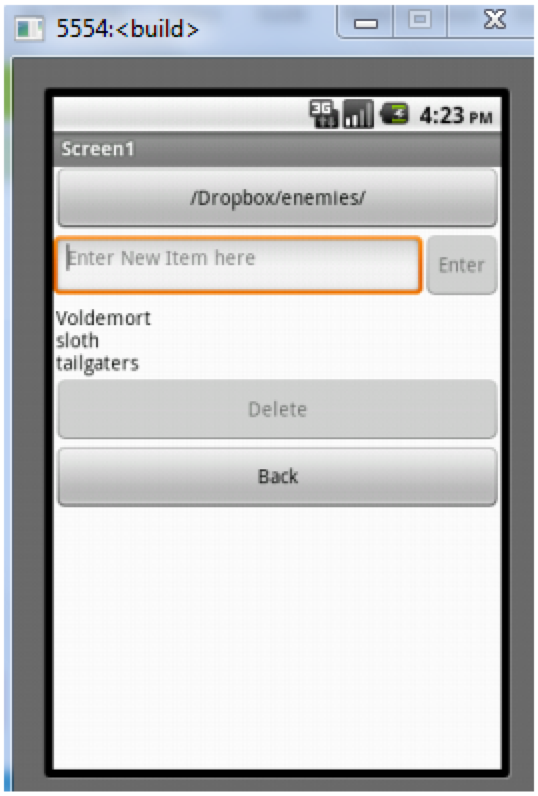
A TinyDB Tree Editor

March 2018

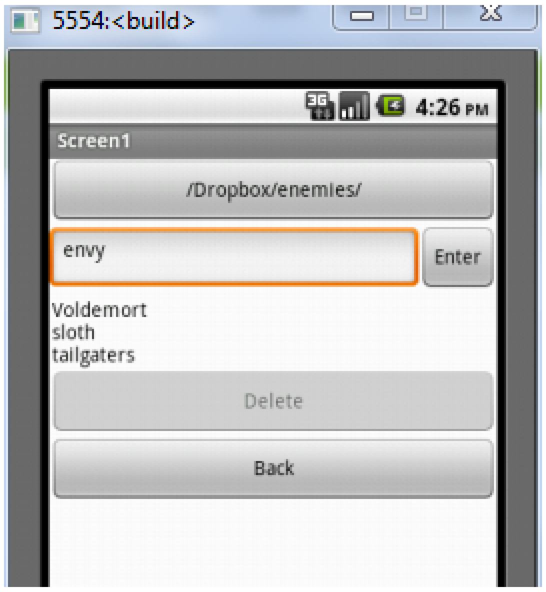
Background

This app is a solution to a problem posed the MIT App Inventor 2 Help Board at https://groups.google.com/d/msg/mitappinventortest/_jLspotjBA4/pQiLq4LkBgAJ, asking for a way to solicit and store user defined topics and subtopics in TinyDB, displaying lists vertically in labels.

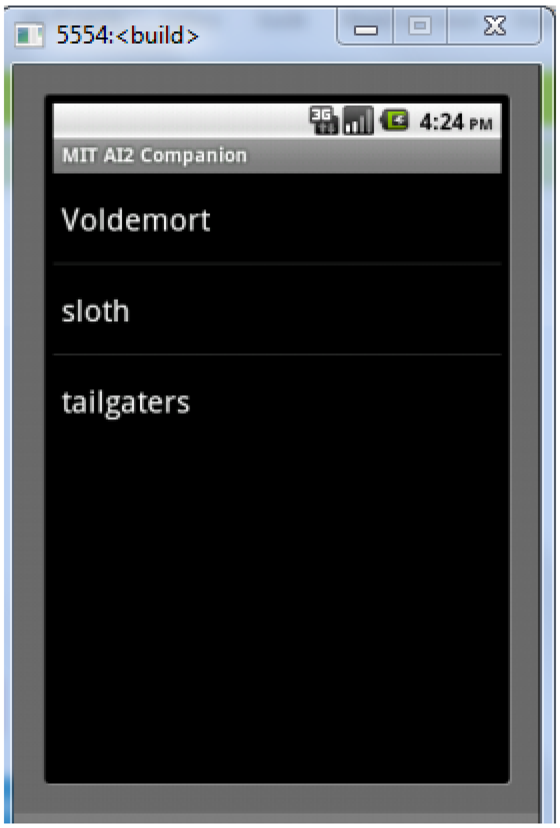
Sample runs



This is a display of three items in the list stored under tag /Dropbox/enemies. Because the list is not empty, the Delete button is disabled. (Only terminal items can be deleted in this implementation, to keep the blocks simple.) Because the text box to enter new items is empty, the Enter button is disabled. We do not allow blank or empty items here.



In this shot, we have typed the enemy **envy** into our data entry text box, but have not yet pressed the Enter key. A Clock component is checking the text box for content every second, controlling the enabling of the Enter key. An additional NewLine sensing feature in the Clock Timer event checks for **\n** in the MultiLine text box and triggers an Enter action for the front of the text.



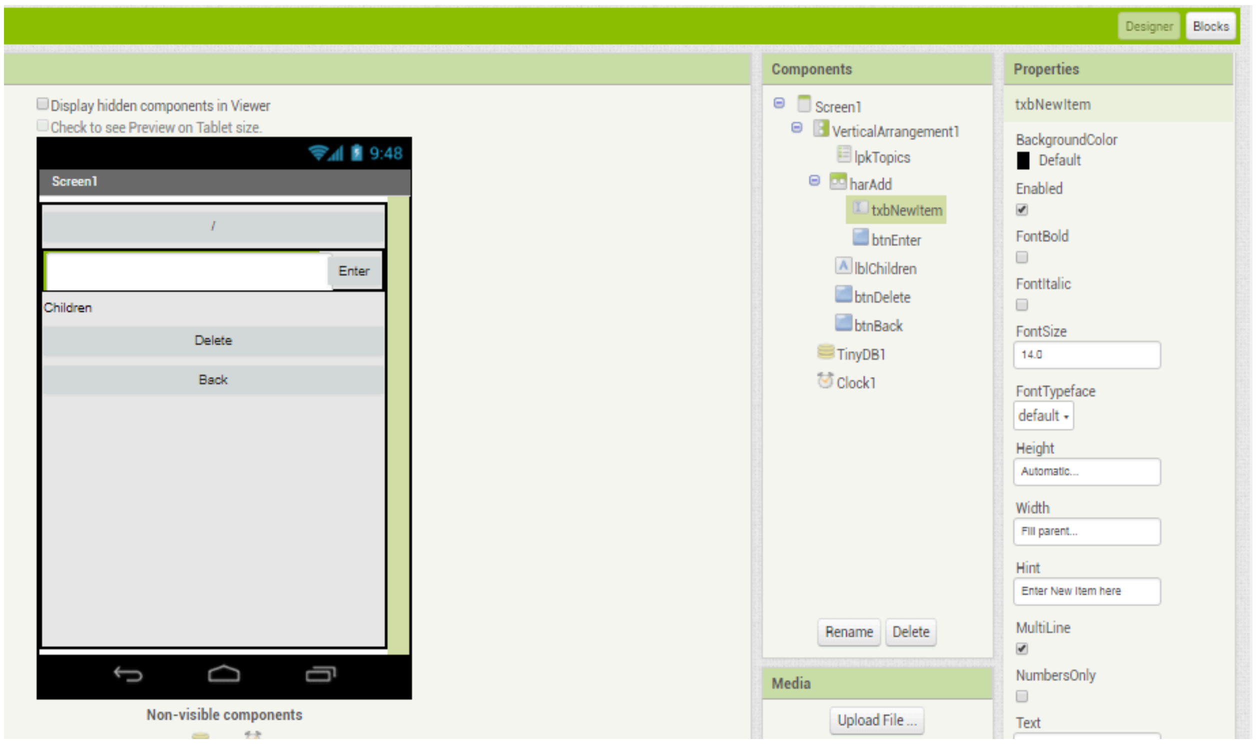
Voldemort

sloth

tailgaters

This is the List Picker offering a chance to select an item from the list at the position in the tree defined by the List Picker Text display. Use the device Back function to cancel the List Picker popup. (It probably would have been nicer to include the current parent as the List Picker Title, to show the current context.)

Designer



txbNewItem

BackgroundColor
■ Default

Enabled
✓

FontBold
☐

FontItalic
☐

FontSize
14.0

FontTypeface
default

Height
Automatic...

Width
Fill parent...

Hint
Enter New Item here

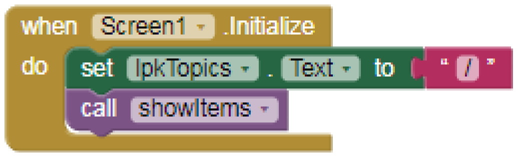
MultiLine
✓

NumbersOnly
☐

Text

Components

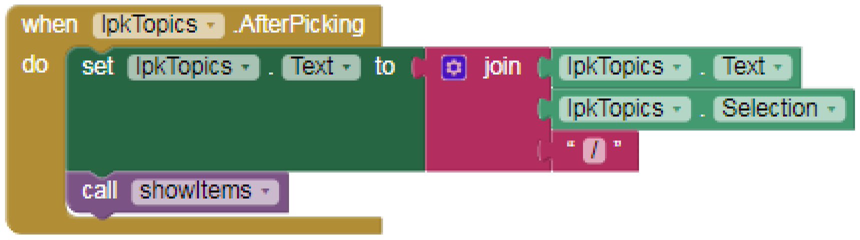
Screen1



```
when Screen1 .Initialize
do  set lpkTopics . Text to " / "
    call showItems
```

At startup, we start from the root of our tree, at position “/”. (Sorry MSDOS users.) We then refresh the display using procedure showItems.

lpkTopics



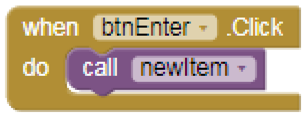
```
when lpkTopics .AfterPicking
do  set lpkTopics . Text to  join  lpkTopics . Text
                                   lpkTopics . Selection
                                   " / "
    call showItems
```

We depend on List Picker lpkTopics's .Text value to show our current position in the tree of Topics. On selection, we follow that Selection value and refresh the display using procedure showItems.

txbNewItem

This text box has no events, and is controlled from button btnEnter and Clock1.Timer. It is set as MultiLine in the Designer, to allow New Lines to be entered for Clock controlled Enter processing.

btnEnter



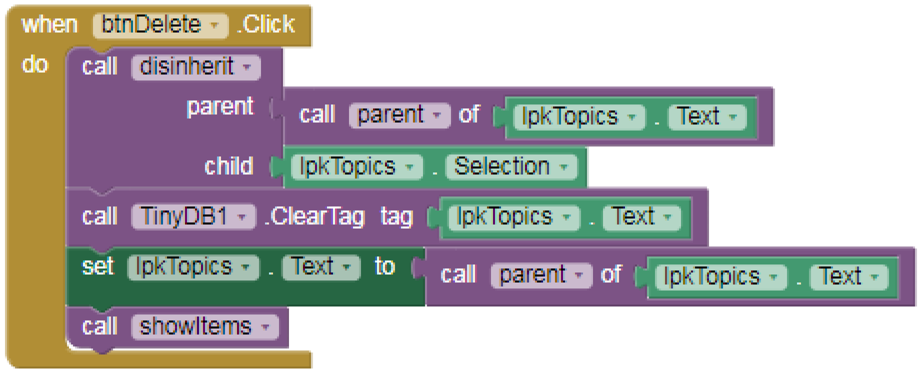
```
when btnEnter .Click
do  call newItem
```

When the Enter button is pressed, we call the newItem procedure to try to add the entered text into the children list of the parent displayed in lpkTopics.Text.

lblChildren

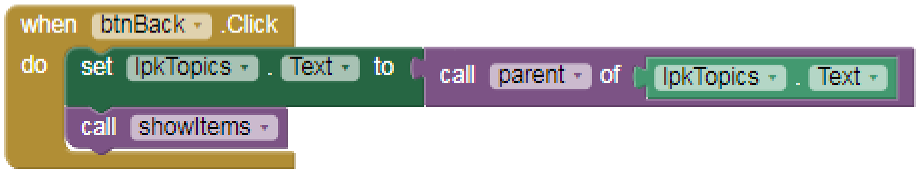
This label shows the children of current tag, based on lpkTopics.Text.

btnDelete



```
when btnDelete .Click
do  call disinherit
        parent  call parent of  lpkTopics . Text
        child   lpkTopics . Selection
    call TinyDB1 .ClearTag tag  lpkTopics . Text
    set lpkTopics . Text to  call parent of  lpkTopics . Text
    call showItems
```

btnBack

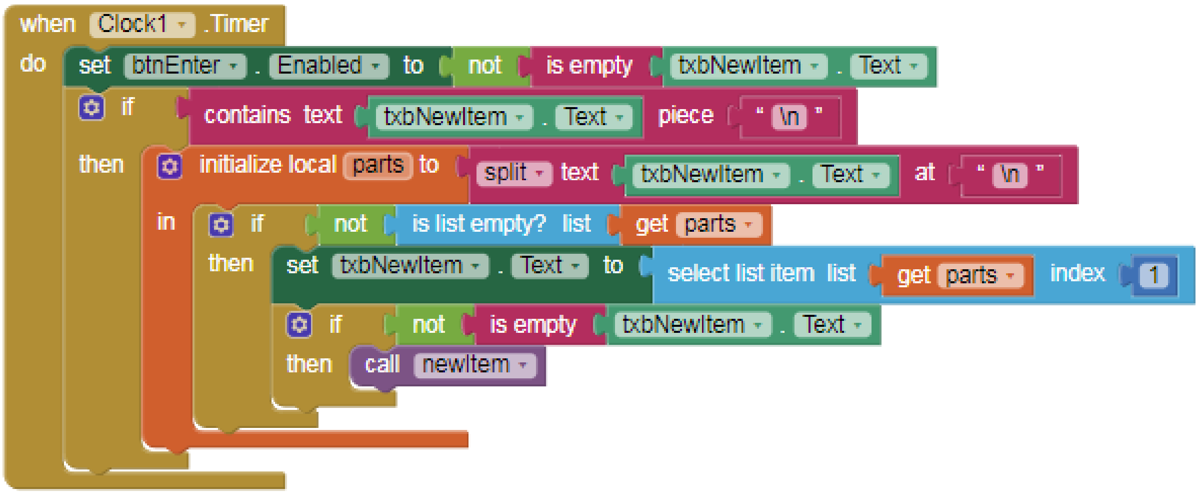


```
when btnBack .Click
do  set lpkTopics . Text to  call parent of  lpkTopics . Text
    call showItems
```

TinyDB

We follow a rule of obtaining values from TinyDB immediately before use, and storing them back immediately after updating them. This keeps TinyDB current at all times.

Clock1

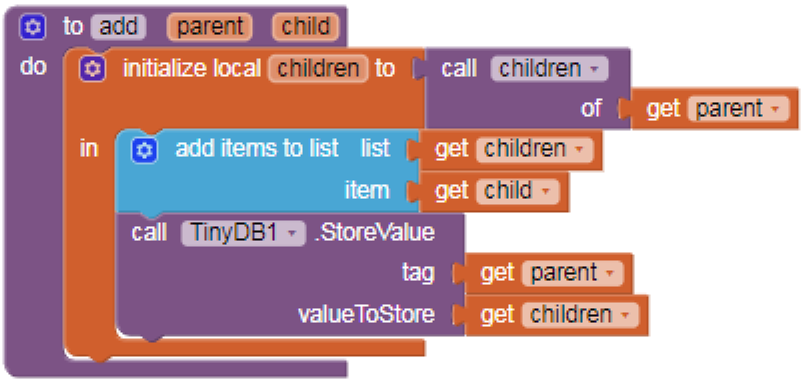


```
when Clock1 .Timer
do  set btnEnter . Enabled to  not  is empty  txbNewItem . Text
    if    contains text  txbNewItem . Text  piece  " \n "
    then  initialize local parts to  split text  txbNewItem . Text  at  " \n "
          in  if    not  is list empty? list  get parts
              then  set txbNewItem . Text to  select list item list  get parts  index  1
                    if    not  is empty  txbNewItem . Text
                    then  call newItem
```

The Clock cycles continually, one cycle per second. It is responsible for checking for non-empty text in text box txbNewItem, and for \n in the text signalling an Enter request. If \n is detected, it eliminates false alarms and empty text and replaces txbNewItem.Text with valid nonblank text, then calls procedure newItem to save the text.

Procedures

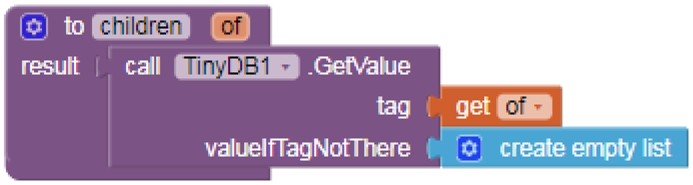
add



```
to add  parent  child
do  initialize local children to  call children
                                       of  get parent
    in  add items to list  list  get children
                           item  get child
        call TinyDB1 .StoreValue
                         tag  get parent
                valueToStore  get children
```

This procedure adds a single child item to the list of children of the supplied parent tag. The current list of children is recovered into a local variable, then the supplied child is added to the list at the end. (To add it at the top of the list for visibility, change that to an **Insert at Position 1**). We immediately store the newly expanded list of children at the given tag in TinyDB.

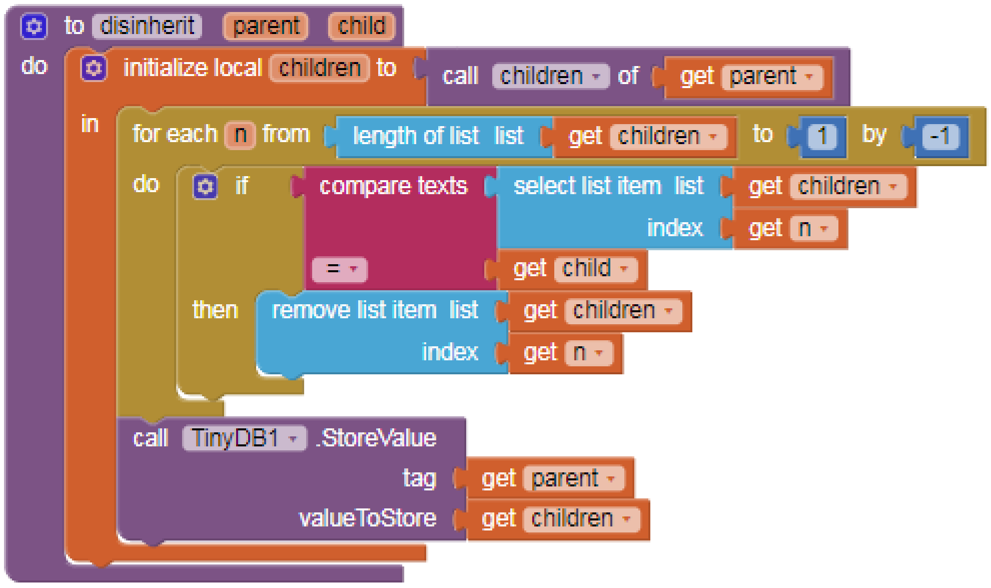
children



```
to children of
result  call TinyDB1 .GetValue
                         tag  get of
          valueIfTagNotThere  create empty list
```

This procedure returns the children of a TinyDB tag, as a list. If none are found, an empty list is returned.

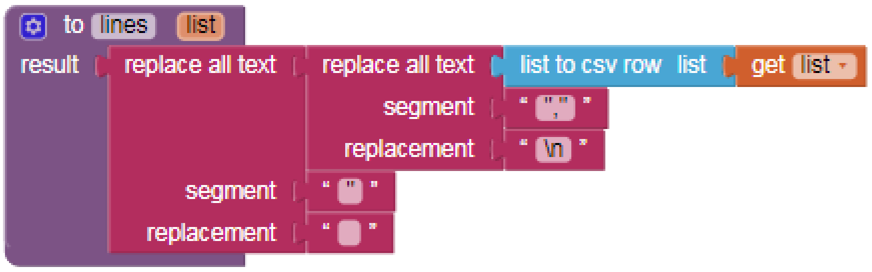
disinherit



```
to disinherit parent child
do  initialize local children to  call children of  get parent
    in  for each n from  length of list list  get children  to 1 by -1
        do  if  compare texts  select list item list  get children
                                                index  get n
                         =     get child
            then  remove list item list  get children
                                  index  get n
        call TinyDB1 .StoreValue
                     tag  get parent
            valueToStore  get children
```

This procedure removes a child from a parent's list of children.   (A better name eludes me.) We get the current list of children into a local variable, the loop through that list backwards, to avoid collapse on deletion of items. Matching items are deleted. We then replace that list of children in TinyDB under the given parent tag.

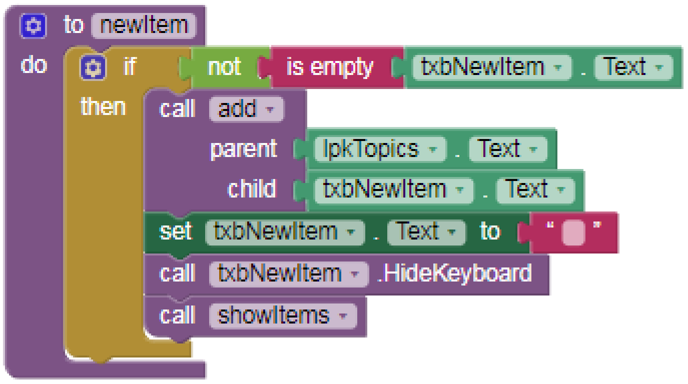
lines



```
to lines list
result  replace all text  replace all text  list to csv row  list  get list
                                            segment      " "," "
                                            replacement  " \n "
                          segment      " " "
                          replacement  "  "
```

This value procedure takes a list, and returns a piece of text with the items of that list separated by \n new line characters. A trick of the **list to csv row** block is used to avoid a loop.

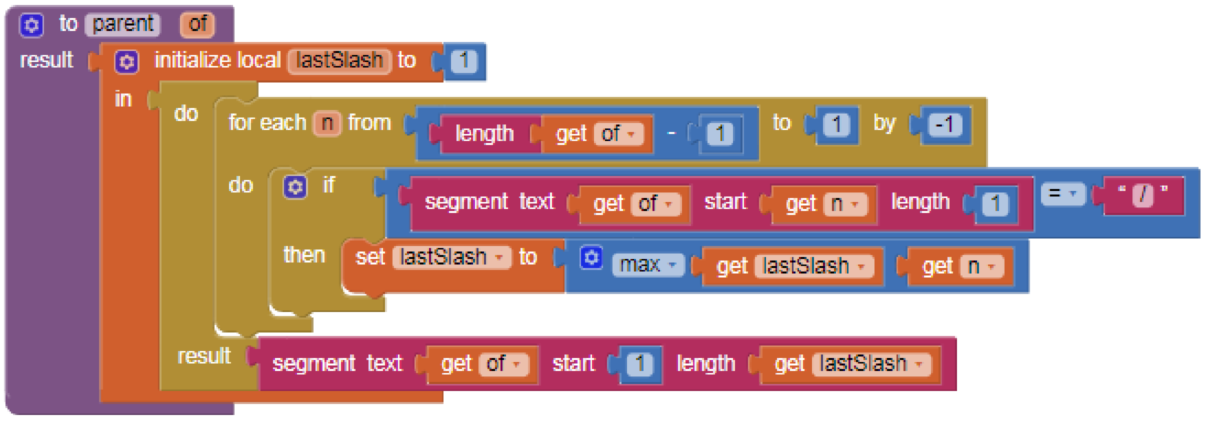
newItem



```
to newItem
do  if    not  is empty  txbNewItem . Text
    then  call add
              parent  lpkTopics . Text
              child   txbNewItem . Text
          set txbNewItem . Text to "  "
          call txbNewItem .HideKeyboard
          call showItems
```

This procedure is called in two places, when the user wants to add the contents of textbox txbNewItem.Text to the children of lpkTopics.Text. We do not allow empty text as children. After adding, we clear the entry as feedback, hide the keyboard, and refresh the display to show the operation worked using procedure showItems.

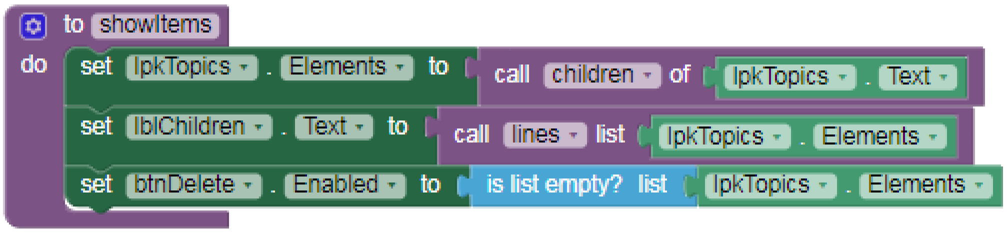
parent



```
to parent of
result  initialize local lastSlash to 1
        in  do  for each n from  length  get of  -  1   to 1  by -1
                do  if    segment text  get of  start  get n  length 1   =   " / "
                    then  set lastSlash to  max  get lastSlash   get n
            result  segment text  get of  start 1  length  get lastSlash
```

This value procedure takes a “/” delimited and terminated path value, and returns it shortened by one level, dropping the end. It loops backwards from just before the end of the input, capturing the rightmost “/” position (lastSlash), and uses that to return the front of the input value. The max() function insures we get the rightmost “/” position.

showItems



```
to showItems
do  set lpkTopics . Elements to   call children of lpkTopics . Text
    set lblChildren . Text to   call lines list lpkTopics . Elements
    set btnDelete . Enabled to   is list empty? list lpkTopics . Elements
```

This procedure is responsible for updating the display, based on our current position in the tree, as shown in lpkTopics.Text. We get its children, and show them as elements of the List Picker lpkTopics. We update label lblChildren with a vertical arrangement of the list of children, formatting the list vertically as text in procedure lines. The Delete button btndelete should be enabled only if the current position in the tree has no children.

Links

This project in the Gallery

http://ai2.appinventor.mit.edu/?galleryId=5967796550893568

More Projects

https://docs.google.com/document/d/1acg2M5KdunKjJgM3Rxpy_Rf6vT6OozxdIWglgbmzroA/edit?usp=sharing