Integrating programming languages with web browsers

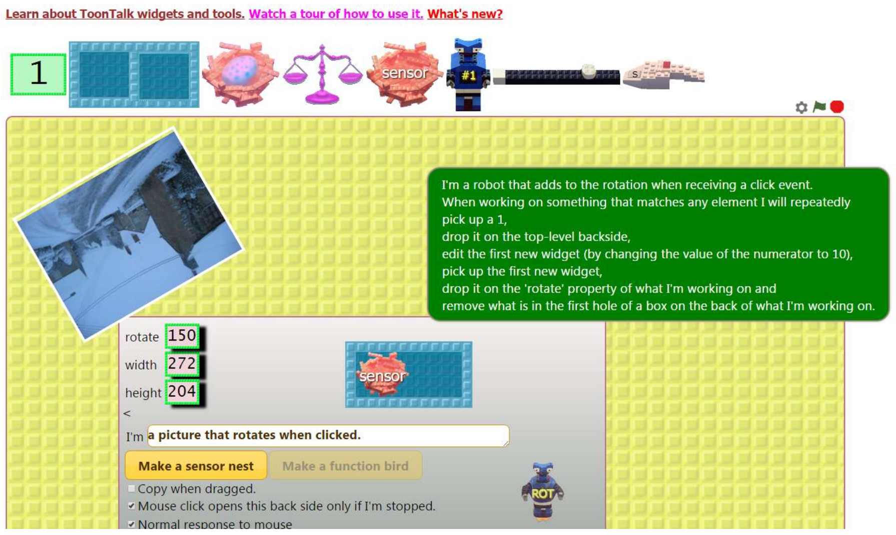
Ken Kahn, *toontalk@gmail.com*
Academic IT Services, University of Oxford

Abstract

Logo, Snap!, NetLogo, ToonTalk, and many other programming languages can run in modern web browsers without the need for plugins or installation. This paper is about the new possibilities this opens up: the integration with the wide range of functionality that browsers afford. Browsers support geometric transformations, animation, web storage, events, flexible styling, 2D and 3D graphics, drag-and-drop, multi-media elements, peer-to-peer communication, disability modes, cameras, microphones, and other sensors. Third-party libraries offer cloud storage, cloud publication, translation to over one hundred languages, and interfaces to social media sites.

A web interface to a programming language and environment can be implemented as web pages whose layout and styling can be customised for different users and contexts. Very different "looks and feels" can be created by users relying only upon HTML and CSS. Customised versions can be embedded into pages to produce interactive tutorials and documentation.

This paper describes ways that a programming language can exploit the web ecosystem. This is illustrated using ToonTalk Reborn, the web version of ToonTalk, as an example. ToonTalk Reborn is tightly integrated with browser elements, events, and styling. Any HTML element can be added to a ToonTalk web page and controlled by ToonTalk programs. Google Drive has been integrated to support sharing and publication of ToonTalk programs. Over one hundred language versions are available due to integration with Google Translate. Drag and drop to and from web pages and other applications supports sharing and saving programs, as well as importing media.


Learn about ToonTalk widgets and tools. Watch a tour of how to use it. What's new?
1
sensor
#1
sensor
ROT

I'm a robot that adds to the rotation when receiving a click event.
When working on something that matches any element I will repeatedly
pick up a 1,
drop it on the top-level backside,
edit the first new widget (by changing the value of the numerator to 10),
pick up the first new widget,
drop it on the 'rotate' property of what I'm working on and
remove what is in the first hole of a box on the back of what I'm working on.

rotate 150
width 272
height 204
<
I'm a picture that rotates when clicked.
Make a sensor nest
Make a function bird
Copy when dragged.
✓ Mouse click opens this back side only if I'm stopped.
✓ Normal response to mouse

Figure 1. Making an image interactive in ToonTalk

Keywords

Web, ToonTalk, programming languages, HTML, DOM, constructionist environments, web programming

Opportunities in the Web Ecosystem

This paper takes as a starting point that programming languages and their programming environments can be implemented to run plugin-free in modern web browsers. Logo (github.com/inexorabletash/jslogo), Snap! (snap.berkeley.edu), NetLogo (netlogoweb.org), and ToonTalk (github.com/ToonTalk/ToonTalk/wiki) are examples[1] from the constructionist community. Nothing need be installed to use them. This is increasingly important as students and teachers lack administrator privileges. System administrators have many fewer security and maintenance concerns with browser-based programs than with executable programs. When a browser-based programming language is combined with cloud storage, users can easily move between computers and operating systems. This is all very important but this paper is about how there is much more that web browsers offer that could be exploited to make programming environments more powerful, more flexible, and more configurable. And yet this is rarely done.

Web Pages

Programming environments of web-based programming languages live on web pages. In most cases the entire page is devoted to the programming environment. Embedding the programming environment in only a part of a page can be very useful. The page can be a tutorial or part of a manual, and the reader can interactively explore the aspect of the language being discussed *in situ*. For example, the Joy and Beauty of Computing MOOC (www.edx.org/course/beauty-joy-computing-cs-principles-part-uc-berkeleyx-bjc-1x) experimented with embedding instances of Snap! in lesson pages. Embeddability enables the creation of more effective interactive learning resources as well as enabling learners to create active essays which blend writing and interactive programs.

Ideally the programming environment can be customised for different embedded uses. Subsets of the full functionality of the language could be provided when appropriate. Half-baked programs (Kynigos, 2007) might be embedded in lessons. The programming interface may be altered to use less screen real estate when sharing the screen with text or images.

If the language is implemented well, customisations should be performable by anyone with sufficient HTML and CSS skills. Different configurations of the programming environment could be designed for users of different ages or abilities (including those with disabilities). Prior to the emergence of web-based implementations of programming environments, such customisations would require knowledge of the language implementation's architecture and programming language. A well-designed web-based implementation with a clear separation between structure (HTML), style (CSS), and interactivity (JavaScript) makes radical customisation possible by many. This could be used to implement version tuned to learners with disabilities or different skill or age levels. In some cases this requires only changes to the CSS styling.

HTML Elements

Web pages consist of HTML elements that include rich text, images, video, audio, links, and interface widgets such as buttons, input areas, and menus. Web-based implementations of programming languages *use* these elements but don't provide interfaces for users to program these elements directly. Typically HTML elements can only be given behaviours using JavaScript. In principle browser-based languages could be implemented to give behaviours to elements. Logo, Snap!, and NetLogo have not been implemented to do so and cannot be used by students to create general interactive web pages.

CSS Styling

If user programs incorporate HTML elements, then the CSS styling of those elements should be controllable by user programs. If, for example, the CSS properties *left*, *top*, *width*, and *height* are

[1] Scratch runs in browsers that support Flash and shares some of the web features discussed here but Snap! being a pure web-based implementation of a superset of Scratch is a better example for the purposes of this paper.

available to user programs, then those programs can alter those properties to move or scale an element. Hundreds of element properties could be made available (www.w3.org/TR/DOM-Level-2-Style/css.html#CSS-ElementCSSInlineStyle). Some properties describe transformations such as rotation or skew. Others define animated transitions. A programming language that provides an interface to element style properties gives the programmer a powerful tool for changing an element's size, colour, font, and much more. The implementers of a constructionist language can ride the wave of well-implemented new browser features with little effort.

DOM Events

A wide range of browser events are available for JavaScript programs to respond to. A web-based implementation of a programming language that provides an interface to these events enables its users to construct programs that respond to the keyboard, mouse, touch devices, media streaming, sensors and changes to interface widgets.

Browsers also support *custom events*. A programming language implementation can define new events that are appropriate for the language's computation model. For example, turtles could signal events such as reaching the edge of the screen.

Drag and Drop

Browsers can respond to drag and drop events. A web application can be programmed to respond to drops of files, URLs, rich text, and custom items. A dropped file can be an image, audio, video, or rich text. This can provide a very powerful easy-to-use means for users to import resources they find on the web into their programs. If a web-based programming language implements drag and drop of its primitive widgets then they can be dragged between browser tabs or windows. Many applications (e.g. Microsoft's WordPad) also support this so widgets can be dragged to a text file, saved or emailed, and later dragged into another instance of the programming environment. A language's support for drag and drop between applications can of course also be used within the language's programming environment and made available to programs constructed in the language.

Canvas and WebGL Graphics

Modern browsers support the construction of *canvases* where a full range of 2D and 3D graphics primitives are available. Browsers typically implement these primitives using the computer's graphics hardware and hence are usually very high performance. While very powerful, many find these APIs difficult to master. A programming language might still benefit beginning students by integrating this functionality since beginners could use higher-level, conceptually simpler primitives implemented by user code that interfaces to canvases directly.

Devices and Sensors

Web browsers provide access to devices and sensors. For example, they can access the camera to integrate photos and videos in a user's program. Web pages that attempt to use devices such as the camera or microphone ask permission before proceeding. While this can be awkward, it provides the user of the program assurance that their privacy will be respected.

Security Issues

Administrators, parents, and even students are rightly concerned about the possibility that they will run some malware that might cause damage, steal secrets, or violate someone's privacy. While not perfect, browsers are probably more secure than most other programs. Much of today's economy relies upon people trusting their browsers. Programming languages implemented in a browser are just as secure as the browser itself. A consequence is that web-based programming languages are severely limited in what operations on the file system are permitted.

There are efforts to define a secure subset of JavaScript (Miller et al, 2013). If a programming language is implemented in this subset then it opens up the possibility that programs written in that language can securely cooperate.

Storage, Sharing, and Publishing

The consequences of the limitations on access to the file system is somewhat remedied by the ubiquity of cloud storage. Instead of relying upon the local machine's file system, users can rely upon their Google Drive, Dropbox, or OneDrive cloud storage. These services provide APIs that language implementers can use so that users need not concern themselves with the technical details of the system. Alternatively, the organisation behind the web-based programming language could provide their own server storage as, for example, the University of California does for Snap! users. When cloud storage is inappropriate, programs can be saved in the browser's *web storage*. It typically provides 50MB of storage that is private to each web application.

Cloud storage greatly simplifies sharing and publishing programs. If the user's own cloud storage is used they can keep it private, share it by URL access or user identities, or make it public to the world. And in some cases (e.g. Dropbox) a URL can be obtained of a public file that the browser will render as a web page instead of downloading it.

Web Workers

Web workers is a browser standard that enables programs to run concurrently on multi-core computers. This ability could be used by browser-based languages to enable users' programs to run significantly faster.

WebRTC

WebRTC (developer.mozilla.org/en-US/docs/Web/API/WebRTC_API) supports peer-to-peer communication between browsers. In addition to data, it supports streaming of audio and video. Currently it is supported by Chrome, Firefox, the Android browser, and Opera. Microsoft is working on a compatible variant but it is unclear if and when Apple's Safari will support it. It could be used to support collaborative programming and the construction of multi-player programs.

Browser Features

The interface of web browsers is familiar to billions of people. The difficulties of learning and using a web-based programming language can be reduced by relying upon the browser's interface. People already know how to navigate within and between web pages, how to zoom in and out of pages, how to bookmark pages, how to use browsers' spell checkers, etc. There is a price to pay, however: for example, right mouse button clicks bring up a browser-specific menu. This can be overridden but not without interfering with the normal operation of the browser.

Browsers maintain a history that users can navigate using back and forward buttons. Initially this history was limited to the history of web pages visited but there is now an API for programs to extend this. One could use it as an interface to an undo/redo functionality.

Third Party APIs

In addition to the functionality that browsers provide, there is a vast number of web services that can be accessed from browsers. Google, for example, provides cloud storage, translations, maps, data visualisations, real-time collaboration, communication with household devices and toys, and much more. A web-based implementation of a programming language can integrate any of these services.

There are many open data sources from government agencies and research labs (e.g. environment, astronomy, or climate). A programming language can provide abstractions that hide many of the technical details involved in accessing these data sources.

ToonTalk Reborn

ToonTalk (Kahn, 1996) was conceived in 1992 with the goal of using video game technology to bring the most advanced computer science research in programming languages to a wide audience. The core idea is that each abstraction in the computation model has a corresponding *concretization* that captures the fundamental properties of the computational abstractions. These concretizations can be playful and understood in their own terms, even by very young children. The computation model chosen was *concurrent constraint programming* (Saraswat, 1993) which provides a safe well-grounded way of introducing concurrency, communication, and synchronisation to programs. The other core idea underlying ToonTalk is the construction of programs by demonstration followed by the removal of details for abstraction (Kahn, 2001).

From Desktop to Web

As described in (Kahn, 2014) ToonTalk Reborn is a re-implementation and re-conceptualisation of desktop ToonTalk for the web. The desktop ToonTalk relies upon many of Microsoft Windows APIs. It uses DirectDraw for 2D graphics, DirectInput for input events, DirectSound for audio, and DirectPlay for peer-to-peer communication. Consequently desktop ToonTalk only runs on Microsoft Windows and exists only as a large executable that must be installed and trusted.

Desktop ToonTalk is implemented in C++ and performance was a critical requirement due to its animated game-like interface. A web-based implementation was inconceivable until the relatively recent adaption of HTML5 and high-performance implementation of JavaScript by modern browsers that makes rich and powerful browser-based programming systems feasible.

Just a Web Page

Any web page can be turned into a ToonTalk Reborn page by including a few lines of HTML that specifies the necessary JavaScript and CSS. ToonTalk widgets (numbers, boxes, birds, nests, scales, robots, and elements) and work areas can be anywhere on the page. The page that implements the default programming environment includes a single work area and an array of widgets that acts like a tool palette (see Figure 2). Straight-forward edits to the HTML of this page can produce very different environments with different tool palettes and layouts.

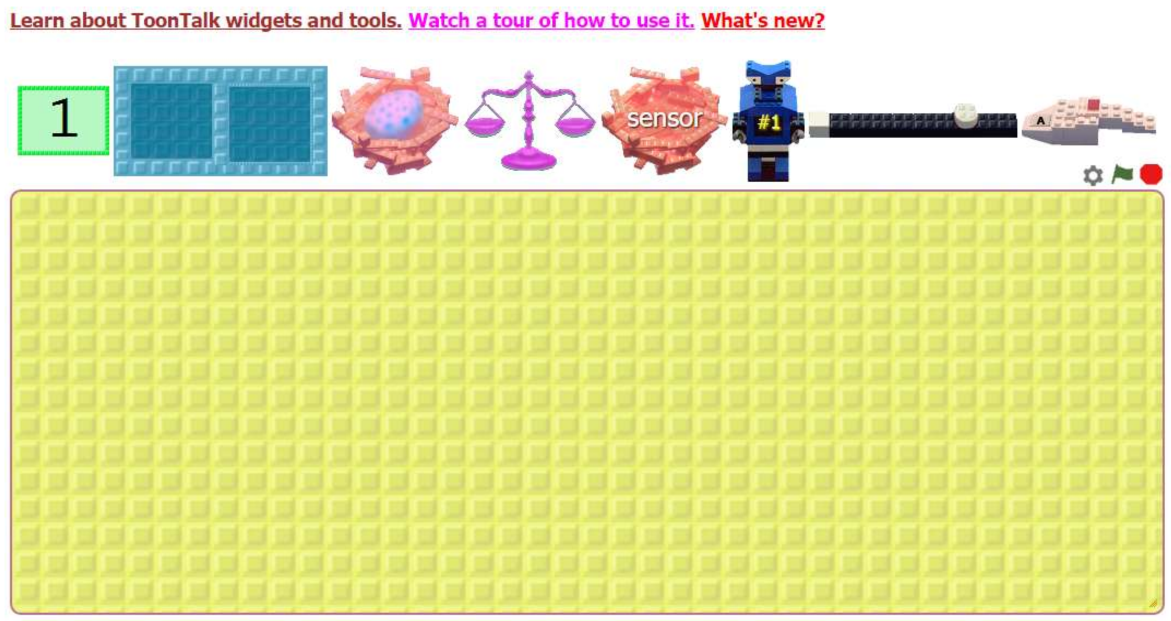

Learn about ToonTalk widgets and tools.
Watch a tour of how to use it.
What's new?
1
sensor
#1
A

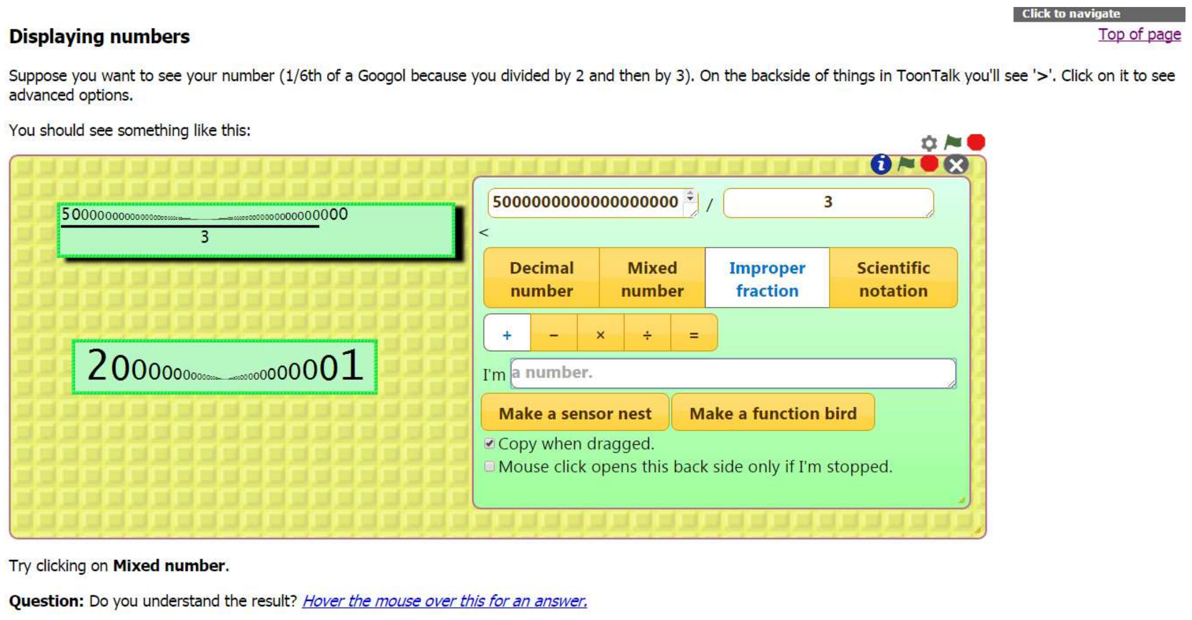
Figure 2. The default ToonTalk Reborn programming environment

A typical ToonTalk Reborn manual page contains many live instances of the widget being documented. Each instance is customised as needed. Small specialised workspaces are available on manual pages to facilitate the exploration of the widget being documented (see Figure 3).

The ToonTalk Reborn CSS style sheet defines more than the colours and fonts of the interface. Most of the widgets can be animated in multiple ways (e.g. a bird hatching, a bird flying in any of eight directions, a bird waiting to be given something). The CSS defines the images and how they are animated. A graphic designer could leave all the JavaScript implementing ToonTalk untouched and create a different CSS with different imagery and animation transitions to give the system a very different 'skin'.

HTML Elements in ToonTalk

Any HTML element can be added to a ToonTalk program. One can drag images, sounds, videos, plain text, rich text, links, and any other HTML element from any web page to a ToonTalk work area. Element widgets, like all other widgets, can be placed in boxes, given to birds, copied, etc. Element widgets can be composed hierarchically (e.g. ball and paddle images can be added to a Pong background image).



Click to navigate

Top of page

Displaying numbers

Suppose you want to see your number (1/6th of a Googol because you divided by 2 and then by 3). On the backside of things in ToonTalk you'll see '>'. Click on it to see advanced options.

You should see something like this:

[✓] Copy when dragged.
[ ] Mouse click opens this back side only if I'm stopped.

Try clicking on **Mixed number**.

Question: Do you understand the result? *Hover the mouse over this for an answer.*

Figure 3. A portion of the manual page describing numbers

The backside of a widget contains a control panel for that widget. An element widget control panel includes access to the CSS style of that element. One can select style properties including those that control the location, dimensions, colour, font, and transformations (rotation, skew, etc.). An property widget of an element widget is a number widget or plain text widget with two additional aspects: (1) it displays the up-to-date value of that property and (2) changes to its value become changes to its style (see Figure 4).

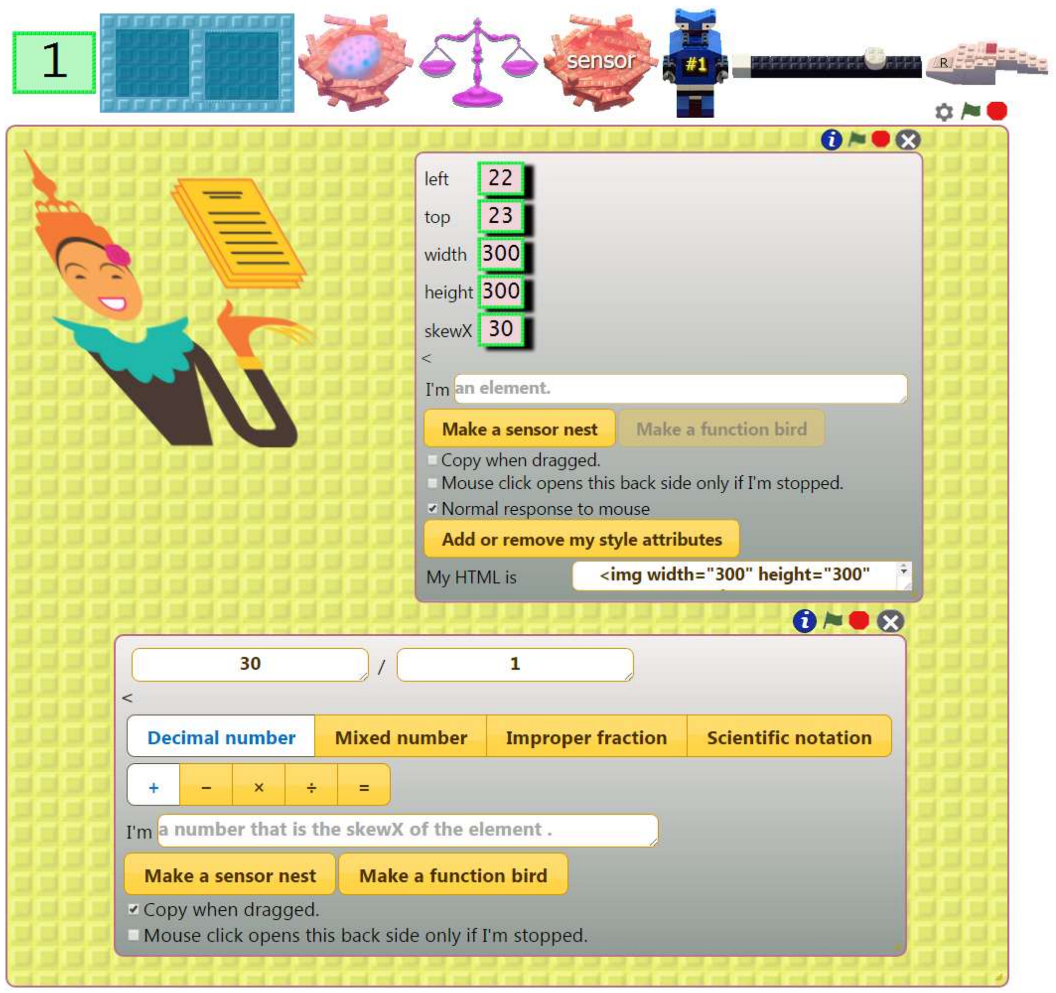
DOM Events in ToonTalk

Communication and coordination in ToonTalk is provided via the metaphor of birds who take messages (widgets) to their nest. Robots who encounter such nests wait for something to arrive before proceeding. ToonTalk sensors are represented by nests whose bird is given an event message "off screen" by the browser. For example, a sensor could be told to listen to 'keydown' events; when they occur, the bird will take a plain text element widget with the description of which key was pressed to the sensor nest. Sensors can also be associated with a widget to, for example, respond to 'click' events on the widget. Custom events are also dispatched in addition to the DOM events that are standard to all browsers. For example, an event is triggered when a ToonTalk widget is added to another widget.

The Many Uses of Drag and Drop in ToonTalk

HTML5 supports the transfer of data when a user drags and drops items. ToonTalk Reborn uses this to transfer the JSON of a widget so the widget can be reconstructed in another browser running ToonTalk. Some applications support the drop of a ToonTalk widget by inserting the text of the JSON. For example, Microsoft WordPad can be used to both receive and send ToonTalk widgets. Selecting the JSON string and dragging it to ToonTalk causes the widget to be reconstructed. This can be used as a way to “manually” save and share ToonTalk programs.

ToonTalk is also able to receive drops of plain text, files (images, audio, video, and text), elements, and URLs. For example, the image in Figure 4 was creating by dragging an image from the Constructionism 2016 website to ToonTalk.



left 22
top 23
width 300
height 300
skewX 30
<
I'm an element.
Make a sensor nest | Make a function bird
☐ Copy when dragged.
☐ Mouse click opens this back side only if I'm stopped.
☑ Normal response to mouse
Add or remove my style attributes
My HTML is <img width="300" height="300"

30 / 1
<
Decimal number | Mixed number | Improper fraction | Scientific notation
+ | − | × | ÷ | =
I'm a number that is the skewX of the element .
Make a sensor nest | Make a function bird
☑ Copy when dragged.
☐ Mouse click opens this back side only if I'm stopped.

Figure 4. An image element widget, its backside, and the skewX property widget backside

Saving and Publishing

ToonTalk Reborn, if granted permission, will use the user’s Google Drive to save programs. Optionally, they are updated automatically as the user changes his or her program. The files are initially private but the user can share or make them public using the Google Drive web interface. Support for other cloud storage providers (Dropbox and OneDrive) is planned.

Programs can also be saved automatically to the web storage associated with the browser. This is particularly important when network connections are not available (ToonTalk Reborn runs fine if its files have been copied to the local disk).

ToonTalk also supports publishing programs to the web. A ToonTalk web page is created containing the widgets in the current workspace. The user can add and edit rich text around those widgets. New widgets can be dropped on these pages. This functionality relies upon Google Drive's support for serving saved documents as web pages which, unfortunately, Google announced it is stopping in 2016. Dropbox continues to support this functionality and will replace Google Drive for publishing web pages with embedded ToonTalk elements. Alternatively a proxy server could fetch pages from Google Drive and serve them as web pages.

Translation

Google Translate can be associated with a dynamic web page. As the page changes under program control text can be translated to the desired language. It is possible to customise the translations of some phrases and to crowdsource corrections.

Future Developments

ToonTalk Reborn has yet to be integrated with the 2D and 3D graphics of canvases and WebGL. Logo turtles have been implemented as a user program in ToonTalk Reborn but there is no fully general way to implement a trail left behind when the turtle's pen is down. After canvases are integrated with ToonTalk this should be relatively easy.

Desktop ToonTalk supports *long-distance birds* that can deliver widgets to nests on other computers. This is used for collaboration and implementing multi-player games. Long-distance birds could be implemented using WebRTC.

Google's real-time API for Google Drive (developers.google.com/google-apps/realtime/overview) could be used to enable simultaneous collaboration when programming ToonTalk. It could enable simultaneous editing of shared programs in a manner similar to collaborative editing of Google Docs.

ToonTalk has yet to be integrated with devices and sensors. Such integration would enable ToonTalk programs to access cameras, microphones, game pads, and smart phone sensors.

Lessons for other Languages

This paper has argued that web browsers offer programming language implementers a wide range of functionality that has mostly been ignored. ToonTalk Reborn was presented as a case study of how some of this functionality could be exploited. Many of the lessons here could be applied to make richer and more powerful versions of Snap!, Logo, NetLogo, and other constructionist programming languages.

References

Kahn, K. (1996) *ToonTalk - An Animated Programming Environment for Children*. Journal of Visual Languages and Computing, June. pp 197–217.

Kahn. K. (2001) *Generalizing by Removing Detail: How Any Program Can Be Created by Working with Examples*, Communications of the ACM, March 2000, pp 21-43. Long version in *Your Wish Is My Command: Programming by Example*, edited by H. Lieberman, Morgan Kaufmann, February.

Kahn, K. (2014) *TOONTALK REBORN - Re-implementing and re-conceptualising ToonTalk for the Web.* Constructionism 2014. Vienna. August.

Kynigos, C. (2007) *Half-Baked Logo Microworlds as Boundary Objects in Integrated Design*. Informatics in education. Volume 6 Issue 2, January. pp 335-358.

Miller, M., Van Cutsem, T., and Tulloh, B. (2013) *Distributed Electronic Rights in JavaScript*. 22nd European Symposium on Programming. Rome. Springer. March. pp 1-20

Saraswat, V. (1993) *Concurrent Constraint Programming*, MIT Press, March.

The source code and the web application are available at github.com/ToonTalk/ToonTalk/wiki.