DynaBook Junior Specification

Ted Kaehler

VPRI Memo M-2011-001

This material is based upon work supported in part by the National Science Foundation under Grant No. 0639876. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Viewpoints Research Institute, 1209 Grand Central Avenue, Glendale, CA 91201 t: (818) 332-3001 f: (818) 244-9761

DynaBook Junior Specification

by Ted Kaehler

14 Jan 2009. (Minor corrections 13 Dec 2010)

A specification for DynaBook Junior shown in itself.

DBJr is a desktop publishing framework and application builder that is modeled after Apple's HyperCard(tm).

Abstract:
DynaBook Junior is a desktop publishing framework and application builder that is modeled after Apple's HyperCard(tm). The purpose of this specification is as a starting point for the discovery of an extremely simple way to describe and automatically generate DynaBook Junior. The specification describes stacks, pages, backgrounds, objects embedded in pages, the front-to-back ordering of objects, and page-specific objects. This document is itself a DBJr stack that shows examples of the features it describes. Algorithms for showing a new page, adding pages, adding backgrounds, and adding objects to a page are given in pseudocode.

DynaBook Junior Spec

A Specification and Manual for DynaBook Junior

by Ted Kaehler, 14 Jan 2009. Minor corrections 13 Dec 2010.

Outline

DynaBook Junior Spec

This is a Page in a Stack

A Page in a Stack

Costumes are objects that can appear on the computer screen in a 2D arrangement with Z-order and overlap. A costume can have other costumes embedded inside it. The embedded costumes are "inside" their parent object. They are "parts" of the parent costume. An embedded costume can have more costumes embedded inside it. (In Squeak a costume is a called Morph. In COLA/Pepsi and Lesserphic2 it is called a Box.)

One type of costume is a text field, such as the one you are reading now. Others are geometric shapes and pictures.

Consider a costume with other objects embedded in it. In addition to being a normal costume, it can be one page of a Dynabook Junior stack. A stack is like a book and can have many pages. At any time, the stack is showing one of its pages on the screen. The page you are reading is a page in a stack.

DynaBook Junior is modeled after Apple's HyperCard.

The same outer costume plays three different roles. A user thinks of is as "the stack". For example, it is the object you move when you want to move the stack to another location on the screen. It also looks like one page in the stack (the currently showing page). It is also the "background costume" of many similar pages. More about backgrounds in a moment.

The outer costume contains a pointer to its background object.

A Text Field

2

DynaBook Junior Spec

Turning a Generic Page

The purpose of a stack is to hold many pages. Often these pages have the same "background". They look the same, but their fields contain different text. The page you are reading and the front page of this stack are identical, except that their text is different. They are both pages of the same background.

It would be wasteful to have a separate tree of costumes for each page. DynaBook Junior (DBJr) reuses the same costume for all pages of a background. Turning to another page of the same background does not involve replacing the costumes. Instead, only the text in each text field is replaced.

To turn from one page to another, the following things must be done:

§ Store the text in each field into its variable in the old DBJrPage object.

§ Load each field with text from a variable in the new DBJrPage object.

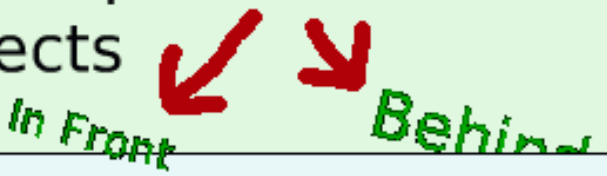
§ Set the pageIndex of the DBJrStack to the index of the new page.

The stack has an array of pages (pageArray) and a pageIndex. pageIndex is the index of the currently-showing page in pageArray.

For each costume embedded in the background costume, the DBJr background object has a variable. It holds the costume and has a getter method with the same name as the costume.

Each DBJr page object has a variable that holds the text for each background field. It has a getter whose name is the name of the field costume followed by 'Contents'.

Page-Specific Objects



Turning a Page that Contains Page-Specific Objects

Any page can contain costumes that appear only on that page. These are page-specific objects. The page object has a variable for each page-specific object, that holds it permanently. The object is marked with the property *pageSpecific*.

In the Z-order of objects, page-specific and background objects can be intermixed. We need to preserve this order. A page has variables to remember the order:

viewOrder, an array of getter selectors of objects in the current page (for this page they are: inFront redArrow1 mainField behind redArrow2 headline docTitle previousButton nextButton pageNumber). The viewOrder list is front to back.

viewOrderMask, a string 'PPPBPPPBBBB' that tells if the corresponding object is page-specific (P) or from the background (B).

When a page is turned,

§ Set viewOrder and viewOrderMask to reflect the current order.

§ Remove page-specific costumes from the costume that owns them (the background costume), but do not damage them.

§ Using the viewOrder and viewOrderMask from the new page, insert the new page-specific costumes into the background costume in the proper order.

(We assume that each costume holds its position as a displacement from its owner. Extra work must be done when costumes use an absolute coordinate system, such as in Morphic.)

DynaBook Junior Spec

Turning to a New Background

This page looks different. It has a different background.

When a page is turned, the new page can be of a different background. To turn the page, do the steps for the old page, change backgrounds, and do the steps for the new page.

To change backgrounds, simply remove the current background costume from its owner, and add the new background costume at the same location.

In a stack, pages of different backgrounds can be intermixed. The pages of one background do not have to be consecutive.

Each page has a way to find its background, and a background has a pointer to its costume and to the stack.

DynaBook Junior Spec

Drag the square to a new location

Move or Modify an Object on a Page

The costumes on a page can be modified in many ways, both from the user interface, and from scripts.

Dynabook Junior only needs to know one thing about this: that a modification has occurred somewhere on a page.

Because costumes are held in variables in the DBJr's internal objects, the variables continue to point to the changed costumes. Changes to these costumes do not need to be recorded anywhere else. When a stack is saved in a file, the current state of all costumes is recorded.

Any costume can be embedded in a page, including another stack. At the right is a stack of names and addresses named Contacts.

Arthur Hull
Village Music Circle
Santa Cruz, CA

Page Menu and Keystroke Shortcuts

Users interact with DBJr via additional menu items in the host system's menus. The menu belonging to the background costume has these additional items:

    save stack in file
    stack and page...

The latter has a sub-menu of:

    new page
    new page of background...
    new blank background (will be the same size and class as current costume, but without any embedded sub-costumes)
    delete page
    rename stack
    saved all stacks in files
    go to page...
    organize pages
    cut page
    copy page
    paste page

Because page turning is so fundamental to DBJr, keystrokes for it are added to be text editing class. The control key (in Windows) combined with left and right arrow mean turn to the previous and next page. On the Macintosh, Command left arrow and Command right arrow are used.

escape (esc)	go to most recently viewed page
Control-1	go to previous page
Control-2	go to next page
Control-3	go to first page
Control-4	go to last page

DynaBook Junior Spec

Scripts

The great flexibility of DBJr comes from user-defined actions within a page. Any costume can have a DBJr script attached to it. Each script is triggered by one of many event triggers, such as mouseUp or pageOpen.

In this implementation, scripts are in the native programming language of the system. Scripts are edited in the same editor used for native programming. When a script is accepted, the editor must register the event that will trigger the script. It must delete the trigger when a script is deleted.

All scripts are defined as messages to the DBJr page object, even those intended for the background. Note that the script is not a method of the costume. This is because costumes belong to the native system, and we don't want to modify the behavior of other instances of them that are not in this stack.

There must be a way to associate a trigger with a script. One convention is to include the event name as part of the script name.

It is important that a script be able to refer to the data that is in fields of the current page, costumes on the page, other pages, and user-defined variables.

```
data in a field of the current page:  (self <name>Contents)
background object:          (self background)
background costume:         (self costume)
a costume on the page:      (self <name>)
stack:                      (self stack)
data in a field of another page:
        (self stack pageArray at: 5) <name>Contents
```

Of course DBJr itself can be controlled from a script. A script in a button can turn the page for example.

DynaBook Junior Spec

Types of objects needed

A major decision is whether the variables holding the text in fields are instance variables.

If instance variables are used (as in DBJr in Squeak), scripts go fast and read naturally because the costumes on pages are stored in instance variables. Variables are at compiled-in offsets from the start of the page object. However, each page with page-specific costumes must have its own light-weight class. As costumes are added removed, instance variable must be added and removed, and this is currently somewhat slow, but could be speeded up. Any page-specific script requires that the page have its own light-weight class.

If a dictionary of properties is used to store costumes, each object reference in a script requires a slower dictionary lookup. Without an extended syntax, the property references will look ugly in scripts. Light-weight class are still needed for page-specific scripts.

If a prototype system is used (as in Pepsi), pages with no extra objects or scripts are siblings, and pages with them are prototypes. Prototypes are much simpler than light-weight classes. The question of fields being properties or instance variables still remains.

(At the moment, Pepsi is not capable of adding and removing instance variables at runtime when sibling instances are present. DBJr in Pepsi uses a dictionary of properties to store costumes. Scripts are not allowed to be high-speed compiled methods.)

Other classes are: DBJrStack, DBJrPage, DBJrBackground, DBJrPageXX, DBJrBackgroundXX, where XX means a specialized subclass or a prototype.

Adding a page or background

To add a page, create a new instance of the current generic page. Put the default text into the field contents variables. Extend the pageArray to make an empty entry after the current page. Store the new page entry there. Turn to the new page.

Issue the event newPageCreated so that a stack or background script can respond to it. The background objects can have scripts, and so can the stack.

To add a page of a different existing background, follow the normal procedure using the designated background. This allows pages of different backgrounds to be intermixed.

Provide a way to add a new background that is the same size, shape, and color as the current background costume, but has no embedded costumes. Copy the current background, but not its variables or scripts. Add a page. This allows new backgrounds to be created.

There needs to be a way to designate an outside costume to be the costume of a new background.

Embedding a New Costume Part

A new costume object can be embedded in an existing page. The user can drag in a new costume, or a script can add it. We need to catch the insertion of the new costume. The new costume is often front-most in the background costume. Its Z-position will be saved in the normal way at the next page turn.

The new costume is automatically page-specific. Force the name of the object to be unique and a legal selector. If the page is generic, create a new light-weight class and replace the page with an instance of it. Create a variable for the costume in the page object and create page accessor methods. If it has per-page data, such as text, create a contents variable and accessors.

There are menu items to move an object from page-specific to background and vice versa.

To move a costume to the background: It may be a field. Force the name of the costume to be unique and a legal selector. Create a variable for it in the background object and create accessor methods. Create page accessor methods. If it has per-page data, such as text, create a contents variable and accessors and add the variable to all page-objects of this background.

A special kind of text field has identical label text on all pages. The text in the field will be a shared label or title in this background. Mark the costume as staticFieldForText. Force the name of the costume to be unique and a legal selector. Create a variable for it in the background object and create accessor methods. Create page accessor methods.

Creating a Stack

Every costume that is not a background has the menu item, 'be a page in a new stack'. When this is chosen, create stack, background, and page objects. Store the costume as the background costume. Store a pointer to the background object into the costume. Set pageIndex to 1.

For each embedded costume, pretend it was just inserted as a new background object. Follow the instructions 'To move a costume to the background:' on the previous page.

Deleting an Object, Field, Page or Background

An embedded costume can be removed by dragging it out, by a menu item or by a script. Catch the removal. Record that the stack is modified.

Page-specific costume. Remove its variable and contents varaible from the page. If this is the last page-specific item, convert the page to a generic page. Remove the pageSpecific property in case the removed costume is used again.

Costume embedded in the background. Ask the user 'This costume part is on several pages. Really delete it from all pages of this background?' Remove its variable from the background. If it is a field with content data, remove its contents variable from all pages of the background.

Remove a page. Ask the user, 'Really erase the information on this page?' If the stack only has one page, tell the user to use 'Delete Stack'. If this is the last page of this background, ask 'This is the last page in this background. Really delete all fields of this background?' The stack removes the page from pageArray and shortens it. If the page has page-specific object, remove its light-weight class. Show the next page.

Remove a background. Ask user, 'Really delete all pages of this background?' The stack removes all pages of this background from pageArray and shortens it. Remove all their page-specific light-weight classes. Remove the background class. Show the next page.


Reorder Pages, Duplicate Pages

The menu item 'Organize Pages' opens a stack organizer window that is not part of the stack. Thumbnail images of the pages are arranged in order in a large rectangle. The user can drag them to new positions. The done button causes the pageArray to change the order of the pages to match the order of the images.

'Cut Page' removes the page and holds it in the paste buffer. The page needs to be encoded so that the native system can accept it in the buffer.

'Copy Page' If the page is generic, duplicate its instance. If not, create a new light-weight class, make an instance, and duplicate each variable into it. Put the page into the paste buffer.

'Paste Page' Insert the page into pageArray after the current page. Show the new page. Dim this menu item when the paste buffer does not contain a DBJrPage.

Each page has a name, and can be referred to by name in a script. Names are not unique. A name refers to the next page in the stack of that name. Each page has a unique id number, which can be used to refer to a page from a script. When a page is duplicated, it is given a new unique id.

Likewise each background has a name and a unique id number.

Likewise each stack has a name and a randomly generated id number. We cannot guarantee that the stack id is unique.

Searching

The user can search the text of a stack in "HyperCard style". The search key is one or more fragments of words that are the beginnings of words. A page satisfies the search if all fragments correspond to the beginning of some word somewhere in the page. Thus, the key 'dyn jun' would be satisfied by any page which contained the words 'Dynabook' and 'Junior' somewhere on it in any order. Turn to the page and select one of the words. Notify the user if no page can be found in the stack.

Modify the user interface of the host system so that Control-f and Control-g are intercepted in any field that is within a page of a stack. Control-f brings up a find dialog for entering key fragments. Control-g finds again starting in the current page. Notify the user when the entire stack has been searched.

Other "fields" such as Image, Number and Date

A main feature of DynaBook Junior text in a background field. As explained already, the field looks the same on many pages, but has different text in it on each.

Other costumes can be parameterized in the same way.

An image field in the background can show a different image on each page.

A numeric field can show a different number on each page.

A date field can show a different date on each page.

A costume that understands the messages contents, contents: and defaultValue, and that appears in the list fieldTypes can have per-page data.

Saving on the Disk

When a stack has been marked modified for more than one minute, it is saved to the disk. If the user is typing or turning pages, saving is delayed. Saved stacks are deleted on a logarithmic schedule, so that every power of two minutes of age has one saved version of the stack.

The format for saving a stack is beyond the scope of this spec.

The End