCode(this, 'PLACES', Blockly.Yail.ORDER_NONE) || 0;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "format-as-decimal"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument0 + Blockly.Yail.YAIL_SPACER + argument1
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "number number"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "format as decimal"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['math_is_a_number'] = function() {
  // Basic is_a_number.
  var argument = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
  var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "is-number?"
      + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_OPEN_COMBINATION
      + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER
      + argument + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE
      + Blockly.Yail.YAIL_OPEN_COMBINATION + "any"
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is a number?"
      + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

/*--------------------------------------*/

Blockly.Yail['now_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};

Blockly.Yail['exec_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};

Blockly.Yail['start_time'] = function(block) {
  var dropdown_func = block.getFieldValue('FUNC');
  // TODO: Assemble Yail into code variable.
  var code = '...';
  // TODO: Change ORDER_NONE to the correct strength.
  return [code, Blockly.Yail.ORDER_NONE];
};

```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Color blocks yail generators for Blockly, modified for MIT App Inve
ntor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.text');
   13:
   14: Blockly.Yail['text'] = function() {
   15:   // Text value.
   16:   var code = Blockly.Yail.quote_(this.getFieldValue('TEXT'));
   17:   return [code, Blockly.Yail.ORDER_ATOMIC];
   18: };
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Lists blocks yail generators for Blockly, modified for MIT App Inventor.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.lists');
   13:
   14: Blockly.Yail.emptyListCode = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "make-yail-list" + Blockly.Yail.YAIL_SPACER;
   15: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   16: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   17: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   18: Blockly.Yail.emptyListCode += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QUOTE + "make a list" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   19:
   20:
   21: Blockly.Yail['lists_create_with'] = function() {
   22:
   23:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "make-yail-list" + Blockly.Yail.YAIL_SPACER;
   24:   code += Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   25:   var itemsAdded = 0;
   26:   for(var i=0;i<this.itemCount_;i++) {
   27:     var argument = Blockly.Yail.valueToCode(this, 'ADD' + i, Blockly.Yail.ORDER_NONE) || null;
   28:     if(argument != null){
   29:       code += argument + Blockly.Yail.YAIL_SPACER;
   30:       itemsAdded++;
   31:     }
   32:   }
   33:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   34:   for(var i=0;i<itemsAdded;i++) {
   35:     code += "any" + Blockly.Yail.YAIL_SPACER;
   36:   }
   37:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
   38:   code += Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_DOUBLE_QUOTE + "make a list" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   39:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   40:
   41: };
   42:
   43: Blockly.Yail['lists_select_item'] = function() {
   44:   // Select from list an item.
   45:
   46:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) || Blockly.Yail.emptyListCode;
   47:   var argument1 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || 1;
   48:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-get-item" + Blockly.Yail.YAIL_SPACER;
   49:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   50:   code = code + argument0;
   51:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   52:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   53:   code = code + "list number" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   54:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "select list item" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   55:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
   56: };
   57:
   58: Blockly.Yail['lists_replace_item'] = function() {
   59:   // Replace Item in list.
   60:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) || Blockly.Yail.emptyListCode;
   61:   var argument1 = Blockly.Yail.valueToCode(this, 'NUM', Blockly.Yail.ORDER_NONE) || 1;
   62:   var argument2 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
   63:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-set-item!" + Blockly.Yail.YAIL_SPACER;
   64:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   65:   code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1
   66:   code = code + Blockly.Yail.YAIL_SPACER + argument2 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   67:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   68:   code = code + "list number any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   69:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "replace list item" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   70:   return code;
   71: };
   72:
   73: Blockly.Yail['lists_remove_item'] = function() {
   74:   // Remove Item in list.
   75:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) || Blockly.Yail.emptyListCode;
   76:   var argument1 = Blockly.Yail.valueToCode(this, 'INDEX', Blockly.Yail.ORDER_NONE) || 1;
   77:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-remove-item!" + Blockly.Yail.YAIL_SPACER;
   78:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUCTOR + Blockly.Yail.YAIL_SPACER;
   79:   code = code + argument0;
   80:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   81:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YAIL_OPEN_COMBINATION;
   82:   code = code + "list number" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
   83:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "remove list item" + Blockly.Yail.YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   84:   return code;
   85: };
   86:
   87: Blockly.Yail['lists_insert_item'] = function() {
   88:   // Insert Item in list.
   89:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
```

```
 Blockly.Yail.emptyListCode;
   90:   var argument1 = Blockly.Yail.valueToCode(this, 'INDEX', Blockly.Yail.ORDER_NONE) |
| 1;
   91:   var argument2 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
   92:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-insert-item!" + Bloc
kly.Yail.YAIL_SPACER;
   93:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
   94:   code = code + argument0 + Blockly.Yail.YAIL_SPACER + argument1;
   95:   code = code + Blockly.Yail.YAIL_SPACER + argument2 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
   96:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
   97:   code = code + "list number any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Ya
il.YAIL_SPACER;
   98:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "insert list item" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
   99:   return code;
  100: };
  101:
  102: Blockly.Yail['lists_length'] = function() {
  103:   // Length of list.
  104:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  105:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-length" + Blockly.Ya
il.YAIL_SPACER;
  106:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  107:   code = code + argument0;
  108:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  109:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  110:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  111:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "length of list" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  112:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  113: };
  114:
  115: Blockly.Yail['lists_append_list'] = function() {
  116:   // Append to list.
  117:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST0', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.emptyListCode;
  118:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST1', Blockly.Yail.ORDER_NONE) |
| Blockly.Yail.emptyListCode;
  119:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-append!" + Blockly.Y
ail.YAIL_SPACER;
  120:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  121:   code = code + argument0;
  122:   code = code + Blockly.Yail.YAIL_SPACER;
  123:   code = code + argument1 + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  124:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  125:   code = code + "list list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAI
L_SPACER;
  126:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "append to list" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  127:   return code;
  128: };
  129:
  130: Blockly.Yail['lists_add_items'] = function() {
  131:   // Add items to list.
  132:   // TODO: (Andrew) Make this handle multiple items.
  133:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  134:   var argument1 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 1;
  135:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-add-to-list!" + Bloc
kly.Yail.YAIL_SPACER;
  136:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  137:   code = code + argument0 + Blockly.Yail.YAIL_SPACER;
  138:
  139:   for(var i=0;i<this.itemCount_;i++) {
  140:     var argument = Blockly.Yail.valueToCode(this, 'ITEM' + i, Blockly.Yail.ORDER_NON
E) || Blockly.Yail.YAIL_FALSE;
  141:     code += argument + Blockly.Yail.YAIL_SPACER;
  142:   }
  143:
  144:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION;
  145:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  146:   code = code + "list ";
  147:   for(var i=0;i<this.itemCount_;i++) {
  148:     code += "any" + Blockly.Yail.YAIL_SPACER;
  149:   }
  150:   code += Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER;
  151:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "add items to list" + Blockly.Yail.
YAIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  152:   return code;
  153: };
  154:
  155: Blockly.Yail['lists_is_in'] = function() {
  156:   // Is in list?.
  157:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  158:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  159:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-member?" + Blockly.Y
ail.YAIL_SPACER;
  160:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  161:   code = code + argument0;
  162:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
  163:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  164:   code = code + "any list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL
_SPACER;
  165:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is in list?" + Blockly.Yail.YAIL_D
OUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  166:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  167: };
  168:
  169: Blockly.Yail['lists_position_in'] = function() {
  170:   // Postion of item in list.
  171:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 1;
  172:   var argument1 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  173:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-index" + Blockly.Yai
l.YAIL_SPACER;
```

```
  174:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  175:   code = code + argument0;
  176:   code = code + Blockly.Yail.YAIL_SPACER + argument1 + Blockly.Yail.YAIL_CLOSE_COMBI
NATION;
  177:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  178:   code = code + "any list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL
_SPACER;
  179:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "position in list" + Blockly.Yail.Y
AIL_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  180:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  181: };
  182:
  183: Blockly.Yail['lists_is_empty'] = function() {
  184:   // Is the list empty?.
  185:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  186:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-empty?" + Blockly.Ya
il.YAIL_SPACER;
  187:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  188:   code = code + argument0;
  189:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  190:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  191:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  192:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is list empty?" + Blockly.Yail.YAI
L_DOUBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  193:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  194: };
  195:
  196: Blockly.Yail['lists_copy'] = function() {
  197:   // Make a copy of list.
  198:   var argument0 = Blockly.Yail.valueToCode(this, 'LIST', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.emptyListCode;
  199:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list-copy" + Blockly.Yail
.YAIL_SPACER;
  200:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  201:   code = code + argument0;
  202:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  203:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  204:   code = code + "list" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPA
CER;
  205:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "copy list" + Blockly.Yail.YAIL_DOU
BLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  206:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  207: };
  208:
  209: Blockly.Yail['lists_is_list'] = function() {
  210:   // Create an empty list.
  211:   // TODO:(Andrew) test whether thing is var or text or number etc...
  212:   var argument0 = Blockly.Yail.valueToCode(this, 'ITEM', Blockly.Yail.ORDER_NONE) ||
 Blockly.Yail.YAIL_FALSE;
  213:   var code = Blockly.Yail.YAIL_CALL_YAIL_PRIMITIVE + "yail-list?" + Blockly.Yail.YAI
L_SPACER;
  214:   code = code + Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_LIST_CONSTRUC
TOR + Blockly.Yail.YAIL_SPACER;
  215:   code = code + argument0;
  216:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  217:   code = code + Blockly.Yail.YAIL_SPACER + Blockly.Yail.YAIL_QUOTE + Blockly.Yail.YA
IL_OPEN_COMBINATION;
  218:   code = code + "any" + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPAC
ER;
  219:   code = code + Blockly.Yail.YAIL_DOUBLE_QUOTE + "is a list?" + Blockly.Yail.YAIL_DO
UBLE_QUOTE + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  220:   return [ code, Blockly.Yail.ORDER_ATOMIC ];
  221: };
```

```
// -*- mode: java; c-basic-offset: 2; -*-
// Copyright 2012 Massachusetts Institute of Technology. All rights reserved.

/**
 * @license
 * @fileoverview Procedure yail generators for Blockly, modified for MIT App Inventor.
 * @author mckinney@mit.edu (Andrew F. McKinney)
 */

'use strict';

goog.provide('Blockly.Yail.procedures');

/**
 * Lyn's History:
 * [lyn, 10/29/13] Fixed bug in handling parameters of zero-arg procedures.
 * [lyn, 10/27/13] Modified procedure names to begin with YAIL_PROC_TAG (currently 'p$')
 *     and parameters to begin with YAIL_LOCAL_VAR_TAG (currently '$').
 *     At least on Kawa-legal first character is necessary to ensure AI identifiers
 *     satisfy Kawa's identifier rules. And the procedure 'p$' tag is necessary to
 *     distinguish procedures from globals (which use the 'g$' tag).
 * [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN)
 */

Blockly.Yail.YAIL_PROC_TAG = 'p$'; // See notes on this in generators/yail/variables.js

// Generator code for procedure call with return
// [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN)
Blockly.Yail['procedures_defreturn'] = function() {
  var argPrefix = Blockly.Yail.YAIL_LOCAL_VAR_TAG
                  + (Blockly.usePrefixInYail && this.arguments_.length != 0 ? "param_" : "");
  var args = this.arguments_.map(function (arg) {return argPrefix + arg;}).join(' ');
  var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('NAME');
  var returnVal = Blockly.Yail.valueToCode(this, 'RETURN', Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
  var code = Blockly.Yail.YAIL_DEFINE + Blockly.Yail.YAIL_OPEN_COMBINATION + procName
      + Blockly.Yail.YAIL_SPACER + args + Blockly.Yail.YAIL_CLOSE_COMBINATION
      + Blockly.Yail.YAIL_SPACER + returnVal + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return code;
};

// Generator code for procedure call with return
Blockly.Yail['procedures_defnoreturn'] = function() {
  var argPrefix = Blockly.Yail.YAIL_LOCAL_VAR_TAG
                  + (Blockly.usePrefixInYail && this.arguments_.length != 0 ? "param_" : "");
  var args = this.arguments_.map(function (arg) {return argPrefix + arg;}).join(' ');
  var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('NAME');
  var body = Blockly.Yail.statementToCode(this, 'STACK', Blockly.Yail.ORDER_NONE)  || Blockly.Yail.YAIL_FALSE;
  var code = Blockly.Yail.YAIL_DEFINE + Blockly.Yail.YAIL_OPEN_COMBINATION + procName
      + Blockly.Yail.YAIL_SPACER + args + Blockly.Yail.YAIL_CLOSE_COMBINATION + body
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return code;
};

Blockly.Yail['procedure_lexical_variable_get'] = function() {
  return Blockly.Yail.lexical_variable_get.call(this);
}

//call the do return in control category
Blockly.Yail['procedures_do_then_return'] = function() {
  return Blockly.Yail.controls_do_then_return.call(this);
}

// Generator code for procedure call with return
Blockly.Yail['procedures_callnoreturn'] = function() {
  var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
  var argCode = [];
  for ( var x = 0;this.getInput("ARG" + x); x++) {
    argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
  }
  var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + procName
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.join(' ')
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return code;
};

// Generator code for procedure call with return
Blockly.Yail['procedures_callreturn'] = function() {
  var procName = Blockly.Yail.YAIL_PROC_TAG + this.getFieldValue('PROCNAME');
  var argCode = [];
  for ( var x = 0; this.getInput("ARG" + x); x++) {
    argCode[x] = Blockly.Yail.valueToCode(this, 'ARG' + x, Blockly.Yail.ORDER_NONE) || Blockly.Yail.YAIL_FALSE;
  }
  var code = Blockly.Yail.YAIL_OPEN_COMBINATION + Blockly.Yail.YAIL_GET_VARIABLE + procName
      + Blockly.Yail.YAIL_CLOSE_COMBINATION + Blockly.Yail.YAIL_SPACER + argCode.join(' ')
      + Blockly.Yail.YAIL_CLOSE_COMBINATION;
  return [ code, Blockly.Yail.ORDER_ATOMIC ];
};

Blockly.Yail['nil_proc'] = function(block) {
  // TODO: Assemble Yail into code variable.
  var code = '...';
  return code;
};
```

```
    1: // -*- mode: java; c-basic-offset: 2; -*-
    2: // Copyright 2012 Massachusetts Institute of Technology. All rights reserved.
    3:
    4: /**
    5:  * @license
    6:  * @fileoverview Procedure yail generators for Blockly, modified for MIT App Invento
r.
    7:  * @author mckinney@mit.edu (Andrew F. McKinney)
    8:  */
    9:
   10: 'use strict';
   11:
   12: goog.provide('Blockly.Yail.qulog');
   13:
   14: /**
   15:  * Lyn's History:
   16:  * [lyn, 10/29/13] Fixed bug in handling parameters of zero-arg procedures.
   17:  * [lyn, 10/27/13] Modified procedure names to begin with YAIL_PROC_TAG (currently '
p$')
   18:  *     and parameters to begin with YAIL_LOCAL_VAR_TAG (currently '$').
   19:  *     At least on Kawa-legal first character is necessary to ensure AI identifiers
   20:  *     satisfy Kawa's identifier rules. And the procedure 'p$' tag is necessary to
   21:  *     distinguish procedures from globals (which use the 'g$' tag).
   22:  * [lyn, 01/15/2013] Edited to remove STACK (no longer necessary with DO-THEN-RETURN
)
   23:  */
   24:
   25: //Blockly.Yail.YAIL_PROC_TAG = 'p$'; // See notes on this in generators/yail/variabl
es.js
   26:
   27: Blockly.Yail['remember_call'] = function(block) {
   28:   var dropdown_dd = block.getFieldValue('DD');
   29:   var value_belief = Blockly.Yail.valueToCode(block, 'belief', Blockly.Yail.ORDER_AT
OMIC);
   30:   var value_time = Blockly.Yail.valueToCode(block, 'time', Blockly.Yail.ORDER_ATOMIC
);
   31:   // TODO: Assemble Yail into code variable.
   32:   var code = '...';
   33:   return code;
   34: };
   35:
   36: Blockly.Yail['forget_call'] = function(block) {
   37:   var dropdown_dd = block.getFieldValue('DD');
   38:   var value_belief = Blockly.Yail.valueToCode(block, 'belief', Blockly.Yail.ORDER_AT
OMIC);
   39:   var value_time = Blockly.Yail.valueToCode(block, 'time', Blockly.Yail.ORDER_ATOMIC
);
   40:   // TODO: Assemble Yail into code variable.
   41:   var code = '...';
   42:   return code;
   43: };
   44:
```